CMR ENGINEERING COLLEGE

KANDLAKOYA (V), MEDCHAL (M), HYDERABAD

STEP MATERIAL ON JAVA PROGRAMMING

Subject : Java Programming

Year: II Year/II Sem

Academic Year:2020-2021

INSTITUTE VISION AND MISSION

VISION

To be recognized as a premier institution in offering value based and futuristic quality technical education to meet the technological needs of the society

MISSION

• To impart value based quality technical education through innovative teaching and learning methods

• To continuously produce employable technical graduates with advanced technical skills to meet the current and future technological needs of the society

• To prepare the graduates for higher learning with emphasis on academic and industrial research

DEPARTMENT VISION AND MISSION

VISION

To produce globally competent and industry ready graduates in Computer Science & Engineering by imparting quality education with a know-how of cutting edge technology and holistic personality

MISSION

- To offer high quality education in Computer Science & Engineering in order to build core competence for the graduates by laying solid foundation in Applied Mathematics, and program framework with a focus on concept building
- The department promotes excellence in teaching, research, and collaborative activities to prepare graduates for professional career or higher studies
- Creating intellectual environment for developing logical skills and problem solving strategies, thus to develop, able and proficient computer engineer to compete in the current global scenario

PART –A
UNIT WISE SHORT QUESTION AND ANSWERS
UNIT-I

1. **What is data abstraction? [2M]

Ans:

Abstraction *refers to the act of representing essential features without including background details or explanations* **(or)** *Abstraction is a process of hiding unnecessary information*

An essential element of object-oriented programming is abstraction. Humans manage complexity through abstraction. For example, people do not think of a car as a set of tens of thousands of individual parts. They think of it as a well-defined object with its own unique behavior. They can ignore the details of how the engine, transmission, and braking systems work. Instead, they are free to utilize the object as a whole.

A powerful way to manage abstraction is through the use of hierarchical classifications. This allows you to layer the semantics of complex systems, breaking them into more manageable pieces. From the outside, the car is a single object. Once inside, you see that the car consists of several subsystems: steering, brakes, sound system, seat belts, heating, cellular phone, and so on.

Hierarchical abstractions of complex systems can also be applied to computer programs. The data from a traditional process-oriented program can be transformed by abstraction into its component objects

2. Explain the feature of Java [3M]

Ans: A list of most important features of Java language is given below.

1. Simple
2. Object-Oriented
3. Platform independent
4. Secured
5. Robust
6. Architecture neutral
7. Portable
8. Dynamic
9. Interpreted
10. High Performance

11. Multithreaded
12. Distributed

- ***Simple:***

Java language is simple because:

1) Syntax is based on C++ (so easier for programmers to learn it after C++).
2) Removed many confusing and/or rarely-used features
   Ex.; explicit pointers, operator overloading etc.
3) No need to remove unreferenced objects ,to do this Automatic Garbage Collection in java.

- ***Object-oriented :***

Object-Oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour. Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some principles. They are :

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

- ***Platform Independence :***

Java provides software-based platform. The Java platform differs from most other platforms in the sense that it's a software-based platform that runs on top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

Java code can be run on multiple platforms e.g.Windows,Linux,SunSolaris,Mac/OS etc. Java code is compiled by the compiler and converted into bytecode.*This bytecode is a platform independent code because it can be run on multiple platforms i.e.* ***Write Once and Run Anywhere(WORA).***

- ***Secured:***

Java is secured because

1. There is no explicit pointers.
2. Programs run inside virtual machine sandbox.

- ***Robust :***

Robust simply means strong. Java uses strong memory management. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points makes java robust.

- ***Architecture-neutral*:**
  There are no implementation dependent features Ex.;size of primitive types is set
- ***Portable :***
  We may carry the java bytecode to any platform.
- ***High-performance:***
  Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)
- ***Distributed :***
  We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet**.**
- ***Multi-threaded:***
  A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it shares the same memory. Threads are important for multi-media, Web applications etc.

3. Explain the types of operators used in java [3M]

Ans: Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups −

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Misc Operators

The Arithmetic Operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators

Examples: +(addition) ,-(substaction), *(multiplication), /(division), %(modulo)

The Relational Operators

There are following relational operators supported by Java language.

Examples: == (equal to), != (not equal to),> (greater than), < (less than), >= (greater than or equal to), <= (less than or equal to)

The Bitwise Operators

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.

Example: & (bitwise and), | (bitwise or), ^ (bitwise XOR), ~ (bitwise compliment), << (left shift), >> (right shift), >>> (zero fill right shift)

The Logical Operators

The lists the logical operators −

Examples: && (logical and), || (logical or), ! (logical not)

The Assignment Operators

Following are the assignment operators supported by Java language –

=,+=,-=,*=,/=,%=,<<=,>>=,&=,^=,|=

Miscellaneous Operators

Miscellaneous operators includes

- *Conditional Operator (? :)*
- instanceof Operator

4. **What is static inner class(or) difference static inner and non static inner classes** [3M]

There are two differences between static inner and non static inner classes.

In case of declaring member fields and methods, non static inner class cannot have static fields and methods. But, in case of static inner class, can have static and non static fields and method.

The instance of non static inner class is created with the reference of object of outer class, in which it has defined, this means it has enclosing instance. But the instance of static inner class

is created without the reference of Outer class, which means it does not have enclosing instance.

See this example

```
class A
{
   class B
   {
      // static int x; not allowed here
   }

   static class C
   {
      static int x; // allowed here
   }
}

class Test
{
   public static void main(String… str)
   {
      A a = new A();

      // Non-Static Inner Class
      // Requires enclosing instance
      A.B obj1 = a.new B();

      // Static Inner Class
      // No need for reference of object to the outer class
      A.C obj2 = new A.C();
   }
}
```

5. List string manipulation functions of Java String class. [3M]

Method	Description
int length()	returns string length
static String format(String format, Object... args)	returns formatted string
String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index
static String join(CharSequence delimiter, CharSequence... elements)	returns a joined string
boolean isEmpty()	checks if string is empty
String concat(String str)	concatenates specified string
String toLowerCase()	Returns string in lowercase.
String toUpperCase()	Returns string in uppercase.
String trim()	Removes beginning and ending spaces of this string.
static String valueOf(int value)	Converts given type into string. It is overloaded.

6. Explain the use of 'final' keyword. [3M]

The final keyword in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

- variable
- method
- class

if a variable is made as final it cannot change its value

if a method is made as final it cannot override it.

If a class is made as final it cannot be extended by another class

Program to demonstrate using final with inheritance

```
final class Bike
{

}

class Honda1 extends Bike    //error since Bike is final we can't inherit it  properties
{
  void run()
{
System.out.println("running safely with 100kmph");
}

  public static void main(String args[]){
  Honda1 honda= new Honda();
  honda.run();
  }
}
```

7. Differentiate between class and object. [2M]

A list of differences between object and class are given below:

s.No.	Class	Object
1)	Class is a **blueprint or template** from which objects are created.	Object is an **instance** of a class.
2)	Class is a **group of similar objects**.	Object is a **real world entity** such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.
3)	Class is a **logical** entity.	Object is a **physical** entity.
4)	Class is declared using **class keyword** e.g. class Student{}	Object is created through **new keyword** mainly e.g. Student s1=new Student();
5)	Class is declared **once**.	Object is created **many times** as per requirement.
6)	Class **doesn't allocated memory when it is created**.	Object **allocates memory when it is created**.
7)	There is only **one way to define class** in java using class keyword.	There are **many ways to create object** in java such as new keyword, newInstance() method, clone() method, factory method and deserialization.

Let's see some real life example of class and object in java to understand the difference well:
Class: Human Object: Man, Woman
Class: Fruit Object: Apple, Banana, Mango, Guava wtc.

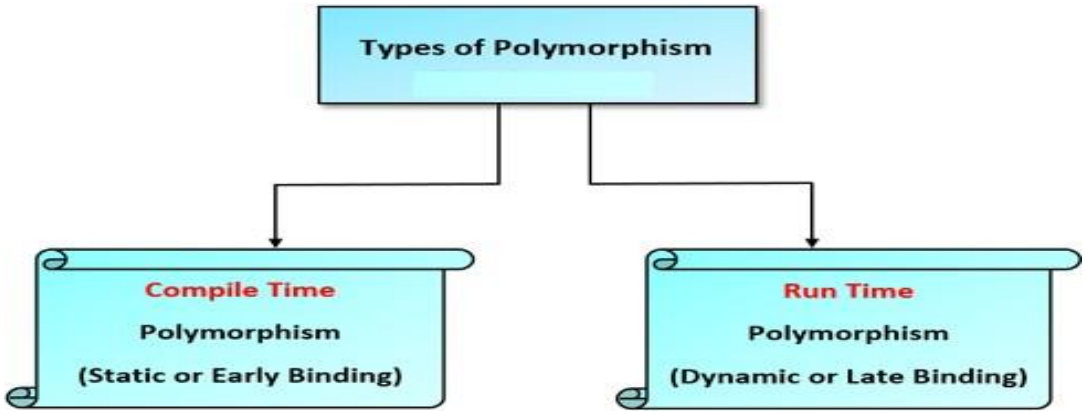
Class: Mobile phone Object: iPhone, Samsung, Moto

Class: Food Object: Pizza, Burger, Samosa

8. **What is meant by ad-hoc polymorphism?(or)What is polymorphism [3M]

Ans:
Polymorphism in Java is a concept by which we can perform a *single action in different ways* There are two types of polymorphism in Java:

1. Compile-time polymorphism or Ad hoc polymorphism
2. Runtime polymorphism. Or Pure polymorphism



Compile -time polymorphism or Ad hoc polymorphism:

Ad hoc polymorphism is also known as function overloading or operator overloading because a polymorphic function can represent a number of unique and potentially different implementations depending on the type of argument it is applied to.

The term ad hoc in this context is not intended to be pejorative; it refers simply to the fact that this type of polymorphism is not a fundamental feature of the type system . Example of Ad hoc polymorphism *i)operator overloading :*ii)Method Overloading:

9. What is abstract class? Give example. **[2M]**

Ans:

Abstract class in Java

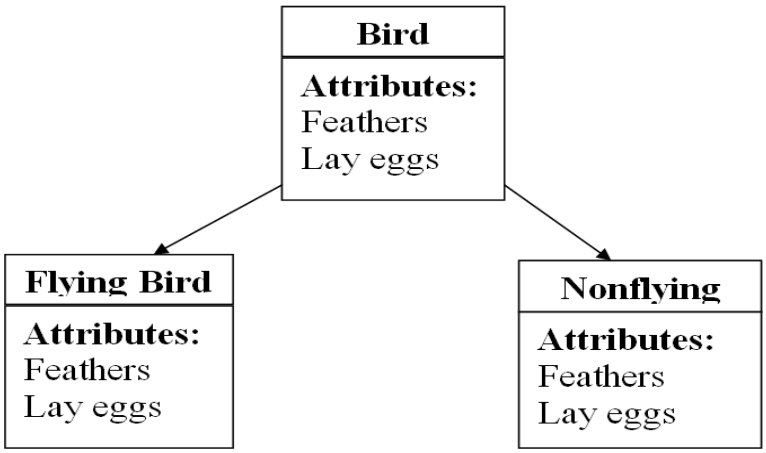
- A class that is declared with ***abstract*** keyword, is known as abstract class in java.
- It can have abstract and non-abstract methods (method with body).
- It needs to be extended and its method implemented.
- It cannot be instantiated.

Example abstract class

```
abstract class Hello
{
        abstract void hai();
}
abstract class Demo extends Hello
{
}
class Welcome  extends Demo
{
        void hai()
        {
                System.out.println("hello");
        }
}
public class AbstractClass {
public static void main(String args[])
{
        //A oa=new A();//error because for abstract class we can't create object
        //Demo ob=new Demo();//error because for abstract class we can't create object
        //ob.hai();//error
        Welcome oh=new Welcome();
        oh.hai();
}
}
```

10. What is inheritance? Give example. **[2M]**

Inheritance *is the process by which objects of one class acquire the properties of objects of another class. Inheritance supports the concept of hierarchical classification as shown below.*



11. Define the basic characteristics of object oriented programming. **[3M]**

The characteristics of OOP are:

Class definitions – Basic building blocks OOP and a single entity which has data and operations on data together

Objects – The instances of a class which are used in real functionality – its variables and operations

Abstraction – Specifying what to do but not how to do ; a flexible feature for having a overall view of an object's functionality.

Encapsulation – Binding data and operations of data together in a single unit – A class adhere this feature

Inheritance and class hierarchy – Reusability and extension of existing classes

Polymorphism – Multiple definitions for a single name - functions with same name with different functionality; saves time in investing many function names Operator and Function overloading

Generic classes – Class definitions for unspecified data. They are known as container classes. They are flexible and reusable.

Class libraries – Built-in language specific classes

Message passing – Objects communicates through invoking methods and sending data to them. This feature of sending and receiving information among objects through function parameters is known as Message Passing.

12. Explain the use of 'for' statement in Java with an example. [3M]

```
for (initialization; condition; increment/decrement)
 {
   statement;
}
```

```
class forLoopTest
 {
   public static void main(String args[])
{
      for (int j = 1; j <= 5; j++)
         System.out.print(j);
   }
}
```

Output:

1 2 3 4 5

For-Each Loop :

For-Each loop is used to traverse through elements in an array. It is easier to use because we don't have to increment the value. It returns the elements from the array or collection one by one.

Example:

```
class foreachDemo {
 public static void main(String args[]) {
  int a[] = {10,15,20,25,30};
   for (int i : a) {
      System.out.print(i);
   }
 }
```

Output:

10 15 20 25 30

13. What is the significance of Java's byte code? [2M]

Javac not only compiles the program but also generates the byte code for the program. Java byte code is the instruction set for the Java Virtual Machine. It acts similar to an assembler which is an alias representation of a C++ code. As soon as a java program is compiled, java byte code is generated. In more apt terms, java byte code is the machine code in the form of a .class file. With the help of java

byte code we achieve platform independence in java.

The set of instructions for the JVM may differ from system to system but all can interpret the byte code. A point to keep in mind is that byte codes are non-runnable codes and rely on the availability of an interpreter to execute and thus the JVM comes into play.

byte code implementation makes Java a **platform-independent** language. This helps to add portability to Java which is lacking in languages like C or C++. Portability ensures that Java can be implemented on a wide array of platforms like desktops, mobile devices, severs and many more. Supporting this, Sun Microsystems captioned JAVA as *"write once, read anywhere" or "WORA"* in resonance to the byte code interpretation

14. List the applications of object oriented programming. [3M]

Main application areas of OOP are:

- User interface design such as windows, menu.
- Real Time Systems
- Simulation and Modeling
- Object oriented databases
- AI and Expert System
- Neural Networks and parallel programming
- Decision support and office automation systems etc.

1. Client-Server Systems

Object-oriented Client-Server Systems provide the IT infrastructure, creating object-oriented Client-Server Internet (OCSI) applications. Here, infrastructure refers to operating systems, networks, and hardware. OSCI consist of three major technologies:

The Client Server
Object-Oriented Programming
The Internet

2. Object-Oriented Databases

They are also called Object Database Management Systems (ODBMS). These databases store objects instead of data, such as real numbers and integers. Objects consist of the following:

Attributes: Attributes are data that defines the traits of an object. This data can be as simple as integers and real numbers. It can also be a reference to a complex object.

Methods: They define the behavior and are also called functions or procedures.

3. Object Oriented Databases

These databases try to maintain a direct correspondence between the real-world and database objects in order to let the object retain their identity and integrity. They can then be identified and operated upon.

4. Real-Time System Design

Real time systems inherit complexities that makes difficult to build them. Object-oriented techniques make it easier to handle those complexities. These techniques present ways of dealing with these complexities by providing an integrated framework which includes schedulability analysis and behavioral specifications.

5. Simulation And Modelling System

It's difficult to model complex systems due to the varying specification of variables. These are prevalent in medicine and in other areas of natural science, such as ecology, zoology, and agronomic systems. Simulating complex systems requires modelling and understanding interactions explicitly. Object-oriented Programming provides an alternative approach for simplifying these complex modelling systems.

6. Hypertext And Hypermedia

OOP also helps in laying out a framework for Hypertext. Basically, hypertext is similar to regular text as it can be stored, searched, and edited easily. The only difference is that hypertext is text with pointers to other text as well.

Hypermedia, on the other hand, is a superset of hypertext. Documents having hypermedia, not only contain links to other pieces of text and information, but also to numerous other forms of media, ranging from images to sound.

7. Neural Networking And Parallel Programming

It addresses the problem of prediction and approximation of complex time-varying systems. Firstly, the entire time-varying process is split into several time intervals or slots. Then, neural networks are developed in a particular time interval to disperse the load of various networks. OOP simplifies the entire process by simplifying the approximation and prediction ability of networks.

8. Office Automation Systems

These include formal as well as informal electronic systems primarily concerned with information sharing and communication to and from people inside as well as outside the organization. Some examples are

Email
Word processing
Web calendars
Desktop publishing

9. CIM/CAD/CAM Systems

OOP can also be used in manufacturing and design applications as it allows people to reduce the effort involved. For instance, it can be used while designing blueprints, flowcharts, etc. OOP makes it possible for the designers and engineers to produce these flowcharts and blueprints accurately.

10. AI Expert Systems

These are computer applications which are developed to solve complex problems pertaining to a specific domain, which is at a level far beyond the reach of a human brain.

It has the following characteristics:Reliable,Highly responsive,Understandable,High-performance

15. Differentiate between break and continue statement. [2M]

ASIS FOR COMPARISON	BREAK	CONTINUE
Task	It terminates the execution of remaining iteration of the loop.	It terminates only the current iteration of the loop.
Control after break/continue	'break' resumes the control of the program to the end of loop enclosing that 'break'.	'continue' resumes the control of the program to the next iteration of that loop enclosing 'continue'.
Causes	It causes early termination of loop.	It causes early execution of the next iteration.
Continuation	'break' stops the continuation of loop.	'continue' do not stops the continuation of loop, it only stops the current iteration.
Other uses	'break' can be used with 'switch', 'label'.	'continue' can not be executed with 'switch' and 'labels'.

16. What is type casting? Explain with an example. **[3M]**

Ans:

The process of converting one data type to another data type is called as **Casting**

Types of castings:

There are two type of castings

1. Explicit type conversion (or) Narrowing conversion
2. Implicit type conversion (or) Widening conversion

1.Casting Incompatible Types or explicit type conversion or narrowing conversion

Casting larger data type into smaller data type may result in a loss of data.

This kind of conversion is sometimes called a ***narrowing conversion***, since you are explicitly making the value narrower.

2.Java's Automatic Conversions or implicit type conversion or widening conversion

When one type of data is assigned to another type of variable, an automatic type conversion

will take place if the following two conditions are met:

a. The two types are compatible.
b. The destination type is larger than the source type.

When these two conditions are met, a widening conversion takes place.

```
class typecastdemo
{
public static void main(String args[])
        {
        float f=3.141,x;
        int I,j=30;
        i=(int)f;//explicit conversion
        x=j;//implicit conversion
        System.out.println("i value is"+i);
        System.out.println("x value    is"+x);
        }
}
```

17. What is the use of super keyword? [2M]

The **super** keyword in java is a reference variable which is used to refer immediate parent class object.Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable

Usage of java super Keyword

1) super is used to refer immediate parent class instance variable.

2) super can be used to invoke parent class method

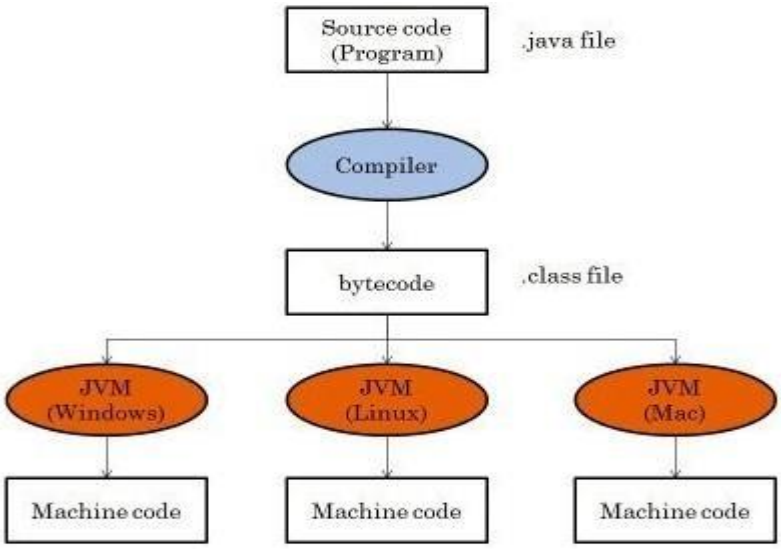
3) super is used to invoke parent class constructor

18. Why is Java known as platform independent? [3M]

Java provides software-based platform. The Java platform differs from most other platforms in the sense that it's a software-based platform that runs on top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

Java code can be run on multiple platforms e.g.Windows,Linux,SunSolaris,Mac/OS etc. Java code is compiled by the compiler and converted into bytecode.*This bytecode is a platform independent code because it can be run on multiple platforms i.e.* ***Write Once and Run Anywhere(WORA).***



19. What is the size of char data type? Why does it differ from C language? [3M]

In java char uses 2 byte in java because java uses unicode system rather than ASCII code system. \u0000 is the lowest range of unicode system.

Unicode System

Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages

UNIT-II

1. List the byte stream classes[3 M]

Ans:

Byte stream classes have been designed to provide functional features for creating and manipulating streams and files for reading and writing Bytes.

Java provides two Byte streams

1. **InputStream class**
2. **OutputStream class**

1. InputStream Class:

- It is an abstract class that defines model of streaming byte input.
- It implements ***Closeable interface***.
- It defines methods for perfoming input functions such as
  - Reading Bytes
  - Closing streams
  - Marking positions in streams
  - Skipping ahead in stream
  - Finding the number of Bytes in a streams

2. OutputStream class:

- It is an abstract class that defines model of streaming byte output.
- It implements ***Closeable and Flushable interface***.
- It defines methods for perfoming output functions such as
  - Writing bytes
  - Closing streams
  - Flushing streams

2. Explain about implicit and explicit import statement[3]

Imports can be categorized as explicit (for example import java.util.List;) or implicit (also known as 'on-demand', for example import java.util.*;):

Implicit imports give access to all visible types in the type (or package) that precedes the ".*"; types imported in this way never shadow other types.

Explicit imports give access to just the named type; they can shadow other types that would normally be visible through an implicit import, or through the normal package visibility rules.

3. How to create and access a package? Or
How to create and use a package in Java program? Or
Define a Package? What is its use in java? Explain. Or
How to define a package in Java?

A java package is a group of similar types of classes, interfaces and sub-packages.
Package in java can be categorized in two form,

i) built-in package and
ii) user-defined package

To create a user defined package in java we use a keyword "package"

Syntax:

```
package packagename[.subpackage1][.subpackage2]…[.subpackageN];
```

//let us save this program in D:/pack folder as A.java

```
package pack;
public class A
{
  public void msg()
      {
System.out.println("Hello");
      }
 }
```

Compile as: D:/pack>javac –d . A.java

```
import pack.*;
  class B
{
```

```
public static void main(String args[])
    {
 A obj = new A();
 obj.msg();
     }
}
```

Compile as: D:/pack>javac –d . A.java

//save it D floder as B.java

Output:Hello

4. Write about the random access file operations[3]

RandamAccessFile class supported by java.io package allows us to create files that can be used for reading and writing data with randam access.
This class implements DataInput,and DataOutput and Closeable Interfaces
It supports positioning request that means we can position the file pointer with in the file.
It has two constructors

RandamAccessFile(**File** *fileObj*,**String** access)throws FileNotFoundException
RandamAccessFile(**String** *filename*,**String** access)throws FileNotFoundException

In both cases,access determines what type of file access is permited**.**

Access	Purpose
r	File opened for reading purpose
rw	File opened for read-write purpose
rws	File opened for read-write purpose and every change to ythe file's data or ymetadata will be immediately written to physical device
rwd	File opened for read-write purpose and every change to ythe file's data will be immediately written to physical device

The method **seek()** is used to set the current position of the file pointer

Syntax:

void seek(long newPos)throws IOException

Here,***newPos*** specifies the new position,in bytes,of the file pointer from the beginning of the file.

5. Differentiate class, abstract class and interface. [2]

Concrete class	Abstract class	Interface
A class can have only non abstract methods.	1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
class **doesn't support multiple inheritance**	2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
class **can have final, non-final, static and non-static variables**.	3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
class **can extend a class and implementation of interface**.	4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
The class **keyword** is used to declare a class.	5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
Class can be initiated using new keyword	**Abstract Class can't be initiated using new keyword**	Interface **can't be initiated using new keyword**
Concrete class can be declared as final	**Abstract Class can't be declared as final**	Interface **can't be declared as final**

Example:
Class Demo extends super implements interface{
}
	6) **Example:**
	public abstract class Shape{
	public abstract void draw();
	}
		Example:
		public interface Drawable{
		void draw();
		}

6. List out the benefits of Stream oriented I/O. [3]

- A stream in java is a path along which data flows.
- A stream presents a uniform,easy-to-use ,object oriented interface between the program and the input/output devices
- It has a source(of data)and destination(for that data).Both the source and destination ma be physical devices or programs or other streams in the same program.

Stream Benefits :

- The streaming interface to I/O in java provides a clean abstraction for a complex and often cumbersome task. The composition of the filtered stream classes allows you to dynamically build the custom streaming interface to suit your data transfer requirements. Java programs written to adhere to the abstract, high-level Java InputStream Class, Java OutputStream Class, Reader Class In Java, and Java Writer Class classes will function properly in the future even when new and improved concrete stream classes are

invented. This model works very well when we switch from a file system-based set of streams to the network and socket streams

7. What is the benefit of Generics[2]

Java Generics programming is introduced in J2SE 5 to deal with type-safe objects. It makes the code stable by detecting the bugs at compile time. Before **generics**, we can store any type of objects in the collection, i.e., non-**generic**. Now **generics** force the **java** programmer to store a specific type of objects.

Generics allow us to provide the type of Object that a **collection** can contain, so if you try to add any element of other type it throws compile time error

Advantages of Generics:

Programs that uses Generics has got many benefits over non-generic code.

Code Reuse: We can write a method/class/interface once and use for any type we want.

Type Safety : Generics make errors to appear compile time than at run time (It's always better to know problems in your code at compile time rather than making your code fail at run time). Suppose you want to create an ArrayList that store name of students and if by mistake programmer adds an integer object instead of string, compiler allows it. But, when we retrieve this data from Array List, it causes problems at runtime

8 Differentiate between Enumeration and Iterator interface.

***Enumeration* and *Iterator* are two interfaces in *java.util* package which are used to traverse over the elements of a *Collection* object. Though they perform the same function i.e traversing the *Collection* object, there are some differences exist between them.**

Enumeration	Iterator
Using Enumeration, you can only traverse the collection. You can't do any modifications to collection while traversing it.	Using Iterator, you can remove an element of the collection while traversing it.
Enumeration is introduced in JDK 1.0	Iterator is introduced from JDK 1.2
Enumeration is used to traverse the legacy classes like Vector, Stack and HashTable.	Iterator is used to iterate most of the classes in the collection framework like ArrayList, HashSet, HashMap, LinkedList etc.
Methods : hasMoreElements() and nextElement()	Methods : hasNext(), next() and remove()
Enumeration is fail-safe in nature.	Iterator is fail-fast in nature.
Enumeration is not safe and secured due to it's fail-safe nature.	Iterator is safer and secured than Enumeration.

9 What are the methods available in the character streams? [2]

- The character stream can be used to read and write Unicode characters.
- There are two kinds of character stream classes,namely
  1.Reader
  2.Writer

1.Reader Stream class:

- It is an abstract class that defines model of streaming character input.
- It implements ***Closeable and Readable interfaces***

Below table gives the methods provided by **Reader class**

Method	Description
abstract void close()	Closes the input source
int read()	Reads the character from the invoking input stream . return -1 when end of the file is encountered
int read(char buffer[ ])	Reads an array of character into *buffer* and return -1 when end of the file is encountered
int read(char buffer[ ],int offset , int	Reads *numchar* bytes into *buffer* starting from

numchar)	*offset*. return -1 when end of the file is encountered
boolean ready()	Returns **true** if the next input request will not wait .otherwise it returns **false**
void mark(int numChars)	Places a mark at the current point in the input stram that will remain valid until *numchars* characters are read
void reset()	Resets the input pointer to the previously set mark

2. Writer Stream class:

- It is an abstract class that defines model of streaming character output.
- It implements ***Closeable,Flushable and Appendable interfaces***

Below table gives the methods provided by **Writer class**

Method	Description
abstract void close()	Closes the output stream.
abstract void flush()	Flushes the output stream
Writer append(char ch)	Appends *ch* to the end of output stream
Writer append(CharSequence chars)	Appends the chars to the end of the output stream
Writer append(CharSequence chars,int begin,int end)	Appends the subrange of *chars* specified by *begin* and *end-1* to the output stream.
void write(int ch)	Write a single character to the output stream
void write(char buffer[])	Writes the complete array of characters to the output stream
void write(String str)	Write a string *str* to the output stream
abstract void write(char buffer[],int offset,int numChars)	Write the subrange of *numchars* characters from *buffer* beging of the *offset* to output

	stream

10. What is the significance of the CLASSPATH environment variable in creating/using a package?[3]

CLASS PATH environment variable setting is used by java run time system to know where to look for package that is created.
For example, consider the following package specification:
package MyPack
the class path ***must not*** *include MyPack, itself.*

It must simply specify the path to MyPack. For example, in a Windows environment, if the path to MyPack is
C:\MyPrograms\Java\MyPack

Then the class path to MyPack is

C:\MyPrograms\Java

Example:

```
package MyPack;
class Demo
{
…….
…….
}
class classpathdemo
{
public static void main(String args[])
{
Demo obj=new Demo();
……
}
}
```

Call this file **classpathdemo.java** and put it in a directory called **MyPack.**

Next, compile the file. Make sure that the resulting **.class** file is also in the MyPack directory.

Then, try executing the classpathdemo class, using the following command line:

java MyPack.**classpathdemo**

Remember, you will need to be in the directory above MyPack when you execute this command.

11. What is Console class? What is its use in java? [2]

- Console class is used to read from and write to the console.
- It implements the ***Flushable interface***
- Console supplies no constructor ,a console object is obtained by calling **System.console()**

Syntax:

```
public static Console console ( )
```

If the console is available,its reference is returned other wise **null** is returned.

Below table gives the methods provided by **Console class**

Method	Description
public String readLine()	used to read a single line of text from the console
public String readLine(String fmt,Object... args)	it provides a formatted prompt then reads the single line of text from the console.
public char[] readPassword()	used to read password that is not being displayed on the console.
public char[] readPassword(String fmt,Object... args)	it provides a formatted prompt then reads the password that is not being displayed on the console.
void flush()	Causes buffered output to be written physically to the console
Console printf(String fmtString,Object …args)	Writes *args* to the console using the format specified by *fmtstring*
Reader reader()	Returns a reference to **Reader** connected to the console
PrintWriter writer()	Returns a reference to the **Writer** connected to the console

Example on Reading Console Input And Writing Console Output

```
import java.io.*;
class ConsoleDemo
{
public static void main(String  args[])
        {
        String str;
        Console con;
        con=System.Console();
        if (con==null)
        return;
        str=con.readLine("Enter a string:");
        con.printf("Here is ur string: %s",str);
        }
}
```

Output:

Enter a string: Have a nice day

Here is ur string: Have a nice day

12. What is the use of auto boxing in java? Explain. [3]

The automatic conversion of primitive data types into its equivalent Wrapper type is known as boxing and opposite operation is known as unboxing.

This is the new feature of Java5. So java programmer doesn't need to write the conversion code**.**

Advantage of Autoboxing and Unboxing:

No need of conversion between primitives and Wrappers manually so less coding is required**.**

Simple Example of Autoboxing in java

```
class BoxingDemo
{
  public static void main(String args[])
                {
                 int a=50;
                Integer a2=new Integer(a);//Boxing

                 Integer a3=5;//Boxing

                System.out.println(a2+" "+a3);
                }
}
```

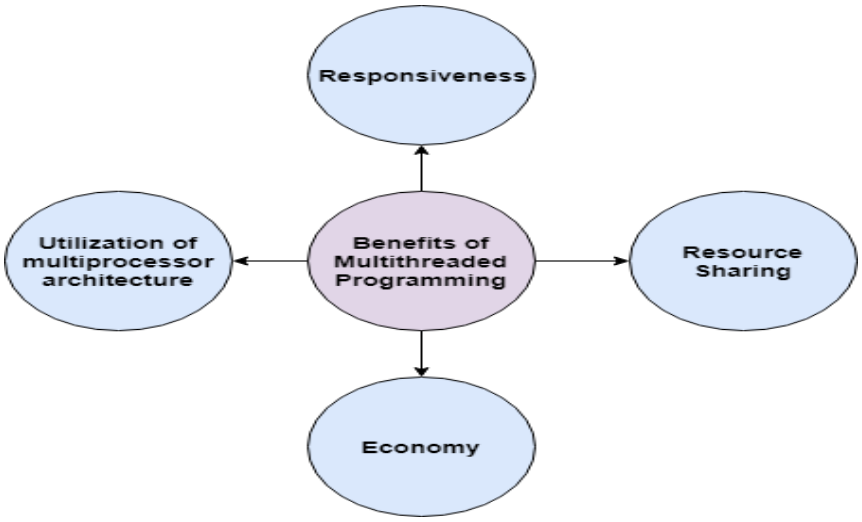
Output:

50 5

UNIT-III

1. What are the advantages of multithreading? [2]

Multithreading allows the execution of multiple parts of a program at the same time. These parts are known as threads and are lightweight processes available within the process. So multithreading leads to maximum utilization of the CPU by multitasking.Some of the benefits of multithreaded programming are given as follows –



- **Resource Sharing**
  All the threads of a process share its resources such as memory, data, files etc. A single application can have different threads within the same address space using resource sharing.
- **Responsiveness**
  Program responsiveness allows a program to run even if part of it is blocked using multithreading. This can also be done if the process is performing a lengthy operation. For example - A web browser with multithreading can use one thread for user contact and another for image loading at the same time.
- **Utilization of Multiprocessor Architecture**
  In a multiprocessor architecture, each thread can run on a different processor in parallel using multithreading. This increases concurrency of the system. This is in direct contrast to a single processor system, where only one process or thread can run on a processor at a time.
- **Economy**
  It is more economical to use threads as they share the process resources. Comparatively, it is more expensive and time-consuming to create processes as they require more memory and resources. The overhead for process creation and management is much higher than thread creation and management.

2. Explain the types of exceptions. [3]

- **Checked Exception**

The exception that can be predicted by the programmer.The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions
e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

- **Unchecked Exception**

Unchecked exceptions are the class that extends RuntimeException. Unchecked exception are ignored at compile time.
*Example :*ArithmeticException, NullPointerException, Array Index out of Bound exception. Unchecked exceptions are checked at runtime.

- **Error**

Errors are typically ignored in code because you can rarely do anything about an error.For example if stack overflow occurs, an error will arise. This type of error is not possible handle in code. Error is irrecoverable
e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

3.Difference between process and thread

BASIS FOR COMPARISON	**PROCESS**	**THREAD**
Basic	An executing program is called a process.	A thread is a small part of a process.
Address Space	Every process has its separate address space.	All the threads of a process share the same address space cooperatively as that of a process.
Multitasking	Process-based multitasking allows a computer to run two or more than two programs concurrently.	Thread-based multitasking allows a single program to run two or more threads concurrently.
Communication	Communication between two processes is expensive and limited.	Communication between two thread is less expensive as compared to process.
Switching	Context switching from one process to another process is expensive.	Context switching from one thread to another thread is less expensive as

BASIS FOR COMPARISON	**PROCESS**	**THREAD**
		compared to process.
Components	A process has its own address space, global variables, signal handlers, open files, child processes, accounting information.	A thread has its own register, stat stack, program counter.
Substitute	Process are also called heavyweight task.	Thread are also called lightweight ta k.
Control	Process-based multitasking is not under the control of Java.	Thread-based multitasking is under t e control of Java.
Example	You are working on text editor it refers to the execution of a process.	You are printing a file from text edi or while working on it that resembles e execution of a thread in the process.

4. Explain how multiple catch statement works

Multiple catch blocks:

A try block can be followed by multiple catch blocks. You can have any number of catch blocks after a single try block.If an exception occurs in the guarded code the exception is passed to the first catch block in the list. If the exception type of exception, matches with the first catch block it gets caught, if not the exception is passed down to the next catch block. This continue until the exception is caught or falls through all catches.

Example for Multiple Catch blocks

```
class Excep {
public static void main(String[] args)
  {
try  {
int arr[]={1,2};
arr[2]=3/0;
  }
catch(ArithmeticExceptionae)
  {
System.out.println("divide by zero");
```

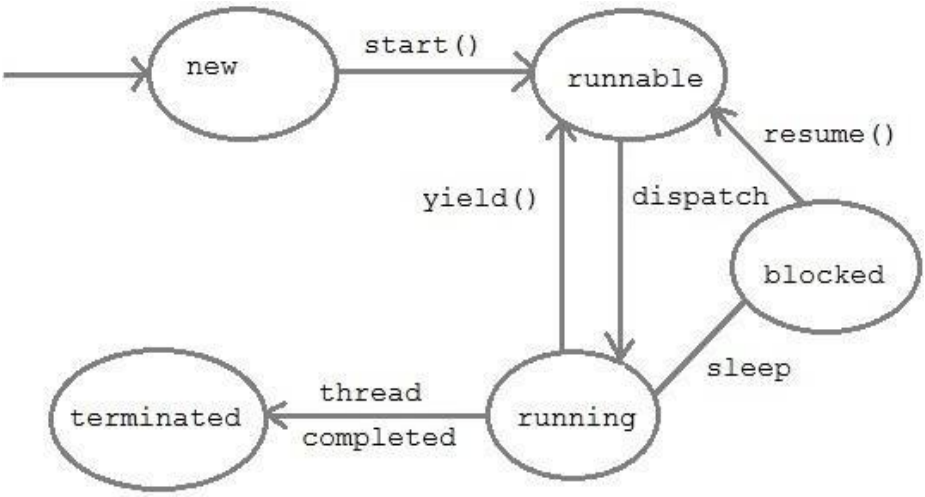
```
  }
catch(ArrayIndexOutOfBoundsException e)
 {
System.out.println("array index out of bound exception");
  }
 }
}
```

Output :

divide by zero

5.List the thread states.

Ans:Life cycle of a Thread



1. **New :** A thread begins its life cycle in the new state. It remains in this state until the start() method is called on it.
2. **Runnable :** After invocation of start() method on new thread, the thread becomes runnable.
3. **Running :** A method is in running thread if the thread scheduler has selected it.
4. **Blocking :** A thread is waiting for another thread to perform a task(such as sleep , I/O operations, block suspend, wait) . In this stage the thread is still alive.
5. **Terminated :** A thread enter the terminated state when it complete its task.

6. What keywords are essential in handling user-defined exception

In java, exception handling is done using five keywords,

1. **try**
2. **catch**
3. **throw**
4. **throws**
5. **finally**

Exception handling is done by transferring the execution of a program to an appropriate exception handler when exception occurs.

Example demonstrating Keyword of Exception handling

```
import java.io.*;
public class ThrowExample {
  void mymethod(int num)throws IOException, ClassNotFoundException{
    if(num==1)
      throw new IOException("Exception Message1");
    else
      throw new ClassNotFoundException("Exception Message2");
  }
}
class Demo
 {
  public static void main(String args[]){
   try{
    ThrowExample obj=new ThrowExample();
    obj.mymethod(1);
      }
catch(Exception ex)
    {
    System.out.println(ex);
   }
 finally
{
system.out.println("message from finally block");

}
}
```

7. Differentiate between error and exception

ERRORS	EXCEPTIONS
Recovering from Error is not possible.	We can recover from exceptions by either using try-catch block or throwing exceptions back to caller.
All errors in java are unchecked type.	Exceptions include both checked as well as unchecked type.
Errors are mostly caused by the environment in which program is running.	Program itself is responsible for causing exceptions.
Errors occur at runtime and not known to the compiler.	All exceptions occurs at runtime but checked exceptions are known to compiler while unchecked are not.
They are defined in java.lang.Error package.	They are defined in java.lang.Exception package
Examples : java.lang.StackOverflowError, java.lang.OutOfMemoryError	Examples : Checked Exceptions : SQLException, IOException Unchecked Exceptions : ArrayIndexOutOfBoundException, NullPointerException, ArithmeticException.

8. How to assign priorities to threads?
Ans:
We can assign priority to a thread by using following function**:**
Threadobject.setPriority(TheardPriority)

Example on Thread Priority

```
class Multi implements Runnable{
 public void run() {
   System.out.println("running thread name is:"+Thread.currentThread().getName());
   System.out.println("running thread priority is:"+Thread.currentThread().getPriority());
    }
 public static void main(String args[]) {
  Multi m1=new Multi  ();
Multi m2=new  Multi  ();
```

```
 m1.setPriority(Thread.MIN_PRIORITY);
 m2.setPriority(Thread.MAX_PRIORITY);
 m1.start();
 m2.start();
   }
}
```

9. How does Java support inter thread communication? [2]

Inter-thread communication or **Co-operation** is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of **Object class**:

- wait()
- notify()
- notifyAll()

1) wait() method

Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

public final void wait()throws InterruptedException---- waits until object is notified.

public final void wait(long timeout)throws InterruptedException-----waits for the specified amount of time.

2) notify() method

Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. Syntax:

public final void notify()

3) notifyAll() method

Wakes up all threads that are waiting on this object's monitor. Syntax:

public final void notifyAll()

10. List any four unchecked exception

Exception	Description
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid cast.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread.
IllegalStateException	Environment or application is in incorrect state.
IllegalThreadStateException	Requested operation not compatible with the current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.
SecurityException	Attempt to violate security.
StringIndexOutOfBounds	Attempt to index outside the bounds of a string.
StringIndexOutOfBounds	UnsupportedOperationException An unsupported operation was encountered.

UNIT-IV

1. What is the difference between array and vector?

Concept name	array	vector
Resize	Length of the array is fixed i.e once t is created we cannot added or emove elements	A vector is a resizable – array which works by reallocating storage
Sychronized	Array is not Sychronized	Vector is Sychronized
Performance	Array is faster	Vector is relatively slow
Storage	It does not reserve any additional storage	Vector reserves O(n) additional storage
trimToSize()	Array doesnot have trimToSize()	Vector uses trimToSize() To remove additional storage
Elements	It can hold both primitives and java objects	It can hold only java objects
size	It uses length property to store length.	It uses size() method to get size .
Dimensions	java support single and multi dimensional array	Vectors doesnot have a concept of dimensions,but it supports vector of Vectors
Generics	It does not support Generics	It support Generics to ensure type safety

2. List the hash table constructors.

Hashtable stores key/value pairs in a hash table. However, neither keys nor values can be **null**. When using a **Hashtable**, you specify an object that is used as a key, and the value that you want linked to that key. The key is then hashed, and the resulting hash code is used as the index at which the value is stored within the table.

Syntax: class Hashtable<K, V>

Here, **K** specifies the type of keys, and **V** specifies the type of values.

*The **Hashtable** constructors are::*

Syntax:

Hashtable( );

Hashtable(int *size*);

Hashtable(int *size*, float *fillRatio*);

Hashtable(Map<? extends K, ? extends V> *m*);

3. **Explain the methods defined by Vector**.

Method defined by Vector.

Method	**Description**
int size()	**It returns the size of the Vector**
int capacity()	**It returns the capacity** of the Vector
addElement(element);	**It add an element to the Vector**
firstElement()	**It returns the first element in Vector**
lastElement()	**It returns the last element in Vector**

4. **Explain the use of String tokenizer with an example*****

Ans:The string tokenizer class allows an application to break a string into tokens.
The StringTokenizer methods do not distinguish among identifiers, numbers, and quoted strings, nor do they recognize and skip comments.

The set of delimiters (the characters that separate tokens) may be specified either at creation time or on a per-token basis.

Example:

```
import java.util.StringTokenizer;

class StringTokenizerDemo
{
   public static void main(String arg[])
   {
      StringTokenizer st1 = new StringTokenizer("Welcome to CMREC" +  " Campus."); //
LINE A
      while (st1.hasMoreTokens()) {
         System.out.println(st1.nextToken());
      }
      System.out.print(" --------\n");
      StringTokenizer st2 = new StringTokenizer("It's an,Education," + "Website." ,  ",","); //
LINE B
      while (st2.hasMoreTokens()) {
```

```
        System.out.println(st2.nextToken());
      }
      System.out.print(" --------\n");
      StringTokenizer st3 = new StringTokenizer("Learn~programming~with Java.", "~", true); //
LINE C
      while (st3.hasMoreTokens()) {
        System.out.println(st3.nextToken());
      }
   }
}
```

5. Write about any 3 methods defined by Iterator

Iterator interface provides the facility of iterating the elements in forward direction only. There are only three methods in the Iterator interface. They are:

1. ***public boolean hasNext()*** it returns true if iterator has more elements.
2. ***public object next()*** it returns the element and moves the cursor pointer to the next element.
3. ***public void remove()*** it removes the last elements returned by the iterator. It is rarely used

Example program:

```
import java.util.*;
        class TestCollection1{
            public static void main(String args[]){
              ArrayList<String> list1=new ArrayList<String>();//Creating arraylist
              list1.add("Ravi");//Adding object in arraylist
              list1.add("Vijay");
              list1.add("Giri");
              list1.add("Ajay");
             System.out.println("arraylist elements using Iterator interface");
              //Traversing list through Iterator
              Iterator itr=list1.iterator();
              while(itr.hasNext()){
              System.out.println(itr.next());
              }
             }
```

6. List the functions of Stack class.

Stack is a subclass of **Vector** that implements a standard "last-in, first-out" stack. **Stack** only defines the default constructor, which creates an empty stack. With the release of JDK 5, **Stack** was retrofitted for generics and is declared as shown here:

Syntax: class Stack<E>

Here, **E** specifies the type of element stored in the stack.

Stack includes all the methods defined by **Vector** and adds several of its own, shown in belowTable.

Method	Description
boolean empty( )	Returns true if the stack is empty, and returns false if the stack contains elements.
E peek( )	Returns the element on the top of the stack, but does not remove it
E pop( )	Returns the element on the top of the stack, removing it in the : process
E push(E element)	Pushes element onto the stack. element is also returned.
int search(Object element)	Searches for element in the stack. If found, its offset from the top of the stack is returned. Otherwise, –1 is returned.

7. What is the use of Iterator class?

Iterator:

It is a universal iterator as we can apply it to any Collection object. By using Iterator, we can perform both read and remove operations. It is improved version of Enumeration with additional functionality of remove-ability of a element.

Iterator must be used whenever we want to enumerate elements in all Collection framework implemented interfaces like Set, List, Queue, Deque and also in all implemented classes of Map interface. Iterator is the only cursor available for entire collection framework.

Iterator object can be created by calling iterator() method present in Collection interface.

Java program to demonstrate Iterator

```
import java.util.ArrayList;
import java.util.Iterator;

public class Test
{
   public static void main(String[] args)
   {
      ArrayList al = new ArrayList();

      for (int i = 0; i < 10; i++)
         al.add(i);

      System.out.println(al);
```

```
        // at beginning itr(cursor) will point to
        // index just before the first element in al
        Iterator itr = al.iterator();

        // checking the next element availabilty
        while (itr.hasNext())
        {
           //  moving cursor to next element
           int i = (Integer)itr.next();

           // getting even elements one by one
           System.out.print(i + " ");

           // Removing odd elements
           if (i % 2 != 0)
              itr.remove();
        }
        System.out.println();
        System.out.println(al);
    }
}
```

Output:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

0 1 2 3 4 5 6 7 8 9

[0, 2, 4, 6, 8]
```

8. What is the benefit of Generics in Collections Framework?

Java 1.5 came with Generics and all collection interfaces and implementations use it heavily. Generics allow us to provide the type of Object that a collection can contain, so if you try to add any element of other type it throws compile time error.

This avoids ClassCastException at Runtime because you will get the error at compilation. Also Generics make code clean since we don't need to use casting and **instanceof** operator. It also adds up to runtime benefit because the bytecode instructions that do type checking are not generated.

9. Differentiate between Enumeration and Iterator interface.

BASIS FOR COMPARISON	ITERATOR	ENUMERATION
Basic	Iterator is a universal cursor as it is applicable for all the collection classes.	Enumeration is not a universal cursor as it applies only to legacy classes.
Access	Using Iterator you can read and remove the elements in the collection.	Using Enumeration you can only read the elements in the collection.

Methods	public boolean hasnext();
	public objects next();
	public void remove();
		public boolean hasMoreElements();
		public object nextElement();

Limitation	Iterator is a unidirectional forward access cursor. Iterator can not replace any element in the collection. Iterator can not add any new element in the collection.	Enumeration is unidirectional forward access cursor. Enumeration support only legacy classes. Enumeration has only read-only access to the elements in a collection.
Overcome	To overcome the limitations of Iterator you must opt for ListIterator.	To overcome the limitations of Enumeration you must opt for Iterator.

10. Make a comparison of List, array and ArrayList.

Ans:

Array: Simple fixed sized arrays that we create in Java, like below

```
int arr[] = new int[10]
```

ArrayList : Dynamic sized arrays in Java that implement List interface.

```
ArrayList<Type> arrL = new ArrayList<Type>();
```

Here Type is the type of elements in ArrayList to be created

Differences between List ,Array and ArrayList

Array

- An array is basic functionality provided by Java and it fixed length.
- Array provide both direct and sequential access to elements
- Array is a static memory allocation so there is a wastage of memory
- Array can sote only objects

- Array is a static memory allocation so there is a wastage of memory
- Insertion and deletion is a time consummation process in array
- All the cells in array should be same data type

List and ArrayList

- List and ArrayList is part of collection framework in Java and have variable length.
- ArrayList has a set of methods to access elements and modify them.
- List provide only sequential access to elements
- List dynamic allocation so no wastage of memory
- Arraylist can store primitive datatype and Object.
- Insertion and deletion is not time consummation process in arraylist and list
- a single cell can be divided into many parts each having info of different data type. but the last necessarily needs to be the pointer to the next cell

// A Java program to demonstrate differences between array and ArrayList

```
import java.util.ArrayList;
import java.util.Arrays;

class Test
{
	public static void main(String args[])
	{
		/* ..... Normal Array ..... */
		int[] arr = new int[2];
		arr[0] = 1;
		arr[1] = 2;
		System.out.println(arr[0]);

		/*.....ArrayList.....*/
		// Create an arrayList with initial capacity 2
		ArrayList<Integer> arrL = new ArrayList<Integer>(2);

		// Add elements to ArrayList
		arrL.add(1);
		arrL.add(2);

		// Access elements of ArrayList
		System.out.println(arrL.get(0));
	}
	}
```

11. What is the significance of Legacy class? Give example.

Java only defined several classes and interfaces that provide methods for storing objects. When Collections framework were added in J2SE 1.2, the original classes were reengineered to support the collection interface. These classes are also known as Legacy classes.

Legacy Classes - Java Collections

- Dictionary.
- HashTable.
- Properties.
- Stack.
- Vector.

.

12. What is a Java Priority queue?

A PriorityQueue is used when the objects are supposed to be processed based on the priority. It is known that a queue follows First-In-First-Out algorithm, but sometimes the elements of the queue are needed to be processed according to the priority, that's when the PriorityQueue comes into play. The PriorityQueue is based on the priority heap.

Mostly used Constructors of PriorityQueue class

- **PriorityQueue():** Creates a PriorityQueue with the default initial capacity (11) that orders its elements according to their natural ordering.
- **PriorityQueue(Collection<E> c):** Creates a PriorityQueue containing the elements in the specified collection.

Methods in PriorityQueue class:

- **boolean add(E element):** This method inserts the specified element into this priority queue.
- **public remove():** This method removes a single instance of the specified element from this queue, if it is present
- **public poll():** This method retrieves and removes the head of this queue, or returns null if this queue is empty.
- **public peek():** This method retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.
- **Iterator iterator():** Returns an iterator over the elements in this queue.
- **boolean contains(Object o):** This method returns true if this queue contains the specified element
- **void clear():** This method is used to remove all of the contents of the priority queue.
- **boolean offer(E e):** This method is used to insert a specific element into the priority queue.
- **int size():** The method is used to return the number of elements present in the set.
- **toArray():** This method is used to return an array containing all of the elements in this queue.
- **Comparator comparator():** The method is used to return the comparator that can be used to order the elements of the queue

13.What is a Collection Class? Give an example.

Collections in java is a framework that provides an architecture to store and manipulate the group of objects.

All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

Each collection classes provide **iterator()** method to return an iterator.

Methods of Iterator:

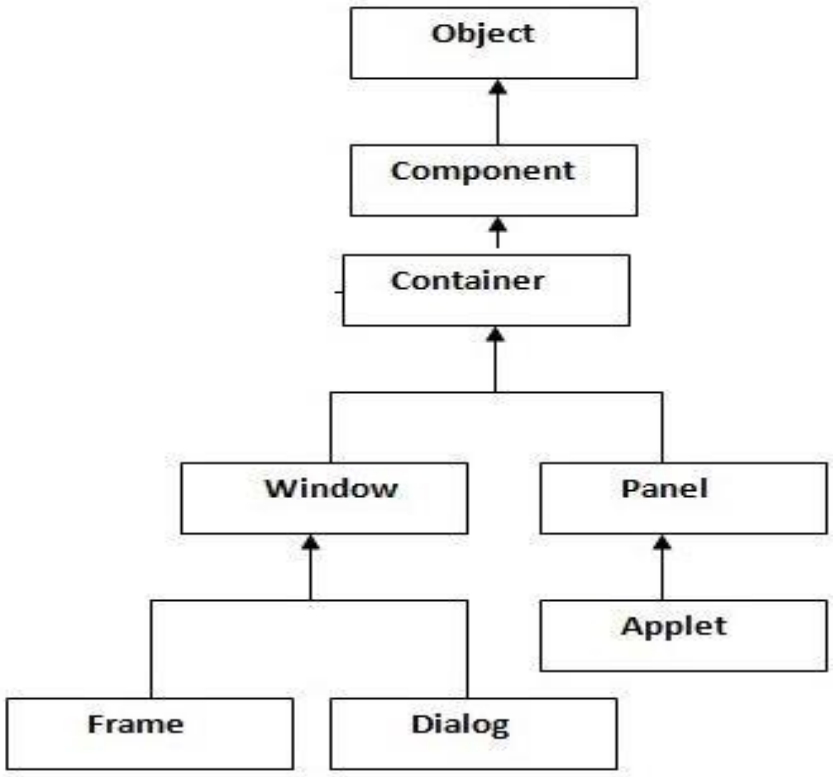
Method	Description
boolean hasNext()	Returns true if there are more elements in the collection. Otherwise, returns false.
E next()	Returns the next element present in the collection. Throw NoSuchElementException if there is not a next element.
void remove()	Removes the current element. Throws IllegalStateException if an attempt is made to call remove() method that is not preceded by a call to next() method.

UNIT-V

1. What are the containers available in swing? [2M]

Abstract Windowing Toolkit (AWT): Abstract Windowing Toolkit (AWT) is used for GUI programming in java.

AWT Container Hierarchy:



Container:

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container.

2. ***Compare Applets with application programs. [3M]

FEATURE	APPLET	APPLICATION
main() method	Not Present	present
Nature	Requires some third party tool help like a browser to execute	Called as stand-alone application as application can be executed from command prompt
Restrictions	cannot access any thing on the system except browser's services	Can access any data or software available on the system
Security	Requires highest security for the system as they are untrusted	Does not require any security
Execution	Applet is portable and can be executed by any JAVA supported browser	Need JDK, JRE, JVM installed on client machine
Creation	Applets are created by extending the java.applet.Applet	Applications are created by writing public static void main(String[] s) method
Methods	Applet application has 5 methods which will be automatically invoked on occurance of specific event	Application has a single start point which is main method
Example	Example:	Example:public class MyClass

	import java.awt.*;
	import java.applet.*;
	
	public class Myclass extends Applet
	{
	public void init() { }
	public void start() { }
	public void stop() {}
	public void destroy() {}
	public void paint(Graphics g) {}
	}
		{
		public static void main(String args[]) {}
		}

3. what is the use of LayOut manager

A container has a so-called *layout manager* to arrange its components. The layout managers provide a level of abstraction to map your user interface on all windowing systems, so that the layout can be *platform-independent*.

AWT provides the following layout managers (in package java.awt):

- Flow Layout
- Border Layout
- Grid Layout
- Card Layout
- Grid Bag Layout

Container's setLayout() method

A container has a setLayout() method to set its layout manager:

// java.awt.Container

public void setLayout(LayoutManager mgr)

To set up the layout of a Container (such as Frame, JFrame, Panel, or JPanel), you have to:

1. Construct an instance of the chosen layout object, via new and constructor, e.g., new FlowLayout())
2. Invoke the setLayout() method of the Container, with the layout object created as the argument;
3. Place the GUI components into the Container using the add() method in the correct order; or into the correct zones.

4. explain about life cycle of Applet
Java Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Lifecycle of Java Applet

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.

5. What are the merits of swing components over AWT? [2]

No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC**.

6.*What is an adapter class? What is its significance? List the adapter classes. [3]**

Java adapter classes provide the default implementation of listener interfaces. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it saves code.

The adapter classes are found in ***java.awt.event, java.awt.dnd*** and ***javax.swing.event*** packages. The Adapter classes with their corresponding listener interfaces are given below

java.awt.event Adapter classes:

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

java.awt.dnd Adapter classes

Adapter class	Listener interface
DragSourceAdapter	DragSourceListener
DragTargetAdapter	DragTargetListener

javax.swing.event Adapter classes:

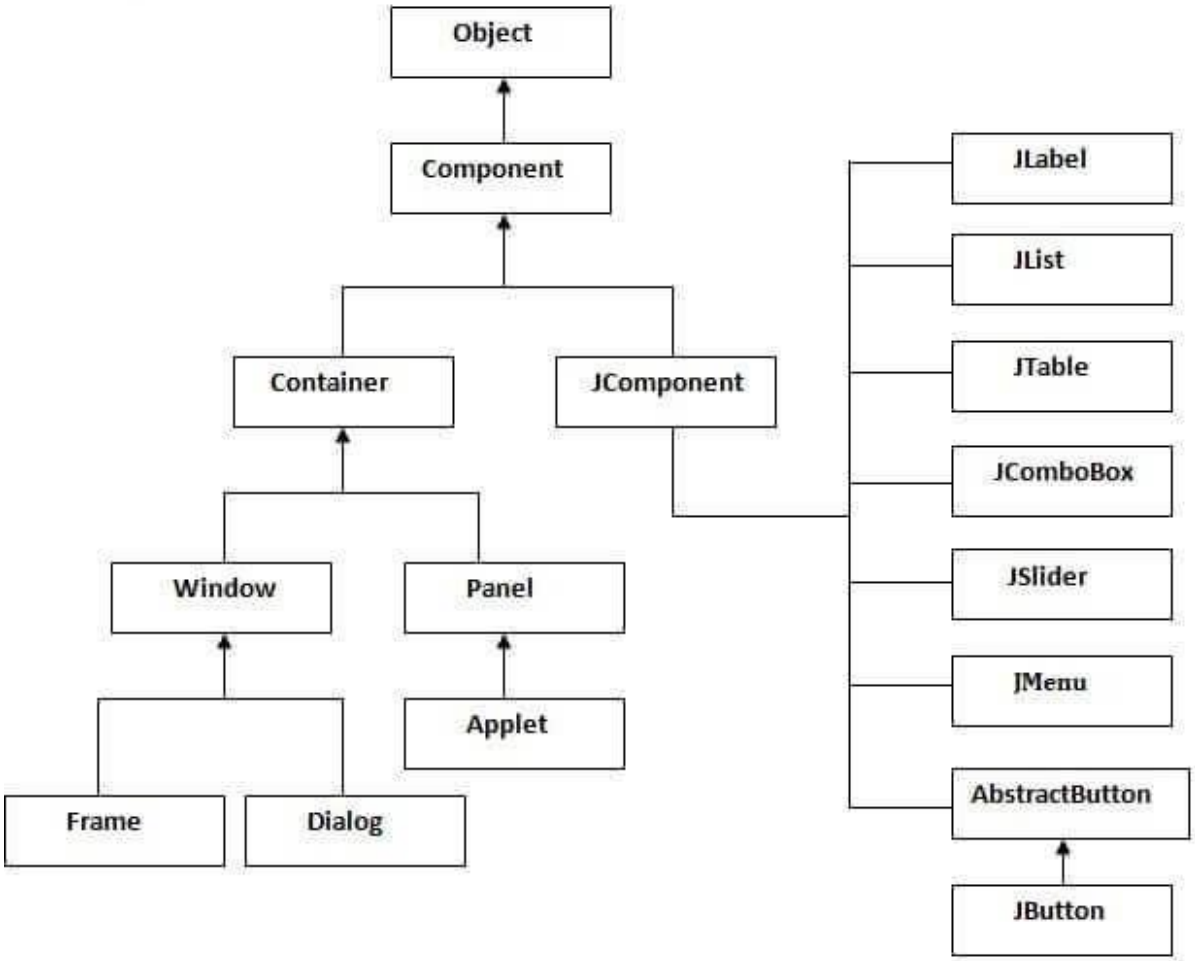
Adapter class	Listener interface
MouseInputAdapter	MouseInputListener
InternalFrameAdapter	InternalFrameListener

7. What are the sources for item event? [2]

JButton
JToggle Button
JCheckBox
JRadio Button
JTabbed Pane
JScroll Pane
JList
JComboBox
Swing Menus
Dialogs

8. Give the hierarchy for swing components. [3]

The hierarchy of java swing API is given below.



9. Give the subclasses of JButton class.

Subclasses:

1. BasicArrowButton,
2. MetalComboBoxButton

BasicArrowButton:

public class BasicArrowButton extends JButton implements SwingConstants MetalComboBoxButton

Constructors

BasicArrowButton(int direction)--Creates a BasicArrowButton whose arrow is drawn in the specified direction.

BasicArrowButton(int direction, Color background, Color shadow, Color darkShadow, Color highlight)--Creates a BasicArrowButton whose arrow is drawn in the specified direction and with the specified colors**.**

Modifier and Type	Method	Description
int	getDirection()	Returns the direction of the arrow.
Dimension	getMaximumSize()	Returns the maximum size of the BasicArrowButton.
Dimension	getMinimumSize()	Returns the minimum size of the BasicArrowButton.
Dimension	getPreferredSize()	Returns the preferred size of the BasicArrowButton.
boolean	isFocusTraversable()	Returns whether the arrow button should ge the focus.
void	paintTriangle(Graphics g, int x, int y, int size, int direction, boolean isEnabled)	Paints a triangle.
void	setDirection(int direction)	Sets the direction of the arrow.

2.MetalComboBoxButton

Constructors

MetalComboBoxButton(JComboBox cb, Icon i, boolean onlyIcon, CellRendererPane pane, JList list)

MetalComboBoxButton(JComboBox cb, Icon i, CellRendererPane pane, JList list)

Methods	
Modifier and Type	Method and Description
JComboBox	getComboBox()
Icon	getComboIcon()
Dimension	getMinimumSize() If the minimum size has been set to a non-null value just returns it.
boolean	isFocusTraversable() Returns whether this Component can become the focus owner.
boolean	isIconOnly()
void	paintComponent(Graphics g) Calls the UI delegate's paint method, if the UI delegate is non-null.
void	setComboBox(JComboBox cb)
void	setComboIcon(Icon i)
void	setEnabled(boolean enabled) Enables (or disables) the button.

void	setIconOnly(boolean isIconOnly)

10.Differentiate between grid layout and border layout managers.

BorderLayout *- Lays out components in BorderLayout.NORTH, EAST, SOUTH, WEST, and CENTER sections.*		
bord =	**new BorderLayout**();	Creates BorderLayout. Widgets added with constraint to tell where.
bord =	**new BorderLayout**(*h, v*);	Creates BorderLayout with horizonal and vertical gaps sizes in pixels.
	p.**add**(*widget, pos*);	Adds *widget* to one of the 5 border layout regions, *pos* (see list above).
GridLayout *- Lays out components in equal sized rectangular grid, added r-t-l, top-to-bottom.*		
grid =	**new GridLayout**(*r, c*);	Creates GridLayout with specified rows and columns.
grid =	**new GridLayout**(*r,c,h,v*);	As above but also specifies horizontal and vertical space between cells.
	p.**add**(*widget*);	Adds *widget* to the next left-to-right, top-to-bottom cell.

GridLayout

- Arranges components into rows and columns
- A GridLayout puts all the components in a rectangular grid and is divided into equal-sized rectangles and each component is placed inside a rectangle
- A **GridLayout** is constructed with parameters
- When a component is added, it is placed in the next position in the grid, which is filled row by row, from the first to last column

BorderLayout

- Arranges components into five areas: North, South, East, West, and Center
- The class BorderLayout arranges the components to fit in the five regions: east, west, north, south and center. Each region is can contain only one component and each component in each region is identified by the corresponding constant NORTH, SOUTH, EAST, WEST, and CENTER.
- A **BorderLayout** can be constructed with no parameters
- We can add a single component at each of the four compass directions (specified by the Strings "North", "South", "East", or "West", as well as at the "Center". Of course, the component we add can be a container, which contains multiple components (managed by another layout manager).

11. What are the limitations of AWT? [2]

limitations of AWT :The AWT defines a basic set of controls, windows, and dialog boxes that support a usable, but limited graphical interface. One reason for the limited nature of the AWT is that it translates its various visual components into their corresponding, platform-specific equivalents or peers. This means that the look and feel of a component is defined by the platform, not by java. Because the AWT components use native code resources, they are referred to as heavy weight.The use of native peers led to several problems. First, because of variations between operating systems, a component might look, or even act, differently on different

platforms. This variability threatened java's philosophy: write once, run anywhere.Second, the look and feel of each component was fixed and could not be changed. Third,the use of heavyweight components caused some frustrating restrictions.

Summary on limitations of AWT

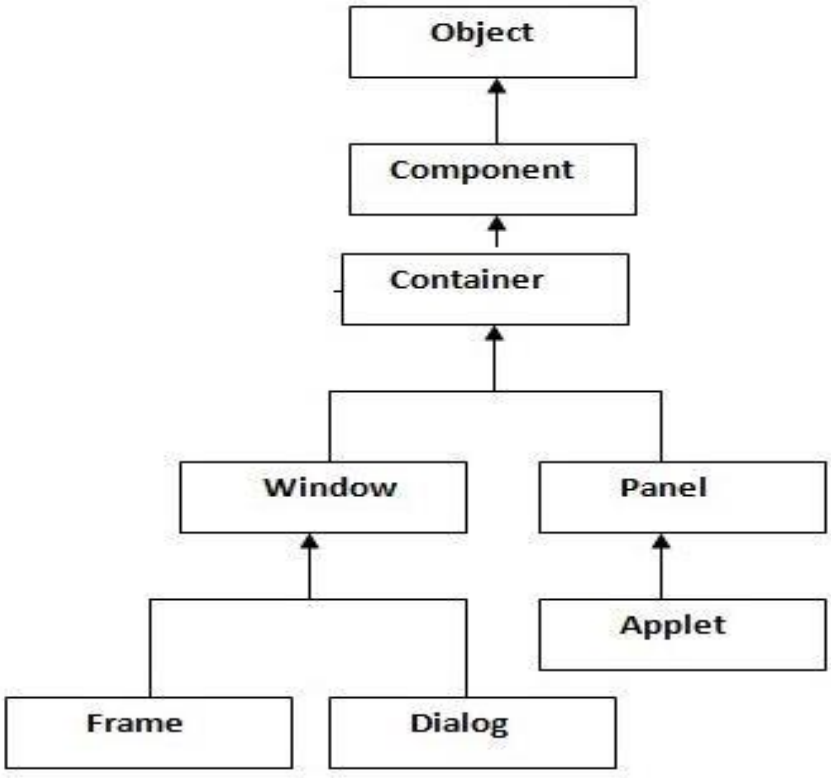
AWT supports limited number of GUI components
AWT component are Heavy weight components
AWT components are developed by using platform specific code
AWT components behaves differently in different Operating Systems.
AWT components is converted by the native code of the Operating System

12. Why do applet classes need to be declared as public? [3]

Applet classes need to be declared as a public becomes the applet classes are access from HTML document that means from outside code. Otherwise, an applet will not be accessible from an outside code i.e an HTML

13. **Give the AWT hierarchy. [2]**

AWT Container Hierarchy:



Container:

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container.

Window:

The window is the container that have no borders and menubars. You must use frame, dialog or

another window for creating a window.

Panel:

The Panel is the container that doesn't contain title bar and MenuBars. It can have other components like button, textfield etc.

Frame:

The Frame is the container that contain title bar and can have MenuBars. It can have other components like button, textfield etc.

There are two ways to create a frame:

1. By extending Frame class (inheritance)

2. By creating the object of Frame class (association)

14. What are the various classes used in creating a swing menu? [3]

JMenuBar, JMenu and JMenuItem

The JMenuBar class is used to display menubar on the window or frame. It may have several menus.
The object of JMenu class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class.
The object of JMenuItem class adds a simple labeled menu item. The items used in a menu must belong to the JMenuItem or any of its subclass.

JMenuBar class declaration

public class JMenuBar extends JComponent implements MenuElement, Accessible

JMenu class declaration

public class JMenu extends JMenuItem implements MenuElement, Accessible

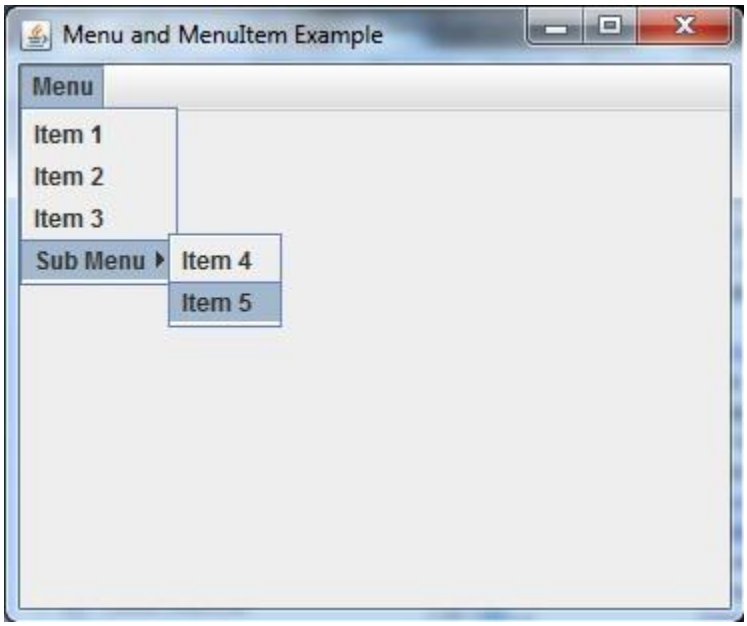
JMenuItem class declaration

public class JMenuItem extends AbstractButton implements Accessible, MenuElement

Example program to demonstrate SWING Menu

```
import javax.swing.*;
class MenuExample
{
        JMenu menu, submenu;
        JMenuItem i1, i2, i3, i4, i5;
        MenuExample(){
        JFrame f= new JFrame("Menu and MenuItem Example");
        JMenuBar mb=new JMenuBar();
        menu=new JMenu("Menu");
        submenu=new JMenu("Sub Menu");
        i1=new JMenuItem("Item 1");
        i2=new JMenuItem("Item 2");
        i3=new JMenuItem("Item 3");
        i4=new JMenuItem("Item 4");
        i5=new JMenuItem("Item 5");
        menu.add(i1); menu.add(i2); menu.add(i3);
        submenu.add(i4); submenu.add(i5);
        menu.add(submenu);
        mb.add(menu);
        f.setJMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
}
public static void main(String args[])
{
new MenuExample();
}}
```

Output:



15. What are the differences between JToggle buttion and Radio buttion? [2]

Toggle Button:

Toggles should be used to represent an action, like turning something on or off, or starting or stopping an activity. It should be clear which state is on and which state is off. As the name suggest a button whose state can be toggled from on to off or vice-versa. For example a Switch in your home to turn a particular light on or off.

Toggle button has immediate effect on selection.

Radio Button:

Radio buttons should be used when the user can select one, and only one, option from a list of items. The button should be circular and become filled in when it is selected. Its name comes from the concept of buttons in Radio where for first station you press first button and for second station you press second button and so forth. So you can choose from multiple options. But at a time only one will be selected.

Radio Button has effect only after pressing Submit button

14. What is Swing in Java? How it differs from Applet. [2]

Swing	Applet
Swing is light weight Component.	**Applet is heavy weight Component.**
Swing have look and feel according to user view you can change look and feel using UIManager.	**Applet Does not provide this facility.**
Swing uses for stand lone Applications, Swing have main method to execute the program.	**Applet need HTML code for Run the Applet.**
Swing uses MVC Model view Controller.	**Applet not.**
Swing have its own Layout like most popular Box Layout.	**Applet uses AWT Layouts like flowlayout.**
Swing have some Thread rules.	**Applet doesn't have any rule.**
To execute Swing no need any browser By which we can create stand alone application But Here we have to add container and maintain all action control with in frame container.	**To execute Applet programe we should need any one browser like Appletviewer, web browser. Because Applet using browser container to run and all action control with in browser container.**

Java swing components are **platform-independent**.

Swing components are **lightweight**.

Swing **supports pluggable look and feel**.

Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.

Swing **follows MVC**.

PART –B

UNIT WISE ESSAY QUESTION AND ANSWERS

UNIT-I

1.Define inheritance: Explain the various forms of inheritance. How to prevent a class from inheritance?

Defination: Inheritance is the mechanism of deriving new class from old one, old class is knows as *superclass* and *new class* is known as **subclass**. The subclass inherits all of its instances variables and methods defined by the superclass and it also adds its own unique elements. Thus we can say that subclass are specialized version of superclass.

Forms of Inheritance:

- **Specification inheritance**
- **Specialization inheritance**
- **Construction inheritance**
- **Extension inheritance**

Specification inheritance:

If the parent class is abstract, we often say that it is providing a specification for the child class, and therefore it is specification inheritance (a variety of specialization inheritance).

Specialization inheritance:

The superclass just specifies which methods should be available but doesn't give code.
This is supported in java by interfaces and abstract methods

Construction inheritance

The superclass just specifies which methods should be available but doesn't give code.
This is supported in java by interfaces and abstract methods

Extension inheritance

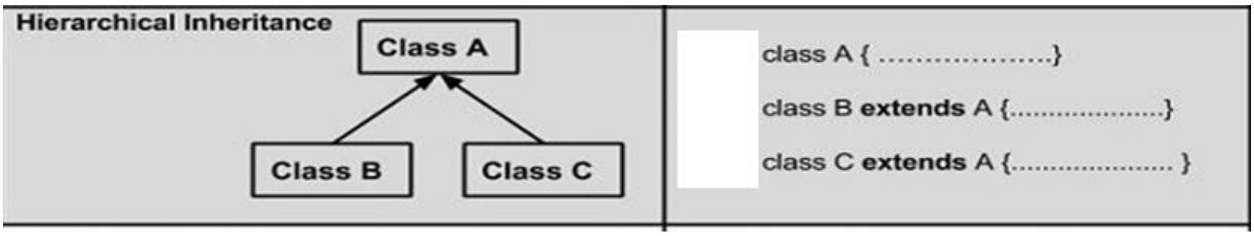
The superclass is just used to provide behavior, but instances of the subclass don't really act like the superclass. Violates substitutability. Exmample: defining Stack as a subclass of Vector. This is not clean -- better to define Stack as having a field that holds a vector
If a child class generalizes or extends the parent class by providing more functionality, but does not override any method, we call it inheritance for generalization.
The child class doesn't change anything inherited from the parent, it simply adds new features.

Preventing a class from inheritance:

If a class is made as final it cannot be extended by another class

```
final Class A
{
….}
Class B extends A    //error since class A is final we can’t inherit it properties
{
….
}
```

2.Write a program to demonstrate hierarchical and multiple inheritance using interfaces



Hierarchical Inheritance program:

```
class A
{
public void display()
{
System.out.println("I am in display method");
}
}
class B extends A
{
public void show()
{
System.out.println(" I am in Show method");
}
}
class C extends A
{
public void printing()
{
System.out.println("I am in printing  method");
}
}


public class Demo
{
public static void main(String args[])
{
B ob=new B();
C oc=new C();
ob.display();//calling display()of Class A using B class object
oc.display();//calling display()of Class A using C class object
ob.show();
oc.printing();
}
}
```

Output:

```
C:\Users\desa\Desktop>java Demo
I am in display method
I am in display method
 I am in Show method
I am in printing  method
```

Multiple Inheritance:

Multiple inheritance is not supported in case of class because of ambiguity. But it is supported in case of interface because there is no ambiguity as implementation is provided by the implementation class.

Program:

```
interface Printable
        {
void print();
        }
interface Showable
        {
void show();
        }

class TestTnterface1  implements Printable, Showable
{
        public void print()
        {
                System.out.println("Hello");
        }
public void show()
    {
System.out.println("Welcome");
          }
        public static void main(String args[])
                {
        TestTnterface1  obj = new TestTnterface1();
        obj.print();
        obj.show();
                }
}
```

Output:

Hello

Welcome

3. Write the significance and internal Architecture of Java Virtual Machine(JVM),. Briefly explain how Java is platform independent

Internal Architecture of JVM

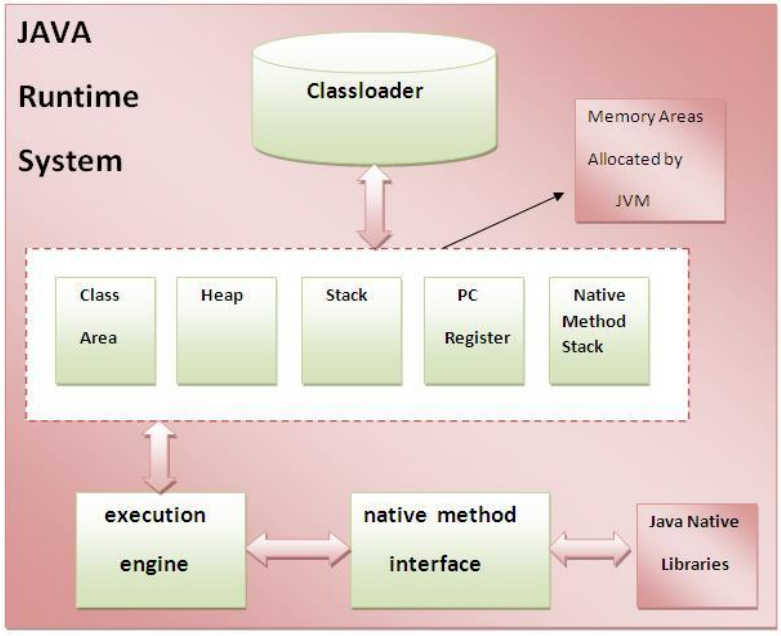
JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent). It is:

1. **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Sun and other companies.
2. **An implementation**Its implementation is known as JRE (Java Runtime Environment).
3. **Runtime Instance** Whenever you write java command on the command prompt to run the java class, and instance of JVM is created.

JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.



Runtime Environment (JRE)

1. **Classloader:**Classloader is a subsystem of JVM that is used to load class files.

2. **Class(Method) Area:**Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.
3. **Heap:**It is the runtime data area in which objects are allocated
4. **Stack**: Java Stack stores frames.It holds local variables and partial results, and plays a part in method invocation and return.Each thread has a private JVM stack, created at the same time as thread. A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.
5. **Program Counter Register:**PC (program counter) register. It contains the address of the Java virtual machine instruction currently being executed.
6. **Native Method Stack**:**It contains all the native methods used in the application**
7. **Execution Engine:**It contains:

    1) **A virtual processor**
    2) **Interpreter:**Readbytecode stream then execute the instructions.

Just-In-Time(JIT) compiler: It is used to improve the performance.JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.Here the term compiler refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

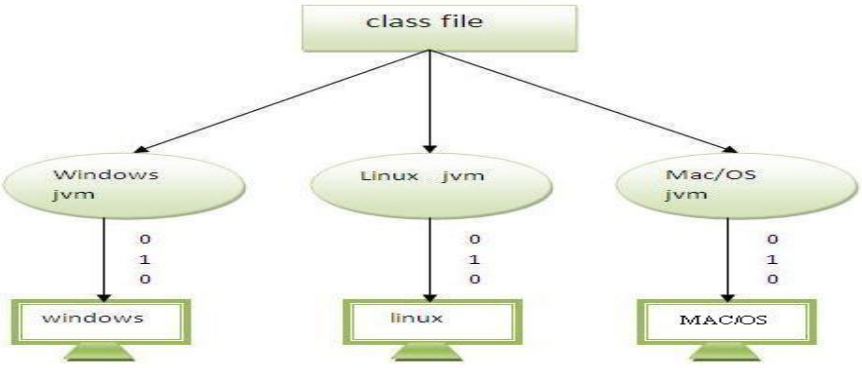
Java is platform independent:

Java provides software-based platform. The Java platform differs from most other platforms in the sense that it's a software-based platform that runs on top of other hardware-based platforms. It has two components:

Runtime Environment

API(Application Programming Interface)

Java code can be run on multiple platforms e.g.Windows,Linux,SunSolaris,Mac/OS etc. Java code is compiled by the compiler and converted into bytecode.*This bytecode is a platform independent code because it can be run on multiple platforms i.e.* ***Write Once and Run Anywhere(WORA).***



4. **What are the drawbacks of procedural languages? Explain the need of object oriented programming with suitable program**.(0r) Write the Difference Between **procedural oriented languages and object oriented programming**

POP (Procedure Oriented Programming)

POP is a programming language that follows a step-by-step approach to break down a task into a collection of variables and routines (or subroutines) through a sequence of instructions. Each step is carried out in order in a systematic manner so that a computer can understand what to do.

Disadvantages

POP (Procedure Oriented Programming)

- Global data are vulnerable(less security)
- Data can move freely within a program
- It is tough to verify the data position.
- Functions are action-oriented.
- Functions are not capable of relating to the elements of the problem.
- Real-world problems cannot be modelled.
- Parts of code are interdependent.
- One application code cannot be used in other application.
- Data is transferred by using the functions.

Basic of Object-Oriented Concepts (OOP)

Object-oriented Programming is a programming language that uses classes and objects to create models based on the real world environment.

- **Objects**: It is considered as a variable of type class and an instance of a class.
- **Class**: It is a set of objects of similar type. A complete set of data and code of an object creates an user-defined data type by using a class.
- **Data abstraction and encapsulation**: Abstraction is nothing but a method of hiding background details and representing essential features. The encapsulation is a method of packing the data and functions into a single unit.
- **Inheritance**: Inheritance is a technique of acquiring features of objects from one class to the other class objects. In other words, it helps in deriving a new class from the existing one.
- **Polymorphism**: Polymorphism provides a method of creating multiple forms of a function by using a single function name.
- **Dynamic binding**: It specifies that the code associated with a particular procedure is not known until the moment of the call at run time.
- **Message passing**: This OOP concept enables interaction between different classes by transmitting and receiving information.

BASIS FOR	POP	OOP

COMPARISON		
Basic	Procedure/Structure oriented .	Object oriented.
Approach	Top-down.	Bottom-up.
Basis	Main focus is on "how to get the task done" i.e. on the procedure or structure of a program .	Main focus is on 'data security'. Hence, only objects are permitted to access the entities of a class.
Division	Large program is divided into units called functions.	Entire program is divided into objects.
Entity accessing mode	No access specifier observed.	Access specifier are "public", "private", "protected".
Overloading or Polymorphism	Neither it overload functions nor operators.	It overloads functions, constructors, and operators.
Inheritance	Their is no provision of inheritance.	Inheritance achieved in three modes public private and protected.
Data hiding & security	There is no proper way of hiding the data, so data is insecure	Data is hidden in three modes public, private, and protected. hence data security increases.
Data sharing	Global data is shared among the functions in the program.	Data is shared among the objects through the member functions.
Friend functions or friend classes	No concept of friend function.	Classes or function can become a friend of another class with the keyword "friend". Note: "friend" keyword is used only in c++
Virtual classes or virtual function	No concept of virtual classes .	Concept of virtual function appear during inheritance.
Example	C, VB, FORTRAN, Pascal	C++, JAVA, VB.NET, C#.NET.

5. Explain about various control statements

A control statement works as a determiner for deciding the next task of the other statements whether to execute or not. An 'If' statement decides whether to execute a statement or which statement has to execute first between the two. In Java, the control statements are divided into three categories which are selection statements, iteration statements, and jump statements. A program can execute from top to bottom but if we use a control statement. We can set order for executing a program based on values and logic.

» **Decision Making in Java**

- Simple if Statement
- if…else Statement
- Nested if statement
- if...else if…else statement
- Switch statement

» **Looping Statements in Java**

- While
- Do…while
- For
- For-Each Loop

» **Branching Statements in Java**

- Break
- Continue
- Return

Decision Making in Java

Decision making statements are statements which decides what to execute and when. They are similar to decision making in real time. Control flow statements control the flow of a program's execution. Here flow of execution will be based on state of a program. We have 4 decision making statements available in Java.

Simple if Statement :

Simple if statement is the basic of decision-making statements in Java. It decides if certain amount of code should be executed based on the condition.

Syn

```
if (condition)
{
Statemen 1; //if condition becomes true then this will be executed
}
    statement 2; //this will be executed irrespective of condition becomes true or false
```

Example:

```
class ifTest  {
   public static void main(String args[])   {
     int x = 5;
      if (x > 10)
         System.out.println("Inside If");
      System.out.println("After if statement");
   }
}
```

Output:

After if statement

if…else Statement :

In if…else statement, if condition is true then statements in if block will be executed but if it comes out as false then else block will be executed.

Syntax:

```
if (condition) {

Statemen 1; //if condition becomes true then this will be executed

}
```

Example:

```
class ifelseTest  {
   public static void main(String args[])   {
      int x = 9;
      if (x > 10)
```

```
        System.out.println("i is greater than 10");
      else
        System.out.println("i is less than 10");
      System.out.println("After if else statement");
   }
}
```

Output:

i is less than 10
After if else statement

Nested if statement :

Nested if statement is if inside an if block. It is same as normal if…else statement but they are written inside another if…else statement.

Syntax:

```
if (condition1)
        {
Statemen 1; //executed when condition1 is true
if (condition2)
        {
Statement 2; //executed when condition2 is true
        }
else
        {
 Statement 3; //executed when condition2 is false
        }
}
```

Example:

```
class nestedifT
   public static
      int x = 25;
      if (x > 10)    {
         if (x%2==0)
         System.out.println("i is greater than 10 and even number");
         else
         System.out.println("i is greater than 10 and odd number");
      }
      else   {
         System.out.println("i is less than 10");
      }
      System.out.println("After nested if statement");
   }
}
```

Output:

i is greater than 10 and odd number
After nested if statement

if…else statement :

if…else if statements will be used when we need to compare the value with more than 2 conditions. They are executed from top to bottom approach. As soon as the code finds the matching condition, that block wil be executed. But if no condition is matching then the last else statement will be executed.

Syntax:

```
if (condition1) {
Statemen 1; //if condition1 becomes true then this will be executed
}
else if (condition2) {
  Statement 2; // if condition2 becomes true then this will be executed
}
....
....
else {
 Statement 3; //executed when no matching condition found
}
```

Example:

```
class ifelseifTest {
   public static void main(String args[]) {
      int x = 2;
        if (x > 10)  {
          System.out.println("i is greater than 10");
      }
      else if (x <10)
          System.out.println("i is less than 10");
      }
      else  {
          System.out.println("i is 10");
      }
      System.out.println("After if else if ladder statement");
   }
}
```

Output:

```
i is less than 10
After if else if ladder statement
```

Switch statement :

Java switch statement compares the value and executes one of the case blocks based on the condition. It is same as if…else if ladder. Below are some points to consider while working with switch statements:

» case value must be of the same type as expression used in switch statement
» case value must be a constant or literal. It doesn't allow variables
» case values should be unique. If it is duplicate, then program will give compile time error

Let us understand it through one example.

```
class switchDemo{
 public static void main(String args[])  {
   int i=2;
   switch(i)
{
     case 0:
      System.out.println("i is 0");
       break;
      case 1:
       System.out.println("i is 1");
       break;
      case 2:
      System.out.println("i is 2");
      break;
      case 3:
      System.out.println("i is 3");
      break;
      case 4:
      System.out.println("i is 4");
      break;
      default:
       System.out.println("i is not in the list");
       break;
 }
}
}
```

Output:
i is 2

Looping Statements in Java :

Looping statements are the statements which execute a block of code repeatedly until some condition mee to the criteria. Loops can be considered as repeating if statements. There are 3 types of loops available in Java.

While :

While loops are simplest kind of loop. It checks and evaluates the condition and if it is true then executes the body of loop. This is repeated until the condition becomes false. Condition in while loop must be given as a Boolean expression. If int or string is used instead, compile will give the error.

Syntax:

```
Initialization;
        while (condition)
        {
 statement1;
increment/decrement;
        }
```

Example:

```
class whileLoopTest
{
   public static void main(String args[])
{
      int j = 1;
      while (j <= 10)     {
         System.out.println(j);
         j = j+2;
      }
   }
}
```

Output:

1 3 5 7 9

Do…while :

Do…while works same as while loop. It has only one difference that in do…while, condition is checked after the execution of the loop body. That is why this loop is considered as exit control loop. In do…while loop, body of loop will be executed at least once before checking the condition

Syntax:

```
do
{
statement1;
}while(condition);
```

Example:

```
class dowhileLoopTest  {

   public static void main(String args[])   {

      int j = 10;

        do     {

          System.out.println(j);

          j = j+1;

      } while (j <= 10) ;

   }

}
```

Output: 10

For Statement :

It is the most common and widely used loop in Java. It is the easiest way to construct a loop structure in code as initialization of a variable, a condition and increment/decrement are declared only in a single line of code. It is easy to debug structure in Java.

Syntax:

```
for (initialization; condition; increment/decrement)
  {
    statement;
}
```

Example:

```
class forLoopTest
 {
   public static void main(String args[])
{
       for (int j = 1; j <= 5; j++)
          System.out.println(j);
   }
}
```

Output:

```
1
 2
  3
  4
  5
```

For-Each Loop :

For-Each loop is used to traverse through elements in an array. It is easier to use because we don't have to increment the value. It returns the elements from the array or collection one by one.

Example:

```
class foreachDemo {

 public static void main(String args[]) {

   int a[] = {10,15,20,25,30};

    for (int i : a) {

        System.out.println(i);

    }

 }
```

Output:

```
10
15
20
```

25
30

Branching Statements in Java :

Branching statements jump from one statement to another and transfer the execution flow. There are 3 branching statements in Java.

Break :

Break statement is used to terminate the execution and bypass the remaining code in loop. It is mostly used in loop to stop the execution and comes out of loop. When there are nested loops then break will terminate the innermost loop.

Example:

```
class breakTest   {

   public static void main(String args[])     {

      for (int j = 0; j < 5; j++)    {

         // come out of loop when i is 4.

         if (j == 4)

            break;

         System.out.println(j);

      }

      System.out.println("After loop");

   }

}
```

Output:

```
0
1
2
3
4
After loop
```

Continue :

Continue statement works same as break but the difference is it only comes out of loop for that iteration and continue to execute the code for next iterations. So it only bypasses the current iteration.

Example:

```
class continueTest   {

   public static void main(String args[])     {

      for (int j = 0; j < 10; j++)     {

         // If the number is odd then bypass and continue with next value

         if (j%2 != 0)

            continue;

         // only even numbers will be printed

         System.out.print(j + " ");

      }

   }

}
```

Output:

0 2 4 6 8

Return :

Return statement is used to transfer the control back to calling method. Compiler will always bypass any sentences after return statement. So, it must be at the end of any method. They can also return a value to the calling method.

6. Explain the significance of public, protected and private access specifies in Inheritance.

(or) Describe different levels of access protection available in Java.

There are two types of modifiers in java: **access modifier** and **non-access modifier**. The access modifiers specifies accessibility (scope) of a datamember, method, constructor or class. There are 4 types of access modifiers:

1. private
2. default
3. protected
4. public

There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient etc.

1) private access modifier:

The private datamembers, method ,contructor are accessible only within class in which their are declared and out of the class the doesn't have scope.

Example of private access modifier:

In this example, we have created two classes 'A' and 'Simple'. 'A' class contains private data member and private method. We are accessing these private members from outside the class, so there is compile time error.

```
class A
{
private int data=40;
private void msg()
     {
System.out.println("Hello java");
     }
}
  public class Simple
{
 public static void main(String args[])
     {
     A obj=new A();
     System.out.println(obj.data);//Compile Time Error
     obj.msg();//Compile Time Error
   }
}
```

2) default access modifier:

If you don't use any modifier, it is treated as default by default. The default modifier is accessible only within package.

Example of default access modifier:

In this example, we have created two packages "pack" and "mypack". We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

//save by A.java

```
package pack;
class A
{
void msg( )
        {
System.out.println("Hello");
        }
}
```

//save by B.java

```
package mypack;
import pack.*;
  class B
{
 public static void main(String args[])
        {
 A obj = new A();//Compile Time Error
 obj.msg();//Compile Time Error
        }
}
```

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

ACCESS MODIFIER	PUBLIC	PROTECTED	PRIVATE
Same class	YES	YES	YES
Subclass in same package	YES	YES	NO
Other classes in same package	YES	YES	NO
Subclass in other packages	YES	YES	NO
Non subclasses in other packages	YES	NO	NO

7.With suitable example demonstrate super, this ,final and static keywords

Super Keyword: The **super** keyword in java is a reference variable which is used to refer immediate paren class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable

Usage of java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor. The use of super keyword

super can be used to invoke immediate parent class method	super() can be used to invoke immediate parent class constructor. The use of super keyword

class Base
{
String color="white";
void display()
{
System.out.println("have a nice day");
}
}
Class Sub extends Base
{
String color="black";
void display()
{
System.out.println("Hello ALL...");
}
void printColor()
{
System.out.println(color);//prints color of sub class
System.out.println(super.color);// prints color of Base class
}
 	
void work()
{
display();
super.display();
 	
}
}
class TestSuper2
{
public static void main(String args[])
{
Sub ob=new Sub();
ob. printColor();
ob.work();
}
	class Parentclass
	{
	Parentclass()
	{
	System.out.println("Constructor of parent class");
	}
	}
	class Subclass extends Parentclass
	{
	Subclass()
	{
	/*Compile implicitly adds super() here as the first statement of this constructor.*/
	System.out.println("Constructor of child class");
	}
	Subclass(int num)
	{
	/*Even though it is a parameterized constructor The compiler still adds the super() here */
	System.out.println("arg constructor of child class");
	}
	void display()
	{
	System.out.println("Hello!");
	}
	public static void main(String args[])
	{
	Subclass obj= new Subclass();
	//Calling sub class method
	obj.display();
	Subclass obj2= new Subclass(10);
	obj2.display();
	}
	}

}	
Output black white Hello ALL… have a nice day	**Output:** Constructor of parent class Constructor of child class Hello! Constructor of parent class arg constructor of child class Hello!

Final Keyword In Java

The final keyword in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

- variable
- method
- class

if a variable is made as final it cannot change its value

if a method is made as final it cannot override it.

If a class is made as final it cannot be extended by another class

```
final Class A
{
….}
Class B extends A    //error since class A is final we can't inherit it properties
{
….
}
```

The this Keyword

Sometimes a method will need to refer to the object that invoked it. To allow this, Java defines

the this keyword. this can be used inside any method to refer to the current object. That is,

this is always a reference to the object on which the method was invoked. You can use this

anywhere as reference to an object of the current class' type is permitted.

To better understand what this refers to, consider the following version of Box( ):

class Box {

double width;

double height;

double depth;

// This is the constructor for Box.

```
Box(double w, double h, double d) {
this.width = w;
this.height = h;
this.depth = d;
}
// compute and return volume
double volume() {
return width * height * depth;
}
}
class BoxDemo {
public static void main(String args[]) {
Box mybox1 = new Box(10, 20, 15);
Box mybox2 = new Box(3, 6, 9);
double vol;
// get volume of first box
vol = mybox1.volume();
System.out.println("Volume is " + vol);
// get volume of second box
vol = mybox2.volume();
System.out.println("Volume is " + vol);
}
}
```

Inside Box( ), this will always refer to the invoking object.

Static keyword:

If a variable or a method is made as static ,then this variables and methods can be called with out using the class object.

Example:

```
class staticdemo
{
static int i=10;
static void display()
{
System.out.println("I am in static display");
}
public static void main(String args[])
{
System.out.println(" i value is"+i);// variable “I”  is accessed with out class object
display();// function is called without class object
}
}
```

Output:

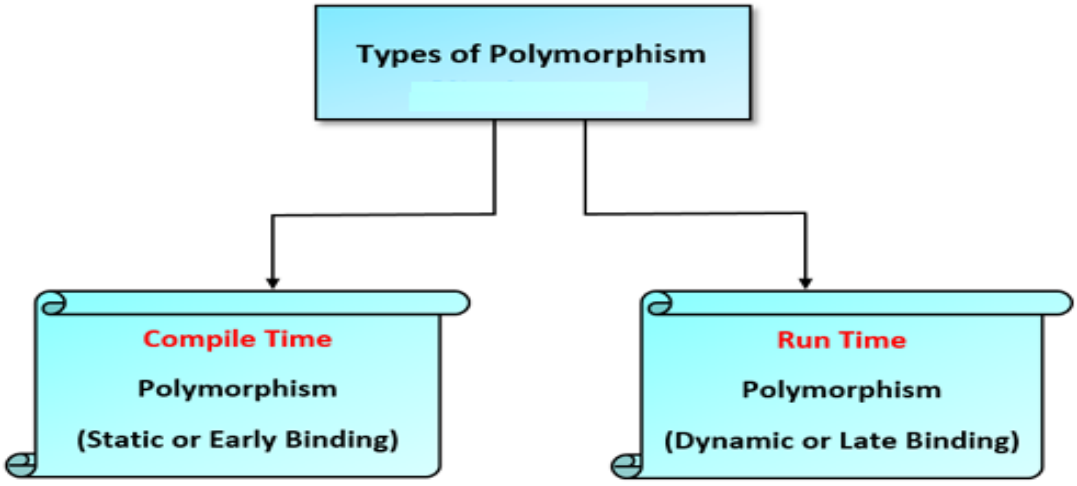
```
C:\Users\desa\Desktop>java staticdemo
 i value is10
I am in static display
```

8.What is polymorphism? Explain different types of polymorphisms with examples.
(or)
Compare and cons tract overloading and overriding with an example

Ans: Polymorphism in Java is a concept by which we can perform a *single action in different ways*. Polymorphism is derived from two Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

There are two types of polymorphism in Java:
1. Compile-time polymorphism or Ad hoc polymorphism
2. Runtime polymorphism. Or Pure polymorphism

We can perform polymorphism in java by method overloading and method overriding.



2.3 Types of polymorphism

1.Compile-time polymorphism or Ad hoc polymorphism:

Ad hoc polymorphism is also known as function overloading or operator overloading because a polymorphic function can represent a number of unique and potentially different implementations depending on the type of argument it is applied to.

i) ***Operator overloading:***

Java also provide option to overload operators. For example, we can make the operator ('+') for string class to concatenate two strings. We know that this is the addition operator whose task is to add two operands. So a single operator '+' when placed between integer operands, adds them and when placed between string operands, concatenates them.In java, Only "+" operator can be overloaded:

2.Method Overloading:

When there are multiple functions with same name but different parameters then these functions are said to be overloaded. Functions can be overloaded by change in number of arguments or/and change in type of arguments. Overloaded methods are generally used when they conceptually execute the same task but with a slightly different set of parameters

Example on Operator overloading	*Example on Method overloading*
// Java program for Operator	class MultiplyFun

overloading
class Operatoroverloading
{
void operator(String str1, String str2)
{
String s = str1 + str2;
System.out.println("Concatinated String - "+ s);
}
void operator(int a, int b)
{
int c = a + b;
System.out.println("Sum = " + c);
}
}
 	
class Main {
public static void main(String[] args)
{
Operatoroverloading obj = new Operatoroverloading ();
obj.operator(2, 3);
obj.operator("hi", "hello");
}
}
 	
Output:
Sum = 5
Concatinated String -hihello
	{
	// Method with 2 parameter
	static int Multiply(int a, int b)
	{
	return a * b;
	}
	// Method with 3 parameter
	static int Multiply(int a, int b,int c)
	{
	return a * b*c;
	}
	// Method with the same name but 2 double parameter
	static double Multiply(double a, double b)
	{
	return a * b;
	}
	}
	
	class Main {
	public static void main(String[] args)
	{
	System.out.println(MultiplyFun.Multiply(2, 4));
	System.out.println(MultiplyFun.Multiply(2, 4,2));
	System.out.println(MultiplyFun.Multiply(5.5,6.3));
	}
	}
	Output:
	16
	34.65

Note:
we overload static methods
we cannot overload methods that differ only by static keyword
we overload main() in Java

2.Runtime polymorphism. Or Pure polymorphism

Runtime polymorphism or **Dynamic Method Dispatch** is a process in which a call to an ***<u>overridden</u>*** method is resolved at runtime rather than compile-time.

Difference between Method overloading and Method overriding

S.no	*Method overloading*	*Method overriding*
1	When a class have same method name with different argument, than it is called method overloading.	Method overriding - Method of superclass is **overridden** in subclass **to provide more specific implementation**.
2	Method overloading is **generally done in same class** but **can also be done in SubClass** .	Method overriding is always done in subClass in java.
3	Both **Static and instance method can be overloaded** in java.	**Only instance methods can be overridden** in java. **Static methods can't be overridden** in java.
4	**Main method can also be overloaded** in java	**Main method can't be overridden** in java, because main is static method and static methods can't be overridden in java (as mentioned in above point)
5	**private methods can be overloaded** in java.	**private methods can't be overridden** in java, because private methods are not inherited in subClass in java.
6	**final methods can be overloaded** in java.	**final methods can't be overridden** in java, because final methods are not inherited in subClass in java.
7	Call to overloaded method is **bonded** at **compile time** in java.	Call to overridden method is **bonded** at **runtime** in java**.**
8	Method overloading concept is also known as **compile time polymorphism or ad hoc polymorphism or static binding** in java.	Method overriding concept is also known as **runtime time polymorphism or pure polymorphism or Dynamic binding** in java.

9)What is a nested class? Differentiate between static nested classes and non-static nested classes.
A nested class (or commonly called inner class) is a class defined inside another class
As anillustration, two nested classes MyNestedClass1 and MyNestedClass2 are defined inside the definition of an outer class called MyOuterClass.

public class MyOuterClass { // outer class defined here
.....
private class MyNestedClass1 { ..... } // an nested class defined inside the outer class**(inner class)**
public static class MyNestedClass2 { ..... } // an "static" nested class defined inside the outer class
.....
}
A nested class has access to the members, including private members, of the class in which it is nested. However, the enclosing class does not have access to the members of the nested class.
There are two types of nested classes: static and non-static. A static nested class is one that has the static modifier applied.
The most important type of nested class is the inner class. An inner class is a non-static nested class. The following program illustrates how to define and use an inner class. The class named **Outer** has one instance variable named *outer_x,* one instance method named test( ), and defines one inner class called **Inner**

```
// Demonstrate an inner class.
class Outer {
int outer_x = 100;
void test() {
Inner inner = new Inner();
inner.display();
}
// this is an inner class as it is a non static class
class Inner {
void display() {
System.out.println("display: outer_x = " + outer_x);
}
}
}
class InnerClassDemo {
public static void main(String args[]) {
Outer outer = new Outer();
outer.test();
}
}
```

Output from this application is shown here:
display: outer_x = 100

Properties of nested class:

1. A nested class is a proper class. That is, it could contain constructors, member variables and member methods. You can create an instance of a nested class via the new operator and constructor.
2. A nested class is a member of the outer class, just like any member variables and methods defined inside a class.

3. Most importantly, a nested class can access the private members (variables/methods) of the enclosing outer class, as it is at the same level as these private members, but the reverse is not true. Members of the inner class are known only within the scope of the inner class and may not be used by the outer class
4. A nested class can have private, public, protected, or the default access, just like any member variables and methods defined inside a class. A private inner class is only accessible by the enclosing outer class, and is not accessible by any other classes. [An top-level outer class cannot be declared private, as no one can use a private outer class.]
5. A nested class can also be declared static, final or abstract, just like any ordinary class.
6. A nested class is NOT a subclass of the outer class. That is, the nested class does not inherit the variables and methods of the outer class. It is an ordinary self-contained class. [Nonetheless, you could declare it as a subclass of the outer class, via keyword "extends OuterClassName", in the nested class's definition.]

The usages of nested class are:

1. To control visibilities (of the member variables and methods) between inner/outer class. The nested class, being defined inside an outer class, can access private members of the outer class.
2. To place a piece of class definition codes closer to where it is going to be used, to make the program clearer and easier to understand.
3. for namespace managemen

10) What is the purpose of constructor in Java programming? Explain the different types of constructors with an example.

- **Constructor** is a **special type of method** that is used to initialize the object.

- Constructor is **invoked at the time of object creation**. It constructs the values i.e. provides data for the object that is why it is known as constructor.

Rules for creating Constructor

There are basically two rules defined for the constructor.

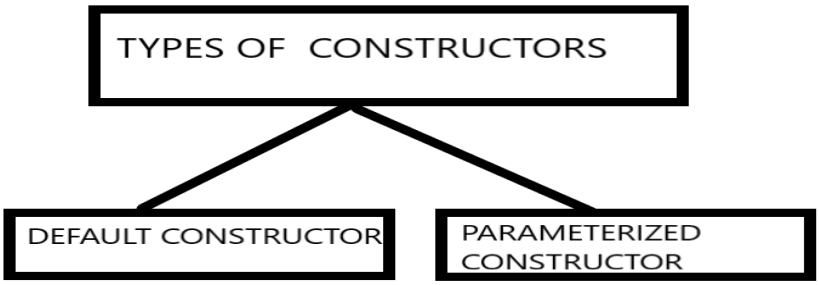
1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

Types of Constructors

There are two types of constructors:

1. default constructor (no-arg constructor)

2. parameterized constructor



2.2 Types of constructors

1) Default Constructor

A constructor that have no parameter is known as default constructor.

Syntax of Default Constructo

```
class_name( )  {
Statements;
}
```

Example of Default Constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
class Bike   {
        Bike()  {
                System.out.println("Bike is created");
                }
        public static void main(String args[]) {
                Bike b=new Bike();
                }
}
```

Output: Bike is created

Rule: If there is no constructor in a class, compiler automatically creates a default constructor.

Purpose of Default Constructor

Default constructor provides the default values to the object like 0, null etc. depending on the type.

2. Parameterized constructor

A constructor that has parameters is known as parameterized constructor**.**

Usage of Parameterized Constructor

Parameterized constructor is used to provide different values to the distinct objects**.**

Example on Parameterized constructor :.

```
class Student   {
                  int id;
                  String name;
                  Student(int i,String n)  //parameterized constructor   {
                            id = i;
                           name = n;
                          }
                void display()  {
                          System.out.println(id+" "+name);
                          }
                public static void main(String args[])  {
                           Student s1 = new Student(111,"abc");
                          Student s2 = new Student(222,"def");
                          s1.display();
                          s2.display();
                           }  }
```

Output:

111 abc

222 def

11) Write a program to find the transpose of a given matrix.

```
public class Transpose {
   public static void main(String[] args) {
```

```
        int row = 2, column = 3;
        int[][] matrix = { {2, 3, 4}, {5, 6, 4} };
        // Display current matrix
        display(matrix);
        // Transpose the matrix
        int[][] transpose = new int[column][row];
        for(int i = 0; i < row; i++) {
            for (int j = 0; j < column; j++) {
                transpose[j][i] = matrix[i][j];
            }
        }
        // Display transposed matrix
        display(transpose);
    }
    public static void display(int[][] matrix) {
        System.out.println("The matrix is: ");
        for(int[] row : matrix) {
            for (int column : row) {
                System.out.print(column + "    ");
            }
            System.out.println();
        }
    }
}
```

```
C:\Users\desa\Desktop>java Transpose
The matrix is:
2    3    4
5    6    4
The matrix is:
2    5
3    6
4    4
```

12) Explain the different parameter passing mechanisms used in Java with an example.

Ans: There are two diffe

a) pass by values (or) call by value

b) pass by objects (or) call by object

program to demonstrate call by value and call by object

```
class Test2 {
int x=10,y=20;
static void display(Test2 obj)
{
System.out.println(" x value is"+obj.x);
System.out.println(" y value is"+obj.y);
}
void display(int a,int b)
```

```
{
 System.out.println("a value is"+a);
System.out.println("b value is"+b);
}
public static void main(String args[]) {
Test2 obj = new Test2();
display(obj);   //call by object
obj.display(100,200); //call by value

}
}
```

```
C:\Users\desa\Desktop>java Test2
 x value is10
 y value is20
a value is100
b value is200
```

UNIT-II

1. How to define a package?Explain with suitable example how to create ,import and access apackage?

Answer:

A java **package** is a group of similar types of classes, interfaces and sub-packages. Package in java can be categorized in two forms,

i) Built-in package and

ii) User-defined package.

Built-in package:

Collection of classes & interfaces which are already defined are called as Build-in packages. There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

User-defined packages:

The collection of classes and interfaces for which definition is provided by the developer or programmer is called as user-defined packages.

Defining a user-defined package

To create a user defined package in java we use a keyword "**package**"

Syntax:

package packagename[.subpackage1][.subpackage2]…[.subpackageN];

Sample program on user defined package:

```
//save as Simple.java
package mypack;
public class Simple {
public static void main(String args[]) {
System.out.println("Welcome to package");
}
}
```

compile java package:

If you are not using any IDE, you need to follow the syntax given below:

Syntax: **javac -d directory javafilename**

For example 1. **javac -d . Simple.java** The -d switch specifies the destination where to put the generated class file.

You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use. (dot).

run java package program: You need to use fully qualified name e.g. mypack.Simpleetc to run the class.

To Compile: javac -d . Simple.java

To Run: java mypack.Simple

Output: Welcome to package

The -d is a switch that tells the compiler where to put the class file i.e. it represents destination. The. (dot) Represent the current folder.

Importing packages

There are three ways to access the package from outside the package.

1. import package.*;
2. import package.classname;

3. fully qualified name.

1) Using packagename.*

- If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.
- The import keyword is used to make the classes and interface of another package accessible to the current package.

Example of package that import the packagename.*

```
//save by A.java
package pack;
public class A {
public void msg() {
System.out.println("Hello");
}
}
//save by B.java
package mypack;
import pack.*;
class B {
public static void main(String args[]) {
A obj = new A();
obj.msg();
}
}
```

Output: Hello

2) Using packagename.classname

- If you import package.classname then only declared class of this package will be accessible.

Example of package by import package.classname

```
//save by A.java
package pack;
public class A {
public void msg() {
System.out.println("Hello");
}
}
//save by B.java
package mypack;
import pack.A;
class B {
public static void main(String args[]) {
A obj = new A(); obj.msg();
}
}
```

Output: Hello

3) Using fully qualified name

- If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.
- It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class. Example of package by import fully qualified name.

//save by A.java

```
package pack;
public class A {
    public void msg() { System.out.println("Hello"); }
}
//save by B.java
package mypack;
class B {
public static void main(String args[]) {
pack.Aobj = new pack.A();//using fully qualified name
obj.msg();
}
}
```

Output: Hello

Note: If you import a package, sub packages will not be imported. If you import a package, all the classes and interface of that package will be imported excluding the classes and interfaces of the sub packages. Hence, you need to import the sub package as well.

2. Write a program to implement the operations of random access file

Answer:

- RandamAccessFile class supported by java.io package allows us to create files that can be used for reading and writing data with random access.
- This class implements DataInput, DataOutput and Closeable Interfaces
- It supports positioning request that means we can position the file pointer with in the file.
- It has two constructors

RandamAccessFile(File fileObj,String access)throws FileNotFoundException
RandamAccessFile(String filename,String access)throws FileNotFoundException
In both cases, access determines what type of file access is permitted.

Access	Purpose
r	File opened for reading purpose
rw	File opened for read-write purpose
rws	File opened for read-write purpose and every change to the file's data or metadata will be immediately written to physical device
rwd	File opened for read-write purpose& every change to the file's data will be immediately written to physical device

The method **seek()** is used to set the current position of the file pointer

Syntax:

void seek(long newPos)throws IOException

Here, newPos specifies the new position, in bytes, of the file pointer from the beginning of the file**.**

```
//writing  and reading with random access
import java.io.*;
class RandomIO {
public static void  main(String args[]) {
RandamAccessFile file=null;
try  {
file=new RandomAccessFile(“rand.dat”, “rw”);
//Writing to the file
file.writeChar(‘x’);
file.writeInt(333);
file.writeDouble(3.1412);
file.seek(0);//go to the beginning
System.out.println(file.readChar());
System.out.println(file.readInt());
System.out.println(file.readDouble());
file.seek(2);  //go to the second item
System.out.println(file.readInt());  //go to the end and append false to the file  file.seek(file.length());
file.writeBoolean(false);
file.seek(4);
System.out.println(file.readBoolean());
file.close();
}
catch(IOException e) {
System.out.println(e);
}
}
}
```

Output:

```
x
333
3.1412
333
False
```

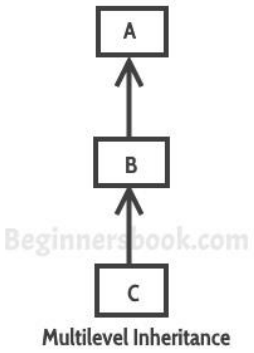
3. Write a program to compute an average of the values in a file

```
package txtfileaverage;
import java.io.*;
import java.util.Scanner;
public class Txtfile {

   public static void main(String args[])  throws IOException
    {
      Scanner file = new Scanner(new File("input.txt"));

intnumTimes = file.nextInt();
file.nextLine();

          for(inti = 0; i<numTimes; i++);
          {
int sum = 0;
int count = 0;
            Scanner split = new Scanner(file.nextLine());
            while(split.hasNextInt())
              //for (int a = 0; a < 4 ; a++)
              {
       sum += split.nextInt();
       count++;
              }
System.out.println("the average is = " + ((double)sum / count));

          }
             }
}
```

4. a)Explain multilevel inheritance with the help of abstract class in your program

When a class extends a class, which extends anther class then this is called **multilevel inheritance**. For example class C extends class B and class B extends class A then this type of inheritance is known as multilevel inheritance.

Lets see this in a diagram:



It's pretty clear with the diagram that in Multilevel inheritance there is a concept of grand parent class. If we take the example of this diagram, then class C inherits class B and class B inherits class A which means B is a parent class of C and A is a parent class of B. So in this case class C is implicitly inheriting the properties and methods of class A along with class B that's what is called multilevel inheritance.

```
abstract class A{ //Abstract Class
void disp1(){
System.out.println("class A");
            }
}
abstract class B extends A{ //Abstract Class     abstract void disp2(); //Abstract Method
}
class C extends B{
void disp2(){                System.out.println("class B");          }
public static void main(String args[]){
  C c = new C();
c.disp1();
c.disp2();
}
}
Output :
class A
class B
```

b)Can inheritance be applied between interfaces? Justify your answer.
(or) **How can you extend one interface by the other interface? Discuss**
Answer: Interfaces Can Be Extended
One interface can inherit another by use of the keyword extends. The syntax is the same as for inheriting classes. When a class implements an interface that inherits another interface, it must provide implementations for all methods defined within the interface inheritance chain.
Following is an example:

```
 // One interface can extend another.
interface A {
void meth1();
 void meth2();
}
// B now includes meth1() and meth2() -- it adds meth3().
interface B extends A {
void meth3();
}
// This class must implement all of A and B class
MyClass implements B {
public void meth1() {
System.out.println("Implement meth1().");
}
public void meth2() {
System.out.println("Implement meth2().");
}
public void meth3() {
System.out.println("Implement meth3().");
                                        }
}
class IFExtend {
public static void main(String arg[]) {
MyClassob = new MyClass(); ob.meth1();
ob.meth2();
ob.meth3();
}
}
```

5. What is an interface? How to design and implement an interface in Java? Differentiate between interface and abstract class.

Interfaces are syntactically similar to classes, but they lack instance variables, and their methods are declared without any body. In practice, this means that you can define interfaces that don't make assumptions about how they are implemented. Once it is defined, any number of classes can implement an interface. Also, one class can implement any number of interfaces. To implement an interface, a class must create the complete set of methods defined by the interface. However, each class is free to determine the details of its own implementation. By providing the interface keyword, Java allows you to fully utilize the "one interface, multiple methods" aspect of polymorphism.

An interface in java is a blueprint of a class. It has static constants and abstract methods.

- The interface in java is a mechanism to achieve abstraction.
- There can be only abstract methods in the java interface not method body.
- It is used to achieve abstraction and multiple inheritance in Java. o
- Java Interface also represents IS-A relationship.
- It cannot be instantiated just like abstract class.

They are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritances.
- It can be used to achieve loose coupling.

Defining an Interface Syntax:

```
public interface NameOfInterface {
// Any number of final, static fields
// Any number of abstract method declarations
}
```

In other words, Interface fields are public, static and final by default, and methods are public and abstract.

Example on class implementing interface

```
interface MyInterface {
public void method1();
}
class Demo implements MyInterface {
public void method1() {
System.out.println("Implementation of method1");
}
public static void main(String arg[]) {
Demo obj=new Demo();
obj.method1() ;
}
}
```

Output: Implementation of method1

Difference between abstract class and interface

- Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods.
- Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.

Abstract class	Interface
1) Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
2) Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
3) Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
4) Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
5) The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.

6) Example: public abstract class Shape{ public abstract void draw(); } Example: public	Interface Drawable{ void draw(); }

6. Write a program to copy the contents of file1 to file 2. Read the names of files as line arguments.

Reading/Writing Characters:

The subclasses of Reader and Writer implements streams that handle characters. The two subclasses used are

FileReader-for reading characters

FileWriter –for writing characters

Example to copy contents of file "input.dat" into file "output.dat"

```
//copying characters from one file into another

import java.io.*;
class CopyCharacters  {
public static void main(String args[])
{  //Declare and create input and output file
File inFile=new File("input.dat");
File outFile=new File("output.dat");
FileReader ins=null;//creates file stream ins
FileWriter outs=null;//creates file stream outs
try  {
 ins=new FileReader(inFile);  //opens inFile
outs=new FileWriter(outFile);//opens outFile
//read and write till the end
intch;
while((ch=ins.read())!=-1)   {
outs.write(ch);
 }
}
catch(IOException e)  {
System.out.println(e);
System.exit(-1);
}
Finally  {   try  {
ns.close();
outs.close();
 }
catch(IOException e)    {
 }
 }}}
```

7. What support is provided by File class for file management? Illustrate with suitable scenarios.

The **File** is a built-in class in Java. In java, the File class has been defined in the **java.io** package. The File class represents a reference to a file or directory. The File class has various methods to perform operations like creating a file or directory, reading from a file, updating file content, and deleting a file or directory.

The File class in java has the following constructors.

Constructor with Description

- File(String pathname)

It creates a new File instance by converting the givenpathname string into an abstract pathname. If the given string isthe empty string, then the result is the empty abstract pathname.

- File(String parent, String child)

It Creates a new File instance from a parent abstractpathname and a child pathname string. If parent is null then the new File instance is created as if by invoking thesingle-argument File constructor on the given child pathname string.

- File(File parent, String child)

It creates a new File instance from a parent abstractpathname and a child pathname string. If parent is null then the new File instance is created as if by invoking thesingle-argument File constructor on the given child pathname string.

- File(URI uri)

It creates a new File instance by converting the given file: URI into an abstract pathname.

Methods with Description

1.String getName()

It returns the name of the file or directory that referenced by the current File object.

2.String getParent()

It returns the pathname of the pathname's parent, or null if the pathname does not name a parent directory.

3.String getPath()

It returns the path of curent File.

4.File getParentFile()

It returns the path of the current file's parent; or null if it does not exist.

5.String getAbsolutePath()

It returns the current file or directory path from the root.

6.boolean isAbsolute()

It returns true if the current file is absolute, false otherwise.

7.boolean isDirectory()

It returns true, if the current file is a directory; otherwise returns false.

8.boolean isFile()

It returns true, if the current file is a file; otherwise returns false.

9.boolean exists()

It returns true if the current file or directory exist; otherwise returns false.

10.boolean canRead()

It returns true if and only if the file specified exists and can be read by the application; false otherwise.

11.boolean canWrite()

It returns true if and only if the file specified exists and the application is allowed to write to the file; false otherwise.

12.long length()

It returns the length of the current file.

13.long lastModified()

It returns the time that specifies the file was last modified.

14.boolean createNewFile()

It returns true if the named file does not exist and was successfully created; false if the named file already exists.

15.boolean delete()

It deletes the file or directory. And returns true if and only if the file or directory is successfully deleted; false otherwise.

16.void deleteOnExit()

It sends a requests that the file or directory needs be deleted when the virtual machine terminates.

17.boolean mkdir()

It returns true if and only if the directory was created; false otherwise.

18.boolean mkdirs()

It returns true if and only if the directory was created, along with all necessary parent directories; false otherwise.

19.boolean renameTo(File dest)

It renames the current file. And returns true if and only if the renaming succeeded; false otherwise.

20.boolean setLastModified(long time)

It sets the last-modified time of the file or directory. And returns true if and only if the operation succeeded; false otherwise.

21.boolean setReadOnly()

It sets the file permission to only read operations; Returns true if and only if the operation succeeded; false otherwise.

22.String[] list()

It returns an array of strings containing names of all the files and directories in the current directory.

23.String[] list(FilenameFilter filter)

It returns an array of strings containing names of all the files and directories in the current directory that satisfy the specified filter.

24.File[] listFiles()

It returns an array of file references containing names of all the files and directories in the current directory.

25.File[] listFiles(FileFilter filter)

It returns an array of file references containing names of all the files and directories in the current directory that satisfy the specified filter.

26.boolean equals(Object obj)

It returns true if and only if the argument is not null and is an abstract pathname that denotes the same file or directory as this abstract pathname.

27.int compareTo(File pathname)

It Compares two abstract pathnames lexicographically. It returns zero if the argument is equal to this abstract pathname, a value less than zero if this abstract pathname is lexicographically less than the argument, or a value greater than zero if this abstract pathname is lexicographically greater than the argument.

28int compareTo(File pathname)

Compares this abstract pathname to another object. Returns zero if the argument is equal to this abstract pathname.

```
importjava.io.*;
publicclassFileClassTest{


    publicstaticvoidmain(Stringargs[]){
            File  f  =newFile("C:\\Raja\\datFile.txt");


            System.out.println("Executable File : "+f.canExecute());
            System.out.println("Name of the file : "+f.getName());
            System.out.println("Path of the file : "+f.getAbsolutePath());
            System.out.println("Parent name :  "+f.getParent());
            System.out.println("Write mode :  "+f.canWrite());
            System.out.println("Read mode :  "+f.canRead());
            System.out.println("Existance :  "+f.exists());
            System.out.println("Last Modified :  "+f.lastModified());
            System.out.println("Length :  "+f.length());
            //f.createNewFile()
            //f.delete();
            //f.setReadOnly()
    }
}
```

8. **a)Demonstrate ordinal( ) method of enum.**

The **java.lang.Enum.ordinal()** method returns the ordinal of this enumeration constant (its position in its enum declaration, where the initial constant is assigned an ordinal of zero).

Declaration

Following is the declaration for **java.lang.Enum.ordinal()** method

public final int ordinal();

ParametersNot Available

Return Value

This method returns the ordinal of this enumeration constant.

ExceptionNot Available

Example

The following example shows the usage of java.lang.Enum.ordinal() method

```
package com.tutorialspoint;
import java.lang.*;
// enum showing Mobile prices
enum Mobile {
   Samsung(400), Nokia(250),Motorola(325);

int price;
   Mobile(int p) {
      price = p;
   }
intshowPrice() {
      return price;
   }
}

public class EnumDemo {

   public static void main(String args[]) {

System.out.println("CellPhone List:");
      for(Mobile m : Mobile.values()) {
System.out.println(m + " costs " + m.showPrice() + " dollars");
      }

      Mobile ret = Mobile.Samsung;
System.out.println("The ordinal is = " + ret.ordinal());
```

```
System.out.println("MobileName = " + ret.name());
   }
}
```

Result:

```
CellPhone List:
Samsung costs 400 dollars
Nokia costs 250 dollars
Motorola costs 325 dollars
The ordinal is = 0
MobileName = Samsung
```

b)What is type wrapper? What is the role of auto boxing and boxing?

A Wrapper class is a class whose object wraps or contains primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store primitive data types. In other words, we can wrap a primitive value into a wrapper class object.

Need of Wrapper Classes

1.They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).
2.The classes in java.util package handles only objects and hence wrapper classes help in this case also.
3.Data structures in the Collection framework, such as ArrayList and Vector, store only objects (reference types) and not primitive types.
4.An object is needed to support synchronization in multithreading.

Primitive Data types and their Corresponding Wrapper class

Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

Autoboxing: Automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing. For example – conversion of int to Integer, long to Long, double to Double etc.
Example:

```
// Java program to demonstrate Autoboxing
import java.util.ArrayList;
class Autoboxing
{
   public static void main(String[] args)
   {
      char ch = 'a';
      // Autoboxing- primitive to Character object conversion
      Character a = ch;
      ArrayList<Integer>arrayList = new ArrayList<Integer>();
      // Autoboxing because ArrayList stores only objects
      arrayList.add(25);
      // printing the values from object
      System.out.println(arrayList.get(0));
   }
}
```

Output:
25

Unboxing: It is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing. For example – conversion of Integer to int, Long to long, Double to double, etc.

```
// Java program to demonstrate Unboxing
import java.util.ArrayList;

class Unboxing
{
   public static void main(String[] args)
   {
      Character ch = 'a';
      // unboxing - Character object to primitive conversion
      char a = ch;
      ArrayList<Integer>arrayList = new ArrayList<Integer>();
      arrayList.add(24);
      // unboxing because get method returns an Integer object
      intnum = arrayList.get(0);
      // printing the values from primitive data types
      System.out.println(num);
   }
}
```

Output:
24

9. Give an example where interface can be used to support multiple inheritance.

An interface contains variables and methods like a class but the methods in an interface are abstract by default unlike a class. Multiple inheritance by interface occurs if a class implements multiple interfaces or also if an interface itself extends multiple interfaces.

A program that demonstrates multiple inheritance by interface in Java is given as follows:

```
interface AnimalEat {
  void eat();
}
interface AnimalTravel {
  void travel();
}
class Animal implements AnimalEat, AnimalTravel {
  public void eat() {
System.out.println("Animal is eating");
  }
  public void travel() {
System.out.println("Animal is travelling");
  }
}
public class Demo {
  public static void main(String args[]) {
    Animal a = new Animal();
a.eat();
a.travel();
  }
}
```

Output

```
Animal is eating
Animal is travelling
```

10. Distinguish between Byte Stream Classes and Character Stream Classes.

Java provides I/O Streams to read and write data where, a Stream represents an input source or an output destination which could be a file, i/o devise, other program etc.

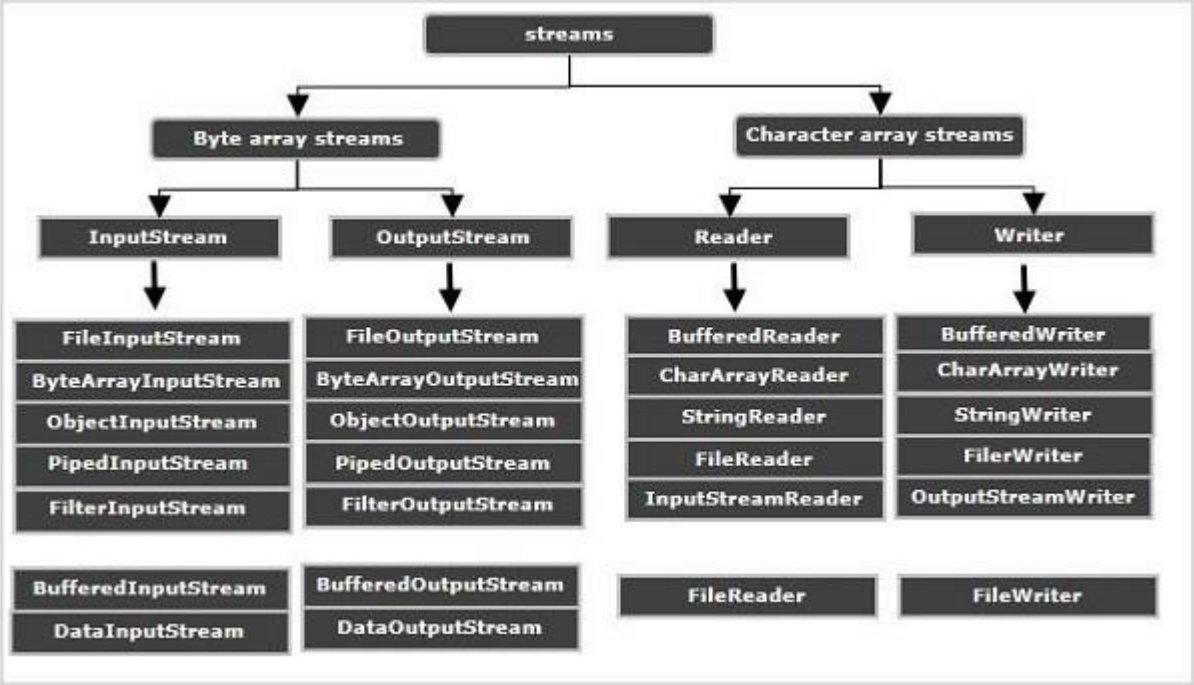
Based on the data they handle there are two types of streams –

Byte Streams – These handle data in bytes (8 bits) i.e., the byte stream classes read/write data of 8 bits. Using these you can store characters, videos, audios, images etc.
Character Streams – These handle data in 16 bit Unicode. Using these you can read and write text data

only.
The Reader and Writer classes (abstract) are the super classes of all the character stream classes: classes that are used to read/write character streams.

Whereas the InputStream and OutputStream classes (abstract) are the super classes of all the input/output stream classes: classes that are used to read/write a stream of bytes.

Following diagram illustrates all the input and output Streams (classes) in Java.



Difference between input/output Streams and Readers/Writers
The major difference between these is that the input/output stream classes read/write byte stream data. Whereas the Reader/Writer classes handle characters.

The methods of input/output stream classes accept byte array as parameter whereas the Reader/Writer classes accept character array as parameter.

The Reader/Writer classes handles all the Unicode characters, comes handy for internalization, comparatively efficient that input/output streams.

Therefore, until you deal with binary data like images it is recommended to use Reader/Writer classes.

Example Input/Output Streams
Following Java program reads data from a particular file using FileInputStream and writes it to another, using FileOutputStream.

```
Import java.io.File;
Import java.io.FileInputStream;
```

```
Import java.io.FileOutputStream;
Import java.io.IOException;
Public class IOStreamsExample {
  Public static void main(String args[]) throws IOException {
    //Creating FileInputStream object
    File file = new File("D:/myFile.txt");
FileInputStreamfis = new FileInputStream(file);
    Byte bytes[] = new byte[(int) file.length()];
    //Reading data from the file
Fis.read(bytes);
    //Writing data to another file
    File out = new File("D:/CopyOfmyFile.txt");
FileOutputStreamoutputStream = new FileOutputStream(out);
    //Writing data to the file
outputStream.write(bytes);
outputStream.flush();
System.out.println("Data successfully written in the specified file");
  }
}
```

Output
Data successfully written in the specified file.

Example Reader/Writer Streams
Following Java program reads data from a particular file using FileReader and writes it to another, using FileWriter.

```
Import java.io.File;
Import java.io.FileReader;
Import java.io.FileWriter;
Import java.io.IOException;
Public class IOStreamsExample {
  Public static void main(String args[]) throws IOException {
    //Creating FileReader object
    File file = new File("D:/myFile.txt");
FileReader reader = new FileReader(file);
    Char chars[] = new char[(int) file.length()];
    //Reading data from the file
Reader.read(chars);
    //Writing data to another file
    File out = new File("D:/CopyOfmyFile.txt");
FileWriter writer = new FileWriter(out);
    //Writing data to the file
Writer.write(chars);
Writer.flush();
System.out.println("Data successfully written in the specified file");
```

```
  }
}
```

OutputData successfully written in the specified file

UNIT –III

1.a) What is an exception? What are keywords used in exception handled in Java programming? Explain with suitable program.
(or)Write a program that includes a try block and a catch clause which processes the arithmetic exception generated by division-by-zero error.

Exception

A Java Exception is an object that describes the exception that occurs in a program. When an exceptional events occurs in java, an exception is said to be thrown. The code that's responsible for doing something about the exception is called an exception handler.

Exception Handling is the mechanism to handle runtime malfunctions. We need to handle such exceptions to prevent abrupt termination of program. The term exception means exceptional condition, it is a problem that may arise during the execution of program. A bunch of things can lead to exceptions, including programmer error, hardware failures, files that need to be opened cannot be found, resource exhaustion etc.
All exception types are subclasses of class Throwable, which is at the top of exception class hierarchy.
Checked Exception The exception that can be predicted by the programmer is called as Checked Exception. The classes that extend Throwable class except Runtime Exception and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time. Unchecked Exception Unchecked exceptions are the class that extends RuntimeException. Unchecked exception is ignored at compile time. Example: ArithmeticException, NullPointerException, ArrayIndexOutOfBound exception. Unchecked exceptions are checked at runtime. Error Errors are typically ignored in code because you can rarely do anything about an error. For example if stack overflow occurs, an error will arise. This type of error is not possible handle in code. Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.
In java, exception handling is done using five keywords,

1. try
2. catch
3. throw
4. throws
5. finally

Exception handling is done by transferring the execution of a program to an appropriate

exception handler when exception occurs.
Using try and catch
Try is used to guard a block of code in which exception may occur. This block of code is called guarded region.
A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in guarded code, the catch block that follows the try is checked, if the type of exception that occurred is listed in the catch block then the exception is handed over to the catch block which then handles it.

Example using Try and catch class

```
ExceptionDemo {
public static void main(String args[]) {
inta,b,c;
 try {
a=0;
b=10;
c=b/a;
System.out.println("This line will not be executed");
}
catch(ArithmeticException e) {
System.out.println("Divided by zero");
}
System.out.println("After exception is handled");
}
}
```

Output:
Divided by zero After exception is handled

2) With a program illustrate user defined exception handling.
(or)
With a program illustrate user defined exception handling. Write a program for user defined exception that check the internal and external marks are greater than 40 it raise the exception "internal marks are exceed" and if external marks greater than 60 exception in raised the exception "external marks Exceed"

User Defined Exception or custom exception is creating your own exception class and throws that exception using 'throw' keyword. This can be done by extending the class Exception.

In java we have already defined, exception classes such as ArithmeticException, NullPointerException

etc. These exceptions are already set to trigger on pre-defined conditions such as when you divide a number by zero it triggers ArithmeticException, In the last tutorial we learnt how to throw these exceptions explicitly based on your conditions using throw keyword.

In java we can create our own exception class and throw that exception using throw keyword. These exceptions are known as user-defined or custom exceptions. In this tutorial we will see how to create your own custom exception and throw it on a particular condition.

EXAMPLE OF USER DEFINED EXCEPTION IN JAVA:

```
Import java.io.IOException;
Import java.util.Scanner;
Public class InternalMarks extends Exception {
Public InternalMarks(String s) {
Super(s);
}
}
Public class ExternalMarks extends Exception {
Public ExternalMarks(String s) {
Super(s);
}
}
Public class Main {
Public static void main(String args[]) throws IOException
{
Intx,y;
Scanner s = new Scanner(System.in);
System.out.println("Enter Internal Marks");
X=s.nextInt();
If(x>40)
{
Try {
Throw new InternalMarks("Internal Marks > 40");
} catch (InternalMarks e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}
Else
{
System.out.println("Internal MARKS = "+ x);
}
System.out.println("Enter External Marks");
Y=s.nextInt();
If(y>60)
{
```

```
Try {
Throw new InternalMarks("External Marks Exceeded");
} catch (InternalMarks e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}
Else
{
System.out.println("External MARKS = "+ y);
}
}
}
```

3 a)Differentiate between Checked and UnChecked Exceptions with examples
Checked exceptions are checked at compile-time. It means if a method is throwing a checked exception then it should handle the exception using try-catch block or it should declare the exception using throws keyword, otherwise the program will give a compilation error.

Lets understand this with the help of an **example**:

Checked Exception Example

In this example we are reading the file myfile.txt and displaying its content on the screen. In this program there are three places where a checked exception is thrown as mentioned in the comments below. FileInputStream which is used for specifying the file path and name, throws FileNotFoundException. The read() method which reads the file content throws IOException and the close() method which closes the file input stream also throws IOException.

```
import java.io.*;
class Example {
  public static void main(String args[])
  {
        FileInputStreamfis = null;
        /*This constructor FileInputStream(File filename)
         * throws FileNotFoundException which is a checked
         * exception
       */
fis = new FileInputStream("B:/myfile.txt");
        int k;

        /* Method read() of FileInputStream class also throws
         * a checked exception: IOException
```

```
        */
          while(( k = fis.read() ) != -1)
          {
                  System.out.print((char)k);
          }

          /*The method close() closes the file input stream
           * It throws IOException*/
          fis.close();
    }
}
```

Exception in thread "main" java.lang.Error: Unresolved compilation problems:
Unhandled exception type FileNotFoundException
Unhandled exception type IOException
Unhandled exception type IOException

Unchecked exceptions are not checked at compile time. It means if your program is throwing an unchecked exception and even if you didn't handle/declare that exception, the program won't give a compilation error. Most of the times these exception occurs due to the bad data provided by user during the user-program interaction. It is up to the programmer to judge the conditions in advance, that can cause such exceptions and handle them appropriately. All Unchecked exceptions are direct sub classes of **RuntimeException** class.

Lets understand this with an example:

```
class Example {
  public static void main(String args[])
  {
        int num1=10;
        int num2=0;
        /*Since I'm dividing an integer with 0
         * it should throw ArithmeticException
        */
        int res=num1/num2;
        System.out.println(res);
  }
}
```

If you compile this code, it would compile successfully however when you will run it, it would throw ArithmeticException. That clearly shows that unchecked exceptions are not checked at compile-time, they occurs at runtime.

b)What is meant by re-throwing exception? Discuss a suitable scenario for this.

Sometimes we may need to rethrow an exception in Java. If a catch block cannot handle the particular exception it has caught, we can rethrow the exception. The rethrow expression causes the originally

thrown object to be rethrown.

Because the exception has already been caught at the scope in which the rethrow expression occurs, it is rethrown out to the next enclosing try block. Therefore, it cannot be handled by catch blocks at the scope in which the rethrow expression occurred. Any catch blocks for the enclosing try block have an opportunity to catch the exception.

Syntax

```
catch(Exception e) {
System.out.println("An exception was thrown");
  throw e;
}
```

Example

Live Demo

```
public class RethrowException {
  public static void test1() throws Exception {
System.out.println("The Exception in test1() method");
    throw new Exception("thrown from test1() method");
  }
  public static void test2() throws Throwable {
    try {
      test1();
    } catch(Exception e) {
System.out.println("Inside test2() method");
      throw e;
    }
  }
  public static void main(String[] args) throws Throwable {
    try {
      test2();
    } catch(Exception e) {
System.out.println("Caught in main");
    }
  }
}
```

Output

```
The Exception in test1() method
Inside test2() method
Caught in main
```

4 a)What are the different ways that are possible to create multiple threaded programs in java? Discuss the differences between them.

Multithreading in Java is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.

Threads can be created by using two mechanisms :

1. Extending the Thread class
2. Implementing the Runnable Interface

Threadcreation by extending the Thread class
We create a class that extends the **java.lang.Thread** class. This class overrides the run() method available in the Thread class. A thread begins its life inside run() method. We create an object of our new class and call start() method to start the execution of a thread. Start() invokes the run() method on the Thread object.

```
// Java code for thread creation by extending
// the Thread class
class MultithreadingDemo extends Thread {
   public void run()

   {

      try {

         // Displaying the thread that is running

System.out.println(

               "Thread " + Thread.currentThread().getId()

               + " is running");

      }

      catch (Exception e) {

         // Throwing an exception
```

```
System.out.println("Exception is caught");

        }

    }
}

// Main Class

public class Multithread {

    public static void main(String[] args)
    {
int n = 8; // Number of threads
        for (inti = 0; i< n; i++) {
MultithreadingDemoobject = new MultithreadingDemo();
object.start();
        }
    }
}
```

Output

```
Thread 15 is running
Thread 14 is running
Thread 16 is running
Thread 12 is running
Thread 11 is running
Thread 13 is running
Thread 18 is running
Thread 17 is running
```

Thread creation by implementing the Runnable Interface

We create a new class which implements java.lang.Runnable interface and override run() method. Then we instantiate a Thread object and call start() method on this object.

```
 // Java code for thread creation by implementing
// the Runnable Interface
class MultithreadingDemo implements Runnable
 {
    public void run()
    {
try {
            // Displaying the thread that is running
System.out.println("Thread " + Thread.currentThread().getId()+ " is running");
        }

        catch (Exception e) {
```

```
            // Throwing an exception
System.out.println("Exception is caught");
        }
    }
}
// Main Class
class Multithread {
    public static void main(String[] args)
    {
int n = 8; // Number of threads
        for (inti = 0; i< n; i++) {
            Thread object= new Thread(new MultithreadingDemo());
object.start();
        }
    }
}
```

Output

```
Thread 13 is running
Thread 11 is running
Thread 12 is running
Thread 15 is running
Thread 14 is running
Thread 18 is running
Thread 17 is running
Thread 16 is running
```

b)Does Java support thread priorities? demonstrate the priority setting in threads with suitable example.

Each thread have a priority. Priorities are represented by a number between 1 and 10.

In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

3 constants defined in Thread class:

1. public static int MIN_PRIORITY

2. public static int NORM_PRIORITY
3. public static int MAX_PRIORITY

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

Example of priority of a Thread:

```
Class TestMultiPriority1 extends Thread{
 Public void run(){
System.out.println("running thread name is:"+Thread.currentThread().getName());
System.out.println("running thread priority is:"+Thread.currentThread().getPriority());
  }
 Public static void main(String args[]){
  TestMultiPriority1 m1=new TestMultiPriority1();
  TestMultiPriority1 m2=new TestMultiPriority1();
  M1.setPriority(Thread.MIN_PRIORITY);
  M2.setPriority(Thread.MAX_PRIORITY);
  M1.start();
  M2.start();
 }
}
```

Output:

```
running thread name is:Thread-0
      running thread priority is:10
      running thread name is:Thread-1
      running thread priority is:1
```

5) Write a program that creates two threads. Fist thread prints the numbers from 1 to 100 and the other thread prints the numbers from 100 to 1.

```
Class Ascending extends Thread

{

Public void run()
```

```
{
For(inti=1; i<=100;i++)
{
System.out.println(“Ascending Thread : “ + i);
}
}
}

Class Descending extends Thread
{
Public void run()
{
For(inti=100; i>0;i--)
{
System.out.println(“Descending Thread : “ + i);
}
}
}

Public class AscendingDescendingThread
{
Public static void main(String[] args)
```

```
{
New Ascending().start();
New Descending().start();
}
}
```

OUTPUT:

```
C:\javac AscendingDescendingThread.java
C:\java AscendingDescendingThread
Ascending Thread: 1
Ascending Thread: 2
Ascending Thread: 3
Ascending Thread : 4
Ascending Thread : 5
Ascending Thread : 6
Ascending Thread : 7
Ascending Thread : 8
Ascending Thread : 9
Ascending Thread : 10
Ascending Thread : 11
Ascending Thread : 12
Ascending Thread : 13
Ascending Thread : 14
```

Ascending Thread : 15

Ascending Thread : .

Ascending Thread : .

Ascending Thread : .

Ascending Thread : 100

Ascending Thread : 100

Ascending Thread : .

Ascending Thread : .

Descending Thread : 15

Descending Thread : 14

Descending Thread : 13

Descending Thread : 12

Descending Thread : 11

Descending Thread : 10

Descending Thread : 9

Descending Thread : 8

Descending Thread : 7

Descending Thread : 6

Descending Thread : 5

Descending Thread : 4

Descending Thread : 3

Descending Thread : 2

Descending Thread : 1

6) Describe the need of thread synchronization. How is it achieved in Java programming? Explain with a suitable program
(or)**Describe producer-consumer pattern using inter-thread communication.

In computing, the producer-consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, which share a common, fixed-size buffer used as a queue.

- The producer's job is to generate data, put it into the buffer, and start again.

- At the same time, the consumer is consuming the data (i.e. removing it from the buffer), one piece at a time.

Problem
To make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.

Solution
The producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer.
An inadequate solution could result in a deadlock where both processes are waiting to be awakened.

Implementation of Producer Consumer Class

- A **LinkedList list** – to store list of jobs in queue.
- **A Variable Capacity** – to check for if the list is full or not
- A mechanism to control the insertion and extraction from this list so that we do not insert into list if it is full or remove from it if it is empty.
- }

```
import java.util.LinkedList;
public class Threadexample {
   public static void main(String[] args)  throws InterruptedException
   {
      // Object of a class that has both produce()
      // and consume() methods
      final PC pc = new PC();


      // Create producer thread
      Thread t1 = new Thread(new Runnable() {
         @Override
         public void run()
         {
            try {
pc.produce();
            }
            catch (InterruptedException e) {
e.printStackTrace();
            }
         }
      });
      // Create consumer thread
     Thread t2 = new Thread(new Runnable() {
         @Override
```

```
            public void run()
            {
        try {
pc.consume();
                }
    catch (InterruptedException e) {
e.printStackTrace();
                }
            }
        });

        // Start both threads
       t1.start();
        t2.start();

        // t1 finishes before t2
        t1.join();
        t2.join();
    }
    // This class has a list, producer (adds items to list
    // and consumber (removes items).

    public static class PC {

        // Create a list shared by producer and consumer
        // Size of list is 2.
LinkedList<Integer> list = new LinkedList<>();
int capacity = 2;
        // Function called by producer thread
        public void produce() throws InterruptedException
    {
int value = 0;
            while (true) {
                synchronized (this)
                {
                  // producer thread waits while list
                    // is full
                    while (list.size() == capacity)
                        wait();
System.out.println("Producer produced-" + value);
                    // to insert the jobs in the list
list.add(value++);
                    // notifies the consumer thread that
                    // now it can start consuming
```

```
                notify();
                // makes the working of program easier
                // to  understand
Thread.sleep(1000);
            }
        }
    }

    // Function called by consumer thread
    public void consume() throws InterruptedException
    {
        while (true) {
            synchronized (this)
            {
                // consumer thread waits while list
                // is empty
                while (list.size() == 0)
                    wait();
                // to retrive the ifrst job in the list

intval = list.removeFirst();
System.out.println("Consumer consumed-"  + val);
                // Wake up producer thread
                notify();
               // and sleep
Thread.sleep(1000);
            }
        }
    }

  }
}
```

UNIT-IV

1 .a) Give an account of Random collection class

Ans:

Methods	Description
doubles()	Returns an unlimited stream of pseudorandom double values.
ints()	Returns an unlimited stream of pseudorandom int values.
longs()	Returns an unlimited stream of pseudorandom long values.
next()	Generates the next pseudorandom number.
nextBoolean()	Returns the next uniformly distributed pseudorandom boolean value from the random number generator's sequence
nextByte()	Generates random bytes and puts them into a specified byte array.
nextDouble()	Returns the next pseudorandom Double value between 0.0 and 1.0 from the random number generator's sequence
nextFloat()	Returns the next uniformly distributed pseudorandom Float value between 0.0 and 1.0 from this random number generator's sequence
nextGaussian()	Returns the next pseudorandom Gaussian double value with mean 0.0 and standard deviation 1.0 from this random number generator's sequence.
nextInt()	Returns a uniformly distributed pseudorandom int value

	generated from this random number generator's sequence
nextLong()	Returns the next uniformly distributed pseudorandom long value from the random number generator's sequence.
setSeed()	Sets the seed of this random number generator using a single long seed.

Java Random class is used to generate a stream of pseudorandom numbers. The algorithms implemented by Random class use a protected utility method than can supply up to 32 pseudorandomly generated bits on each invocation.

1.b) Discuss the methods of Stack class

Ans. The Stack class in java has the following methods.

S.No.	Methods with Description
1	**Object push(Object element)** It pushes the element onto the stack and returns the same.
2	**Object pop( )** It returns the element on the top of the stack and removes the same.
3	**int search(Object element)** If element found, it returns offset from the top. Otherwise, -1 is returned.
4	**Object peek( )** It returns the element on the top of the stack.

S.No.	Methods with Description
5	**boolean empty()** It returns true if the stack is empty, otherwise returns false.

1. c) What is the need of Generics?

Ans. There are mainly 3 advantages of generics. They are as follows:

1) Type-safety: We can hold only a single type of objects in generics. It doesn?t allow to store other objects.

Without Generics, we can store any type of objects.

```
1. List list = new ArrayList();
2. list.add(10);
3. list.add("10");
4. With Generics, it is required to specify the type of object we need to store.
5. List<Integer> list = new ArrayList<Integer>();
6. list.add(10);
7. list.add("10");// compile-time error
```

2) Type casting is not required: There is no need to typecast the object.

Before Generics, we need to type cast.

```
1. List list = new ArrayList();
2. list.add("hello");
3. String s = (String) list.get(0);//typecasting
4. After Generics, we don't need to typecast the object.
5. List<String> list = new ArrayList<String>();
6. list.add("hello");
```

7. String s = list.get(0);

3) Compile-Time Checking: It is checked at compile time so problem will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.

```
1. List<String> list = new ArrayList<String>();
2. list.add("hello");
3. list.add(32);//Compile Time Error
```

Syntax to use generic collection

```
1. ClassOrInterface<Type>
```

Example to use Generics in java

```
1. ArrayList<String>
```

2.a) Differentiate between ArrayList and a Vector? Why ArrayList is faster than Vector? Explain.

Ans.

ArrayList and Vector both implements List interface and maintains insertion order.

However, there are many differences between ArrayList and Vector classes that are given below.

ArrayList	**Vector**
1) ArrayList is **not synchronized**.	Vector is **synchronized**.

2) ArrayList **increments 50%** of current array size if the number of elements exceeds from its capacity.	Vector **increments 100%** means doubles the array size if the total number of elements exceeds than its capacity.
3) ArrayList is **not a legacy** class. It is introduced in JDK 1.2.	Vector is a **legacy** class.
4) ArrayList is **fast** because it is non-synchronized.	Vector is **slow** because it is synchronized, i.e., in a multithreading environment, it holds the other threads in runnable or non-runnable state until current thread releases the lock of the object.
5) ArrayList uses the **Iterator** interface to traverse the elements.	A Vector can use the **Iterator** interface or **Enumeration** interface to traverse the elements.

2.b)Write a program which stores a list of strings in an Array List and then displays the contents of the list

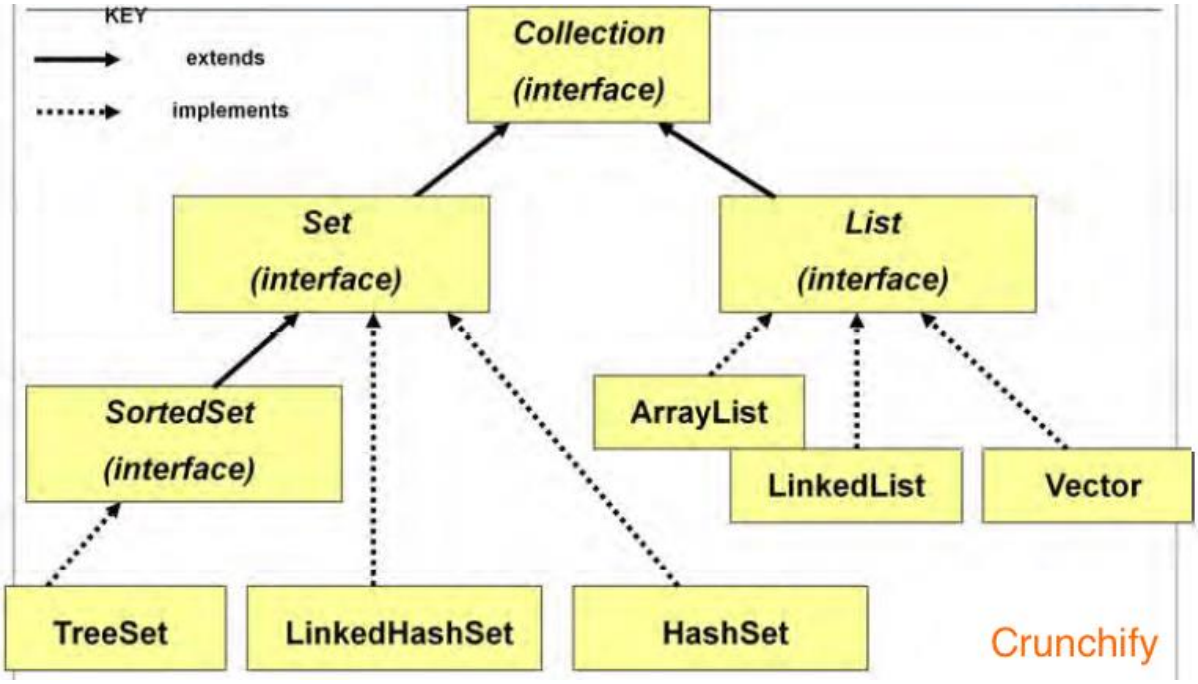
Ans.

```
importjava.util.*;
classJavaExample{
publicstaticvoid main(Stringargs[]){
ArrayList<String>alist=newArrayList<String>();
alist.add("Steve");
alist.add("Tim");
alist.add("Lucy");
alist.add("Pat");
alist.add("Angela");
alist.add("Tom");

//displaying elements
System.out.println(alist);
```

```
//Adding "Steve" at the fourth position
alist.add(3, "Steve");

//displaying elements
System.out.println(alist);
   }
}
```

3.a) What is Java Collections Framework? List out some benefits of Collections framework and explain.

Ans.



Java Collections are used in every programming language and initial java release contained few classes for collections: Vector, Stack, Hashtable, Array. But looking at the larger scope and usage, Java 1.2 came up with Collections Framework that group all the collections interfaces, implementations and algorithms.

Java Collections have come through a long way with usage of Generics and Concurrent Collection classes for thread-safe operations. It also includes blocking interfaces and their implementations in java concurrent package.

Some of the benefits of collections framework are:

- Reduced development effort by using core collection classes rather than implementing our own collection classes.
- Code quality is enhanced with the use of well tested collections framework classes.
- Reduced effort for code maintenance by using collection classes shipped with JDK.
- Reusability and Interoperability

3.b) What is the importance of hashCode() and equals() methods?

Ans. The equals() and hashcode() are the two important methods provided by the **Object** class for comparing objects. Since the Object class is the parent class for all Java objects, hence all objects inherit the default implementation of these two methods. In this topic, we will see the detailed description of equals() and hashcode() methods, how they are related to each other, and how we can implement these two methods in Java.

Java equals()

- The java equals() is a method of *lang.Object* class, and it is used to compare two objects.
- To compare two objects that whether they are the same, it compares the values of both the object's attributes.
- By default, two objects will be the same only if stored in the same memory location.

Syntax:

1. **public boolean** equals(Object obj)

Parameter:

obj: It takes the reference object as the parameter, with which we need to make the comparison.

Returns:

It returns the true if both the objects are the same, else returns false.

Java hashcode()

- A **hashcode** is an integer value associated with every object in Java, facilitating the hashing in hash tables.
- To get this hashcode value for an object, we can use the hashcode() method in Java. It is the means ***hashcode() method that returns the integer hashcode value of the given object***.
- Since this method is defined in the Object class, hence it is inherited by user-defined classes also.
- The hashcode() method returns the same hash value when called on two objects, which are equal according to the equals() method. And if the objects are unequal, it usually returns different hash values.

Syntax:

1. **public int** hashCode()

Returns:

It returns the hash code value for the given objects.

4.What is difference between

a) ArrayList and LinkedList

Ans.

4.b Ans.

ArrayList	LinkedList
1) ArrayList internally uses a **dynamic array** to store the elements.	LinkedList internally uses a **doubly linked list** to store the elements.
2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the bits are shifted in memory.	Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.
3) An ArrayList class can **act as a list** only because it implements List only.	LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces.
4) ArrayList is **better for storing and accessing** data.	LinkedList is **better for manipulating** data.

The **HashMap** and **HashSet** in Java are the most popular Collection classes. Both are used for the data structure. The following table describes the difference between HashMap and HashSet:

Basis	HashMap	HashSet

Definition	Java HashMap is a hash table based implementation of Map interface.	HashSet is a Set. It creates a collection that uses a hash table for storage.
Implementation	HashMap implements **Map, Cloneable, and Serializable** interface es.	HashSet implements **Set, Cloneable, Serializable, Iterable** and **Collection** interf aces.
Stores	In HashMap we store a **key-value pair**. It maintains the mapping of key and value.	In HashSet, we store **objects**.
Duplicate values	It does not allow **duplic ate keys**, but **duplicat e values** are **al**	It does not allow **duplicate values**.

	lowed.	
Null values	It can contain a **single null key** and **mul tiple null values**.	It can contain **a single null value**.
Method of insertion	HashMap uses the **put()** me thod to add the elements in the HashMap.	HashSet uses the **add()** method to add elements in the HashSet.
Performance	HashMap is **faster/ tha n HashSet because values are associated with a unique key.**	HashSet is **slower** than HashMap because the member object is used for calculating hashcode value, which can be same for two objects.
The Number of objects	Only **one** obj ect is created	There are **two** objects created during put operation, one

	during the add operation.	for **key** and one for **value**.
Storing Mechanism	HashMap internally uses **hashing** to store objects.	HashSet internally uses a **HashMap** object to store objects.
Uses	Always prefer when we do not maintain the **uniqueness**.	It is used when we need to maintain the **uniqueness** of data.
Example	**{a->4, b->9, c->5}** Where **a, b, c** are **keys** and **4, 9, 5** are **values** associated with key.	**{6, 43, 2, 90, 4}** It denotes a set.

4.c

S.No	List	Set

1.	The list implementation allows us to add the same or duplicate elements.	The set implementation doesn't allow us to add the same or duplicate elements.
2.	The insertion order is maintained by the List.	It doesn't maintain the insertion order of elements.
3.	List allows us to add any number of null values.	Set allows us to add at least one null value in it.
4.	The List implementation classes are LinkedList and ArrayList.	The Set implementation classes are TreeSet, HashSet and LinkedHashSet.
5.	We can get the element of a specified index from the list using the get() method.	We cannot find the element from the Set based on the index because it doesn't provide any get method().
6.	It is used when we want to frequently access the elements by using the index.	It is used when we want to design a collection of distinct elements.
7.	The method of List interface listiterator() is used to iterate the List elements.	The iterator is used when we need to iterate the Set elements.

4.d) Iterator and ListIterator

Sr. No.	Key	Iterator	ListIterator

Sr. No.	Key	Iterator	ListIterator
1	Applicable	Iterator can be used to traverse any collection irrespective of the type of collection.	List iterator can only be used to iterate only List collection implemented classes like arraylist,linkedlist etc.
2	Calling	As mentioned Iterator must be used to enumerate elements in all Collections implemented interfaces like Set, List, Queue, Deque and also in all implemented classes of Map interface. Iterator object can be created by calling iterator() method of Collection interface.	On the other hand, ListIterator must be used when we want to enumerate elements of List.The object of ListIterator can be created by calling listIterator() method present in List interface.
3	Data traverse	Data traverse in case of the iterator is possible only in one direction as	List iterator could traverses both in

Sr. No.	Key	Iterator	ListIterator
		Iterator can traverse in the forward direction only	forward and backward directions which makes data traverse in both directions.
4	Deletion	The deletion of an element is not allowed in the case of the iterator.	ListIterator can replace an element in list collection.
5	Addition	The addition of an element is not allowed in case of an iterator as it throws ConcurrentModificationException.	ListIterator can add an element in list collection any time easily.
6	Modification	Modification of an element is not allowed in case of an iterator as it throws ConcurrentModificationException.	ListIterator can modify an element in list collection any time easily by calling set() method.

Sr. No.	Key	Iterator	ListIterator
7	Index of element	One can't get the index of the traversed element in collection while using an iterator.	ListIterator has methods like nextIndex() and previousIndex() to obtain indexes of elements at any time while traversing List.

4.e) Comparable and Comparator interface

Comparable	Comparator
1) Comparable provides a **single sorting sequence**. In other words, we can sort the collection on the basis of a single element such as id, name, and price.	The Comparator provides **multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc.
2) Comparable **affects the original class**, i.e., the actual class is modified.	Comparator **doesn't affect the original class**, i.e., the actual class is not modified.

3) Comparable provides **compareTo() method** to sort elements.	Comparator provides **compare() method** to sort elements.
4) Comparable is present in **java.lang** package.	A Comparator is present in the **java.util** package.
5) We can sort the list elements of Comparable type by **Collections.sort(List)** method.	We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method.

5) What is the purpose of BitSet class and Calendar classes? What is the functionality of the following functions of BitSet class: cardinality( ) , flip( ) and intersects( )

Ans.

The BitSet class creates a special type of array that holds bit values. The BitSet array can increase in size as needed. This makes it similar to a vector of bits. This is a legacy class but it has been completely re-engineered in Java 2, version 1.4.

The BitSet defines the following two constructors.

int cardinality( )

Returns the number of set bits in the invoking object.

void flip(int index)

Reverses the bit specified by the index.

void flip(int startIndex, int endIndex)

Reverses the bits from startIndex to endIndex.

boolean intersects(BitSet bitSet)

Returns true if at least one pair of corresponding bits within the invoking object and bitSet are 1.

Calendar class in Java is an abstract class that provides methods for converting date between a specific instant in time and a set of calendar fields such as MONTH, YEAR, HOUR, etc. It inherits Object class and implements the Comparable, Serializable, Cloneable interfaces.

As it is an Abstract class, so we cannot use a constructor to create an instance. Instead, we will have to use the static method Calendar.getInstance() to instantiate and implement a sub-class.

- Calendar.getInstance(): return a Calendar instance based on the current time in the default time zone with the default locale.
- Calendar.getInstance(TimeZone zone)
- Calendar.getInstance(Locale aLocale)
- Calendar.getInstance(TimeZone zone, Locale aLocale)

6. How an Hashtable can change the iterator? Explain.

Ans.

First of all, we **cannot** iterate a Map directly using iterators, because Map are not Collection. Also before going further, you must know a little-bit about Map.Entry<K, V> interface.

Since all maps in Java implement Map interface, following techniques will work for any map implementation (HashMap, TreeMap, LinkedHashMap, Hashtable, etc.)

Hashtable will change the iterator by using Enumerator interface.Yes,we can iterate a hashmap using the entrySet() method.

1. Iterating over Map.entrySet() using For-Each loop :

Map.entrySet() method returns a collection-view(*Set<Map.Entry<K, V>>*) of the mappings contained in this map. So we can iterate over key-value pair using *getKey()* and *getValue()* methods of Map.Entry<K, V>. This method is most common and should be used if you need both map keys and values in the loop. Below is the java program to demonstrate it.

// Java program to demonstrate iteration over

```
// Map.entrySet() entries using for-each loop

import java.util.Map;
import java.util.HashMap;
class IterationDemo
{
	public static void main(String[] arg)
	{
		Map<String,String> gfg = new HashMap<String,String>();

		// enter name/url pair
		gfg.put("GFG", "geeksforgeeks.org");
		gfg.put("Practice", "practice.geeksforgeeks.org");
		gfg.put("Code", "code.geeksforgeeks.org");
		gfg.put("Quiz", "quiz.geeksforgeeks.org");

		// using for-each loop for iteration over Map.entrySet()
		for (Map.Entry<String,String> entry : gfg.entrySet())
			System.out.println("Key = " + entry.getKey() +
							", Value = " + entry.getValue());
	}
}
```

7. Explain different ways to iterate over a list.

Ans.

There are 7 ways you can iterate through List

1. Simple For **loop**.
2. Enhanced For **loop**.
3. **Iterator**.
4. ListIterator.

5. While **loop**.
6. Iterable.**forEach**() util.
7. Stream.**forEach**() util.

```
package crunchify.com.tutorials;
 import java.util.*;
public class CrunchifyIterateThroughList {
    public static void main(String[] argv) {
       // create list
       List<String> crunchifyList = new ArrayList<String>();
       // add 4 different values to list
       crunchifyList.add("Facebook");
       crunchifyList.add("Paypal");
       crunchifyList.add("Google");
       crunchifyList.add("Yahoo");

       // Other way to define list is - we will not use this list :)
       List<String> crunchifyListNew = Arrays.asList("Facebook", "Paypal", "Google", "Yahoo");

       // Simple For loop
       System.out.println("==============> 1. Simple For loop Example.");
       for (int i = 0; i < crunchifyList.size(); i++) {
          System.out.println(crunchifyList.get(i));
       }

       // New Enhanced For loop
       System.out.println("\n==============> 2. New Enhanced For loop Example..");
       for (String temp : crunchifyList) {
          System.out.println(temp);
       }
```

```
// Iterator - Returns an iterator over the elements in this list in proper sequence.
System.out.println("\n==============> 3. Iterator Example...");
Iterator<String> crunchifyIterator = crunchifyList.iterator();
while (crunchifyIterator.hasNext()) {
   System.out.println(crunchifyIterator.next());
}

// ListIterator - traverse a list of elements in either forward or backward order
// An iterator for lists that allows the programmer to traverse the list in either direction, modify the list during iteration,
// and obtain the iterator's current position in the list.
System.out.println("\n==============> 4. ListIterator Example...");
ListIterator<String> crunchifyListIterator = crunchifyList.listIterator();
while (crunchifyListIterator.hasNext()) {
   System.out.println(crunchifyListIterator.next());
}

// while loop
System.out.println("\n==============> 5. While Loop Example....");
int i = 0;
while (i < crunchifyList.size()) {
   System.out.println(crunchifyList.get(i));
   i++;
}

// Iterable.forEach() util: Returns a sequential Stream with this collection as its source
System.out.println("\n==============> 6. Iterable.forEach() Example....");
crunchifyList.forEach((temp) -> {
   System.out.println(temp);
```

```
        });

        // collection Stream.forEach() util: Returns a sequential Stream with this collection as its source
        System.out.println("\n==============> 7. Stream.forEach() Example....");
        crunchifyList.stream().forEach((crunchifyTemp) -> System.out.println(crunchifyTemp));
    }
}
```

Output:

```
==============> 1. Simple For loop Example.
Facebook
Paypal
Google
Yahoo

==============> 2. New Enhanced For loop Example..
Facebook
Paypal
Google
Yahoo

==============> 3. Iterator Example...
Facebook
Paypal
Google
Yahoo

==============> 4. ListIterator Example...
```

Facebook
Paypal
Google
Yahoo

==========> 5. While Loop Example....
Facebook
Paypal
Google
Yahoo

==========> 6. Iterable.forEach() Example....
Facebook
Paypal
Google
Yahoo
==========> 7. Stream.forEach() Example....
Facebook
Paypal
Google
Yahoo
Process finished with exit code 0

UNIT-V

1a) What is the significance of layout managers? Discuss briefly various types of layout managers.

Ans.

The Layout managers enable us to control the way in which visual components are arranged in the GUI forms by determining the size and position of components within the containers.

Types of LayoutManager

There are 6 layout managers in Java

- **FlowLayout**: It arranges the components in a container like the words on a page. It fills the top line from **left to right and top to bottom**. The components are arranged in the order as they are added i.e. first components appears at top left, if the container is not wide enough to display all the components, it is wrapped around the line. Vertical and horizontal gap between components can be controlled. The components can be **left, center or right aligned.**
- **BorderLayout**: It arranges all the components along the edges or the middle of the container i.e. **top, bottom, right and left** edges of the area. The components added to the top or bottom gets its preferred height, but its width will be the width of the container and also the components added to the left or right gets its preferred width, but its height will be the remaining height of the container. The components added to the center gets neither its preferred height or width. It covers the remaining area of the container.
- **GridLayout**: It arranges all the components in a grid of **equally sized cells**, adding them from the **left to right** and **top to bottom**. Only one component can be placed in a cell and each region of the grid will have the same size. When the container is resized, all cells are automatically resized. The order of placing the components in a cell is determined as they were added.
- **GridBagLayout**: It is a powerful layout which arranges all the components in a grid of cells and maintains the aspect ration of the object whenever the container is resized. In this layout, cells may be different in size. It assigns a consistent horizontal and vertical gap among components. It allows us to specify a default alignment for components within the columns or rows.
- **BoxLayout**: It arranges multiple components in either **vertically or horizontally**, but not both. The components are arranged from **left to right or top to bottom**. If the components are aligned **horizontally**, the height of all components will be the same and equal to the largest sized components. If the components are aligned **vertically**, the width of all components will be the same and equal to the largest width components.
- **CardLayout**: It arranges two or more components having the same size. The components are **arranged in a deck**, where all the cards of the same size and the **only top card are visible at any time**. The first component added in the container will be kept at the top of the deck. The default gap at the left, right, top and bottom edges are zero and the card components are displayed either **horizontally or vertically.**

1. b) Give an overview of JButton class and subclasses of JButton class of swing package.

Ans.

The class **JButton** is an implementation of a push button. This component has a label and generates an event when pressed. It can also have an Image.

Class Declaration

Following is the declaration for **javax.swing.JButton** class −

```
public class JButton
  extends AbstractButton
    implements Accessible
```

Class Constructors

Sr.No.	Constructor & Description
1	**JButton()** Creates a button with no set text or icon.
2	**JButton(Action a)** Creates a button where properties are taken from the Action supplied.
3	**JButton(Icon icon)** Creates a button with an icon.
4	**JButton(String text)** Creates a button with the text.
5	**JButton(String text, Icon icon)** Creates a button with an initial text and an icon.

Class Methods

Sr.No.	Method & Description
1	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this JButton.

2	**String getUIClassID()** Returns a string that specifies the name of the L&F class which renders this component.
3	**boolean isDefaultButton()** Gets the value of the defaultButton property, which if true means that this button is the current default button for its JRootPane.
4	**boolean isDefaultCapable()** Gets the value of the defaultCapable property.
5	**protected String paramString()** Returns a string representation of this JButton.
6	**void removeNotify()** Overrides JComponent.removeNotify to check if this button is currently set as the default button on the RootPane. And if so, sets the RootPane's default button to null to ensure the RootPane doesn't hold onto an invalid button reference.
7	**void setDefaultCapable(boolean defaultCapable)** Sets the defaultCapable property, which determines whether this button can be made the default button for its root pane.
8	**void updateUI()** Resets the UI property to a value from the current look and feel.

Swing defines four types of buttons: **JButton**, JToggleButton, JCheckBox, and JRadioButton. All are **subclasses** of the AbstractButton **class**, which extends JComponent. Thus, all buttons share a set of common traits. Here, di, pi, si, and ri are the icons to be used for the indicated purpose

JToggleButton

JToggleButton is used to create toggle button, it is two-states button to switch on or off

.

Java JCheckBox

The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".It inherits JToggleButton class.

Java JRadioButton

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

2a)Explain delegation event model.

Ans.

Delegation Event Model in Java

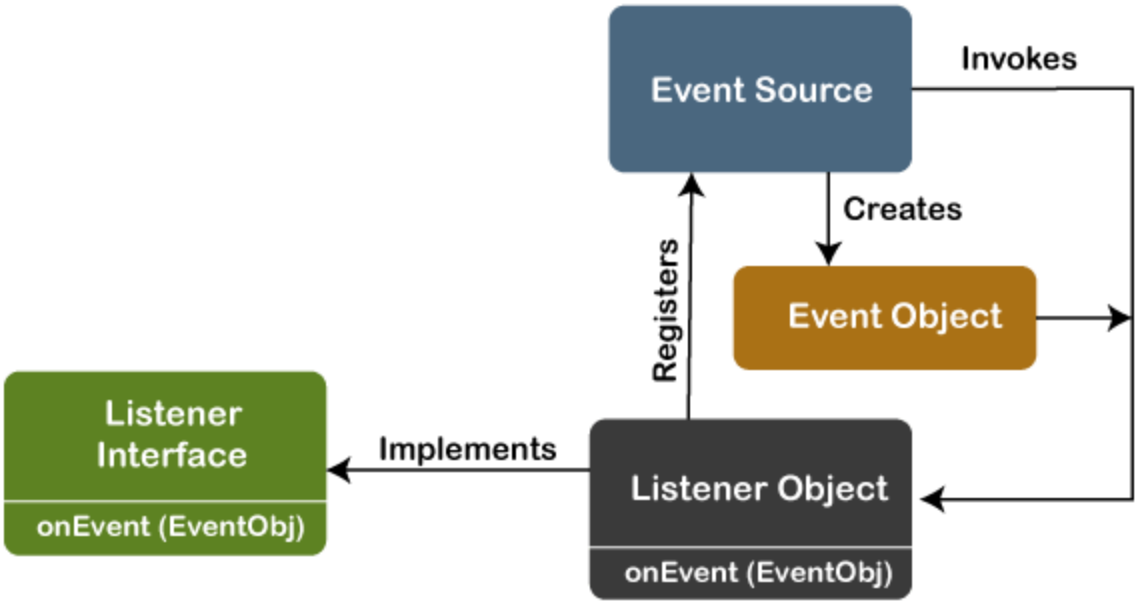
The Delegation Event model is defined to handle events in GUI programming languages. The GUI stands for Graphical User Interface, where a user graphically/visually interacts with the system.

The GUI programming is inherently event-driven; whenever a user initiates an activity such as a mouse activity, clicks, scrolling, etc., each is known as an event that is mapped to a code to respond to functionality to the user. This is known as event handling.

In this section, we will discuss event processing and how to implement the delegation event model in Java. We will also discuss the different components of an Event Model.

Event Processing in Java

Java support event processing since Java 1.0. It provides support for AWT (Abstract Window Toolkit), which is an API used to develop the Desktop application. In Java 1.0, the AWT was based on inheritance. To catch and process GUI events for a program, it should hold subclass GUI components and override action() or handleEvent() methods. The below image demonstrates the event processing.



But, the modern approach for event processing is based on the Delegation Model. It defines a standard and compatible mechanism to generate and process events. In this model, a source generates an event and forwards it to one or more listeners. The listener waits until it receives an event. Once it receives the event, it is processed by the listener and returns it. The UI elements are able to delegate the processing of an event to a separate function.

The key advantage of the Delegation Event Model is that the application logic is completely separated from the interface logic.

In this model, the listener must be connected with a source to receive the event notifications. Thus, the events will only be received by the listeners who wish to receive them. So, this approach is more convenient than the inheritance-based event model (in Java 1.0).

In the older model, an event was propagated up the containment until a component was handled. This needed components to receive events that were not processed, and it took lots of time. The Delegation Event model overcame this issue.

Basically, an Event Model is based on the following three components:

- Events
- Events Sources
- Events Listeners

2.b) Write an Applet to draw a smiley picture accept user name as a parameter and display welcome message.

```
1.  import java.applet.Applet;
```

```
2.      import java.awt.*;
3.
4.      public class SmileyExc extends Applet {
5.
6.       public void paint(Graphics g) {
7.
8.        g.setColor(Color.yellow);
9.        g.fillOval(20,20,150,150);   // For face
10.       g.setColor(Color.black);
11.       g.fillOval(50,60,15,25);     // Left Eye
12.       g.fillOval(120,60,15,25);    // Right Eye
13.       int x[] = {95,85,106,95};
14.       int y[] = {85,104,104,85};
15.       g.drawPolygon(x, y, 4);      // Nose
16.       g.drawArc(55,95,78,50,0,-180);  // Smile
17.       g.drawLine(50,126,60,116);   // Smile arc1
18.       g.drawLine(128,115,139,126);  // Smile arc2
19.      }
20.     }
21.
22.     /* <applet code="SmileyExc.class" width="200" height="200">
23.        </applet>
24.     */
```

3a) Describe about various components in AWT and SWINGS

Ans.

Java AWT (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

components in AWT

Container

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc.

The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Window

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panel

The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Swing is the collection of user interface components for Java programs. It is part of Java Foundation classes that are referred to as JFC. In simple words, Swing is the graphical user interface toolkit that is used for developing windows based java applications or programs. Swing is the successor of AWT which is known as Abstract window toolkit API for Java and AWT components are mainly heavyweight.

The components are lightweight as compared to AWT components. It provides a good interface to the user for all the platforms. It is not specifically for one platform. The components are written in Java and platform-independent as well. The Java foundation classes were first appeared in 1997 and then later on it is termed as Swing. To use the swing in java, javax. swing package needs to be used or import. It is also known as Java Swing

Below are the different components of swing in java:

1. ImageIcon

The ImageIcon component creates an icon sized-image from an image residing at the source URL.

2. JButton

JButton class is used to create a push-button on the UI. The button can contain some display text or image. It generates an event when clicked and double-clicked. A JButton can be implemented in the application by calling one of its constructors.

3. JLabel

JLabel class is used to render a read-only text label or images on the UI. It does not generate any event.

4. JTextField

JTextField renders an editable single-line text box. A user can input non-formatted text in the box. To initialize the text field, call its constructor and pass an optional integer parameter to it. This parameter sets the width of the box measured by the number of columns. It does not limit the number of characters that can be input in the box.

5. JTextArea

JTextArea class renders a multi-line text box. Similar to the JTextField, a user can input non-formatted text in the field. The constructor for JTextArea also expects two integer parameters which define the height and width of the text-area in columns. It does not restrict the number of characters that the user can input in the text-area.

6. JPasswordField

JPasswordField is a subclass of JTextField class. It renders a text-box that masks the user input text with bullet points. This is used for inserting passwords into the application.

7. JCheckBox

JCheckBox renders a check-box with a label. The check-box has two states – on/off. When selected, the state is on and a small tick is displayed in the box.

8. JRadioButton

JRadioButton is used to render a group of radio buttons in the UI. A user can select one choice from the group.

9. JList

JList component renders a scrollable list of elements. A user can select a value or multiple values from the list. This select behavior is defined in the code by the developer.

10. JComboBox

JComboBox class is used to render a dropdown of the list of options.

3.b)Write an applet program to handle various mouse events and key events

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class MouseApplet extends Applet implements MouseListener
{
        String msg="Initial Message";
        public void init()
        {
                addMouseListener(this);
```

```
	}
	public void mouseClicked(MouseEvent me)
	{
		msg = "Mouse Clicked";
		repaint();
	}
	public void mousePressed(MouseEvent me)
	{
		msg = "Mouse Pressed";
		repaint();
	}
	public void mouseReleased(MouseEvent me)
	{
		msg = "Mouse Released";
		repaint();
	}
	public void mouseEntered(MouseEvent me)
	{
		msg = "Mouse Entered";
		repaint();
	}
	public void mouseExited(MouseEvent me)
	{
		msg = "Mouse Exited";
		repaint();
	}
	public void paint(Graphics g)
	{
		g.drawString(msg,20,20);
	}
}
/*
<applet code="MouseApplet" height="300" width="500">
</applet>
*/
```

4 a) What are the different types of Event listeners supported by java.

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener

KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

4.b)Write a java program that design scientific calculator using AWT

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

class Calculator extends JFrame {
   private final Font BIGGER_FONT = new Font("monspaced",Font.PLAIN, 20);
   private JTextField textfield;
   private boolean number = true;
   private String equalOp = "=";
   private CalculatorOp op = new CalculatorOp();

   public Calculator() {
      textfield = new JTextField("", 12);
      textfield.setHorizontalAlignment(JTextField.RIGHT);
      textfield.setFont(BIGGER_FONT);
      ActionListener numberListener = new NumberListener();
      String buttonOrder = "1234567890 ";
      JPanel buttonPanel = new JPanel();
      buttonPanel.setLayout(new GridLayout(4, 4, 4, 4));
      for (int i = 0; i < buttonOrder.length(); i++) {
         String key = buttonOrder.substring(i, i+1);
         if (key.equals(" ")) {
            buttonPanel.add(new JLabel(""));
         } else {
            JButton button = new JButton(key);
            button.addActionListener(numberListener);
```

```
            button.setFont(BIGGER_FONT);
            buttonPanel.add(button);
        }
    }
    ActionListener operatorListener = new OperatorListener();
    JPanel panel = new JPanel();
    panel.setLayout(new GridLayout(4, 4, 4, 4));
    String[] opOrder = {"+", "-", "*", "/","=","C","sin","cos","log"};
    for (int i = 0; i < opOrder.length; i++) {
        JButton button = new JButton(opOrder[i]);
        button.addActionListener(operatorListener);
        button.setFont(BIGGER_FONT);
        panel.add(button);
    }
    JPanel pan = new JPanel();
    pan.setLayout(new BorderLayout(4, 4));
    pan.add(textfield, BorderLayout.NORTH );
    pan.add(buttonPanel , BorderLayout.CENTER);
    pan.add(panel , BorderLayout.EAST);
    this.setContentPane(pan);
    this.pack();
    this.setTitle("Calculator");
    this.setResizable(false);
}
private void action() {
    number = true;
    textfield.setText("");
    equalOp = "=";
    op.setTotal("");
}
class OperatorListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        String displayText = textfield.getText();
        if (e.getActionCommand().equals("sin"))
        {
            textfield.setText("" + Math.sin(Double.valueOf(displayText).doubleValue()));

        }else
        if (e.getActionCommand().equals("cos"))
        {
            textfield.setText("" + Math.cos(Double.valueOf(displayText).doubleValue()));

        }
        else
        if (e.getActionCommand().equals("log"))
```

```
        {
            textfield.setText("" + Math.log(Double.valueOf(displayText).doubleValue()));

        }
        else if (e.getActionCommand().equals("C"))
        {
            textfield.setText("");
        }

        else
        {
            if (number)
            {

                action();
                textfield.setText("");

            }
            else
            {
                number = true;
                if (equalOp.equals("="))
                {
                    op.setTotal(displayText);
                }else
                if (equalOp.equals("+"))
                {
                            op.add(displayText);
                        }
                        else if (equalOp.equals("-"))
                        {
                            op.subtract(displayText);
                        }
                        else if (equalOp.equals("*"))
                        {
                            op.multiply(displayText);
                        }
                        else if (equalOp.equals("/"))
                        {
                            op.divide(displayText);
                        }

                        textfield.setText("" + op.getTotalString());
                        equalOp = e.getActionCommand();
                    }
```

```
            }
        }
    }
    class NumberListener implements ActionListener {
        public void actionPerformed(ActionEvent event) {
            String digit = event.getActionCommand();
            if (number) {
                textfield.setText(digit);
                number = false;
            } else {
                textfield.setText(textfield.getText() + digit);
            }
        }
    }
    public class CalculatorOp {
        private int total;
        public CalculatorOp() {
            total = 0;
        }
        public String getTotalString() {
            return ""+total;
        }
        public void setTotal(String n) {
            total = convertToNumber(n);
        }
        public void add(String n) {
            total += convertToNumber(n);
        }
        public void subtract(String n) {
            total -= convertToNumber(n);
        }
        public void multiply(String n) {
            total *= convertToNumber(n);
        }
        public void divide(String n) {
            total /= convertToNumber(n);
        }
        private int convertToNumber(String n) {
            return Integer.parseInt(n);
        }
    }
}
class SwingCalculator {
    public static void main(String[] args) {
        JFrame frame = new Calculator();
```

```
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setVisible(true);
        }
    }
```

5 a) Is Applet more secure than application program? Justify your answer

Ans.

An **applet** is executed in a **more** restricted environment with **more security** restrictions. They can only access the browser specific services. An **application** can access data and resources available on the system without any **security** restrictions

Java Application	Java Applet
Applications are just like a Java programs that can be execute independently without using the web browser.	Applets are small Java programs that are designed to be included with the HTML web document. They require a Java-enabled web browser for execution.
Application program requires a main function for its execution.	Applet does not require a main function for its execution.
Java application programs have the full access to the local file system and network.	Applets don't have local disk and network access.
Applications can access all kinds of resources available on the system.	Applets can only access the browser specific services. They don't have access to the local system.
Applications can executes the programs from the local system.	Applets cannot execute programs from the local machine.
An application program is needed to perform some task directly for the user.	An applet program is needed to perform small tasks or the part of it.

5.b)Why swing components are preferred over AWT components?

Swing is the latest GUI toolkit, and provides a richer set of interface **components** than the **AWT**. In addition, **Swing components** offer the following advantages **over AWT components**: The behavior and appearance of **Swing components** is consistent across platforms, whereas **AWT components** will differ from platform to platform.

Each ION Java component is shipped in two forms, one built on AWT classes, the other on Swing classes. This section discusses the difference between AWT and Swing, the advantages and disadvantages of each, and how to distinguish between the ION AWT classes and the ION Swing classes.

AWT and Swing are both part of a group of Java class libraries called the Java Foundation Classes (JFC). The Abstract Windowing Toolkit (AWT) is the original GUI toolkit shipped with the Java Development Kit (JDK). The AWT provides a basic set of graphical interface components similar to those available with HTML forms. Swing is the latest GUI toolkit, and provides a richer set of interface components than the AWT. In addition, Swing components offer the following advantages over AWT components:

- The behavior and appearance of Swing components is consistent across platforms, whereas AWT components will differ from platform to platform
- Swing components can be given their own "look and feel"
- Swing uses a more efficient event model than AWT; therefore, Swing components can run more quickly than their AWT counterparts

On the other hand, Swing components can take longer to load than AWT components.

ION Applications should use either all AWT-based components, or all Swing-based components. Mixing AWT and Swing components in the same application can cause problems with the stacking order of your components.

The ION Swing components can be identified by a "J." For example, the Swing version of the IONPlot class is called IONJPlot.

6 a) What is an adapter class? With example demonstrate their role in event handling?
(or) Write a program to demonstrate key and mouse Adapter class
Ans.

Java Adapter Classes

Java adapter classes *provide the default implementation of listener interfaces*. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it *saves code*.

The adapter classes are found in **java.awt.event**, **java.awt.dnd** and **javax.swing.event** packages. The Adapter classes with their corresponding listener interfaces are given below.

Adapter classes are a set of classes which can be used to simplify the event handling process. As we can see in the mouse event handling program here, even though we want to handle only one or two mouse events, we have to define all other remaining methods available in the listener interface(s).

To remove the aforementioned disadvantage, we can use adapter classes and define only the required event handling method(s). Some of the adapter classes available in the *java.awt.event* package are listed below:

Adapter Class	Listener Interface
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener and (as of JDK 6) MouseMotionListener and MouseWheelListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener, WindowFocusListener, and WindowStateListener

```
1  import java.awt.*;
2  import java.awt.event.*;
3  public class AdapterExample{
4      Frame f;
5      AdapterExample(){
6          f=new Frame("Window Adapter");
7          f.addWindowListener(new WindowAdapter(){
8              public void windowClosing(WindowEvent e) {
9                  f.dispose();
1.             }
1.         });
1.
1.         f.setSize(400,400);
1.         f.setLayout(null);
1.         f.setVisible(true);
1.     }
1. public static void main(String[] args) {
1.     new AdapterExample();
1. }
2. }
```

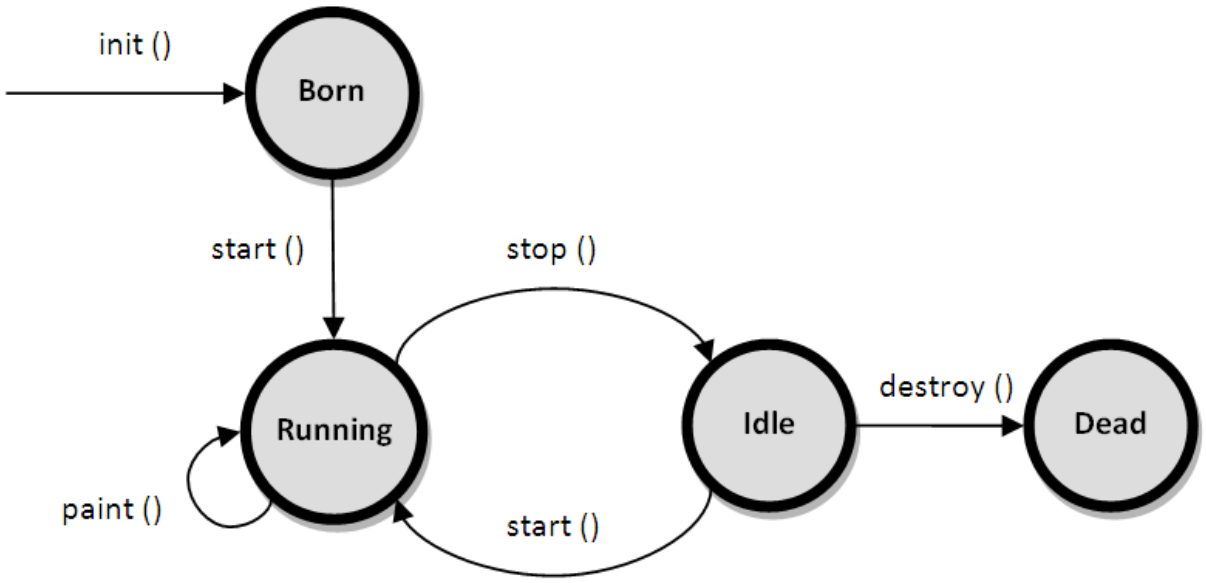
7a) What is an applet? Explain the life cycle of Applet with a neat sketch.

An **applet** is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

There are some important differences between an applet and a standalone Java application, including the following −

- An applet is a Java class that extends the java.applet.Applet class.

- A main() method is not invoked on an applet, and an applet class will not define main().
- Applets are designed to be embedded within an HTML page.
- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
- A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
- The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
- Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.
- Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file.



Life Cycle of an Applet

Four methods in the Applet class gives you the framework on which you build any serious applet −

- **init** − This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.
- **start** − This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.
- **stop** − This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.

- **destroy** − This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.
- **paint** − Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

7.b) Write the applets to draw the Cube,Circle and Cylinder shapes

```
/*Cylinder*/
g.drawString("(a).Cylinder",10,110);
g.drawOval(10,10,50,10);
g.drawOval(10,80,50,10);
g.drawLine(10,15,10,85);
g.drawLine(60,15,60,85);
/*Cube*/
g.drawString("(b).Cube",95,110);
g.drawRect(80,10,50,50);
g.drawRect(95,25,50,50);
g.drawLine(80,10,95,25);
g.drawLine(130,10,145,25);
g.drawLine(80,60,95,75);
g.drawLine(130,60,145,75);
```

Applet program to draw Square

- import java. **applet**. **Applet**;
- import java. awt. Graphics;
- public class DrawRectanglesExample extends **Applet**{
- public void paint(Graphics g){
- g. drawRect(10,10,50,100);
- g. drawRect(100,100,50,50)

8. Discuss various AWT containers with examples.
Ans.

Containers are integral part of AWT GUI components. A container provides a space where a component can be located. A Container in AWT is a component itself and it adds the capability to add component to itself. Following are noticable points to be considered.

- Sub classes of Container are called as Containter. For example Panel, Frame and Window.
- Container can add only Component to itself.
- A default layout is present in each container which can be overridden using setLayout method.

Sr. No.	Container & Description

1	Container It is a generic container object which can contain other AWT components.

AWT UI Elements:

Following is the list of commonly used containers while designed GUI using AWT.

Sr. No.	Container & Description
1	Panel Panel is the simplest container. It provides space in which any other component can be placed, including other panels.
2	Frame A Frame is a top-level window with a title and a border
3	Window A Window object is a top-level window with no borders and no menubar.

```
import java.awt.*;
import javax.swing.*;
public class ContainerTest extends JFrame { // top-level container
  JPanel panel; // low-level container
  JTextField field;
  JButton btn;
  public ContainerTest() {
    setTitle("Container Test");
    panel = new JPanel();
```

```
    field = new JTextField(20);
    panel.add(field);
    btn = new JButton("Submit");
    panel.add(btn);
    add(panel, BorderLayout.CENTER);
    setSize(350, 275);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLocationRelativeTo(null);
    setVisible(true);
  }
  public static void main(String args[]) {
    new ContainerTest();
  }
}
```

9)Discuss about anonymous inner classes.

Anonymous Inner Classes

An anonymous inner class is one that is not assigned a name. This section illustrates how an anonymous inner class can facilitate the writing of event handlers. Consider the applet shown in the following listing. As before, its goal is to display the string “Mouse Pressed” in the status bar of the applet viewer or browser when the mouse is pressed.

```
// Anonymous inner class demo.
import java.applet.*;
import java.awt.event.*;
/*
<applet code="AnonymousInnerClassDemo" width=200 height=100>
</applet>
*/
public class AnonymousInnerClassDemo extends Applet {
public void init() {
addMouseListener(new MouseAdapter() {
public void mousePressed(MouseEvent me) {
showStatus("Mouse Pressed");
}
});
```

```
}
}
```

There is one top-level class in this program: AnonymousInnerClassDemo. The init( ) method calls the addMouseListener( ) method. Its argument is an expression that defines and instantiates an anonymous inner class. Let's analyze this expression carefully.

The syntax new MouseAdapter( ) { ... } indicates to the compiler that the code between the braces defines an anonymous inner class. Furthermore, that class extends MouseAdapter. This new class is not named, but it is automatically instantiated when this expression is executed.

Because this anonymous inner class is defined within the scope of AnonymousInnerClassDemo, it has access to all of the variables and methods within the scope of that class. Therefore, it can call the showStatus( ) method directly.

As just illustrated, both named and anonymous inner classes solve some annoying problems in a simple yet effective way. They also allow you to create more efficient code

PART- C

UNIT WISE UNIVERSITY PREVIOUS QUESTION PAPER QUESTIONS

UNIT –I PREVIOUS ASKED ESSAY QUESTIONS

1.What are the responsibilities of an agent? [5 m]

2.Define inheritance Explain the various forms of inheritance with suitable code segments. How to prevent a class from inheritance? [10 M]

3.Write a program to demonstrate hierarchical and multiple inheritance using interfaces [10 M]

4 Write the significance and internal Architecture of Java Virtual Machine(JVM),. Briefly explain how Java is platform independent[10 M]

5.What are the drawbacks of procedural languages? Explain the need of object oriented programming with suitable program. [10 M]

6. Explain about various control statements[10 M]

7.Explain the significance of public, protected and private access specifies in Inheritance. [10 M]

(or) Describe different levels of access protection available in Java.

8.With suitable example demonstrate super, this ,final and[5 m] static keywords[10 M]

9.What is polymorphism? Explain different types of polymorphisms with examples. [10 M] (or)

Compare and cons tract overloading and overriding with an example

10)What is a nested class? Differentiate between static nested classes and non-static nested classes. [10 M]

11)Discuss about anonymous inner classes. [5 m]

12) What is the purpose of constructor in Java programming? Explain the different types of constructors with an example.

13) Write a program to find the transpose of a given matrix. [10 M]

14) Explain the different parameter passing mechanisms used in Java with an example. [5 m]

UNIT –II PREVIOUS ASKED ESSAY QUESTIONS

1. How to define a package?Explain with suitable example how to create ,import and access a package?

2. What is the need of Generics?[4 marks]

3. Write a program to implement the operations of random access file [5 m]

5. Explain the file management using File class.[5m]

6. Explain how multiple inheritance is achieved in java .

7. Create an interface with at least one method and implement that interface by within a method which returns a reference to your interface.[5m]

8. Write a program to compute an average of the values in a file.[5m]

9. Explain multilevel inheritance with the help of abstract class in your program. .[5m]

10. Can inheritance be applied between interfaces? Justify your answer. .[5m]

11. Differentiate between interface and abstract class. .[5m]

12. Write a program to copy the contents of file1 to file 2. Read the names of files as command line arguments.[5m]

13. What support is provided by File class for file management? Illustrate with suitable scenarios. .[5m]

14. What is an interface? What are the similarities between interfaces and classes? .[5m]

15. How can you extend one interface by the other interface? Discuss. .[5m]

16. Discuss about CLASSPATH environment variables. [5m]

17. Discuss the different levels of access protection available in Java. [5m]

18. Demonstrate ordinal( ) method of enum. [5m]

19. What is type wrapper? What is the role of auto boxing? [5m]

20. Explain the process of defining and creating a package with suitable examples. [5m]

21. Give an example where interface can be used to support multiple inheritance. [5m]

22. Describe the process of importing and accessing a package with suitable examples.[5m]

23. How to design and implement an interface in Java? Give an example[5m]

24. Give an example where interface can be used to support multiple inheritance. [5m]

25. What are the methods available in the Character Streams? Discuss. [5m]

26. Distinguish between Byte Stream Classes and Character Stream Classes. [5m]

27. What is the accessibility of a public method or field inside a nonpublic class or interface? Explain. [5m]

UNIT –III PREVIOUS ASKED ESSAY QUESTIONS

1.a)With a suitable Java program explain user-defined exception handling.
b)What is meant by re-throwing exception? Discuss a suitable scenario for this. [5+5]

2.Does Java support thread priorities? Justify your answer with suitable discussion. b)**Describe producer-consumer pattern using inter-thread communication. [5+5]

3.a)Write a program that demonstrate the priority setting in threads.
b)Write a program that includes a try block and a catch clause which processes the arithmetic exception generated by division-by-zero error. [5+5]

4.a)Write a program that creates a thread that forces preemptive scheduling for lower- priority threads. b)Explain the checked and unchecked exception With an example [5+5]

5 a) Write a program for user defined exception that check the internal and external marks are greater than 40 it raise the exception "internal marks are exceed" and if external marks greater than 60 exception in raised the exception "external marks Exceed"

6a) Explain about Synchronization method with an example. b)Explain about built in Exceptions

7. What is an exception? How are exceptions handled in Java programming? Explain with suitable program. [10]

8. Describe the need of thread synchronization. How is it achieved in Java programming? Explain with a suitable program[10]

9a)Write a program to illustrate the use of multiple catch blocks for a try block. b)What are the uses of 'throw' and 'throws' clauses for exception handling? [5+5]

10.a) What is the difference between a thread and a process?
b)How to achieve synchronization among threads? Write suitable code. [5+5]

11.a)With a program illustrate user defined exception handling
b)How to handle multiple catch blocks for a nested try block? Explain with an example. [5+5]

12.a)Describe how to create a thread with an example.
b)Write a program to explain thread priorities usage. [5+5]

13a)What are advantages of using Exception handling mechanism in a program?
b)Write a java program that demonstrates how certain exception types are not allowed to be thrown. [5+5]

14a)What are the different ways that are possible to create multiple threaded programs in java? Discuss the differences between them.

b)Write a program to create four threads using Runnable interface.

15Write a program to create three threads in your program and context switch among the threads using sleep functions. [10]

16a)Write a program with nested try statements for handling exception. b)How to create a user defined exception? [5+5]

17a)Differentiate between Checked and UnChecked Exceptions with examples. b)Write a program to create four threads using Runnable interface. [5+5]

18a)What are the different ways to handle exceptions? Explain.
b)How many ways are possible in java to create multiple threaded programs? Discuss the differences between them. [5+5]

19a)What is an Exception? How is an Exception handled in JAVA?
b)Write a java program that illustrates the application of multiple catch statements. [5+5]

20a)Differentiate between multiprocessing and multithreading.What is to be done to implement these in a program?
b)Write a program that creates two threads. Fist thread prints the numbers
from 1 to 100 and the other thread prints the numbers from 100 to 1. [5+5]

UNIT –IV PREVIOUS ASKED ESSAY QUESTIONS

1 .a) Give an account of Random collection class
b) Discuss the methods of Stack class
c) What is the need of Generics? [3+3+4]

2.a) Explain the file management using File class.
b) Write a program which stores a list of strings in an Array List and then displays the contents of the list. [5+5]
3 .Write the name of bank depositor and their balance using Hash table.

4. a) Differentiate between ArrayList and Vector.
b) List the methods of Stack class. [5+5]
5 .a) What is a vector? How does it differ from array, list?
b) Write a program to count number of words in a given sentence. [5+5]

6 .a) What is Java Collections Framework? List out some benefits of Collections framework and explain.
b) What is the importance of hashCode() and equals() methods? [5+5]
7 a) What are the common algorithms implemented in Collections Framework? Discuss.
b) What is difference between ArrayList and LinkedList in collection framework? Explain. [5+5]

8 a) Contrast sorted map and navigable map interfaces.
b) What is the purpose of BitSet class? What is the functionality of the following functions of BitSet class: cardinality( ) , flip( ) and intersects( ) [5+5]

9 a) Differentiate between ArrayList and a Vector? Why ArrayList is faster than Vector? Explain.
b) How an Hashtable can change the iterator? Explain. [5+5]

10 a) Explain the Bit Set and Calander classes in detail.
b) Discuss the differences between HashList and HashMap, Set and List. [5+5]

11 a) What are similarities and difference between ArrayList and Vector? Explain.
b) What is different between Iterator and ListIterator? Explain different ways to iterate over a list. [5+5]
12 .a) What are the best practices related to Java Collections Framework? Discuss.
b) What is Comparable and Comparator interface? Differentiate between them. [5+5]

UNIT-V JNTUH-PREVIOUSLY ASKED ESSAY QUESTIONS

1a) What is the significance of layout managers? Discuss briefly various layout managers.
b) Give an overview of JButton class. [5+5]

2a) Explain delegation event model.
b) Write an Applet to draw a smiley picture accept user name as a parameter and display welcome message. [5+5]

3a) Describe about various components in AWT.
b) Write an applet program to handle all mouse events. [5+5]

4a) Write a Java program to create AWT radio buttons using check box group.
b) Explain the various event listener interfaces. [5+5]

5 a)Write a java program that design scientific calculator using AWT
b)What are the different types of Event listeners supported by java

6 a) Is Applet more secure than application program? Justify your answer.
b) Design a user interface to collect data from the student for admission application using swing components. [5+5]

7. Write a program to demonstrate various keyboard events with suitable functionality. [10]

8.a) Why swing components are preferred over AWT components?
b) What is an adapter class? What is their role in event handling? [5+5]

9a) Explain the life cycle of an applet.
b) What are the various layout managers used in Java? [5+5]

10.a) What is the role of event listeners in event handling? List the Java event listeners
b) Write an applet to display the mouse cursor position in that applet window.[5+5]

11.a) Discuss various AWT containers with examples.
b) Explain about the adapter class with an example. [5+5]

12.a) What is an applet? Explain the life cycle of Applet with a neat sketch.
b) Write the applets to draw the Cube and Cylinder shapes. [5+5]

13.a) What is an Layout manager? Explain different types of Layout managers.
b) Write a program to create a frame window that responds to key strokes. [5+5]

14.a) Illustrate the use of Grid Bag layout.
b) What are the subclasses of JButton class of swing package? [5+5]

15.a) Create a simple applet to display a smiley picture using Graphics class methods.
b) Write a short note on delegation event model. [5+5]

16 a) List and explain different types of Layout managers with suitable examples.
b) How to move/drag a component placed in Swing Container? Explain. [5+5]

17.a) Discuss about different applet display methods in brief.
b) What are the various components of Swing? Explain. [5+5]

18 a) What is the difference between init( ) and start ( ) methods in an Applet? When will each be executed?
b) Write the applets to draw the Cube and Circle shapes. [5+5]

19. a) Explain various layout managers in JAVA.
b) Write a program to create a frame window that responds to mouse clicks. [5+5]