CMR ENGINEERING COLLEGE

(Approved by AICTE, Affiliated to JNTU, Hyderabad)
KANDLAKOYA (V), MEDCHAL ROAD, HYDERABAD-501401.
Ph: 08418 200037, 92470 22662, Fax: 08418 200240, www.cmrec.ac.in
Department of Information Technology

STEP MATERIAL

SUBJECT: Java Programming

I. Short Answer Questions:

Q1. What is the difference between an Inner Class and a Sub-Class?

Ans: An Inner class is a class which is nested within another class. An Inner class has access rights for the class which is nesting it and it can access all variables and methods defined in the outer class.

A sub-class is a class which inherits from another class called super class. Sub-class can access all public and protected methods and fields of its super class.

Q2. What are the various access specifiers for Java classes?

Ans: In Java, access specifiers are the keywords used before a class name which defines the access scope. The types of access specifiers for classes are:

1. Public : Class,Method,Field is accessible from anywhere.

2. Protected:Method,Field can be accessed from the same class to which they belong or from the sub-classes,and from the class of same package,but not from outside.

3. Default: Method,Field,class can be accessed only from the same package and not from outside of it's native package.

4. Private: Method,Field can be accessed from the same class to which they belong.

Q3. What's the purpose of Static methods and static variables?

Ans: When there is a requirement to share a method or a variable between multiple objects of a class instead of creating separate copies for each

object, we use static keyword to make a method or variable shared for all objects.

Q4. What is data encapsulation and what's its significance?

Ans: Encapsulation is a concept in Object Oriented Programming for combining properties and methods in a single unit.

Encapsulation helps programmers to follow a modular approach for software development as each object has its own set of methods and variables and serves its functions independent of other objects. Encapsulation also serves data hiding purpose.

Q5. What is a singleton class? Give a practical example of its usage.

A singleton class in java can have only one instance and hence all its methods and variables belong to just one instance. Singleton class concept is useful for the situations when there is a need to limit the number of objects for a class.

The best example of singleton usage scenario is when there is a limit of having only one connection to a database due to some driver limitations or because of any licensing issues.

Q6. What are Loops in Java? What are three types of loops?

Ans: Looping is used in programming to execute a statement or a block of statement repeatedly. There are three types of loops in Java:

1) For Loops

For loops are used in java to execute statements repeatedly for a given number of times. For loops are used when number of times to execute the statements is known to programmer.

2) While Loops

While loop is used when certain statements need to be executed repeatedly until a condition is fulfilled. In while loops, condition is checked first before execution of statements.

3) Do While Loops

Do While Loop is same as While loop with only difference that condition is checked after execution of block of statements. Hence in case of do while loop, statements are executed at least once.

Q7: What is an infinite Loop? How infinite loop is declared?

Ans: An infinite loop runs without any condition and runs infinitely. An infinite loop can be broken by defining any breaking logic in the body of the statement blocks.

Infinite loop is declared as follows:

```
for (;;)
{
    // Statements to execute

    // Add any loop breaking logic
}
```

Q8. What is the difference between continue and break statement?

Ans: break and continue are two important keywords used in Loops. When a break keyword is used in a loop, loop is broken instantly while when continue keyword is used, current iteration is broken and loop continues with next iteration.

In below example, Loop is broken when counter reaches 4.

```
for (counter = 0; counter & lt; 10; counter++)
    system.out.println(counter);

if (counter == 4) {

    break;
}

}
```

In the below example when counter reaches 4, loop jumps to next iteration and any statements after the continue keyword are skipped for current iteration.

```
for (counter = 0; counter < 10; counter++)
    system.out.println(counter);

if (counter == 4) {

    continue;
}
system.out.println("This will not get printed when counter is 4");
```

```
}
```

Q9. What is the difference between double and float variables in Java?

Ans: In java, float takes 4 bytes in memory while Double takes 8 bytes in memory. Float is single precision floating point decimal number while Double is double precision decimal number.

Q10. What is Final Keyword in Java? Give an example.

Ans: In java, a constant is declared using the keyword Final. Value can be assigned only once and after assignment, value of a constant can't be changed.

In below example, a constant with the name const_val is declared and assigned avalue:

Private Final int const_val=100

When a method is declared as final,it can NOT be overridden by the subclasses.This method are faster than any other method,because they are resolved at complied time.

When a class is declares as final,it cannot be subclassed. Example String,Integer and other wrapper classes.

Q11. What is ternary operator? Give an example.

Ans: Ternary operator , also called conditional operator is used to decide which value to assign to a variable based on a Boolean value evaluation. It's denoted as ?

In the below example, if rank is 1, status is assigned a value of "Done" else "Pending".

```
public class conditionTest {
    public static void main(String args[]) {
        String status;
        int rank = 3;
        status = (rank == 1) ? "Done" : "Pending";
        System.out.println(status);
    }
}
```

Q12: How can you generate random numbers in Java?

Ans:

- Using Math.random() you can generate random numbers in the range greater than or equal to 0.1 and less than 1.0
- Using Random class in package java.util

Q13. What is default switch case? Give example.

Ans: In a switch statement, default case is executed when no other switch condition matches. Default case is an optional case .It can be declared only once all other switch cases have been coded.

In the below example, when score is not 1 or 2, default case is used.

```
public class switchExample {
    int score = 4;
    public static void main(String args[]) {
        switch (score) {
            case 1:
                system.out.println("Score is 1");
                break;
            case 2:
                system.out.println("Score is 2");
                break;
            default:
                system.out.println("Default Case");
        }
    }
}
```

Q14. What's the base class in Java from which all classes are derived?

Ans: java.lang.object

Q15. Can main() method in Java can return any data?

Ans: In java, main() method can't return any data and hence, it's always declared with a void return type.

Q16. What are Java Packages? What's the significance of packages?

Ans: In Java, package is a collection of classes and interfaces which are bundled together as they are related to each other. Use of packages helps developers to modularize the code and group the code for proper re-use.

Once code has been packaged in Packages, it can be imported in other classes and used.

Q17. Can we declare a class as Abstract without having any abstract method?

Ans: Yes we can create an abstract class by using abstract keyword before class name even if it doesn't have any abstract method. However, if a class has even one abstract method, it must be declared as abstract otherwise it will give an error.

Q18. What's the difference between an Abstract Class and Interface in Java?

Ans: The primary difference between an abstract class and interface is that an interface can only possess declaration of public static methods with no concrete implementation while an abstract class can have members with any access specifiers (public, private etc) with or without concrete implementation.

Another key difference in the use of abstract classes and interfaces is that a class which implements an interface must implement all the methods of the interface while a class which inherits from an abstract class doesn't require implementation of all the methods of its super class.

A class can implement multiple interfaces but it can extend only one abstract class.

Q19. What are the performance implications of Interfaces over abstract classes?

Ans: Interfaces are slower in performance as compared to abstract classes as extra indirections are required for interfaces. Another key factor for developers to take into consideration is that any class can extend only one abstract class while a class can implement many interfaces.

Use of interfaces also puts an extra burden on the developers as any time an interface is implemented in a class; developer is forced to implement each and every method of interface.

Q20. Does Importing a package imports its sub-packages as well in Java?

Ans: In java, when a package is imported, its sub-packages aren't imported and developer needs to import them separately if required.

For example, if a developer imports a package university.*, all classes in the package named university are loaded but no classes from the sub-package are loaded. To load the classes from its sub-package (say department), developer has to import it explicitly as follows:

Import university.department.*

Q21. Can we declare the main method of our class as private?

Ans: In java, main method must be public static in order to run any application correctly. If main method is declared as private, developer won't get any compilation error however, it will not get executed and will give a runtime error.

Q22. How can we pass argument to a function by reference instead of pass by value?

Ans: In java, we can pass argument to a function only by value and not by reference.

Q23. How an object is serialized in java?

Ans: In java, to convert an object into byte stream by serialization, an interface with the name Serializable is implemented by the class. All objects of a class implementing serializable interface get serialized and their state is saved in byte stream.

Q24. When we should use serialization?

Ans: Serialization is used when data needs to be transmitted over the network. Using serialization, object's state is saved and converted into byte stream .The byte stream is transferred over the network and the object is re-created at destination.

Q25. Is it compulsory for a Try Block to be followed by a Catch Block in Java for Exception handling?

Ans: Try block needs to be followed by either Catch block or Finally block or both. Any exception thrown from try block needs to be either caught in the catch block or else any specific tasks to be performed before code abortion are put in the Finally block.

Q26. Is there any way to skip Finally block of exception even if some exception occurs in the exception block?

Ans: If an exception is raised in Try block, control passes to catch block if it exists otherwise to finally block. Finally block is always executed when an exception occurs and the only way to avoid execution of any statements in Finally block is by aborting the code forcibly by writing following line of code at the end of try block:

```
System.exit(0);
```

Q27. When the constructor of a class is invoked?

Ans: The constructor of a class is invoked every time an object is created with new keyword.

For example, in the following class two objects are created using new keyword and hence, constructor is invoked two times.

```
public class const_example {

    const_example() {

        system.out.println("Inside constructor");
    }
    public static void main(String args[]) {

        const_example c1 = new const_example();

        const_example c2 = new const_example();
    }
}
```

Q28. Can a class have multiple constructors?

Ans: Yes, a class can have multiple constructors with different parameters. Which constructor gets used for object creation depends on the arguments passed while creating the objects.

Q29. Can we override static methods of a class?

Ans: We cannot override static methods. Static methods belong to a class and not to individual objects and are resolved at the time of compilation (not at runtime).Even if we try to override static method,we will not get an complitaion error,nor the impact of overriding when running the code.

Q30. In the below example, what will be the output?

```
public class superclass {
```

```
    public void displayResult() {

        system.out.println("Printing from superclass");

    }

}

public class subclass extends superclass {

    public void displayResult() {

        system.out.println("Displaying from subClass");

        super.displayResult();

    }

    public static void main(String args[]) {

        subclass obj = new subclass();

        obj.displayResult();

    }

}
```

Ans: Output will be:

Displaying from subclass

Displaying from superclass

Q31. Is String a data type in java?

Ans: String is not a primitive data type in java. When a string is created in java, it's actually an object of Java.Lang.String class that gets created. After creation of this string object, all built-in methods of String class can be used on the string object.

Q32. In the below example, how many String Objects are created?

```
String s1="I am Java Expert";
```

```
String s2="I am C Expert";

String s3="I am Java Expert";
```

Ans: In the above example, two objects of Java.Lang.String class are created. s1 and s3 are references to same object.

Q33. Why Strings in Java are called as Immutable?

Ans: In java, string objects are called immutable as once value has been assigned to a string, it can't be changed and if changed, a new object is created.

In below example, reference str refers to a string object having value "Value one".

```
String str="Value One";
```

When a new value is assigned to it, a new String object gets created and the reference is moved to the new object.

```
str="New Value";
```

Q34. What's the difference between an array and Vector?

Ans: An array groups data of same primitive type and is static in nature while vectors are dynamic in nature and can hold data of different data types.

Q35. What is multi-threading?

Ans: Multi threading is a programming concept to run multiple tasks in a concurrent manner within a single program. Threads share same process stack and running in parallel. It helps in performance improvement of any program.

Q36. Why Runnable Interface is used in Java?

Ans: Runnable interface is used in java for implementing multi threaded applications. Java.Lang.Runnable interface is implemented by a class to support multi threading.

Q37. What are the two ways of implementing multi-threading in Java?

Ans: Multi threaded applications can be developed in Java by using any of the following two methodologies:

1. By using Java.Lang.Runnable Interface. Classes implement this interface to enable multi threading. There is a Run() method in this interface which is implemented.

2. By writing a class that extend Java.Lang.Thread class.

Q38. When a lot of changes are required in data, which one should be a preference to be used? String or StringBuffer?

Ans: Since StringBuffers are dynamic in nature and we can change the values of StringBuffer objects unlike String which is immutable, it's always a good choice to use StringBuffer when data is being changed too much. If we use String in such a case, for every data change a new String object will be created which will be an extra overhead.

Q39. What's the purpose of using Break in each case of Switch Statement?

Ans: Break is used after each case (except the last one) in a switch so that code breaks after the valid case and doesn't flow in the proceeding cases too.

If break isn't used after each case, all cases after the valid case also get executed resulting in wrong results.

Q40. How garbage collection is done in Java?

Ans: In java, when an object is not referenced any more, garbage collection takes place and the object is destroyed automatically. For automatic garbage collection java calls either System.gc() method or Runtime.gc() method.

Q41. How we can execute any code even before main method?

Ans: If we want to execute any statements before even creation of objects at load time of class, we can use a static block of code in the class. Any statements inside this static block of code will get executed once at the time of loading the class even before creation of objects in the main method.

Q42. Can a class be a super class and a sub-class at the same time? Give example.

Ans: If there is a hierarchy of inheritance used, a class can be a super class for another class and a sub-class for another one at the same time.

In the example below, continent class is sub-class of world class and it's super class of country class.

```
public class world {

..........

}
public class continenet extends world {

............

}
public class country extends continent {

......................

}
```

Q43. How objects of a class are created if no constructor is defined in the class?

Ans: Even if no explicit constructor is defined in a java class, objects get created successfully as a default constructor is implicitly used for object creation. This constructor has no parameters.

Q44. In multi-threading how can we ensure that a resource isn't used by multiple threads simultaneously?

Ans: In multi-threading, access to the resources which are shared among multiple threads can be controlled by using the concept of synchronization. Using synchronized keyword, we can ensure that only one thread can use shared resource at a time and others can get control of the resource only once it has become free from the other one using it.

Q45. Can we call the constructor of a class more than once for an object?

Ans: Constructor is called automatically when we create an object using new keyword. It's called only once for an object at the time of object creation and hence, we can't invoke the constructor again for an object after its creation.

Q46. There are two classes named classA and classB. Both classes are in the same package. Can a private member of classA can be accessed by an object of classB?

Ans: Private members of a class aren't accessible outside the scope of that class and any other class even in the same package can't access them.

Q47. Can we have two methods in a class with the same name?

Ans: We can define two methods in a class with the same name but with different number/type of parameters. Which method is to get invoked will depend upon the parameters passed.

For example in the class below we have two print methods with same name but different parameters. Depending upon the parameters, appropriate one will be called:

```
public class methodExample {

    public void print() {

        system.out.println("Print method without parameters.");

    }

    public void print(String name) {

        system.out.println("Print method with parameter");

    }

    public static void main(String args[]) {

        methodExample obj1 = new methodExample();

        obj1.print();

        obj1.print("xx");

    }

}
```

Q48. How can we make copy of a java object?

Ans: We can use the concept of cloning to create copy of an object. Using clone, we create copies with the actual state of an object.

Clone() is a method of Cloneable interface and hence, Cloneable interface needs to be implemented for making object copies.

Q49. What's the benefit of using inheritance?

Ans: Key benefit of using inheritance is reusability of code as inheritance enables sub-classes to reuse the code of its super class. Polymorphism (Extensibility) is another great benefit which allow new functionality to be introduced without effecting existing derived classes.

Q50. What's the default access specifier for variables and methods of a class?

Ans: Default access specifier for variables and method is package protected i.e variables and class is available to any other class but in the same package,not outside the package.

Q51. Give an example of use of Pointers in Java class.

Ans: There are no pointers in Java. So we can't use concept of pointers in Java.

Q52. How can we restrict inheritance for a class so that no class can be inherited from it?

Ans: If we want a class not to be extended further by any class, we can use the keyword **Final** with the class name.

In the following example, Stone class is Final and can't be extend

```
public Final Class Stone {
        // Class methods and Variables
}
```

Q53. What's the access scope of Protected Access specifier?

Ans: When a method or a variable is declared with Protected access specifier, it becomes accessible in the same class,any other class of the same package as well as a sub-class.

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Q54. What's difference between Stack and Queue?

Ans: Stack and Queue both are used as placeholder for a collection of data. The primary difference between a stack and a queue is that stack is based on Last in First out (LIFO) principle while a queue is based on FIFO (First In First Out) principle.

Q55. In java, how we can disallow serialization of variables?

Ans: If we want certain variables of a class not to be serialized, we can use the keyword **transient** while declaring them. For example, the variable trans_var below is a transient variable and can't be serialized:

```
public class transientExample {
        private transient trans_var;
    // rest of the code
}
```

Q56. How can we use primitive data types as objects?

Ans: Primitive data types like int can be handled as objects by the use of their respective wrapper classes. For example, Integer is a wrapper class for primitive data type int. We can apply different methods to a wrapper class, just like any other object.

Q57. Which types of exceptions are caught at compile time?

Ans: Checked exceptions can be caught at the time of program compilation. Checked exceptions must be handled by using try catch block in the code in order to successfully compile the code.

Q58. Describe different states of a thread.

Ans: A thread in Java can be in either of the following states:

- Ready: When a thread is created, it's in Ready state.
- Running: A thread currently being executed is in running state.
- Waiting: A thread waiting for another thread to free certain resources is in waiting state.
- Dead: A thread which has gone dead after execution is in dead state.

Q59. Can we use a default constructor of a class even if an explicit constructor is defined?

Ans: Java provides a default no argument constructor if no explicit constructor is defined in a Java class. But if an explicit constructor has

been defined, default constructor can't be invoked and developer can use only those constructors which are defined in the class.

Q60. Can we override a method by using same method name and arguments but different return types?

Ans: The basic condition of method overriding is that method name, arguments as well as return type must be exactly same as is that of the method being overridden. Hence using a different return type doesn't override a method.

Q61.What will be the output of following piece of code?

```
public class operatorExample {

    public static void main(String args[]) {

        int x = 4;

        system.out.println(x++);
        }
}
```

Ans: In this case postfix ++ operator is used which first returns the value and then increments. Hence it's output will be 4.

Q61. A person says that he compiled a java class successfully without even having a main method in it? Is it possible?

Ans: main method is an entry point of Java class and is required for execution of the program however; a class gets compiled successfully even if it doesn't have a main method. It can't be run though.

Q62. Can we call a non-static method from inside a static method?

Ans: Non-Static methods are owned by objects of a class and have object level scope and in order to call the non-Static methods from a static block (like from a static main method), an object of the class needs to be created first. Then using object reference, these methods can be invoked.

Q63. What are the two environment variables that must be set in order to run any Java programs?

Ans: Java programs can be executed in a machine only once following two environment variables have been properly set:

1. PATH variable
2. CLASSPATH variable

Q64. Can variables be used in Java without initialization?

Ans: In Java, if a variable is used in a code without prior initialization by a valid value, program doesn't compile and gives an error as no default value is assigned to variables in Java.

Q65. Can a class in Java be inherited from more than one class?

Ans: In Java, a class can be derived from only one class and not from multiple classes. Multiple inheritances is not supported by Java.

Q66. Can a constructor have different name than a Class name in Java?

Ans: Constructor in Java must have same name as the class name and if the name is different, it doesn't act as a constructor and compiler thinks of it as a normal method.

Q67. What will be the output of Round(3.7) and Ceil(3.7)?

Ans: Round(3.7) returns 4 and Ceil(3.7) returns 4.

Q68: Can we use goto in Java to go to a particular line?

Ans: In Java, there is not goto keyword and java doesn't support this feature of going to a particular labeled line.

Q69. Can a dead thread be started again?

Ans: In java, a thread which is in dead state can't be started again. There is no way to restart a dead thread.

Q70. Is the following class declaration correct?

Ans:

```
public abstract final class testClass {
        // Class methods and variables
}
```

Ans: The above class declaration is incorrect as an abstract class can't be declared as Final.

Q71. Is JDK required on each machine to run a Java program?

Ans: JDK is development Kit of Java and is required for development only and to run a Java program on a machine, JDK isn't required. Only JRE is required.

Q72. What's the difference between comparison done by equals method and == operator?

Ans: In Java, equals() method is used to compare the contents of two string objects and returns true if the two have same value while == operator compares the references of two string objects.

In the following example, equals() returns true as the two string objects have same values. However == operator returns false as both string objects are referencing to different objects:

```
public class equalsTest {

    public static void main(String args[]) {

        String str1 = new String("Hello World");

        String str2 = new String("Hello World");

        if (str1.equals(str2))

        { // this condition is true

            System.out.println("str1 and str2 are equal in terms of values");

        }

        if (str1 == str2) {

            //This condition is true

            System.out.println("Both strings are referencing same object");

        } else

        {

            // This condition is NOT true

            System.out.println("Both strings are referencing different objects");
```

```
        }

    }

}
```

Q73. Is it possible to define a method in Java class but provide it's implementation in the code of another language like C?

Ans: Yes, we can do this by use of native methods. In case of native method based development, we define public static methods in our Java class without its implementation and then implementation is done in another language like C separately.

Q74. How are destructors defined in Java?

Ans: In Java, there are no destructors defined in the class as there is no need to do so. Java has its own garbage collection mechanism which does the job automatically by destroying the objects when no longer referenced.

Q75. Can a variable be local and static at the same time?

Ans: No a variable can't be static as well as local at the same time. Defining a local variable as static gives compilation error.

Q76. Can we have static methods in an Interface?

Ans: Static methods can't be overridden in any class while any methods in an interface are by default abstract and are supposed to be implemented in the classes being implementing the interface. So it makes no sense to have static methods in an interface in Java.

Q77. In a class implementing an interface, can we change the value of any variable defined in the interface?

Ans: No, we can't change the value of any variable of an interface in the implementing class as all variables defined in the interface are by default public, static and Final and final variables are like constants which can't be changed later.

Q78. Is it correct to say that due to garbage collection feature in Java, a java program never goes out of memory?

Ans: Even though automatic garbage collection is provided by Java, it doesn't ensure that a Java program will not go out of memory as there is a

possibility that creation of Java objects is being done at a faster pace compared to garbage collection resulting in filling of all the available memory resources.

So, garbage collection helps in reducing the chances of a program going out of memory but it doesn't ensure that.

Q79. Can we have any other return type than void for main method?

Ans: No, Java class main method can have only void return type for the program to get successfully executed.

Nonetheless , if you absolutely must return a value to at the completion of main method , you can use System.exit(int status)

Q80. I want to re-reach and use an object once it has been garbage collected. How it's possible?

Ans: Once an object has been destroyed by garbage collector, it no longer exists on the heap and it can't be accessed again. There is no way to reference it again.

Q81. In Java thread programming, which method is a must implementation for all threads?

Ans: Run() is a method of Runnable interface that must be implemented by all threads.

Q82. I want to control database connections in my program and want that only one thread should be able to make database connection at a time. How can I implement this logic?

Ans: This can be implemented by use of the concept of synchronization. Database related code can be placed in a method which hs **synchronized** keyword so that only one thread can access it at a time.

Q83. How can an exception be thrown manually by a programmer?

Ans: In order to throw an exception in a block of code manually, **throw** keyword is used. Then this exception is caught and handled in the catch block.

```
public void topMethod() {
    try {
        excMethod();
    } catch (ManualException e) {}
```

```
}

public void excMethod {
    String name = null;
    if (name == null) {
        throw (new ManualException("Exception thrown manually ");
        }
    }
```

Q84. I want my class to be developed in such a way that no other class (even derived class) can create its objects. How can I do so?

Ans: If we declare the constructor of a class as private, it will not be accessible by any other class and hence, no other class will be able to instantiate it and formation of its object will be limited to itself only.

Q85. How objects are stored in Java?

Ans: In java, each object when created gets a memory space from a heap. When an object is destroyed by a garbage collector, the space allocated to it from the heap is re-allocated to the heap and becomes available for any new objects.

Q86. How can we find the actual size of an object on the heap?

Ans: In java, there is no way to find out the exact size of an object on the heap.

Q87. Which of the following classes will have more memory allocated?

Class A: Three methods, four variables, no object

Class B: Five methods, three variables, no object

Ans: Memory isn't allocated before creation of objects. Since for both classes, there are no objects created so no memory is allocated on heap for any class.

Q88. What happens if an exception is not handled in a program?

Ans: If an exception is not handled in a program using try catch blocks, program gets aborted and no statement executes after the statement which caused exception throwing.

Q89. I have multiple constructors defined in a class. Is it possible to call a constructor from another constructor's body?

Ans: If a class has multiple constructors, it's possible to call one constructor from the body of another one using **this()**.

Q90. What's meant by anonymous class?

Ans: An anonymous class is a class defined without any name in a single line of code using new keyword.

For example, in below code we have defined an anonymous class in one line of code:

```
public java.util.Enumeration testMethod()

{

    return new java.util.Enumeration()

    {

        @Override

        public boolean hasMoreElements()

        {

            // TODO Auto-generated method stub

            return false;

        }

        @Override

        public Object nextElement()

        {

            // TODO Auto-generated method stub

            return null;

        }
```

```
    }
```

Q91. Is there a way to increase the size of an array after its declaration?

Ans: Arrays are static and once we have specified its size, we can't change it. If we want to use such collections where we may require a change of size (no of items), we should prefer vector over array.

Q92. If an application has multiple classes in it, is it okay to have a main method in more than one class?

Ans: If there is main method in more than one classes in a java application, it won't cause any issue as entry point for any application will be a specific class and code will start from the main method of that particular class only.

Q93. I want to persist data of objects for later use. What's the best approach to do so?

Ans: The best way to persist data for future use is to use the concept of serialization.

Q94. What is a Local class in Java?

Ans: In Java, if we define a new class inside a particular block, it's called a local class. Such a class has local scope and isn't usable outside the block where its defined.

Q95. String and StringBuffer both represent String objects. Can we compare String and StringBuffer in Java?

Ans: Although String and StringBuffer both represent String objects, we can't compare them with each other and if we try to compare them, we get an error.

Q96. Which API is provided by Java for operations on set of objects?

Ans: Java provides a Collection API which provides many useful methods which can be applied on a set of objects. Some of the important classes provided by Collection API include ArrayList, HashMap, TreeSet and TreeMap.

Q97. Can we cast any other type to Boolean Type with type casting?

Ans: No, we can neither cast any other primitive type to Boolean data type nor can cast Boolean data type to any other primitive data type.

Q98. Can we use different return types for methods when overridden?

Ans: The basic requirement of method overriding in Java is that the overridden method should have same name, and parameters.But a method can be overridden with a different return type as long as the new return type extends the original.

For example , method is returning a reference type.

```
Class B extends A {

    A method(int x) {

        //original method

    }

    B method(int x) {

        //overridden method

    }

}
```

Q99. What's the base class of all exception classes?

Ans: In Java, **Java.lang.Throwable** is the super class of all exception classes and all exception classes are derived from this base class.

Q100. What's the order of call of constructors in inheritance?

Ans: In case of inheritance, when a new object of a derived class is created, first the constructor of the super class is invoked and then the constructor of the derived class is invoked.

Q #1) What is JAVA?

Answer: Java is a high-level programming language and is platform-independent. Java is a collection of objects. It was developed by Sun Microsystems. There are a lot of applications, websites, and games that are developed using Java.

Q #2) What are the features in JAVA?

Answer: Features of Java are as follows:

- **OOPs concepts**
  - Object-oriented
  - Inheritance
  - Encapsulation
  - Polymorphism
  - Abstraction
- **Platform independent:** A single program works on different platforms without any modification.
- **High Performance:** JIT (Just In Time compiler) enables high performance in Java. JIT converts the bytecode into machine language and then JVM starts the execution.
- **Multi-threaded:** A flow of execution is known as a Thread. JVM creates a thread which is called the main thread. The user can create multiple threads by extending the thread class or by implementing the Runnable interface.

Q #3) How does Java enable high performance?

Answer: Java uses Just In Time compiler to enable high performance. It is used to convert the instructions into bytecodes.

Q #4) Name the Java IDE's?

Answer: Eclipse and NetBeans are the IDE's of JAVA.

Q #5) What do you mean by Constructor?

Answer: Constructor can be explained in detail with enlisted points:

- When a new object is created in a program a constructor gets invoked corresponding to the class.
- The constructor is a method which has the same name as the class name.
- If a user doesn't create a constructor implicitly a default constructor will be created.
- The constructor can be overloaded.
- If the user created a constructor with a parameter then he should create another constructor explicitly without a parameter.

Q #6) What is meant by the Local variable and the Instance variable?

Answer:

Local variables are defined in the method and scope of the variables existed inside the method itself.

Instance variable is defined inside the class and outside the method and the scope of the variables exists throughout the class.

Q #7) What is a Class?

Answer: All Java codes are defined in a Class. It has variables and methods.

Variables are attributes which define the state of a class.

Methods are the place where the exact business logic has to be done. It contains a set of statements (or) instructions to satisfy the particular requirement.

Example:

```
public class Addition{ //Class name declaration
int a = 5; //Variable declaration
int b= 5;
```

```
public void add(){ //Method declaration
int c = a+b;
}
}
```

Q #8) What is an Object?

Answer: An instance of a class is called an object. The object has state and behavior. Whenever the JVM reads the "new()" keyword then it will create an instance of that class.

Example:

```
public class Addition{
public static void main(String[] args){
Addion add = new Addition();//Object creation
}
}
```

The above code creates the object for the Addition class.

Q #9)What are the OOPs concepts?

Answer: OOPs concepts include:

- Inheritance
- Encapsulation
- Polymorphism
- Abstraction
- Interface

Q #10) What is Inheritance?

Answer: Inheritance means one class can extend to another class. So that the codes can be reused from one class to another class. The existing class is known as the Super class whereas the derived class is known as a sub class.

Example:

```
Super class:
public class Manupulation(){
}
Sub class:
public class Addition extends Manipulation(){
}
```

Inheritance is applicable for public and protected members only. Private members can't be inherited.

Q #11) What is Encapsulation?

Answer: Purpose of Encapsulation:

- Protects the code from others.
- Code maintainability.

Example:

We are declaring 'a' as an integer variable and it should not be negative.

```
public class Addition(){
int a=5;
}
```

If someone changes the exact variable as "***a = -5"*** then it is bad.

In order to overcome the problem we need to follow the below steps:

- We can make the variable as private or protected one.
- Use public accessor methods such as set<property> and get<property>.

So that the above code can be modified as:

```
public class Addition(){
```

```
private int a = 5; //Here the variable is marked as private
}
```

The below code shows the getter and setter.
Conditions can be provided while setting the variable.

```
get A(){
}
set A(int a){
if(a>0){// Here condition is applied
.........
}
}
```

For encapsulation, we need to make all the instance variables as private and create setter and getter for those variables. Which in turn will force others to call the setters rather than access the data directly.

Q #12) What is Polymorphism?

Answer: Polymorphism means many forms.
A single object can refer to the super-class or sub-class depending on the reference type which is called polymorphism.

Example:

```
Public class Manipulation(){ //Super class
public void add(){
}
}
public class Addition extends Manipulation(){ // Sub class
public void add(){
}
public static void main(String args[]){
Manipulation addition = new Addition();//Manipulation is reference type and Addition is
addition.add();
}
}
```

Using Manipulation reference type we can call the Addition class “add()” method. This ability is known as Polymorphism. Polymorphism is applicable for **overriding** and not for **overloading**.

Q #13) What is meant by Method Overriding?

Answer: Method overriding happens if the sub-class method satisfies the below conditions with the Super-class method:

- Method name should be the same
- The argument should be the same
- Return type also should be the same

The key benefit of overriding is that the Sub-class can provide some specific information about that sub-class type than the super-class.

Example:

```
public class Manipulation{ //Super class
public void add(){
.....
}
}

Public class Addition extends Manipulation(){
Public void add(){
```

```
……..
}
Public static void main(String args[]){
Manipulation addition = new Addition(); //Polimorphism is applied
addition.add(); // It calls the Sub class add() method
}
}
```

addition.add() method calls the add() method in the Sub-class and not the parent class. So it overrides the Super-class method and is known as Method Overriding.

Q #14) What is meant by Overloading?

Answer: Method overloading happens for different classes or within the same class.

For method overloading, sub-class method should satisfy the below conditions with the Super-class method (or) methods in the same class itself:

- Same method name
- Different argument type
- May have different return types

Example:

```
public class Manipulation{ //Super class
public void add(String name){ //String parameter
……..
}
}

Public class Addition extends Manipulation(){
Public void add(){//No Parameter
……..
}
Public void add(int a){ //integer parameter

}
Public static void main(String args[]){
Addition addition = new Addition();
addition.add();
}
}
```

Here the add() method has different parameters in the Addition class is overloaded in the same class as well as with the super-class.

Note: Polymorphism is not applicable for method overloading.

Q #15) What is meant by Interface?

Answer: Multiple inheritances cannot be achieved in java. To overcome this problem Interface concept is introduced.

An interface is a template which has only method declarations and not the method implementation.

Example:

```
Public abstract interface IManupulation{ //Interface declaration
Public abstract void add();//method declaration
public abstract void subtract();
}
```

- All the methods in the interface are internally **public abstract void**.
- All the variables in the interface are internally **public static final** that is constants.
- Classes can implement the interface and not extends.
- The class which implements the interface should provide an implementation for all the methods declared in the interface.

```
public class Manupulation implements IManupulation{ //Manupulation class uses the interfa
Public void add(){
……
}
Public void subtract(){
…….
}
}
```

Q #16) What is meant by Abstract class?

Answer: We can create the Abstract class by using the "Abstract" keyword before the class name. An abstract class can have both "Abstract" methods and "Non-abstract" methods that are a concrete class.

Abstract method:

The method which has only the declaration and not the implementation is called the abstract method and it has the keyword called "abstract". Declarations are the ends with a semicolon.

Example:

```
public abstract class Manupulation{
public abstract void add();//Abstract method declaration
Public void subtract(){
}
}
```

- An abstract class may have a non- abstract method also.
- The concrete Subclass which extends the Abstract class should provide the implementation for abstract methods.

Q #17) Difference between Array and Array List.

Answer: The Difference between Array and Array List can be understood from the below table:

Array	Array List
Size should be given at the time of array declaration. String[] name = new String[2]	Size may not be required. It changes the size dynamic ArrayList name = new ArrayList
To put an object into array we need to specify the index. name[1] = "book"	No index required. name.add("book")
Array is not type parameterized	ArrayList in java 5.0 are parameterized. Eg: This angle bracket is a type parameter which mea String.

Q #18) Difference between String, String Builder, and String Buffer.

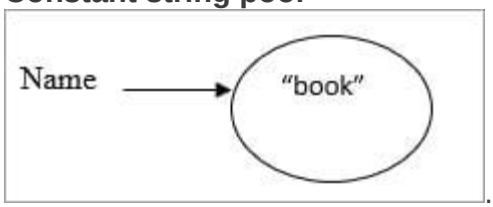
Answer:

String: String variables are stored in a "constant string pool". Once the string reference changes the old value that exists in the "constant string pool", it cannot be erased.

Example:

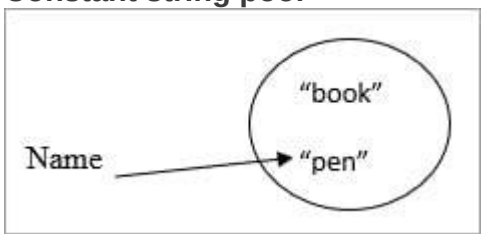
String name = "book";

Constant string pool



If the name-value has changed from “book” to “pen”.

Constant string pool



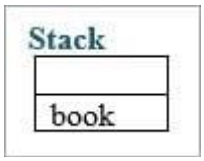
Then the older value retains in the constant string pool.

String Buffer:

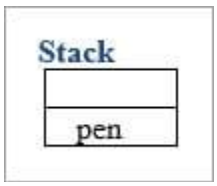
- Here string values are stored in a stack. If the values are changed then the new value replaces the older value.
- The string buffer is synchronized which is thread-safe.
- Performance is slower than the String Builder.

Example:

String Buffer name =”book”;



Once the name value has been changed to “pen” then the “book” is erased in the stack.



String Builder:

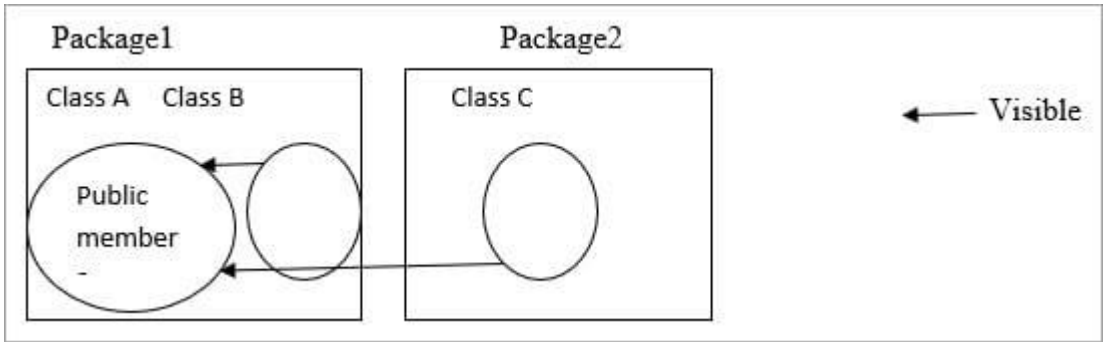
This is the same as String Buffer except for the String Builder which is not threaded safety that is not synchronized. So obviously performance is fast.

Q #19) Explain about Public and Private access specifiers.

Answer: Methods and instance variables are known as members.

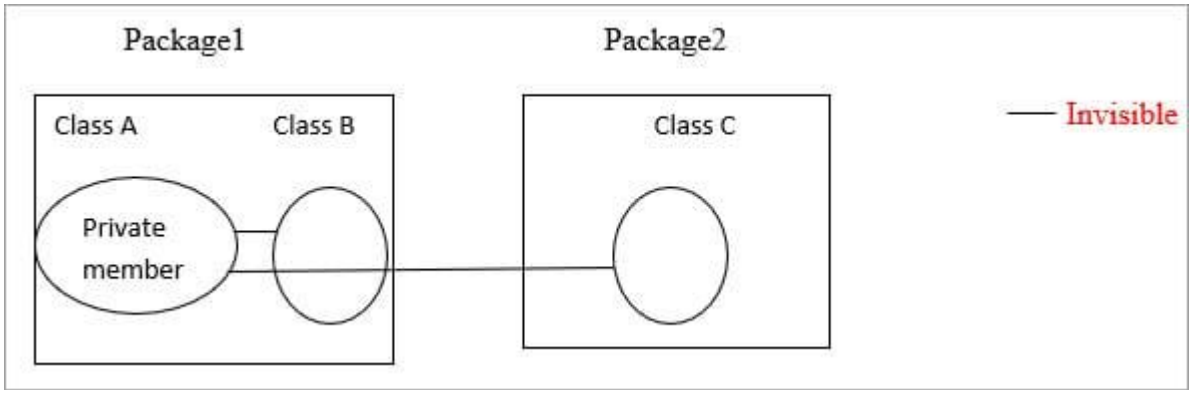
Public:

Public members are visible in the same package as well as the outside package that is for other packages.



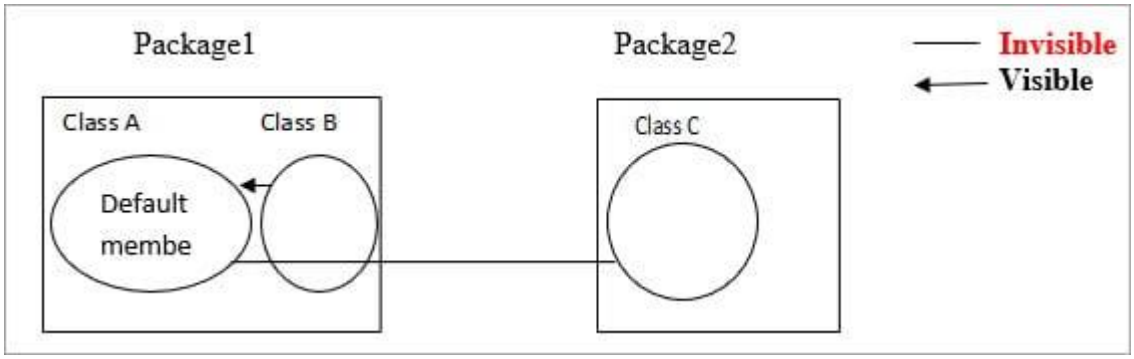
Public members in Class A are visible to Class B (same package) as well as Class C (different packages).

Private:
Private members are visible in the same class only and not for the other classes in the same package as well as classes in the outside packages.



Private members in class A are visible only in that class. It is invisible for class B as well as class C.

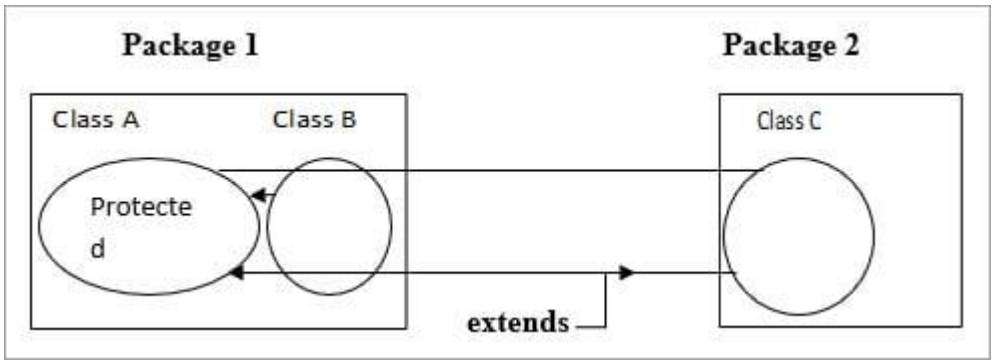
Q #20) Difference between Default and Protected access specifiers.
Answer:
Default: Methods and variables declared in a class without any access specifiers are called default.



Default members in Class A are visible to the other classes which are inside the package and invisible to the classes which are outside the package.

So Class A members are visible to the Class B and invisible to the Class C.

Protected:



Protected is the same as Default but if a class extends then it is visible even if it is outside the package.

Class A members are visible to Class B because it is inside the package. For Class C it is invisible but if Class C extends Class A then the members are visible to the Class C even if it is outside the package.

Q #21) Difference between HashMap and HashTable.

Answer: Difference between HashMap and HashTable can be seen below:

HashMap	HashTable
Methods are not synchronized	Key methods are synchronized
Not thread safety	Thread safety
Iterator is used to iterate the values	Enumerator is used to iterate the values
Allows one null key and multiple null values	Doesn't allow anything that is null
Performance is high than HashTable	Performance is slow

Q #22) Difference between HashSet and TreeSet.

Answer: Difference between HashSet and TreeSet can be seen below:

HashSet	TreeSet
Inserted elements are in random order	Maintains the elements in the sorted order
Can able to store null objects	Couldn't store null objects
Performance is fast	Performance is slow

Q #23) Difference between Abstract class and Interface.

Answer: Difference between Abstract Class and Interface are as follows:

Abstract Class:

- Abstract classes have a default constructor and it is called whenever the concrete subclass is instantiated.
- It contains Abstract methods as well as Non-Abstract methods.
- The class which extends the Abstract class shouldn't require the implementation of all the methods, only Abstract methods need to be implemented in the concrete sub-class.
- Abstract class contains instance variables.

Interface:

- It doesn't have any constructor and couldn't be instantiated.
- The abstract method alone should be declared.
- Classes that implement the interface should provide the implementation for all the methods.
- The interface contains only constants.

Q #24) What is mean by Collections in Java?

Answer: Collection is a framework that is designed to store the objects and manipulate the design to store the objects.

Collections are used to perform the following operations:

- Searching
- Sorting
- Manipulation
- Insertion
- Deletion

A group of objects is known as collections. All the classes and interfaces for collecting are available in Java util package.

Q #25) What are all the Classes and Interfaces that are available in the collections?

Answer: Given below are the Classes and Interfaces that are available in Collections:

Interfaces:

- Collection
- List
- Set
- Map
- Sorted Set
- Sorted Map
- Queue

Classes:

- Lists:
- Array List
- Vector
- Linked List

Sets:

- Hash set
- Linked Hash Set
- Tree Set

Maps:

- Hash Map
- Hash Table
- TreeMap
- Linked Hashed Map

Queue:

- Priority Queue

Q #26) What is meant by Ordered and Sorted in collections?

Answer:

Ordered: It means the values that are stored in a collection is based on the values that are added to the collection. So we can iterate the values from the collection in a specific order.

Sorted: Sorting mechanisms can be applied internally or externally so that the group of objects sorted in a particular collection is based on the properties of the objects.

Q #27) Explain the different lists available in the collection.

Answer: Values added to the list is based on the index position and it is ordered by index position. Duplicates are allowed.

Types of Lists are:

a) Array List:

- Fast iteration and fast Random Access.
- It is an ordered collection (by index) and not sorted.
- It implements the Random Access Interface.

Example:

```
public class Fruits{
public static void main (String [ ] args){
ArrayList <String>names=new ArrayList <String>();
names.add ("apple");
names.add ("cherry");
names.add ("kiwi");
names.add ("banana");
names.add ("cherry");
System.out.println (names);
}
}
```

Output:

[Apple, cherry, kiwi, banana, cherry]

From the output, Array List maintains the insertion order and it accepts the duplicates. But not sorted.

b) Vector:

It is the same as Array List.

- Vector methods are synchronized.
- Thread safety.
- It also implements the Random Access.
- Thread safety usually causes a performance hit.

Example:

```
public class Fruit {
public static void main (String [ ] args){
Vector <String> names = new Vector <String> ( );
 names.add ("cherry");
names.add ("apple");
names.add ("banana");
names.add ("kiwi");
names.add ("apple");
System.out.println ("names");
}
}
```

Output:

[cherry,apple,banana,kiwi,apple]

Vector also maintains the insertion order and accepts the duplicates.

c) Linked List:

- Elements are doubly linked to one another.
- Performance is slower than the Array list.
- Good choice for insertion and deletion.

- In Java 5.0 it supports common queue methods peek( ), Pool ( ), Offer ( ) etc.

Example:

```
public class Fruit {
public static void main (String [ ] args){
Linkedlist <String> names = new linkedlist <String> ( ) ;
names.add("banana");
names.add("cherry");
names.add("apple");
names.add("kiwi");
names.add("banana");
System.out.println (names);
}
}
```

Output:

[banana,cherry,apple,kiwi,banana]

Maintains the insertion order and accepts the duplicates.

Q #28) Explain about Set and their types in a collection?

Answer: Set cares about uniqueness. It doesn't allow duplications. Here "equals ( )" method is used to determine whether two objects are identical or not.

a) Hash Set:

- Unordered and unsorted.
- Uses the hash code of the object to insert the values.
- Use this when the requirement is "no duplicates and don't care about the order".

Example:

```
public class Fruit {
public static void main (String[ ] args){
HashSet<String> names = new HashSet <=String>( ) ;
names.add("banana");
names.add("cherry");
names.add("apple");
names.add("kiwi");
names.add("banana");
System.out.println (names);
}
}
```

Output:

[banana, cherry, kiwi, apple]

It doesn't follow any insertion order. Duplicates are not allowed.

b) Linked Hash set:

- An ordered version of the hash set is known as Linked Hash Set.
- Maintains a doubly-Linked list of all the elements.
- Use this when the iteration order is required.

Example:

```
public class Fruit {
public static void main (String[ ] args){
LinkedHashSet<String>; names = new LinkedHashSet <String>( ) ;
 names.add("banana");
 names.add("cherry");
 names.add("apple");
 names.add("kiwi");
 names.add("banana");
 System.out.println (names);
```

```
 }
}
```

Output:
[banana, cherry, apple, kiwi]

It maintains the insertion order in which they have been added to the Set. Duplicates are not allowed.

c) Tree Set:

- It is one of the two sorted collections.
- Uses the "Read-Black" tree structure and guarantee that the elements will be in ascending order.
- We can construct a tree set with the constructor by using a comparable (or) comparator.

Example:

```
public class Fruits{
public static void main (String[ ]args) {
Treeset<String> names= new TreeSet<String>( ) ;
names.add("cherry");
names.add("banana");
names.add("apple");
names.add("kiwi");
names.add("cherry");
System.out.println(names);
}
}
```

Output:
[apple, banana, cherry, kiwi]

TreeSet sorts the elements in ascending order. And duplicates are not allowed.

Q #29). Explain about Map and their types.

Answer: Map cares about the unique identifier. We can map a unique key to a specific value. It is a key/value pair. We can search a value, based on the key. Like the set, Map also uses the "equals ( )" method to determine whether two keys are the same or different.

Map is of following types:

a) Hash Map:

- Unordered and unsorted map.
- Hashmap is a good choice when we don't care about the order.
- It allows one null key and multiple null values.

Example:

```
Public class Fruit{
Public static void main(String[ ] args){
HashMap<Sting,String> names =new HashMap<String,String>( );
names.put("key1","cherry");
names.put ("key2","banana");
names.put ("key3","apple");
names.put ("key4","kiwi");
names.put ("key1","cherry");
System.out.println(names);
}
 }
```

Output:
{key2 =banana, key1=cherry, key4 =kiwi, key3= apple}

Duplicate keys are not allowed in Map.

It doesn't maintain any insertion order and is unsorted.

b) Hash Table:

- Like the vector key, methods of the class are synchronized.
- Thread safety and therefore slows the performance.
- It doesn't allow anything that is null.

Example:

```
public class Fruit{
public static void main(String[ ]args){
Hashtable<Sting,String> names =new Hashtable<String,String>( );
names.put("key1","cherry");
names.put("key2","apple");
names.put("key3","banana");
names.put("key4","kiwi");
names.put("key2","orange");
System.out.println(names);
}
 }
```

Output:

{key2=apple, key1=cherry,key4=kiwi, key3=banana}

Duplicate keys are not allowed.

c) Linked Hash Map:

- Maintains insertion order.
- Slower than Hash map.
- I can expect a faster iteration.

Example:

```
public class Fruit{
public static void main(String[ ] args){
LinkedHashMap<Sting,String> names =new LinkedHashMap<String,String>( );
 names.put("key1","cherry");
 names.put("key2","apple");
 names.put("key3","banana");
 names.put("key4","kiwi");
 names.put("key2","orange");
 System.out.println(names);
 }
 }
```

Output:

{key2=apple, key1=cherry,key4=kiwi, key3=banana}

Duplicate keys are not allowed.

d) TreeMap:

- Sorted Map.
- Like Tree set, we can construct a sort order with the constructor.

Example:

```
public class Fruit{
public static void main(String[ ]args){
TreeMap<Sting,String> names =new TreeMap<String,String>( );
names.put("key1","cherry");
names.put("key2","banana");
```

```
names.put("key3","apple");
names.put("key4","kiwi");
names.put("key2","orange");
System.out.println(names);
}
}
```

Output:

{key1=cherry, key2=banana, key3 =apple, key4=kiwi}

It is sorted in ascending order based on the key. Duplicate keys are not allowed.

Q #30) Explain the Priority Queue.

Answer: Queue Interface

Priority Queue: Linked list class has been enhanced to implement the queue interface. Queues can be handled with a linked list. The purpose of a queue is "Priority-in, Priority-out".

Hence elements are ordered either naturally or according to the comparator. The elements ordering represents their relative priority.

Q #31) What is meant by Exception?

Answer: An Exception is a problem that can occur during the normal flow of execution. A method can throw an exception when something wails at runtime. If that exception couldn't be handled, then the execution gets terminated before it completes the task. If we handled the exception, then the normal flow gets continued. Exceptions are a subclass of java.lang.Exception.

Example for handling Exception:

```
try{
//Risky codes are surrounded by this block
}catch(Exception e){
//Exceptions are caught in catch block
}
```

Q #32) What are the types of Exceptions?

Answer: There are two types of Exceptions. They are explained below in detail.

a) Checked Exception:

These exceptions are checked by the compiler at the time of compilation. Classes that extend Throwable class except Runtime exception and Error are called checked Exception.

Checked Exceptions must either declare the exception using throws keyword (or) surrounded by appropriate try/catch.

For Example, ClassNotFound Exception

b) Unchecked Exception:

These exceptions are not checked during the compile time by the compiler. The compiler doesn't force to handle these exceptions. **It includes:**

- Arithmetic Exception
- ArrayIndexOutOfBounds Exception

Q #33) What are the different ways to handle exceptions?

Answer: Two different ways to handle exception are explained below:

a) Using try/catch:

A risky code is surrounded by try block. If an exception occurs, then it is caught by the catch block which is followed by the try block.

Example:

```
class Manipulation{
public static void main(String[] args){
add();
}
Public void add(){
try{
addition();
}catch(Exception e){
e.printStacktrace();
}
}
}
```

b) By declaring throws keyword:

At the end of the method, we can declare the exception using throws keyword.

Example:

```
class Manipulation{
public static void main(String[] args){
add();
}
public void add() throws Exception{
addition();
}
}
```

Q #34) What are the advantages of Exception handling?

Answer: The advantages are as follows:

- The normal flow of the execution won't be terminated if exception got handled
- We can identify the problem by using catch declaration

Q #35) What are the Exception handling keywords in Java?

Answer: Enlisted below are the two Exception Handling Keywords:

a) try:

When a risky code is surrounded by a try block. An exception occurring in the try block is caught by a catch block. Try can be followed either by catch (or) finally (or) both. But any one of the blocks is mandatory.

b) catch:

This is followed by try block. Exceptions are caught here.

c) finally:

This is followed either by try block (or) catch block. This block gets executed regardless of an exception. So generally clean up codes are provided here.

Q #36) Explain about Exception Propagation.

Answer: Exception is first thrown from the method which is at the top of the stack. If it doesn't catch, then it pops up the method and moves to the previous method and so on until they are got.

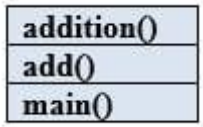
This is called Exception propagation.

Example:

```
public class Manipulation{
```

```
public static void main(String[] args){
add();
}
public void add(){
addition();
}
```

From the above example, the stack looks like as shown below:



If an exception occurred in the **addition()** method is not caught, then it moves to the method **add()**. Then it is moved to the **main()** method and then it will stop the flow of execution. It is called Exception Propagation.

Q #37) What is the final keyword in Java?

Answer:

Final variable: Once a variable is declared as final, then the value of the variable could not be changed. It is like a constant.

Example:

final int = 12;

Final method: A final keyword in a method, couldn't be overridden. If a method is marked as a final, then it can't be overridden by the subclass.

Final class: If a class is declared as final, then the class couldn't be subclassed. No class can extend the final class.

Q #38) What is a Thread?

Answer: In Java, the flow of execution is called Thread. Every java program has at least one thread called the main thread, the main thread is created by JVM. The user can define their own threads by extending the Thread class (or) by implementing the Runnable interface. Threads are executed concurrently.

Example:

```
public static void main(String[] args){//main thread starts here
}
```

Q #39) How do you make a thread in Java?

Answer: There are two ways available to make a thread.

a) Extend Thread class: Extending a Thread class and override the run method. The thread is available in java.lang.thread.

Example:

```
Public class Addition extends Thread {
public void run () {
}
}
```

The disadvantage of using a thread class is that we cannot extend any other classes because we have already extended the thread class. We can overload the run () method in our class.

b) Implement Runnable interface: Another way is by implementing the runnable interface. For that, we should provide the implementation for the run () method which is defined in the interface.

Example:

```
Public class Addition implements Runnable {
public void run () {
}
}
```

Q #40) Explain about join () method.

Answer: Join () method is used to join one thread with the end of the currently running thread.

Example:

```
public static void main (String[] args){
Thread t = new Thread ();
t.start ();
t.join ();
}
```

From the above code, the main thread has started the execution. When it reaches the code ***t.start()*** then 'thread t' starts the own stack for the execution. JVM switches between the main thread and 'thread t'.

Once it reaches the code ***t.join()*** then 'thread t' alone is executed and completes its task, then only the main thread started the execution.

It is a non-static method. The Join () method has an overloaded version. So we can mention the time duration in join () method also ".s".

Q #41) What does the yield method of the Thread class do?

Answer: A yield () method moves the currently running thread to a runnable state and allows the other threads for execution. So that equal priority threads have a chance to run. It is a static method. It doesn't release any lock.

Yield () method moves the thread back to the Runnable state only, and not the thread to sleep (), wait () (or) block.

Example:

```
public static void main (String[] args){
Thread t = new Thread ();
t.start ();
}
public void run(){
Thread.yield();
}
}
```

Q #42) Explain about wait () method.

Answer: wait () method is used to make the thread to wait in the waiting pool. When the wait () method is executed during a thread execution then immediately the thread gives up the lock on the object and goes to the waiting pool. Wait () method tells the thread to wait for a given amount of time.

Then the thread will wake up after notify () (or) notify all () method is called.

Wait() and the other above-mentioned methods do not give the lock on the object immediately until the currently executing thread completes the synchronized code. It is mostly used in synchronization.

Example:

```
public static void main (String[] args){
Thread t = new Thread ();
t.start ();
Synchronized (t) {
Wait();
}
}
```

Q #43) Difference between notify() method and notifyAll() method in Java.

Answer: The differences between notify() method and notifyAll() method are enlisted below:

notify()	notifyAll()
This method is used to send a signal to wake up a single thread in the waiting pool.	This method sends the signal to wake up all t in a waiting spool.

Q #44) How to stop a thread in java? Explain about sleep () method in a thread?

Answer: We can stop a thread by using the following thread methods:

- Sleeping
- Waiting
- Blocked

Sleep: Sleep () method is used to sleep the currently executing thread for the given amount of time. Once the thread is wake up it can move to the runnable state. So sleep () method is used to delay the execution for some period.
It is a static method.

Example:

Thread. Sleep (2000)

So it delays the thread to sleep 2 milliseconds. Sleep () method throws an uninterrupted exception, hence we need to surround the block with try/catch.

```
public class ExampleThread implements Runnable{
public static void main (String[] args){
Thread t = new Thread ();
t.start ();
}
public void run(){
try{
Thread.sleep(2000);
}catch(InterruptedException e){
}
}
```

Q #45) When to use the Runnable interface Vs Thread class in Java?

Answer: If we need our class to extend some other classes other than the thread then we can go with the runnable interface because in java we can extend only one class. If we are not going to extend any class then we can extend the thread class.

Q #46) Difference between start() and run() method of thread class.

Answer: Start() method creates a new thread and the code inside the run () method is executed in the new thread. If we directly called the run() method then a new thread is not created and the currently executing thread will continue to execute the run() method.

Q #47) What is Multi-threading?

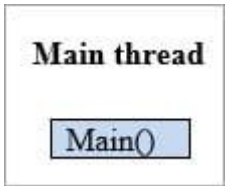
Answer: Multiple threads are executed simultaneously. Each thread starts its own stack based on the flow (or) priority of the threads.

Example Program:

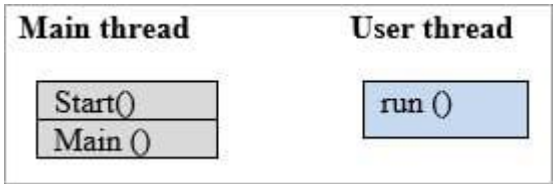
```
public class MultipleThreads implements Runnable
{
public static void main (String[] args){//Main thread starts here
Runnable r = new runnable ();
Thread t=new thread ();
t.start ();//User thread starts here
Addition add=new addition ();
}
```

```
public void run(){
go();
}//User thread ends here
}
```

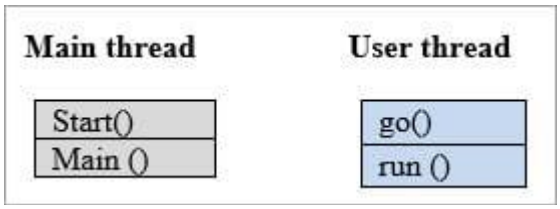
On the 1st line execution, JVM calls the main method and the main thread stack looks as shown below.



Once the execution reaches, **t.start ()** line then a new thread is created and the new stack for the thread is also created. Now JVM switches to the new thread and the main thread are back to the runnable state.
The two stacks look as shown below.



Now, the user thread executed the code inside the run() method.



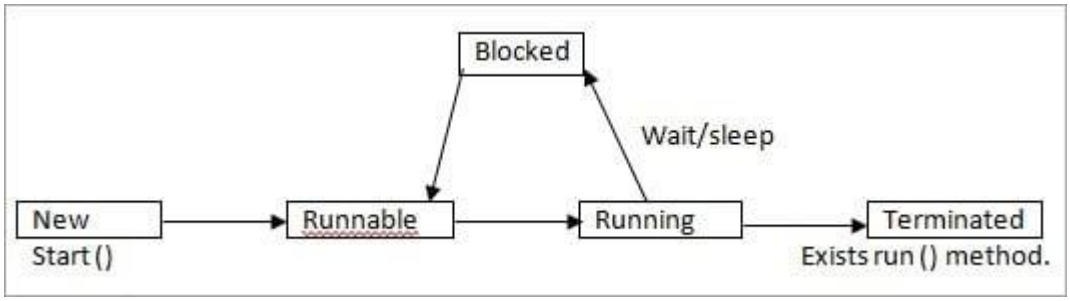
Once the run() method has completed, then JVM switches back to the main thread and the user thread has completed the task and the stack was disappeared.

JVM switches between each thread until both the threads are completed. This is called Multi-threading.

Q #48) Explain the thread life cycle in Java.

Answer: Thread has the following states:

- New
- Runnable
- Running
- Non-runnable (Blocked)
- Terminated



- **New:** In New state, a Thread instance has been created but start () method is not yet invoked. Now the thread is not considered alive.
- **Runnable**: The Thread is in the runnable state after the invocation of the start () method, but before the run () method is invoked. But a thread can also return to the runnable state from waiting/sleeping. In this state, the thread is considered alive.
- **Running**: The thread is in running state after it calls the run () method. Now the thread begins the execution.
- **Non-Runnable**(Blocked): The thread is alive but it is not eligible to run. It is not in the runnable state but also, it will return to the runnable state after some time. **Example:** wait, sleep, block.
- **Terminated**: Once the run method is completed then it is terminated. Now the thread is not alive.

Q #49) What is Synchronization?

Answer: Synchronization makes only one thread to access a block of code at a time. If multiple threads accesses the block of code, then there is a chance for inaccurate results at the end. To avoid this issue, we can provide synchronization for the sensitive block of codes.
The synchronized keyword means that a thread needs a key in order to access the synchronized code.

Locks are per objects. Every Java object has a lock. A lock has only one key. A thread can access a synchronized method only if the thread can get the key to the objects to lock.

For this, we use the “Synchronized” keyword.

Example:

```
public class ExampleThread implements Runnable{
 public static void main (String[] args){
 Thread t = new Thread ();
 t.start ();
 }
 public void run(){
 synchronized(object){
 {
 }
}
```

Q #50) What is the disadvantage of Synchronization?

Ans: Synchronization is not recommended to implement all the methods. Because if one thread accesses the synchronized code then the next thread should have to wait. So it makes a slow performance on the other end.

Q #51) What is meant by Serialization?

Answer: Converting a file into a byte stream is known as Serialization. The objects in the file are converted to the bytes for security purposes. For this, we need to implement a java.io.Serializable interface. It has no method to define.
Variables that are marked as transient will not be a part of the serialization. So we can skip the serialization for the variables in the file by using a transient keyword.

Q #52) What is the purpose of a transient variable?

Answer: Transient variables are not part of the serialization process. During deserialization, the values of the transient variables are set to the default value. It is not used with static variables.

Example:
transient int numbers;

Q #53) Which methods are used during the Serialization and Deserialization process?

Answer: ObjectOutputStream and ObjectInputStream classes are higher level java.io. package. We will use them with lower level classes FileOutputStream and FileInputStream.
ObjectOutputStream.writeObject **—->**Serialize the object and write the serialized object to a file.
ObjectInputStream.readObject **—>** Reads the file and deserializes the object.
To be serialized, an object must implement the serializable interface. If superclass implements Serializable, then the subclass will automatically be serializable.

Q #54) What is the purpose of a Volatile Variable?

Answer: Volatile variable values are always read from the main memory and not from thread's cache memory. This is used mainly during synchronization. It is applicable only for variables.
Example:
volatile int number;

Q #55) Difference between Serialization and Deserialization in Java.

Answer: These are the difference between serialization and deserialization in java:

Serialization	Deserialization
Serialization is the process which is used to convert the objects into byte stream	Deserialization is the opposite process of serialization v can get the objects back from the byte stream.
An object is serialized by writing it an ObjectOutputStream.	An object is deserialized by reading it from an ObjectInputStream.

Q #56) What is SerialVersionUID?

Answer: Whenever an object is Serialized, the object is stamped with a version ID number for the object class. This ID is called the SerialVersionUID. This is used during deserialization to verify that the sender and receiver that are compatible with the Serialization.

Q3. Why Java is platform independent?

Java is called platform independent because of its byte codes which can run on any system irrespective of its underlying operating system.

Q4. Why Java is not 100% Object-oriented?

Java is not 100% Object-oriented because it makes use of eight primitive data types such as boolean, byte, char, int, float, double, long, short which are not objects.

Q5. What are wrapper classes in Java?

Wrapper classes convert the Java primitives into the reference types (objects). Every primitive data type has a class dedicated to it. These are known as wrapper classes because they “wrap” the primitive data type into an object of that class. Refer to the below image which displays different primitive type, wrapper class and constructor argument.

Q6. What are constructors in Java?

In Java, constructor refers to a block of code which is used to initialize an object. It must have the same name as that of the class. Also, it has no return type and it is automatically called when an object is created.

There are two types of constructors:

1. **Default Constructor:** In Java, a default constructor is the one which does not take any inputs. In other words, default constructors are the no argument constructors which will be created by default in case you no other constructor is defined by the user. Its main purpose is to initialize the instance variables with the default values. Also, it is majorly used for object creation.
2. **Parameterized Constructor:** The parameterized constructor in Java, is the constructor which is capable of initializing the instance variables with the provided values. In other words, the constructors which take the arguments are called parameterized constructors.

II. Long Answer Questions:

Q #1) Write a Java Program to reverse a string without using String inbuilt function.

Answer: Here, we are initializing a string variable str and are making use of the string builder class.
The object of the string builder class str2 will be further used to append the value stored in the string variable str.
Thereafter, we are using the inbuilt function of string builder (reverse()) and storing the new reversed string in str2.

Finally, we are printing str2.

```
public class FinalReverseWithoutUsingStringMethods {

     public static void main(String[] args) {
          // TODO Auto-generated method stub
          String str = "Automation";
          StringBuilder str2 = new StringBuilder();
          str2.append(str);
          str2 = str2.reverse();      // used string builder to reverse
         System.out.println(str2);
          }

}
```

Output:
noitamotuA

Q #2) Write a Java Program to reverse a string without using String inbuilt function reverse().

Answer:

Method 1:

There are several ways with which you can reverse your string if you are allowed to use the other string inbuilt functions.

In this method, we are initializing a string variable called str with the value of your given string. Then, we are converting that string into character array with toCharArray() function. Thereafter, we are using for loop to iterate between each character in reverse order and printing each character.

```
public class FinalReverseWithoutUsingInbuiltFunction {
    public static void main(String[] args) {
        String str = "Saket Saurav";
        char chars[] = str.toCharArray();  // converted to character array and printed i
        for(int i= chars.length-1; i>=0; i--) {
            System.out.print(chars[i]);
        }
    }
}
```

Output:
varuaS tekaS

Method 2:
This is another method in which you are declaring your string variable str and then using Scanner class to declare an object with predefined standard input object.

This program will accept the string value through the command line (when executed).

We have used nextLine() which will read the input with the spaces between the words of a string. Thereafter, we have used a split() method to split the string into its substrings(no delimiter given here). Finally, we have printed the string in reverse order using for loop.

```
import java.util.Scanner;

public class ReverseSplit {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        String str;
        Scanner in = new Scanner(System.in);
        System.out.println("Enter your String");
        str = in.nextLine();
        String[] token = str.split("");    //used split method to print in reverse orde
        for(int i=token.length-1; i>=0; i--)
        {
            System.out.print(token[i] + "");
        }

    }

}
```

Output:
Enter your String
Softwaretestinghelp
plehgnitseterawtfoS

Method 3:
This is almost like method 2, but here we did not use the split() method. We have used the scanner class and nextLine() for reading the input string. Then, we have declared an integer length which has the length of the input string.

Thereafter, we have printed the string in the reverse order using for loop. However, we have used charAt(index) method which will return the character at any specific index. After each iteration, the character will be concatenated to reverse the string variable.

Finally, we have printed the reverse string variable.

```
import java.util.Scanner;

public class Reverse {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        String original, reverse = "";
```

```
        System.out.println("Enter the string to be reversed");
        Scanner in = new Scanner(System.in);
        original = in.nextLine();
        int length = original.length();
        for(int i=length-1; i>=0; i--) {
            reverse = reverse + original.charAt(i);    //used inbuilt method charAt() to
        }
        System.out.println(reverse);
    }

}
```

Output:

Enter the string to be reversed
automation testing
gnitset noitamotua

Q #3) Write a Java Program to swap two numbers with using the third variable.

Answer: In this Example, we have made use of the Scanner class to declare an object with a predefined standard input object. This program will accept the values of x and y through the command line (when executed).
We have used nextInt() which will input the value of an integer variable ‘x' and ‘y' from the user. A temp variable is also declared.

Now, the logic of the program goes like this – we are assigning temp or third variable with the value of x, and then we are assigning x with the value of y and again we are assigning y with the value of temp. So, after the first complete iteration, the temp will have a value of x, x will have a value of y and y will have a value of temp (which is x).

```
import java.util.Scanner;

public class SwapTwoNumbers {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int x, y, temp;
        System.out.println("Enter x and y");
        Scanner in = new Scanner(System.in);
        x = in.nextInt();
        y = in.nextInt();
        System.out.println("Before Swapping" + x + y);
        temp = x;
        x = y;
        y = temp;
        System.out.println("After Swapping" + x + y);

    }

}
```

Output:

Enter x and y
45
98
Before Swapping4598
After Swapping9845

Q #4) Write a Java Program to swap two numbers without using the third variable.

Answer: Rest all things will be the same as the above program. Only the logic will change. Here, we are assigning x with the value x + y which means x will have a sum of both x and y.
Then, we are assigning y with the value x – y which means we are subtracting the value of y from the sum of (x + y). Till here, x still has the sum of both x and y. But y has the value of x.

Finally, in the third step, we are assigning x with the value x – y which means we are subtracting y (which has the value of x) from the total (x + y). This will assign x with the value of y and vice versa.

```
import java.util.Scanner;

class SwapTwoNumberWithoutThirdVariable
{
   public static void main(String args[])
   {
      int x, y;
      System.out.println("Enter x and y");
      Scanner in = new Scanner(System.in);

      x = in.nextInt();
      y = in.nextInt();

      System.out.println("Before Swapping\nx = "+x+"\ny = "+y);

      x = x + y;
      y = x - y;
      x = x - y;

      System.out.println("After Swapping without third variable\nx = "+x+"\ny = "+y);
   }
}
```

Output:
Enter x and y
45
98
Before Swapping
x = 45
y = 98
After Swapping without a third variable
x = 98
y = 45

Q #5) Write a Java Program to count the number of words in a string using HashMap.

Answer: This is a collection class program where we have used HashMap for storing the string.
First of all, we have declared our string variable called str. Then we have used split() function delimited by single space so that we can split multiple words in a string.

Thereafter, we have declared HashMap and iterated using for loop. Inside for loop, we have an if else statement in which wherever at a particular position, the map contains a key, we set the counter at that position and add the object to the map.

Each time, the counter is incremented by 1. Else, the counter is set to 1.

Finally, we are printing the HashMap.

Note: The same program can be used to count the number of characters in a string. All you need to do is to remove one space (remove space delimited in split method) in String[] split = str.split("");

```
import java.util.HashMap;

public class FinalCountWords {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        String str = "This this is is done by Saket Saket";
        String[] split = str.split(" ");
        HashMap<String,Integer> map = new HashMap<String,Integer>();
        for (int i=0; i<split.length-1; i++) {
            if (map.containsKey(split[i])) {
                int count = map.get(split[i]);
                map.put(split[i], count+1);
            }
            else {
                map.put(split[i], 1);
            }
        }
        System.out.println(map);
    }

}
```

Output:

{Saket=1, by=1, this=1, This=1, is=2, done=1}

Q #6) Write a Java Program to iterate HashMap using While and advance for loop.

Answer: Here we have inserted three elements in HashMap using put() function. The size of the map can get using size() method. Thereafter, we have used While loop for iterating through the map which contains one key-value pair for each element. Keys and Values can be retrieved through getKey() and getValue().

Likewise, we have used advanced for loop where we have "me2" object for the HashMap.

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

public class HashMapIteration {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        HashMap<Integer,String> map = new HashMap<Integer,String>();
```

```
        map.put(2, "Saket");
        map.put(25, "Saurav");
        map.put(12, "HashMap");
        System.out.println(map.size());
        System.out.println("While Loop:");
        Iterator itr = map.entrySet().iterator();
        while(itr.hasNext()) {
            Map.Entry me = (Map.Entry) itr.next();
            System.out.println("Key is " + me.getKey() + " Value is " + me.getValue());
        }
        System.out.println("For Loop:");
        for(Map.Entry me2: map.entrySet()) {
            System.out.println("Key is: " + me2.getKey() + " Value is: " + me2.getValue(
        }
    }

}
```

Output:

3
While Loop:
Key is 2 Value is Saket
Key is 25 Value is Saurav
Key is 12 Value is HashMap
For Loop:
Key is: 2 Value is: Saket
Key is: 25 Value is: Saurav
Key is: 12 Value is: HashMap

Q #7) Write a Java Program to find whether a number is prime or not.

Answer: Here, we have declared two integers temp and num and used Scanner class with nextInt(as we have integer only).
One boolean variable isPrime is set to true. Thereafter, we have used for loop starting from 2, less than half of the number are entered and incremented by 1 for each iteration. Temp will have remainder for every iteration. If the remainder is 0, then isPrime will be set to False.

Based on isPrime value, we are coming to the conclusion that whether our number is prime or not.

```
import java.util.Scanner;

public class Prime {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int temp, num;
        boolean isPrime = true;
        Scanner in = new Scanner(System.in);
        num = in.nextInt();
        in.close();
        for (int i = 2; i<= num/2; i++) {
            temp = num%i;
            if (temp == 0) {
                isPrime = false;
                break;
            }
```

```
        }
        if(isPrime)
            System.out.println(num + "number is prime");
            else
                System.out.println(num + "number is not a prime");


    }

}
```

Output:
445
445number is not a prime

Q #8) Write a Java Program to find whether a string or number is palindrome or not.

Answer: You can use any of the reverse string program explained above to check whether the number or string is palindrome or not.
What you need to do is to include one if-else statement. If the original string is equal to a reversed string then the number is a palindrome, otherwise not.

```
import java.util.Scanner;

public class Palindrome {
    public static void main (String[] args) {
        String original, reverse = "";
        Scanner in = new Scanner(System.in);
        int length;
        System.out.println("Enter the number or String");
        original = in.nextLine();
        length = original.length();
        for (int i =length -1; i>=0; i--) {
            reverse = reverse + original.charAt(i);
        }
        System.out.println("reverse is:" +reverse);

        if(original.equals(reverse))
            System.out.println("The number is palindrome");
        else
            System.out.println("The number is not a palindrome");

    }
}
```

Output:
For String-
Enter the number or String
vijay
reverse is:yajiv
The number is not a palindrome

For Number-
Enter the number or String
99
reverse is:99
The number is palindrome

Q #9) Write a Java Program for Fibonacci series.

Answer: Fibonacci series is a series of numbers where after the initial two numbers, every occurring number is the sum of two preceding numbers.
For Example 0,1,1,2,3,5,8,13,21……
In this program, we have used Scanner class again with nextInt (discussed above). Initially, we are entering (through command line) the number of times the Fibonacci has to iterate. We have declared integer num and initialized a,b with zero and c with one. Then, we have used for loop to iterate.

The logic goes like a is set with the value of b which is 0, then b is set with the value of c which is 1. Then, c is set with the sum of both a and b.

```
import java.util.Scanner;

public class Fibonacci {
    public static void main(String[] args) {
        int num, a = 0,b=0, c =1;
        Scanner in = new Scanner(System.in);
        System.out.println("Enter the number of times");
        num = in.nextInt();
        System.out.println("Fibonacci Series of the number is:");
        for (int i=0; i<=num; i++) {
            a = b;
            b = c;
            c = a+b;
            System.out.println(a + "");    //if you want to print on the same line, use
        }
    }
}
```

Output:

```
Enter the number of times
9
Fibonacci Series of the number is:
0
1
1
2
3
5
8
13
21
34
```

Q #10) Write a Java Program to iterate ArrayList using for-loop, while-loop, and advance for-loop.

Answer: In this program, we have inserted three elements and printed the size of the ArrayList.
Then, we have used While Loop with an iterator. Whenever the iterator has (next) element, it will display that element until we reach the end of the list. So it will iterate three times.

Likewise, we have done for Advanced For Loop where we have created an object called obj for the ArrayList called list. Then printed the object.

Thereafter, we have put the condition of For Loop where the iterator i is set to 0 index, then it is incremented by 1 until the ArrayList limit or size is reached. Finally, we have printed each element using a get(index) method for each iteration of For Loop.

```
import java.util.*;

public class arrayList {
    public static void main(String[] args) {
        ArrayList list = new ArrayList();
        list.add("20");
        list.add("30");
        list.add("40");
        System.out.println(list.size());
        System.out.println("While Loop:");
        Iterator itr = list.iterator();
        while(itr.hasNext()) {
            System.out.println(itr.next());
        }
        System.out.println("Advanced For Loop:");
        for(Object obj : list) {
            System.out.println(obj);
    }
        System.out.println("For Loop:");
        for(int i=0; i<list.size(); i++) {
            System.out.println(list.get(i));
        }
}
}
```

Output:

3
While Loop:
20
30
40
Advanced For Loop:
20
30
40
For Loop:
20
30
40

Q #11) Write a Java Program to demonstrate explicit wait condition check.

Answer: There are two main types of wait – implicit and explicit. (We are not considering Fluent wait in this program)

Implicit wait is those waits that are executed irrespective of any condition. In the below program, you can see that it is for Google Chrome and we have used some inbuilt methods to set the property, maximizing window, URL navigation, and web element locating.

```
WebDriverWait wait = new WebDriverWait(driver, 20);
WebElement element2 = wait.until(ExpectedConditions.visibilityOfElementLocated(By.parti
element2.click();
```

In the above piece of code, you can see that we have created an object wait for WebDriverWait and then we have searched for WebElement called element2.

The condition is set in such a way that webdriver will have to wait until we see the link "Software testing – Wikipedia" on a web page. It won't execute if it does not find this link. If it does, then it will do a mouse click on that link.

```
package Codes;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

public class explicitWaitConditionCheck {

            public static void main(String[] args) {
                     // TODO Auto-generated method stub
                     System.setProperty("webdriver.chrome.driver", "C:\\webdriver\\chrom
                    ChromeOptions options = new ChromeOptions();
                    options.addArguments("--disable-arguments");
                    WebDriver driver = new ChromeDriver();
                    driver.manage().window().maximize();
                    driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);
                    driver.navigate().to("https://www.google.com");
                    WebElement element = driver.findElement(By.name("q"));
                     element.sendKeys("Testing");
                     element.submit();
                     WebDriverWait wait = new WebDriverWait(driver, 20);

                      WebElement element2 = wait.until(ExpectedConditions.visibilityOfEl
element2.click();
}}
```

Q #12) Write a Java Program to demonstrate Scroll up/ Scroll down.

Answer: All the lines of codes are easily relatable as we have discussed in our previous example.
However, in this program, we have included our JavascriptExecutor js which will do the scrolling. If you see the last line of the code, we have passed window.scrollBy(arg1,arg2).

If we want to scroll up then pass some value in arg1 if you want to scroll down then pass some value in arg2.

```
package Codes;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
```

```
public class ScrollDown {

          public static void main(String[] args) {
                   // TODO Auto-generated method stub
                   System.setProperty("webdriver.chrome.driver", "C:\\webdriver\\chrome
                   WebDriver driver = new ChromeDriver();
                   JavascriptExecutor js = (JavascriptExecutor) driver;
                   driver.manage().window().maximize();
                   driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);
                   driver.get("https://www.google.com");
                   WebElement element = driver.findElement(By.name("q"));
                   element.sendKeys("SoftwareTestingHelp");
                   element.sendKeys(Keys.ENTER);
                    js.executeScript("window.scrollBy(0,1000)");

}

}
```

Q #13) Write a Java Program to open all links of gmail.com.

Answer: It is a typical example of advanced for loop which we have seen in our previous programs.
Once you have opened a website such as Gmail using get() or navigate().to(), you can use a tagName locator to find the tag name of a website which will return all the tags.

We have advanced for loop where we have created a new WebElement link2 for a link(which already has located all the tags), then we have got all the links through getAttribute("href") and got all the texts through getText().

```
package Codes;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class openAllLinks {

         public static void main(String[] args) {
         // TODO Auto-generated method stub
         System.setProperty("webdriver.chrome.drive", "C:\\webdriver\\chromedriver.exe")
         WebDriver driver = new ChromeDriver();
         driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);
         driver.manage().window().maximize();
         driver.get("https://www.gmail.com/");
         java.util.List<WebElement> link = driver.findElements(By.tagName("a"));
        System.out.println(link.size());

        for (WebElement link2: link) {

        //print the links i.e. http://google.com or https://www.gmail.com
       System.out.println(link2.getAttribute("href"));

       //print the links text
```

```
     System.out.println(link2.getText());
}
}
}
```

Output:

Starting ChromeDriver 2.38.551601 (edb21f07fc70e9027c746edd3201443e011a61ed) on port 16163
Only local connections are allowed.
4
https://support.google.com/chrome/answer/6130773?hl=en-GB
Learn more
https://support.google.com/accounts?hl=en-GB
Help
https://accounts.google.com/TOS?loc=IN&hl=en-GB&privacy=true
Privacy
https://accounts.google.com/TOS?loc=IN&hl=en-GB
Terms

Q #14) Write a Selenium code to switch to the previous tab.

Answer: We have demonstrated the use of the Robot class. We see this as an important third party because we can achieve the different navigation within a browser and its tabs if you know the shortcut keys.

For example, if you have three tabs open in your chrome and you want to go to the middle tab, then you have to press control + 2 from your keyboard. The same thing can be achieved through the code as well.

Observe the following code (just after we see the instantiation of Robot class). we have used Robot class object called a robot with two inbuilt methods keyPress(KeyEvenet.VK_*) and keyRelease(KeyEvenet.VK_*).

```
package Codes;

import java.awt.AWTException;
import java.awt.Robot;
import java.awt.event.KeyEvent;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
public class PreviousTab {
          public static void main(String[] args) throws AWTException {
               // TODO Auto-generated method stub
              System.setProperty("webdriver.chrome.driver", "C:\\webdriver\\chromedrive
             WebDriver driver = new ChromeDriver();
             driver.manage().window().maximize();
             driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);
             driver.get("https://www.google.com");
             WebElement element1 = driver.findElement(By.name("q"));
             element1.sendKeys("software testing help");
             element1.sendKeys(Keys.ENTER);
             String a = Keys.chord(Keys.CONTROL,Keys.RETURN);
             driver.findElement(By.partialLinkText("Software Testing Help - A Must Visi
             Robot robot = new Robot(); // instantiated robot class
             robot.keyPress(KeyEvent.VK_CONTROL); // with robot class you can easily ac
             robot.keyPress(KeyEvent.VK_2); // here, we have just pressed ctrl+2
```

```
            robot.keyRelease(KeyEvent.VK_CONTROL); // once we press and release ctrl+2
            robot.keyRelease(KeyEvent.VK_2); //if you again want to go back to first t
            }
}
```

Q #15) Write a Java Program to find the duplicate characters in a string.

Answer: In this program, we have created a string variable str and initialized an integer count with zero.
Then, we have created a character array to convert our string variable to the character. With the help of for loop, we are performing a comparison between different characters at different indexes.

If two characters of consecutive index match, then it will print that character and the counter will be incremented by 1 after each iteration.

```
public class DuplicateCharacters {

        public static void main(String[] args) {
                // TODO Auto-generated method stub
               String str = new String("Sakkett");
               int count = 0;
               char[] chars = str.toCharArray();
               System.out.println("Duplicate characters are:");
               for (int i=0; i<str.length();i++) {
                           for(int j=i+1; j<str.length();j++) {
                                       if (chars[i] == chars[j]) {
                                                   System.out.println(chars[j]);
                                                   count++;
                                                   break;
                                        }
                            }
                }
         }

}
```

Output:
Duplicate characters are:
k
t

Q #16) Write a Java Program to find the second highest number in an array.

Answer: In this program, we have initialized an array with 10 random elements out of which we are going to find the second highest number. Here, we have two integers- the largest and second largest. Both set to the first index of the element. Then, we have printed all the elements using for loop.
Now the logic is when the element at the 0th index is greater than the largest, then assign arr[0] to largest and secondLargest to largest. Again, if the element at the 0th index is greater than the secondLargest, then assign secondLargest to arr[0].

This will be repeated for each iteration and ultimately after comparing or completing iterations up to array length will give you the secondLargest element.

```
package codes;
public class SecondHighestNumberInArray {
public static void main(String[] args)
    {
```

```
        int arr[] = { 100,14, 46, 47, 94, 94, 52, 86, 36, 94, 89 };
        int largest = 0;
        int secondLargest = 0;
        System.out.println("The given array is:");
        for (int i = 0; i < arr.length; i++)
        {
            System.out.print(arr[i] + "\t");
        }
        for (int i = 0; i < arr.length; i++)
        {
            if (arr[i] > largest)
            {
                secondLargest = largest;
                largest = arr[i];
            }
            else if (arr[i] > secondLargest)
            {
                secondLargest = arr[i];
            }
        }
        System.out.println("\nSecond largest number is:" + secondLargest);
        System.out.println("Largest Number is: " +largest);
    }
}
```

Output:

The given array is:
100 14 46 47 94 94 52 86 36 94 89
Second largest number is:94
Largest Number is: 100

Q #17) Write a Java Program to check Armstrong number.

Answer: First of all we need to understand what Armstrong Number is. Armstrong number is the number which is the sum of the cubes of all its unit, tens and hundred digits for three-digit number.
153 = 1*1*1 + 5*5*5 + 3*3*3 = 1 + 125 + 27 = 153

If you have a four-digit number lets say

1634 = 1*1*1*1 + 6*6*6*6 + 3*3*3*3 + 4*4*4*4 = 1 + 1296 + 81 + 256 = 1634

Now, in this program, we have a temp and integers declared. We have initialized c with value 0. Then, we need to assign the integer value which we are going to check for Armstrong (in our case, let us say 153). Then we have assigned our temp variable with that number which we are going to check.

Thereafter, we have used while conditional check where the remainder is assigned to a and the number is divided by 10 and assigned to n. Now, our c variable which was set to zero initially is assigned with c+(a*a*a). Suppose we have to evaluate a four-digit number then c should be assigned with c + (a*a*a*a).

Lastly, we have put an if-else statement for conditional checking where we have compared the value contained in c against temp(which has the actual number stored at this point). If it matches, then the number is Armstrong otherwise not.

```
class Armstrong{
 public static void main(String[] args)  {
   int c=0,a,temp;
   int n=153;//It is the number to check Armstrong
   temp=n;
   while(n>0)
   {
   a=n%10;
   n=n/10;
    c=c+(a*a*a);
    }
    if(temp==c)
    System.out.println("armstrong number");
    else
        System.out.println("Not armstrong number");
   }
}
```

Output:

armstrong number

Q #18) Write a Java Program to remove all white spaces from a string with using replace().

Answer: This is a simple program where we have our string variable str1.
Another string variable str2 is initialized with the replaceAll option which is an inbuilt method to remove n number of whitespaces. Ultimately, we have printed str2 which has no whitespaces.

```
class RemoveWhiteSpaces
{
    public static void main(String[] args)
    {
        String str1 = "Saket Saurav        is a QualityAna    list";

        //1. Using replaceAll() Method

        String str2 = str1.replaceAll("\\s", "");

        System.out.println(str2);

           }
}
}
```

Output:

SaketSauravisaQualityAnalist

Q #19) Write a Java Program to remove all white spaces from a string without using replace().

Answer: This is another approach to removing all white spaces. Again, we have one string variable str1 with some value. Then, we have converted that string into a character array using toCharArray().
Then, we have one StringBuffer object sb which will be used to append the value stored at chars[i] index after we have included for loop and one if condition.

If the condition is set such that then the element at i index of the character array should not be equal to space or tab. Finally, we have printed our StringBuffer object sb.

```
class RemoveWhiteSpaces
{
    public static void main(String[] args)
    {
        String str1 = "Saket Saurav        is an Autom ation Engi ne      er";

        char[] chars = str1.toCharArray();

        StringBuffer sb = new StringBuffer();

        for (int i = 0; i < chars.length; i++)
        {
            if( (chars[i] != ' ') && (chars[i] != '\t') )
            {
                sb.append(chars[i]);
            }
        }
        System.out.println(sb);            //Output : CoreJavajspservletsjdbcstrutshiber
    }
}
```

Output:
SaketSauravisanAutomationEngineer

20. Explain JDK, JRE and JVM?

JDK vs JRE vs JVM		
JDK	**JRE**	**JVM**
It stands for Java Development Kit.	It stands for Java Runtime Environment.	It stands for Java Virtual Machine.
It is the tool necessary to compile, document and package Java programs.	JRE refers to a runtime environment in which Java bytecode can be executed.	It is an abstract machine. It is a specification that provides a run-time environment in which Java bytecode can be executed.
It contains JRE + development tools.	It's an implementation of the JVM which physically exists.	JVM follows three notations: Specification, **Implementation,** and **Runtime Instance**.

21. Explain public static void main(String args[]) in Java.

main() in Java is the entry point for any Java program. It is always written as **public static void main(String[] args)**.

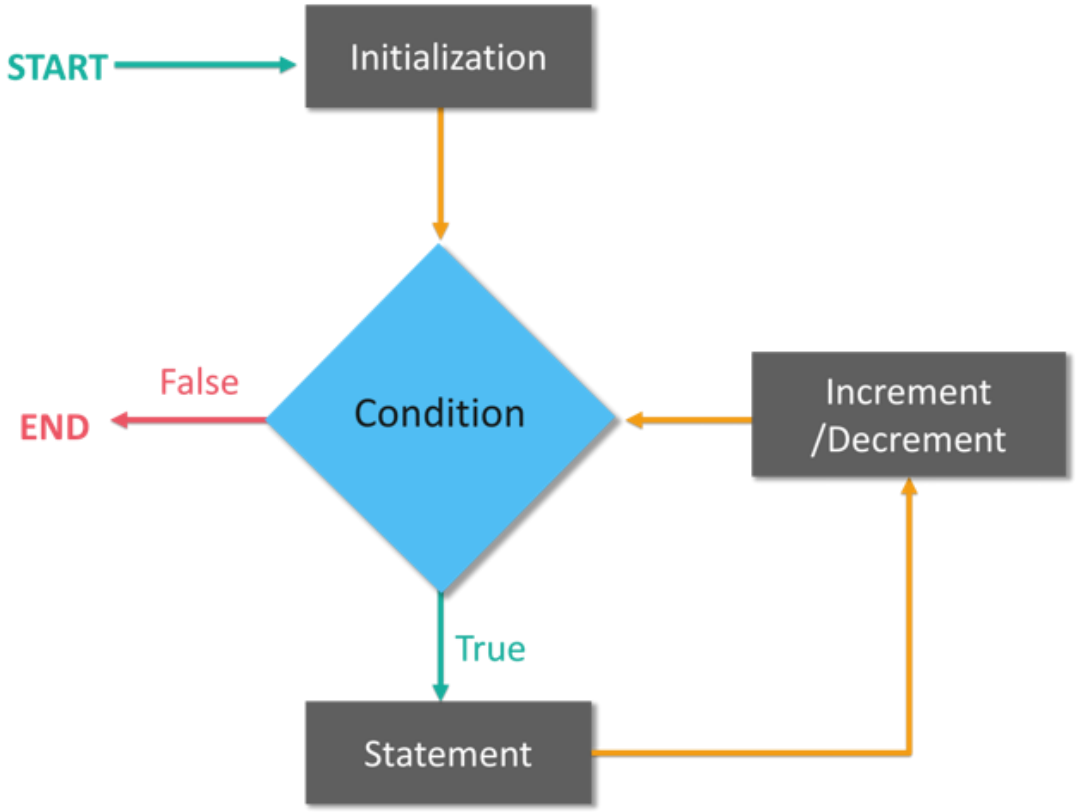
- **public**: Public is an access modifier, which is used to specify who can access this method. Public means that this Method will be accessible by any Class.
- **static**: It is a keyword in java which identifies it is class-based. main() is made static in Java so that it can be accessed without creating the instance of a Class. In case, main is not made static then the compiler will throw an error as **main**() is called by the JVM before any objects are made and only static methods can be directly invoked via the class.
- **void**: It is the return type of the method. Void defines the method which will not return any value.
- **main**: It is the name of the method which is searched by JVM as a starting point for an application with a particular signature only. It is the method where the main execution occurs.
- **String args[]**: It is the parameter passed to the main method.

22. What is for loop? Explain with an example.

Programmers usually use loops to execute a set of statements. ***For*** loop is used when they need to iterate a part of the programs multiple times. It is particularly used in cases where the number of iterations is fixed!

For a better understanding, let me give you a pictorial representation!

Flow diagram



Here, after initialization, the condition that you have assigned in the code is scanned, in case the condition is true, it would increment/decrement (according to your code) the value, and again iterate the code according to the condition that you have assigned. But, if your condition is false, it will exit the loop.

After this theoretical explanation, let me show you the syntax of the ***for*** loop!

Syntax

```
for (statement 1; statement 2; statement 3) {
// code block to be executed
}
```

The syntax is pretty simple. It goes as follows
Statement 1: condition before the code block is executed
Statement 2: specifies the condition for execution of the code
Statement 3: condition once the code has been executed

To make things clearer, let us implement the above-explained syntax in a Java code.

Example of for loop

The code written below depicts how for loop is implemented in Java Language

```
public class MyClass {
{
public static void main(String[] args) {
{for (int i = 0; i < 5; i++) {
```

```
System.out.println(i);
}
}
}}
```

Output:

```
0
1
2
3
4
```

I have taken a simple code to get you all acquainted with the concept of for loop. Inside the for loop, there are three statements that I have talked about in the previous segment. I hope you can now relate to them easily!

- Firstly, Int i=0, is the initialization of an integer variable whose value has been assigned to 0.
- Secondly, i<5 is the condition that I have applied in my code
- Thirdly, i++, means that I want the value of my variable to be incremented.

After understanding the working of for loop, let me take you to another concept, that is Java nested ***for*** loop!

Java nested for loop

If you have a for loop inside a for loop, you have encountered a Java nested for loop. The inner loop executes completely when the outer loop executes.

I am presenting an example to show you the working of a Java nested for loop.

Example

A Java code for a nested for loop:

```

public class Example{
public static void main(String[] args) {
for(int i=1;i<=3;i++){
for(int j=1;j<=3;j++){
System.out.println(i+" "+j);
}
}
}
}
```

Output:

```
1 1
1 2
1 3
```

```
2 1
2 2
2 3
3 1
3 2
3 3
```

Now that you have understood the concept of a nested for loop, let me show you a very famous example that you might have heard of! The pyramid examples!

Pyramid Example: Case 1

```
1
2  public class PyramidExample {
3  public static void main(String[] args) {
4  for(int i=1;i<=5;i++){
5  for(int j=1;j<=i;j++){
6  System.out.print("* ");
7  }
8  System.out.println();//new line
9  }
10 }
11 }
```

Output:

```
*
* *
* * *
* * * *
* * * * *
```

Moving on with next example.

Pyramid Example: Case 2

```
1
2  package MyPackage;
3  public class Demo {
4  public static void main(String[] args) {
5  int term=6;
6  for(int i=1;i<=term;i++){ for(int j=term;j>=i;j--){
7  System.out.print("* ");
8  }
9  System.out.println();//new line
10 }
11 }
12 }
```

Output:

```
* * * * *
* * * *
```

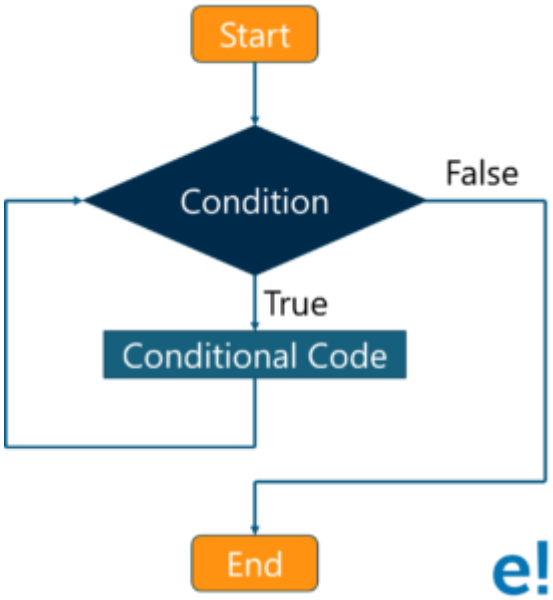
```
* * *
* *
*
```

I am sure that you would be familiar with these two patterns.

23. What is a while loop in Java? Explain with examples.

The Java while loop is used to iterate a part of the program again and again. If the number of iteration is not fixed, then you can use while loop.

A pictorial representation of how a while loop works:



In the above diagram, when the execution begins and the condition returns false, then the control jumps out to the next statement after the while loop. On the other hand, if the condition returns true then the statement inside the while loop is executed.

Moving on with this article on While Loop in Java, Let's have a look at the syntax:

Syntax:

```
while (condition) {

  // code block to be executed

}
```

Now that I have shown you the syntax, here is an example:

Practical Implementation:

```
class Example {
    public static void main(String args[]){
         int i=10;
         while(i>1){
              System.out.println(i);
              i--;
         }
    }
}
```

Output:

```
10
9
8
7
6
5
4
3
2
```

Next, let's take a look at another example:

Another example of While Loop in Java:

```
// Java While Loop example

package Loops;

import java.util.Scanner;

public class WhileLoop {
    private static Scanner sc;

    public static void main(String[] args) {
         int number, sum = 0;
         sc = new Scanner(System.in);

         System.out.println("n Please Enter any integer Value below 10: ");
         number = sc.nextInt();

         while (number <= 10)  {
              sum = sum + number;
              number++;
         }
         System.out.format(" Sum of the Numbers From the While  Loop is: %d ", sum)
    }
}
```

Output:

Please Enter any integer Value below 10: 7
Sum of the Numbers From the While Loop is: 34

Above illustrated example is a bit complex as compared to the previous one. Let me explain it step by step.

In this Java while loop example, the machine would ask the user to enter any integer value below 10. Next, the While loop and the Condition inside the While loop will assure that the given number is less than or equal to 10.

Now, User Entered value = 7 and I have initialized the sum = 0

This is how the iteration would work: (concentrate on the while loop written in the code)

First Iteration:

sum = sum + number
sum = 0 + 7 ==> 7
Now, the number will be incremented by 1 (number ++)

Second Iteration

Now in the first iteration the values of both Number and sum has changed as:
Number = 8 and sum = 7
sum = sum + number
sum = 7 + 8 ==> 15
Again, the number will be incremented by 1 (number ++)

Third Iteration

Now, in the Second Iteration, the values of both Number and sum has changed as:
Number = 9 and sum = 15
sum = sum + number
sum = 15 + 9 ==> 24
Following the same pattern, the number will be incremented by 1 (number ++) again.

Fourth Iteration

In the third Iteration of the Java while loop, the values of both Number and sum has changed as: Number = 10 and sum = 24

sum = sum + number
sum = 24 + 10 ==> 34

Finally, the number will be incremented by 1 (number ++) for the last time.

Here, Number = 11. So, the condition present in the while loop fails.

In the end, System.out.format statement will print the output as you can see above!

Moving further,

One thing that you need to keep in mind is that you should use increment or decrement statement inside while loop so that the loop variable gets changed on each iteration so that at some point, the condition returns false. This way you can end the execution of the while loop. Else, the loop would execute indefinitely. In such cases, where the loop executes indefinitely, you'll encounter a concept of the infinite while loop in Java, which is our next topic of discussion!

Infinite while loop in Java

The moment you pass 'true' in the while loop, the infinite while loop will be initiated.

Syntax:

```
while (true){
    statement(s);
}
```

Practical Demonstration

Let me show you an example of Infinite While Loop in Java:

```

class Example {
    public static void main(String args[]){
          int i=10;
          while(i>1)
          {
              System.out.println(i);
               i++;
          }
    }
}

```

It's an infinite while loop, hence it won't end. This is because the condition in the code says i>1 which would always be true as we are incrementing the value of i inside the while loop.

24. What Is A Switch Case In Java? Explain with examples.

Java switch statement is like a conditional statement which tests multiple values and gives one output. These multiple values that are tested are called cases. It is like a multi-branch statement. After the release of java 7 we can even use strings in the cases. Following is the syntax of using a switch case in Java.

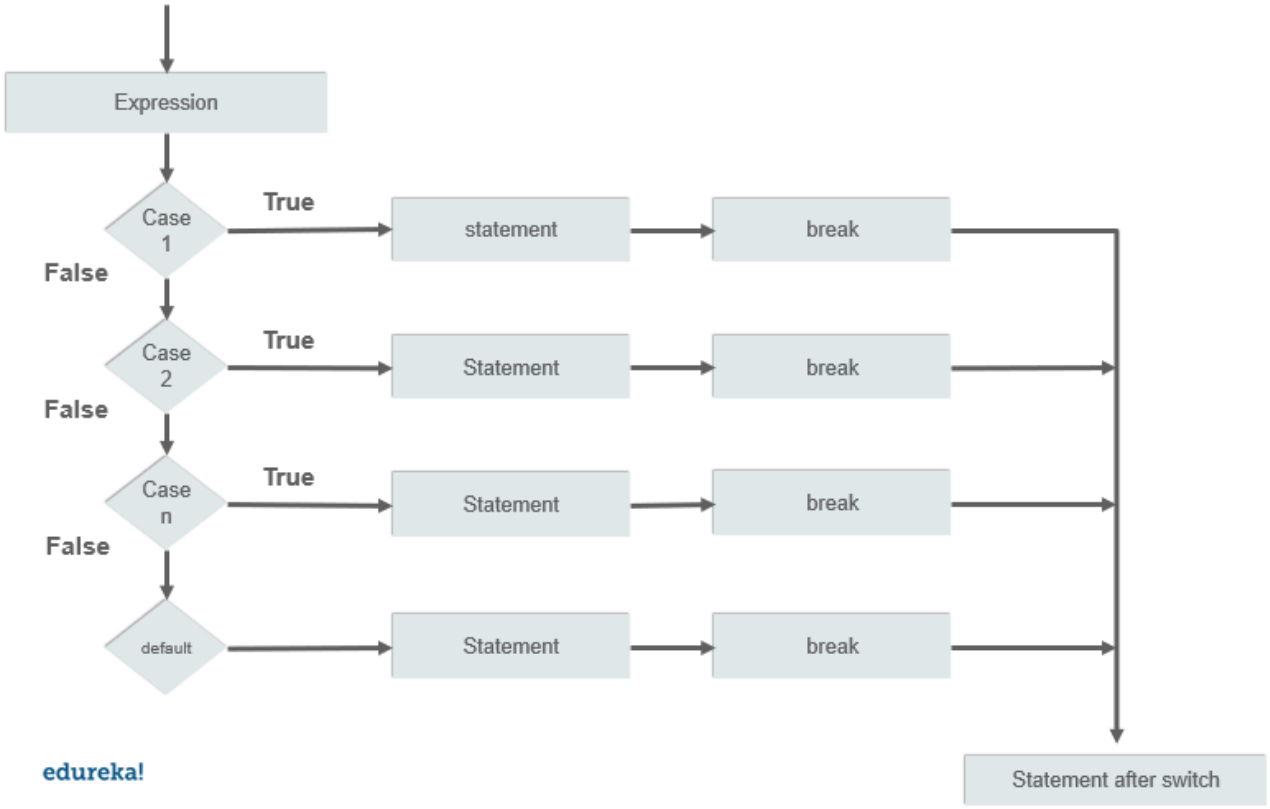
```
switch(expression)
{
case value:
      //statement
      break;
case value n :
     //statement
     break;
default:
     //statement
}
```

Rules To Remember

There are a certain rules one must keep in mind while declaring a switch case in java. Following are a certain points to remember while writing a switch case in java.

1. We cannot declare duplicate values in a switch case.
2. The values in the case and the data type of the variable in a switch case must be same.
3. Variables are not allowed in a case, it must be a constant or a literal.
4. The break statement fulfills the purpose of terminating the sequence during execution.
5. It is not necessary to include the break statement, the execution will move to the next statement if the break statement is missing.
6. The default statement is optional as well, it can appear anywhere in the block.

Flow Chart



Examples

Break Statement In Switch Case

Break statement is used to control the flow of the execution, as soon as the expression is satisfied the execution moves out the switch case block.

```
1  public class Example{
2  public static void main(String args[]){
3  int month= 7;
4
5  switch(month){
6  case 1 :
7        System.out.println("january");
8        break;
9  case 2:
10       System.out.println("february");
11       break;
12 case 3:
13       System.out.println("march");
14       break;
15 case 4:
16       System.out.println("april");
17       break;
18 case 5:
       System.out.println("may");
       break;
case 6:
       System.out.println("june");
```

```
        break;
case 7:
        System.out.println("july");
        break;
case 8:
        System.out.println("august");
        break;
case 9:
        System.out.println("september");
        break;
case 10:
        System.out.println("October");
        break;
case 11:
       System.out.println("november");
       break;
case 12:
       System.out.println("december");
       break;
default:
       System.out.println("not valid");
}
}
}
```

Output: july

Nested Switch Case

Nested switch case incorporates another switch case in an existing switch case. Following is an example showing a nested switch case.

```
public class Example{
public static void main(String args[]){
int tech = 2;
int course = 2;

switch(tech){
case 1:
        System.out.println("python");
        break;
case 2:
        switch(course){
        case 1:
                System.out.println("J2EE");
                break;
        case 2:
               System.out.println("advance java");
                }
   }
```

```
}
}
```

Output: advance java

Fall Through Switch Case

Whenever there is no break statement involved in a switch case block. All the statements are executed even if the test expression is satisfied. Following is an example of a fall through switch case.

```
public class Example{
public static void main( String args[])
{
int courses = 2;

switch(courses){
case 1:
      System.out.println("java");
case 2:
      System.out.println("python");
case 3:
      System.out.println("Devops");
case 4:
      System.out.println("Automation testing");
case 5:
      System.out.println("Hadoop");
case 6:
      System.out.println("AWS");
default:
      System.out.println("check out edureka.co for more");
}
}
}
```

Output: java
python
Devops
Automation testing
Hadoop
AWS
check out edureka.co for more

Enum In Switch Case

Switch case allows enum as well. Enum is basically a list of named constants. Following is an example of the use of enum in a switch case.

```
public class Example{
public enum day { s , m , t , w , th, fr, sa };
public static void main(String args[]){
course[] c = day.values();
 for(day today : c)
{
  switch (today){
 case s :
        System.out.println("Sunday");
        break;
case m:
        System.out.println("Monday");
        break;
case t:
        System.out.println("Tuesday");
        break;
case w:
        System.out.println("Wednesday");
        break;
case th:
        System.out.println("Thursday");
        break;
case fr:
        System.out.println("Friday");
        break;
case sa:
        System.out.println("Saturday");
        break;
     }
   }
}
}
```

Output: Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday

String In Switch Case

After the release of Java 7, a switch case can have strings as a case. Following is an example of using string as cases in a switch statement.

```
public class Example{
public static void main(String args[]){
String player = "batsmen";

switch(player){
case "batsmen":
       System.out.println(" Batsmen are players who plays with a bat");
```

```
        break;
case "bowler":
        System.out.println("who throws the ball");
        break;
case "wicket-keeper":
        System.out.println("who keeps the ball behind the wickets");
        break;
case "fielder":
        System.out.println("who fields in the ground");
        break;
default:
        System.out.println("no entry present");
}
}
}
```

Output: Batsmen are players who play with a bat

25. What is the difference between Array list and vector in Java?

ArrayList	Vector
Array List is not synchronized.	Vector is synchronized.
Array List is fast as it's non-synchronized.	Vector is slow as it is thread safe.
If an element is inserted into the Array List, it increases its Array size by 50%.	Vector defaults to doubling size of its array.
Array List does not define the increment size.	Vector defines the increment size.
Array List can only use Iterator for traversing an Array List.	Vector can use both Enumeration and Iterator for traversing.

26. What is the difference between equals() and == in Java?

Equals() method is defined in Object class in Java and used for checking equality of two objects defined by business logic.

"==" or equality operator in Java is a binary operator provided by Java programming language and used to compare primitives and objects. *public boolean equals(Object*

o) is the method provided by the Object class. The default implementation uses == operator to compare two objects. For example: method can be overridden like String class. equals() method is used to compare the values of two objects.

27. What are the differences between Heap and Stack Memory in Java?

The major difference between Heap and Stack memory are:

Features	Stack	Heap
Memory	Stack memory is used only by one thread of execution.	Heap memory is used by all the parts of the application.
Access	Stack memory can't be accessed by other threads.	Objects stored in the heap are globally accessible.
Memory Management	Follows LIFO manner to free memory.	Memory management is based on the generation associated with each object.
Lifetime	Exists until the end of execution of the thread.	Heap memory lives from the start till the end of application execution.
Usage	Stack memory only contains local primitive and reference variables to objects in heap space.	Whenever an object is created, it's always stored in the Heap space.

28. What is final keyword in Java? How it is used in java?

final is a special keyword in Java that is used as a non-access modifier. A final variable can be used in different contexts such as:

- **final variable**

When the final keyword is used with a variable then its value can't be changed once assigned. In case the no value has been assigned to the final variable then using only the class constructor a value can be assigned to it.

- **final method**

When a method is declared final then it can't be overridden by the inheriting class.

- **final class**

When a class is declared as final in Java, it can't be extended by any subclass class but it can extend other class.

29. What is the difference between break and continue statements?

break	continue
1. Can be used in switch and loop (for, while, do while) statements	1. Can be only used with loop statements
2. It causes the switch or loop statements to terminate the moment it is executed	2. It doesn't terminate the loop but causes the loop to jump to the next iteration
3. It terminates the innermost enclosing loop or switch immediately	3. A continue within a loop nested with a switch will cause the next loop iteration to execute

Example break:

```
1
2  for (int i = 0; i < 5; i++)
3  {
4  if (i == 3)
5  {
6  break;
7  }
8  System.out.println(i);
   }
```

Example continue:

```
1
2  for (int i = 0; i < 5; i++)
3  {
4  if(i == 2)
5  {
6  continue;
7  }
8  System.out.println(i);
   }
```

30.What is an infinite loop in Java? Explain with an example.

An infinite loop is an instruction sequence in Java that loops endlessly when a functional exit isn't met. This type of loop can be the result of a programming error or may also be a deliberate action based on the application behavior. An infinite loop will terminate automatically once the application exits.

For example:

```
1  public class InfiniteForLoopDemo
2  {
3  public static void main(String[] arg) {
4  for(;;)
5  System.out.println("Welcome to Edureka!");
   // To terminate this program press ctrl + c in the console.
```

```
6  }
7  }
8
```

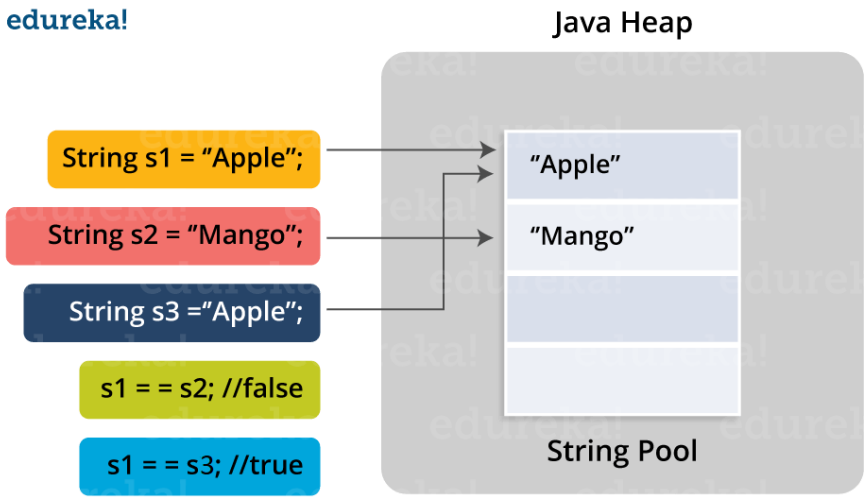
31. What is the difference between this() and super() in Java?

In Java, super() and this(), both are special keywords that are used to call the constructor.

this()	super()
1. this() represents the current instance of a class	1. super() represents the current instance of a parent/base class
2. Used to call the default constructor of the same class	2. Used to call the default constructor of the parent/base class
3. Used to access methods of the current class	3. Used to access methods of the base class
4. Used for pointing the current class instance	4. Used for pointing the superclass instance
5. Must be the first line of a block	5. Must be the first line of a block

32. What is Java String Pool?

Java String pool refers to a collection of Strings which are stored in heap memory. In this, whenever a new object is created, String pool first checks whether the object is already present in the pool or not. If it is present, then the same reference is returned to the variable else new object will be created in the String pool and the respective reference will be returned.



33. Differentiate between static and non-static methods in Java.

Static Method	Non-Static Method
1. *The static* keyword must be used before the method name	1. No need to use the *static* keyword before the method name
2. It is called using the class (className.methodName)	2. It is can be called like any general method
3. They can't access any non-static instance variables or methods	3. It can access any static method and any static variable without creating an instance of the class

34. What is constructor chaining in Java?

In Java, constructor chaining is the process of calling one constructor from another with respect to the current object. Constructor chaining is possible only through legacy where a subclass constructor is responsible for invoking the superclass' constructor first. There could be any number of classes in the constructor chain. Constructor chaining can be achieved in two ways:

1. Within the same class using this()
2. From base class using super()

35. Difference between String, StringBuilder, and StringBuffer.

Factor	String	StringBuilder	StringBuffer
Storage Area	Constant String Pool	Heap Area	Heap Area
Mutability	Immutable	Mutable	Mutable
Thread Safety	Yes	No	Yes
Performance	Fast	More efficient	Less efficient

36. What is the difference between an array and an array list?

Array	ArrayList
Cannot contain values of different data types	Can contain values of different data types.
Size must be defined at the time of declaration	Size can be dynamically changed
Need to specify the index in order to add data	No need to specify the index
Arrays are not type parameterized	Arraylists are type

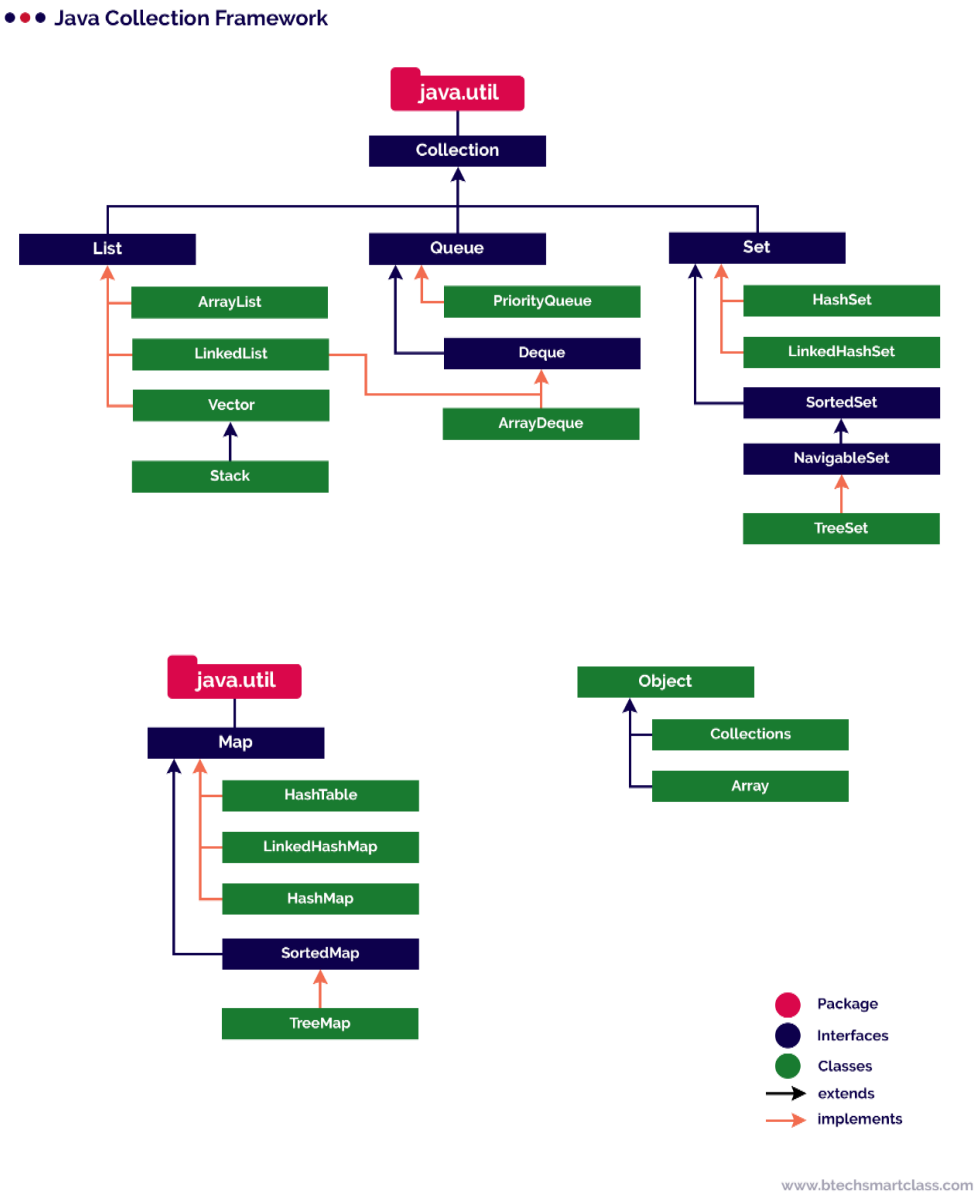
Arrays can contain primitive data types as well as objects	Arraylists can contain only objects, no primitive data types are allowed

37. What is collection class in Java? List down its methods and interfaces.

In Java, the collection is a framework that acts as an architecture for storing and manipulating a group of objects. Using Collections you can perform various tasks like searching, sorting, insertion, manipulation, deletion, etc. Java collection framework includes the following:

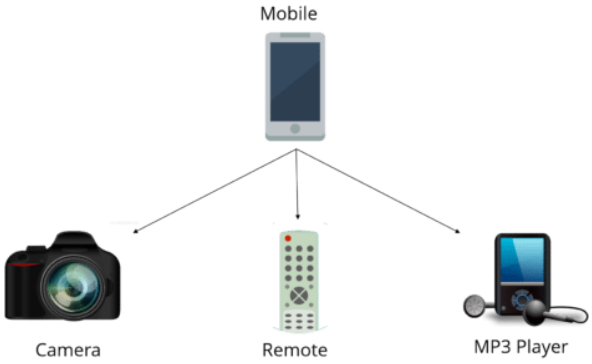
- Interfaces
- Classes
- Methods

The below image shows the complete hierarchy of the Java Collection.



38. What is Polymorphism?

Polymorphism is briefly described as “one interface, many implementations”. Polymorphism is a characteristic of being able to assign a different meaning or usage to something in different contexts – specifically, to allow an entity such as a variable, a function, or an object to have more than one form. There are two types of polymorphism:



1. Compile time polymorphism
2. Run time polymorphism

Compile time polymorphism is method overloading whereas Runtime time polymorphism is done using inheritance and interface.

39. What is runtime polymorphism or dynamic method dispatch?

In Java, runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this process, an overridden method is called through the reference variable of a superclass. Let’s take a look at the example below to understand it better.

```
1
2
3  class Car {
4  void run()
5  {
6  System.out.println(&ldquo;car is running&rdquo;);
7  }
8  }
9  class Audi extends Car {
10 void run()
11 {
12 System.out.prinltn(&ldquo;Audi is running safely with 100km&rdquo;);
13 }
14 public static void main(String args[])
15 {
16 Car b= new Audi();     //upcasting
17 b.run();
18 }
19 }
```

40. What is the difference between abstract classes and interfaces?

Abstract Class	Interfaces
An abstract class can provide complete, default code and/or just the details that have to be overridden	An interface cannot provide any code at all, just the signature
In the case of an abstract class, a class may extend only one abstract class	A Class may implement several interfaces
An abstract class can have non-abstract methods	All methods of an Interface are abstract
An abstract class can have instance variables	An Interface cannot have instance variables
An abstract class can have any visibility: public, private, protected	An Interface visibility must be public (or) none
If we add a new method to an abstract class then we have the option of providing default implementation and therefore all the existing code might work properly	If we add a new method to an Interface then we have to track down all the implementations of the interface and define implementation for the new method
An abstract class can contain constructors	An Interface cannot contain constructors
Abstract classes are fast	Interfaces are slow as it requires extra indirection to find the corresponding method in the actual class

41. What are the different types of inheritance in Java?

Java supports four types of inheritance which are:

1. **Single Inheritance:** In single inheritance, one class inherits the properties of another i.e there will be only one parent as well as one child class.
2. **Multilevel Inheritance:** When a class is derived from a class which is also derived from another class, i.e. a class having more than one parent class but at different levels, such type of inheritance is called Multilevel Inheritance.
3. **Hierarchical Inheritance:** When a class has more than one child classes (subclasses) or in other words, more than one child classes have the same parent class, then such kind of inheritance is known as hierarchical.
4. **Hybrid Inheritance:** Hybrid inheritance is a combination of two *or more types* of inheritance.

42. What is method overloading and method overriding? Explain with example.

Method Overloading :

- In Method Overloading, Methods of the same class shares the same name but each method must have a different number of parameters or parameters having different types and order.
- Method Overloading is to “add” or “extend” more to the method’s behavior.
- It is a compile-time polymorphism.
- The methods must have a different signature.
- It may or may not need inheritance in Method Overloading.

Let’s take a look at the example below to understand it better.

```
1
2  class Adder {
3  Static int add(int a, int b)
4  {
5  return a+b;
6  }
7  Static double add( double a, double b)
8  {
9  return a+b;
10 }
11 public static void main(String args[])
12 {
13 System.out.println(Adder.add(11,11));
14 System.out.println(Adder.add(12.3,12.6));
15 }}
16
```

Method Overriding:

- In Method Overriding, the subclass has the same method with the same name and exactly the same number and type of parameters and same return type as a superclass.
- Method Overriding is to “Change” existing behavior of the method.
- It is a run time polymorphism.
- The methods must have the same signature.
- It always requires inheritance in Method Overriding.

Let’s take a look at the example below to understand it better.

```
1  class Car {
2  void run(){
3  System.out.println(&ldquo;car is running&rdquo;);
4  }
5  Class Audi extends Car{
6  void run()
7  {
8  System.out.prinltn("Audi is running safely with 100km");
9  }
10 public static void main( String args[])
11 {
12 Car b=new Audi();
13 b.run();
14 }
15 }
```

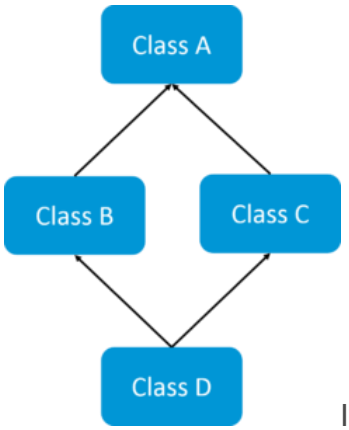
```
13
14
15
```

43. Can you override a private or static method in Java?

You cannot override a private or static method in Java. If you create a similar method with the same return type and same method arguments in child class then it will hide the superclass method; this is known as method hiding. Similarly, you cannot override a private method in subclass because it's not accessible there. What you can do is create another private method with the same name in the child class. Let's take a look at the example below to understand it better.

```
1
2
3  class Base {
4  private static void display() {
5  System.out.println("Static or class method from Base");
6  }
7  public void print() {
8  System.out.println("Non-static or instance method from Base");
9  }
10 class Derived extends Base {
11 private static void display() {
12 System.out.println("Static or class method from Derived");
13 }
14 public void print() {
15 System.out.println("Non-static or instance method from Derived");
16 }
17 public class test {
18 public static void main(String args[])
19 {
20 Base obj= new Derived();
21 obj1.display();
22 obj1.print();
   }
   }
```

44. What is multiple inheritance? Is it supported by Java?



If a child class inherits the property from multiple classes is known as multiple inheritance. Java does not allow to extend multiple classes.

The problem with multiple inheritance is that if multiple parent classes have the same method name, then at runtime it becomes difficult for the compiler to decide which method to execute from the child class.

Therefore, Java doesn't support multiple inheritance. The problem is commonly referred to as Diamond Problem.

45. What is the difference between Error and Exception?

An error is an irrecoverable condition occurring at runtime. Such as OutOfMemory error. These JVM errors you cannot repair them at runtime. Though error can be caught in the catch block but the execution of application will come to a halt and is not recoverable.

While exceptions are conditions that occur because of bad input or human error etc. e.g. FileNotFoundException will be thrown if the specified file does not exist. Or a NullPointerException will take place if you try using a null reference. In most of the cases it is possible to recover from an exception (probably by giving the user feedback for entering proper values etc.

46. How can you handle Java exceptions?

There are five keywords used to handle exceptions in Java:

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

47. What are the differences between Checked Exception and Unchecked Exception?

Checked Exception

- The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions.
- Checked exceptions are checked at compile-time.
- Example: IOException, SQLException etc.

Unchecked Exception

- The classes that extend RuntimeException are known as unchecked exceptions.
- Unchecked exceptions are not checked at compile-time.
- Example: ArithmeticException, NullPointerException etc.

48. What purpose do the keywords final, finally, and finalize fulfill?

Final:

Final is used to apply restrictions on class, method, and variable. A final class can't be inherited, final method can't be overridden and final variable value can't be changed. Let's take a look at the example below to understand it better.

```
1
2  class FinalVarExample {
3  public static void main( String args[])
4  {
5  final int a=10;    // Final variable
6  a=50;               //Error as value can't be changed
7  }
8
```

Finally

Finally is used to place important code, it will be executed whether the exception is handled or not. Let's take a look at the example below to understand it better.

```
1
2
3   class FinallyExample {
4   public static void main(String args[]){
5   try {
6   int x=100;
7   }
8   catch(Exception e) {
9   System.out.println(e);
10  }
11  finally {
12  System.out.println("finally block is executing");}
13  }}
14  }
15
```

Finalize

Finalize is used to perform clean up processing just before the object is garbage collected. Let's take a look at the example below to understand it better.

```
1   class FinalizeExample {
2   public void finalize() {
```

```
3   System.out.println("Finalize is called");
4   }
5   public static void main(String args[])
6   {
7   FinalizeExample f1=new FinalizeExample();
8   FinalizeExample f2=new FinalizeExample();
9   f1= NULL;
10  f2=NULL;
11  System.gc();
12  }
13  }
```

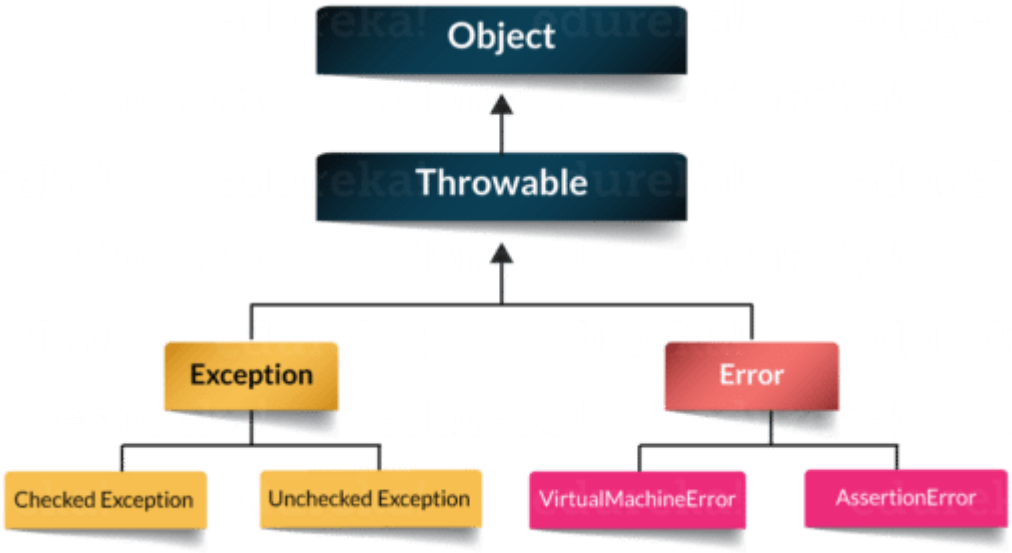
49. What are the differences between throw and throws?

throw keyword	throws keyword
Throw is used to explicitly throw an exception.	Throws is used to declare an exception.
Checked exceptions can not be propagated with throw only.	Checked exception can be propagated with throw
Throw is followed by an instance.	Throws is followed by class.
Throw is used within the method.	Throws is used with the method signature.
You cannot throw multiple exception	You can declare multiple exception e.g. public voi method()throws IOException,SQLException.

50. What is exception hierarchy in java?

The hierarchy is as follows:

Throwable is a parent class of all Exception classes. There are two types of Exceptions: Checked exceptions and UncheckedExceptions or RunTimeExceptions. Both type of exceptions extends Exception class whereas errors are further classified into Virtual Machine error and Assertion error.



51. What are the important methods of Java Exception Class?

Exception and all of it's subclasses doesn't provide any specific methods and all of the methods are defined in the base class Throwable.

1. **String getMessage()** – This method returns the message String of Throwable and the message can be provided while creating the exception through it's constructor.
2. **String getLocalizedMessage()** – This method is provided so that subclasses can override it to provide locale specific message to the calling program. Throwable class implementation of this method simply use getMessage() method to return the exception message.
3. **Synchronized Throwable getCause()** – This method returns the cause of the exception or null id the cause is unknown.
4. **String toString()** – This method returns the information about Throwable in String format, the returned String contains the name of Throwable class and localized message.
5. **void printStackTrace()** – This method prints the stack trace information to the standard error stream, this method is overloaded and we can pass PrintStream or PrintWriter as an argument to write the stack trace information to the file or stream.

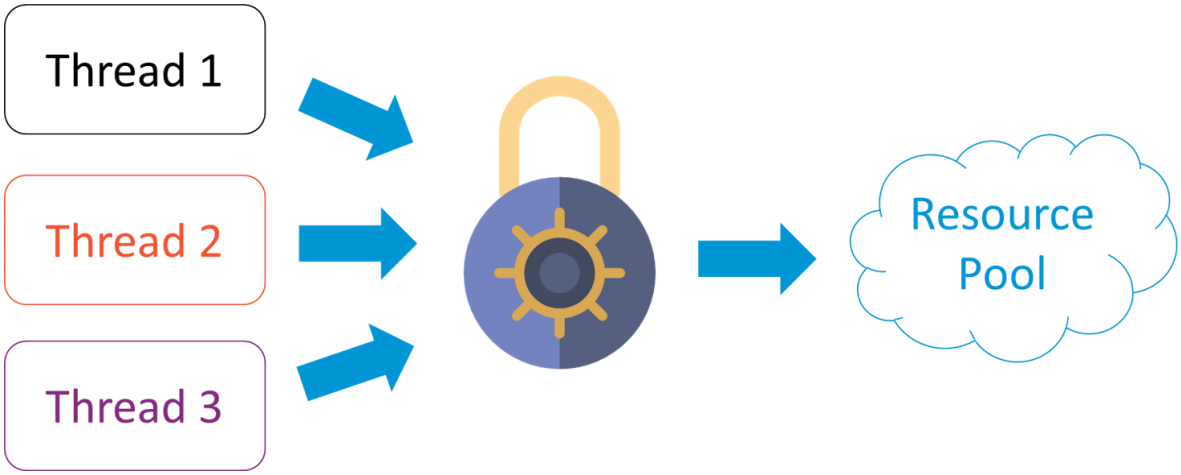
52. What are the differences between processes and threads?

	Process	Thread
Definition	An executing instance of a program is called a process.	A thread is a subset of the process.
Communication	Processes must use inter-process communication to communicate with sibling processes.	Threads can directly communicate with o threads of its process.

Control	Processes can only exercise control over child processes.	Threads can exercise considerable cont threads of the same process.
Changes	Any change in the parent process does not affect child processes.	Any change in the main thread may affe behavior of the other threads of the proc
Memory	Run in separate memory spaces.	Run in shared memory spaces.
Controlled by	Process is controlled by the operating system.	Threads are controlled by programmer i program.
Dependence	Processes are independent.	Threads are dependent.

53. What is synchronization?

Synchronization refers to multi-threading. A synchronized block of code can be executed by only one thread at a time. As Java supports execution of multiple threads, two or more threads may access the same fields or objects. Synchronization is a process which keeps all concurrent threads in execution to be in sync. Synchronization avoids memory consistency errors caused due to inconsistent view of shared memory. When a method is declared as synchronized the thread holds the monitor for that method's object. If another thread is executing the synchronized method the thread is blocked until that thread releases the monitor.



54. Can we write multiple catch blocks under single try block?

Yes we can have multiple catch blocks under single try block but the approach should be from specific to general. Let's understand this with a programmatic example.

```
1  public class Example {
2  public static void main(String args[]) {
3  try {
4  int a[]= new int[10];
5  a[10]= 10/0;
6  }
7  catch(ArithmeticException e)
   {
```

```
8  System.out.println("Arithmetic exception in first catch block");
9  }
10 catch(ArrayIndexOutOfBoundsException e)
11 {
12 System.out.println("Array index out of bounds in second catch block");
13 }
14 catch(Exception e)
15 {
16 System.out.println("Any exception in third catch block");
   }
   }
```

55. What are the important methods of Java Exception Class?

Methods are defined in the base class Throwable. Some of the important methods of Java exception class are stated below.

1. **String getMessage()** – This method returns the message String about the exception. The message can be provided through its constructor.
2. **public StackTraceElement[] getStackTrace() –** This method returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack whereas the last element in the array represents the method at the bottom of the call stack.
3. **Synchronized Throwable getCause()** – This method returns the cause of the exception or null id as represented by a Throwable object.
4. **String toString()** – This method returns the information in String format. The returned String contains the name of Throwable class and localized message.
5. **void printStackTrace()** – This method prints the stack trace information to the standard error stream.

56. What are Collection related features in Java 8?

Java 8 has brought major changes in the Collection API. Some of the changes are:

1. Java Stream API for collection classes for supporting sequential as well as parallel processing
2. Iterable interface is extended with forEach() default method that we can use to iterate over a collection. It is very helpful when used with lambda expressions because its argument Consumer is a function interface.
3. Miscellaneous Collection API improvements such as `forEachRemaining(Consumer action)` method in `Iterator` interface, Map `replaceAll()`, `compute()`, `merge()` methods.

57. What is Java Collections Framework? List out some benefits of Collections framework?

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 the 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

58. 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. I would highly recommend to go through **Java Generic Tutorial** to understand generics in a better way.

59. What are the basic interfaces of Java Collections Framework?

Collection is the root of the collection hierarchy. A collection represents a group of objects known as its elements. The Java platform doesn't provide any direct implementations of this interface.

Set is a collection that cannot contain duplicate elements. This interface models the mathematical set abstraction and is used to represent sets, such as the deck of cards.

List is an ordered collection and can contain duplicate elements. You can access any element from its index. The list is more like an array with dynamic length.

A Map is an object that maps keys to values. A map cannot contain duplicate keys: Each key can map to at most one value.

Some other interfaces
are `Queue`, `Dequeue`, `Iterator`, `SortedSet`, `SortedMap` and `ListIterator`.

60.Why Collection doesn't extend Cloneable and Serializable interfaces?

Collection interface specifies group of Objects known as elements. How the elements are maintained is left up to the concrete implementations of Collection. For example, some Collection implementations like List allow duplicate elements whereas other implementations like Set don't.
A lot of the Collection implementations have a public clone method. However, it doesn't make sense to include it in all implementations of Collection. This is because Collection is an abstract representation. What matters is the implementation.
The semantics and the implications of either cloning or serializing come into play when dealing with the actual implementation; so concrete implementation should decide how it should be cloned or serialized, or even if it can be cloned or serialized.
So mandating cloning and serialization in all implementations is less flexible and more restrictive. The specific implementation should decide as to whether it can be cloned or serialized.

61.Why Map interface doesn't extend Collection interface?

Although Map interface and its implementations are part of the Collections Framework, Map is not collections and collections are not Map. Hence it doesn't make sense for Map to extend Collection or vice versa.
If Map extends Collection interface, then where are the elements? The map contains key-value pairs and it provides methods to retrieve the list of Keys or values as Collection but it doesn't fit into the "group of elements" paradigm.

62.What is an Iterator?

The Iterator interface provides methods to iterate over any Collection. We can get iterator instance from a Collection using *iterator()* method. Iterator takes the place of Enumeration in the Java Collections Framework. Iterators allow the caller to remove elements from the underlying collection during the iteration. Java Collection iterator provides a generic way for traversal through the elements of a collection and implements **Iterator Design Pattern**.

63.What is difference between Enumeration and Iterator interface?

Enumeration is twice as fast as Iterator and uses very little memory. Enumeration is very basic and fits basic needs. But the Iterator is much safer as compared to Enumeration because it always denies other threads to modify the collection object which is being iterated by it.
Iterator takes the place of Enumeration in the Java Collections Framework. Iterators allow the caller to remove elements from the underlying collection that is not possible with Enumeration. Iterator method names have been improved to make its functionality clear.

64.Why there is not method like Iterator.add() to add elements to the collection?

The semantics are unclear, given that the contract for Iterator makes no guarantees about the order of iteration. Note, however, that ListIterator does provide an add operation, as it does guarantee the order of the iteration.

65.Why Iterator don't have a method to get next element directly without moving the cursor?

It can be implemented on top of current Iterator interface but since its use will be rare, it doesn't make sense to include it in the interface that everyone has to implement.

66.What is different between Iterator and ListIterator?

We can use Iterator to traverse Set and List collections whereas ListIterator can be used with Lists only.

Iterator can traverse in forward direction only whereas ListIterator can be used to traverse in both the directions.

ListIterator inherits from Iterator interface and comes with extra functionalities like adding an element, replacing an element, getting index position for previous and next elements.

67.What are different ways to iterate over a list?

We can iterate over a list in two different ways – using iterator and using for-each loop.

```
List<String> strList = new ArrayList<>();

//using for-each loop
for(String obj : strList){
    System.out.println(obj);
}

//using iterator
Iterator<String> it = strList.iterator();
while(it.hasNext()){
    String obj = it.next();
    System.out.println(obj);
}
```

Using iterator is more thread-safe because it makes sure that if underlying list elements are modified, it will throw `ConcurrentModificationException`.

68.What do you understand by iterator fail-fast property?

Iterator fail-fast property checks for any modification in the structure of the underlying collection everytime we try to get the next element. If there are any modifications found, it throws `ConcurrentModificationException`. All the implementations of Iterator in Collection classes are fail-fast by design

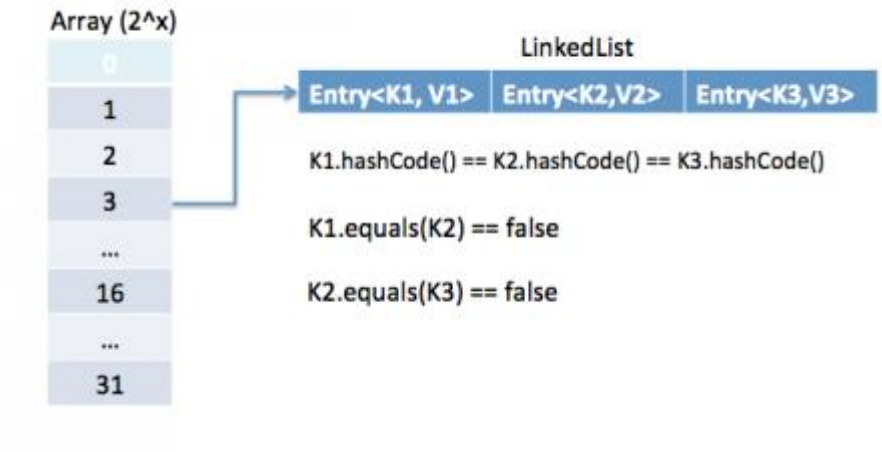
except the concurrent collection classes like ConcurrentHashMap and CopyOnWriteArrayList.

69.What is difference between fail-fast and fail-safe?

Iterator fail-safe property work with the clone of underlying collection, hence it's not affected by any modification in the collection. By design, all the collection classes in `java.util` package are fail-fast whereas collection classes in `java.util.concurrent` are fail-safe.
Fail-fast iterators throw ConcurrentModificationException whereas fail-safe iterator never throws ConcurrentModificationException.
Check this post for CopyOnWriteArrayList Example.

70.How HashMap works in Java?

HashMap stores key-value pair in `Map.Entry` static nested class implementation. HashMap works on hashing algorithm and uses hashCode() and equals() method in `put` and `get` methods.
When we call `put` method by passing key-value pair, HashMap uses Key hashCode() with hashing to find out the index to store the key-value pair. The Entry is stored in the LinkedList, so if there is an already existing entry, it uses equals() method to check if the passed key already exists, if yes it overwrites the value else it creates a new entry and stores this key-value Entry.
When we call `get` method by passing Key, again it uses the hashCode() to find the index in the array and then use equals() method to find the correct Entry and return its value. The below image will explain these detail clearly.



The other important things to know about HashMap are capacity, load factor, threshold resizing. HashMap initial default capacity is **16** and load factor is 0.75. The threshold is capacity multiplied by load factor and whenever we try to add an entry if map size is greater than the threshold, HashMap rehashes the contents of the map into a new array with a larger capacity. The capacity is always the power of 2, so if you know that you need to store a large number of key-value pairs, for example in caching data from the database, it's a good idea to initialize the HashMap with correct capacity and load factor.

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

HashMap uses the Key object hashCode() and equals() method to determine the index to put the key-value pair. These methods are also used when we try to get value from HashMap. If these methods are not implemented correctly, two different Key's might produce the same hashCode() and equals() output and in that case, rather than storing it at a different location, HashMap will consider the same and overwrite them.

Similarly all the collection classes that doesn't store duplicate data use hashCode() and equals() to find duplicates, so it's very important to implement them correctly. The implementation of equals() and hashCode() should follow these rules.

If `o1.equals(o2)`, then `o1.hashCode() == o2.hashCode()` should always be `true`.

If `o1.hashCode() == o2.hashCode` is true, it doesn't mean that `o1.equals(o2)` will be `true`.

72.Can we use any class as Map key?

We can use any class as Map Key, however following points should be considered before using them.

If the class overrides equals() method, it should also override hashCode() method.

The class should follow the rules associated with equals() and hashCode() for all instances. Please refer earlier question for these rules.

If a class field is not used in equals(), you should not use it in hashCode() method.

Best practice for user defined key class is to make it immutable, so that hashCode() value can be cached for fast performance. Also immutable classes make sure that hashCode() and equals() will not change in future that will solve any issue with mutability.
For example, let's say I have a class `MyKey` that I am using for the HashMap key.

```
        //MyKey name argument passed is used for equals() and
hashCode()
        MyKey key = new MyKey("Pankaj"); //assume hashCode=1234
        myHashMap.put(key, "Value");

        // Below code will change the key hashCode() and
equals()
        // but its location is not changed.
        key.setName("Amit"); //assume new hashCode=7890

        //below will return null because HashMap will try to
look for key
        //in the same index as it was stored but since the key
is mutated,
        //there will be no match and it will return null.
        myHashMap.get(new MyKey("Pankaj"));
```

This is the reason why String and Integer are mostly used as HashMap keys

73. **What is difference between HashMap and Hashtable?**

HashMap and Hashtable both implements Map interface and looks similar, however, there is the following difference between HashMap and Hashtable.

HashMap allows null key and values whereas Hashtable doesn't allow null key and values.

Hashtable is synchronized but HashMap is not synchronized. So HashMap is better for single threaded environment, Hashtable is suitable for multi-threaded environment.

`LinkedHashMap` was introduced in Java 1.4 as a subclass of HashMap, so incase you want iteration order, you can easily switch from HashMap to

LinkedHashMap but that is not the case with Hashtable whose iteration order is unpredictable.

HashMap provides Set of keys to iterate and hence it's fail-fast but Hashtable provides Enumeration of keys that doesn't support this feature.

Hashtable is considered to be legacy class and if you are looking for modifications of Map while iterating, you should use ConcurrentHashMap.

74.How to decide between HashMap and TreeMap?

For inserting, deleting, and locating elements in a Map, the HashMap offers the best alternative. If, however, you need to traverse the keys in a sorted order, then TreeMap is your better alternative. Depending upon the size of your collection, it may be faster to add elements to a HashMap, then convert the map to a TreeMap for sorted key traversal.

75.What are similarities and difference between ArrayList and Vector?

ArrayList and Vector are similar classes in many ways.

Both are index based and backed up by an array internally.

Both maintains the order of insertion and we can get the elements in the order of insertion.

The iterator implementations of ArrayList and Vector both are fail-fast by design.

ArrayList and Vector both allows null values and random access to element using index number.

These are the differences between ArrayList and Vector.

Vector is synchronized whereas ArrayList is not synchronized. However if you are looking for modification of list while iterating, you should use CopyOnWriteArrayList.

ArrayList is faster than Vector because it doesn't have any overhead because of synchronization.

ArrayList is more versatile because we can get synchronized list or read-only list from it easily using Collections utility class.

76.What is difference between Array and ArrayList? When will you use Array over ArrayList?

Arrays can contain primitive or Objects whereas ArrayList can contain only Objects.
Arrays are fixed-size whereas ArrayList size is dynamic.
Arrays don't provide a lot of features like ArrayList, such as addAll, removeAll, iterator, etc.

Although ArrayList is the obvious choice when we work on the list, there are a few times when an array is good to use.

If the size of list is fixed and mostly used to store and traverse them.

For list of primitive data types, although Collections use autoboxing to reduce the coding effort but still it makes them slow when working on fixed size primitive data types.

If you are working on fixed multi-dimensional situation, using [][] is far more easier than List<List<>>

77.What is difference between ArrayList and LinkedList?

ArrayList and LinkedList both implement List interface but there are some differences between them.

ArrayList is an index based data structure backed by Array, so it provides random access to its elements with performance as O(1) but LinkedList stores data as list of nodes where every node is linked to its previous and next node. So even though there is a method to get the element using index, internally it traverse from start to reach at the index node and then return the element, so performance is O(n) that is slower than ArrayList.

Insertion, addition or removal of an element is faster in LinkedList compared to ArrayList because there is no concept of resizing array or updating index when element is added in middle.

LinkedList consumes more memory than ArrayList because every node in LinkedList stores reference of previous and next elements.

78.What is Queue and Stack, list their differences?

Both Queue and Stack are used to store data before processing them. `java.util.Queue` is an interface whose implementation classes are present in java concurrent package. Queue allows retrieval of element in First-In-First-Out (FIFO) order but it's not always the case. There is also Deque interface that allows elements to be retrieved from both end of the queue.
The stack is similar to queue except that it allows elements to be retrieved in Last-In-First-Out (LIFO) order.
Stack is a class that extends Vector whereas Queue is an interface.

79.What is Collections Class?

`java.util.Collections` is a utility class consists exclusively of static methods that operate on or return collections. It contains polymorphic algorithms that operate on collections, "wrappers", which return a new collection backed by a specified collection, and a few other odds and ends.

This class contains methods for collection framework algorithms, such as binary search, sorting, shuffling, reverse, etc.

80.What is Comparable and Comparator interface?

Java provides a Comparable interface which should be implemented by any custom class if we want to use Arrays or Collections sorting methods. The comparable interface has a compareTo(T obj) method which is used by sorting methods. We should override this method in such a way that it returns a negative integer, zero, or a positive integer if "this" object is less than, equal to, or greater than the object passed as an argument.

But, in most real-life scenarios, we want sorting based on different parameters. For example, as a CEO, I would like to sort the employees based on Salary, an HR would like to sort them based on age. This is the situation where we need to use `Comparator` interface because `Comparable.compareTo(Object o)` method implementation can sort based on one field only and we can't choose the field on which we want to sort the Object.
Comparator interface `compare(Object o1, Object o2)` method need to be implemented that takes two Object argument, it should be implemented in such a way that it returns negative int if the first argument is less than the

second one and returns zero if they are equal and positive int if the first argument is greater than the second one.

Check this post for use of Comparable and Comparator interface to sort objects.

81. What is Event-Dispatcher-Thread (EDT) in Swing?
The Event Dispatcher Thread or EDT is a special thread in Swing and AWT. The Event-Driven Thread is used to render graphics and listen for events in Swing. You will get a bonus point if you are able to highlight that time-consuming operation like connecting to database, opening a file, or connecting to network should not be done on EDT thread because it could lead to freezing GUI because of blocking and time-consuming nature of these operations instead they should be done on separate thread and EDT can just be used to spawn those thread on a button click or mouse click.

82. Does Swing is thread-safe? What do you mean by swing is not thread-safe?
Though it's pretty basic many developers don't understand **thread-safety issue in Swing**. Since Swing components are not thread-safe it means you can not update these components in any thread other than Event-Dispatcher-Thread.

If you do so you will see unexpected behavior e.g. freezing GUI due to the deadlock of incorrect values etc. Some time interviewer will also ask what are thread-safe methods in the Swing API? Those are the methods which can be safely called from any thread and there are only a couple of them
e.g. `repaint()` and `revalidate()`.

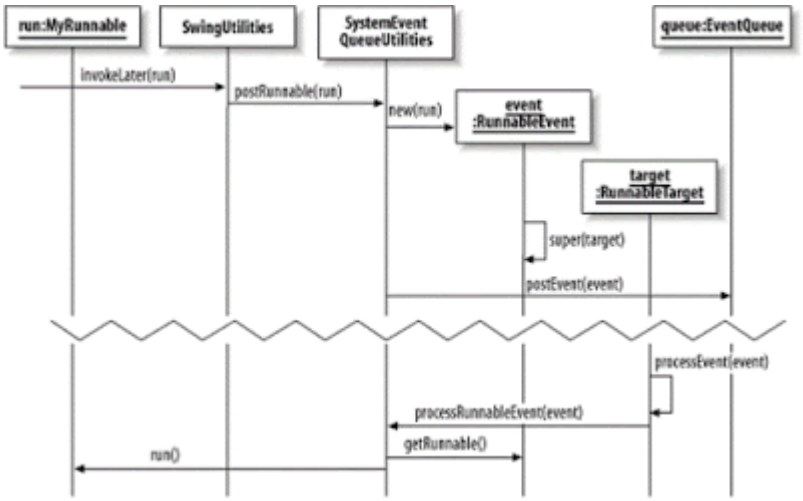
83. What are differences between Swing and AWT?
There are a couple of differences between Swing and AWT:

1. The AWT component are considered to be **heavyweight** while Swing component are considered **lightweights**
2. The Swing GUI has pluggable look and feel.
3. The AWT is platform dependent i.e. the same GUI will look different on different platforms because they use native components e.g. a Frame will look different on Windows and Linux if you pick native components. On the other hand, Swing components are developed in Java and are platform dependent. It also provide consistent GUIs across platform e.g. if you run NetBeans which is created in Java on Linux or Windows you will see consistency in GUI.

84. Why Swing components are called lightweight components?
Since the AWT components are associated with native screen resource they are called heavyweight component while Swing components use the screen resource of an ancestor instead of having their own and that's why called lightweight or lighter component.

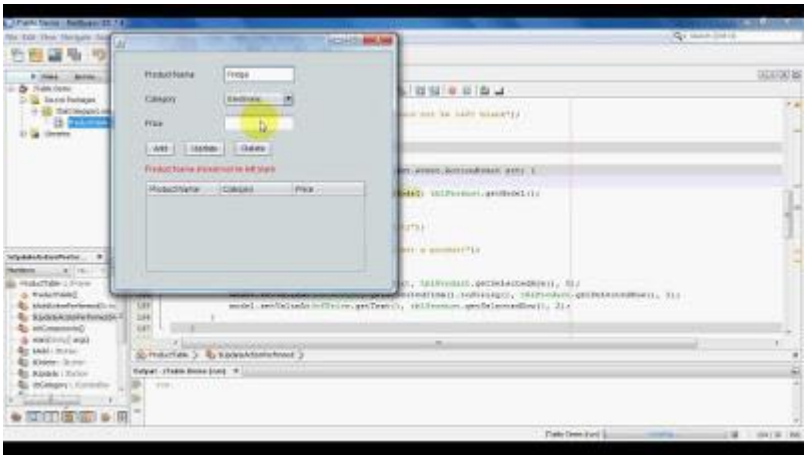
85. What is the difference between invokeAndWait and invokeLater?
We use `SwingUtilities.invokeAndWait(Runnable r)` and `SwingUtilities.invokeLetter(Runnable r)` though there are quite a few differences between these two, the major one is that `invokeAndWait()` is a blocking call and wait until GUI update is finished while invokeLater is a non-blocking asynchronous call, so it won't wait for GUI to be updated. The update request will be picked by Event Dispatcher thread from the request queue.

In my opinion, these question has its own value and every swing developer should be familiar with these questions or concept not just for interview point of view but on application perspective.



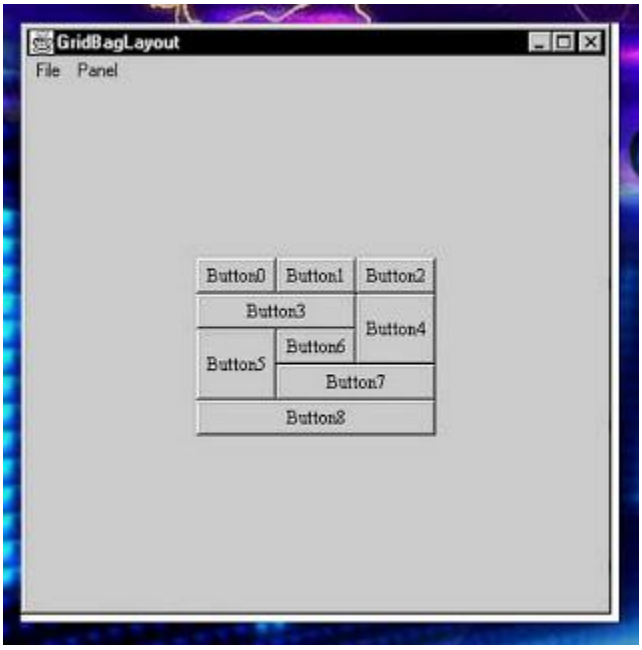
86. Write code for JTable with custom cell editor and custom cell renderer?

The GUI used for *online stock trading uses JTable* to show data in tabular format so an in-depth knowledge of JTable is required to work on online trading GUI developed in Swing. While this question is just an example questions around JTable are mostly centered around ***updating table***, **how do you handle large volume of data in table**, using customize cell renderer and editor, sorting table data based on any column etc. so just make sure you have done quite a few handsome exercise on JTable before appearing for any Java Swing interview in IB.



87. Write code to print following layout (mainly focused on GridBag layout)?
The `GridBagLayout` in the swing is most powerful but at the same time, most complex layout and a clear cut experience and expertise around GridBagLayout is desired for developing Swing GUI for trading systems.

No matter whether you are developing GUI for equities trading, futures or options trading or forex trading you always required GridBagLayout. *Swing interview question on GridBagLayout* will be mostly on writing code for a particular layout just like an example shown here. In which six buttons A, B, C, D, E and F are organized in a certain fashion.



88. What is a layout manager and what are different types of layout managers available in Java Swing?

A layout manager is an object that is used to organize components in a container.

1. FlowLayout: The elements of a Flow Layout are organized in a top to bottom, left to right fashion.
2. Border Layout: The elements of a BorderLayout are organized at the borders(North, South East and West) and the center of a container.
3. Card Layout: The elements of a CardLayout are stacked, on top of the other, like a deck of cards.
4. Grid Layout: The elements of a GridLayout are of equal size and laid out using the square of a grid.
5. Grid Bag Layout: The elements of a GridBagLayout are organized according to a grid. However, the elements may be different sizes and may occupy more than one row or column of the grid. In addition, the rows and columns may have different sizes.

89. Explain the JTable and TableDateModel interface in Swing?

1. JTable is one of the powerful features given by Swing. This class, present in the swing.table package, shows the data in the form of tables, in a much better way. We can also select an entire column or row at a time.
2. JTable (TableDataaModel) is the constructor for a JTable.
3. The method addColumn (JTableColumn) appends a column to the end of the JTable's array of columns and JTableHeader's getTableHeader () method gives a Header to the table.
4. The TableDataModel interface specifies the interface for objects that provides data for cells in a JTable. We can have an object of this interface by creating an object of JTableDataModelAdapter after overriding the methods.

90. How do you classify Swing Components?

Swing components are classified under the following headings:
Top level containers – The containers at the top of any swing component hierarchy are:

1. Applet
2. Dialog
3. Frame

. General purpose containers – The intermediate containers are

1. Panel
2. Scroll pane
3. Split pane
4. Tabbed pane
5. Tool bar

Special purpose containers – These are intermediate containers which play specific roles in the user interface:

1. Internal frame
2. Layered pane
3. Root pane

Basic controls : The atomic components that exist primarily to get input from the user are

1. Buttons
2. Combo box
3. List
4. Menu
5. Slider
6. Spinner
7. TextField

Uneditable information displays: The atomic components which give information to the user are

1. Label
2. Progress bar
3. Tool tip

Interactive displays of highly formatted information – The atomic components that display formatted information that can be modified by the user are

1. Color chooser
2. File chooser
3. Table
4. Text
5. Tree

91. What is MVC?

Model-View-Controller (MVC) is a well known object oriented design for GUI components. The architecture of swing components is based on MVC design. When a GUI component is developed using MVC architecture, it is divided into three parts:

1. Model-An object that defines the component's state
2. View-The visual screen representation of component.
3. Controller-An object that controls a component in such a way that it responds to user input.
4. The model part of an MVC-based component provides information that can be used to specify the component's value, provided the component has any value properties, e.g. the min and max values of a slider control are stored in the component's model part.

5. The Controller part modifies information maintained by the component's model part in response to input from the user.
6. The View part manages the way in which the object is drawn on the screen.