**UNIT – IV**

**The Collections Framework (java.util)-** Collections overview, Collection Interfaces, The Collection classes- Array List, Linked List, Hash Set, Tree Set, Priority Queue, Array Deque. Accessing a Collection via an Iterator, Using an Iterator, The For-Each alternative, Map Interfaces and Classes, Comparators, Collection algorithms, Arrays, The Legacy Classes and Interfaces- Dictionary, Hashtable ,Properties, Stack, Vector More Utility classes, String Tokenizer, Bit Set, Date, Calendar, Random, Formatter, Scanner

**SHORT ANSWER QUESTIONS**

**1. Define collections?**

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

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

**2. What is Java collection framework?**

Collection framework represents a unified architecture for storing and manipulating group of objects.

It has:

1. Interfaces and its implementations i.e. classes

2. Algorithm

**3. Define ArrayList with syntax?**

Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.

**Syntax**

public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable, Serializable

**4. Define HashTable with syntax?**

Java Hashtable class implements a hashtable, which maps keys to values. It inherits Dictionary class and implements the Map interface.

Syntax:

public class Hashtable<K,V> extends Dictionary<K,V> implements Map<K,V>, Cloneable, Serializable

**5. What is an iterator interface in java ?**

Iterator interface provides the facility of iterating the elements in forward direction only.

There are three methods in the Iterator interface.

public boolean hasNext(),

public object next(),

public void remove()

**6. Define properties class in java?**

The properties object contains key and value pair both as a string. The java.util.Properties class is the subclass of Hashtable.

It can be used to get property value based on the property key. The Properties class provides methods to get data from properties file and store data into properties file. Moreover, it can be used to get properties of system.

**7. Define Random class in java?**

The java.util.Random class instance is used to generate a stream of pseudorandom numbers.

**Syntax:**

public class Random

extends Object

implements Serializable

**8. Define Scanner class in java?**

The Java Scanner class breaks the input into tokens using a delimiter that is whitespace bydefault. It provides many methods to read and parse various primitive values. Java Scanner class is widely used to parse text for string and primitive types using regular expression.

Java Scanner class extends Object class and implements Iterator and Closeable interfaces.

**9. Define stream?**

A stream can be defined as a sequence of data. There are two kinds of Streams.

* InPutStream − The InputStream is used to read data from a source.
* OutPutStream − The OutputStream is used for writing data to a destination.

**10. Define byte stream?**

Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently used classes are, FileInputStream and FileOutputStream.

**11. Define character stream?**

Java Character streams are used to perform input and output for 16-bit unicode. Though there are many classes related to byte streams but the most frequently used classes are, FileInputStream and FileOutputStream.

**Long Answer Questions.**

**1. Write about Vector with an explain its working with an example program?**

Vector implements a dynamic array. It is similar to ArrayList, but with two differences −

* Vector is synchronized.
* Vector contains many legacy methods that are not part of the collections framework.
* Vector proves to be very useful if you don't know the size of the array in advance or you just need one that can change sizes over the lifetime of a program.

**Code for Vector implementation.**

import java.util.\*;

public class VectorDemo {

public static void main(String args[]) {

// initial size is 3, increment is 2

Vector v = new Vector(3, 2);

System.out.println("Initial size: " + v.size());

System.out.println("Initial capacity: " + v.capacity());

v.addElement(new Integer(1));

v.addElement(new Integer(2));

v.addElement(new Integer(3));

v.addElement(new Integer(4));

System.out.println("Capacity after four additions: " + v.capacity());

v.addElement(new Double(5.45));

System.out.println("Current capacity: " + v.capacity());

v.addElement(new Double(6.08));

v.addElement(new Integer(7));

System.out.println("Current capacity: " + v.capacity());

v.addElement(new Float(9.4));

v.addElement(new Integer(10));

System.out.println("Current capacity: " + v.capacity());

v.addElement(new Integer(11));

v.addElement(new Integer(12));

System.out.println("First element: " + (Integer)v.firstElement());

System.out.println("Last element: " + (Integer)v.lastElement());

if(v.contains(new Integer(3)))

System.out.println("Vector contains 3.");

// enumerate the elements in the vector.

Enumeration vEnum = v.elements();

System.out.println("\nElements in vector:");

while(vEnum.hasMoreElements())

System.out.print(vEnum.nextElement() + " ");

System.out.println();

}

}

This will produce the following result −

**Output:**

Initial size: 0

Initial capacity: 3

Capacity after four additions: 5

Current capacity: 5

Current capacity: 7

Current capacity: 9

First element: 1

Last element: 12

Vector contains 3.

Elements in vector:

1 2 3 4 5.45 6.08 7 9.4 10 11 12

**2. Explain HashTable class in java with an example program?**

Java Hashtable class implements a hashtable, which maps keys to values. It inherits Dictionary class and implements the Map interface.

The important points about Java Hashtable class are:

* A Hashtable is an array of list. Each list is known as a bucket. The position of bucket is identified by calling the hashcode() method. A Hashtable contains values based on the key.
* It contains only unique elements.
* It may have not have any null key or value.
* It is synchronized.

Hashtable class Parameters

The Parameters for java.util.Hashtable class are:

* K: It is the type of keys maintained by this map.
* V: It is the type of mapped values.

**Example program (HashTable):**

import java.util.\*;

class TestCollection16{

public static void main(String args[]){

Hashtable<Integer,String> hm=new Hashtable<Integer,String>();

hm.put(100,"Amit");

hm.put(102,"Ravi");

hm.put(101,"Vijay");

hm.put(103,"Rahul");

for(Map.Entry m:hm.entrySet()){

System.out.println(m.getKey()+" "+m.getValue());

}

}

}

**OUTPUT:**

103 Rahul

102 Ravi

101 Vijay

100 Amit

**3. Explain the features of ArrayList with an example Program?**

Java ArrayList class uses a dynamic array for storing the elements. It inherits

AbstractList class and implements List interface.

**The important features Java ArrayList class are:**

* Java ArrayList class can contain duplicate elements.
* Java ArrayList class maintains insertion order.
* Java ArrayList class is non synchronized.
* Java ArrayList allows random access because array works at the index basis.
* In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.



**Fig: Hierarchy of ArrayList**

**Example Program:**

import java.util.\*;

class TestCollection1{

public static void main(String args[]){

ArrayList<String> list=new ArrayList<String>();//Creating arraylist

list.add("Ravi");//Adding object in arraylist

list.add("Vijay");

list.add("Ravi");

list.add("Ajay");

//Traversing list through Iterator

Iterator itr=list.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

**Output:** Ravi

Vijay

Ravi

Ajay

**4. What is stack? Perform stack operations push and pop in java?**

Stack is a subclass of Vector that implements a standard last-in, first-out stack.

Stack only defines the default constructor, which creates an empty stack. Stack includes all the methods defined by Vector, and adds several of its own.

Syntax: Stack()

import java.util.\*;

public class StackDemo {

static void showpush(Stack st, int a) {

st.push(new Integer(a));

System.out.println("push(" + a + ")");

System.out.println("stack: " + st);

}

static void showpop(Stack st) {

System.out.print("pop -> ");

Integer a = (Integer) st.pop();

System.out.println(a);

System.out.println("stack: " + st);

}

public static void main(String args[]) {

Stack st = new Stack();

System.out.println("stack: " + st);

showpush(st, 42);

showpush(st, 66);

showpush(st, 99);

showpop(st);

showpop(st);

showpop(st);

try {

showpop(st);

}catch (EmptyStackException e) {

System.out.println("empty stack");

}

}

}

OUTPUT:

stack: [ ]

push(42)

stack: [42]

push(66)

stack: [42, 66]

push(99)

stack: [42, 66, 99]

pop -> 99

stack: [42, 66]

pop -> 66

stack: [42]

pop -> 42

stack: [ ]

pop -> empty stack

**5. Explain the differences between iterator and enumeration in java?**



**6. What is Enumeration interface in java collections? Explain with an example program program.**

It is used to navigate through legacy collection classes. It is a legacy collection interface, which

is used to iterate through legacy collections.

The Enumeration interface defines the methods by which can be enumerated (obtain one at a

time) the elements in a collection of objects.

This legacy interface has been superceded by Iterator. Although not deprecated, Enumeration is

considered obsolete for new code. However, it is used by several methods defined by the legacy

classes such as Vector and Properties, is used by several other API classes.

The methods declared by Enumeration are summarized in the following table:

**SN Methods with Description**

1

**boolean hasMoreElements( )**

When implemented, it must return true while there are still more

elements to extract, and false when all the elements have been

enumerated.

2

**Object nextElement( )**

This returns the next object in the enumeration as a generic Object

reference.

**EXAMPLE:**

Following is the example showing usage of Enumeration.

import java.util.Vector;

import java.util.Enumeration;

public class EnumerationTester {

public static void main(String args[]) {

Enumeration days;

Vector dayNames = new Vector();

dayNames.add("Sunday");

dayNames.add("Monday");

dayNames.add("Tuesday");

dayNames.add("Wednesday");

dayNames.add("Thursday");

dayNames.add("Friday");

dayNames.add("Saturday");

days = dayNames.elements();

while (days.hasMoreElements()){

System.out.println(days.nextElement());

}

}

}

This would produce the following result:

14

Sunday

Monday

Tuesday

Wednesday

Thursday

Friday

Saturday

**7. Explain the LinkedList collection class in java along with methods and an example program.**

Java **LinkedList** class is used to store the elements. It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.

The important points about Java LinkedList are:

* Java LinkedList class can contain duplicate elements.
* Java LinkedList class maintains insertion order.
* Java LinkedList class is non synchronized.
* In Java LinkedList class, manipulation is fast because no shifting needs to be occurred.
* Java LinkedList class can be used as list, stack or queue.

|  |  |  |
| --- | --- | --- |
|  | **Methods of Linked List** |  |
| 1 | void add(int index, Object element) | It is used to insert the specified element at the specified position index in a list. |
| 2 | void addFirst(Object o) | It is used to insert the given element at the beginning of a list. |
| 3 | void addLast(Object o) | It is used to append the given element to the end of a list. |
| 4 | int size() | It is used to return the number of elements in a list |
| 5 | boolean add(Object o) | It is used to append the specified element to the end of a list. |
| 6 | boolean remove(Object o) | It is used to remove the first occurence of the specified element in a list. |
| 7 | Object getFirst() | It is used to return the first element in a list. |
| 8 | Object getLast() | It is used to return the last element in a list. |
| 9 | int indexOf(Object o) | It is used to return the index in a list of the first occurrence of the specified element, or -1 if the list does not contain any element. |
| 10 | int lastIndexOf(Object o) | It is used to return the index in a list of the last occurrence of the specified element, or -1 if the list does not contain any element. |

**8. Explain the working of TreeSet with an example program.**

Java TreeSet class implements the Set interface that uses a tree for storage.

A TreeSet

* Contains unique elements only like HashSet.
* Access and retrieval times are quiet fast.
* Maintains ascending order.

Methods used with TreeSet



Java program to demonstrate the working of TreeSet

**import** java.util.\*;

**class** TestCollection11{

 **public** **static** **void** main(String args[]){

  //Creating and adding elements

  TreeSet<String> al=**new** TreeSet<String>();

  al.add("Ravi");

  al.add("Vijay");

  al.add("Ravi");

  al.add("Ajay");

  //Traversing elements

  Iterator<String> itr=al.iterator();

  **while**(itr.hasNext()){

   System.out.println(itr.next());

  }

 }

}

OUTPUT:

Ajay

Ravi

Vijay

**9. What is PriorityQueue in java? Demonstrate its working with a java program.**

**Priority Queue**

The PriorityQueue class provides the facility of using queue. But it does not orders the elements in FIFO manner.

* PriorityQueue doesn’t allow **null**
* The elements of the priority queue are ordered according to their natural ordering
* The queue retrieval operations poll, remove, peek, and element access the element at the head of the queue.
* It inherits methods from AbstractQueue, AbstractCollection, Collection and Object class.

**Methods with PriorityQueue**

|  |  |  |
| --- | --- | --- |
| Sl no. | Method | Description |
| 1 | booleanadd(E element) | This method inserts the specified element into this priority queue. |
| 2 | public remove() | This method removes a single instance of the specified element from this queue, if it is present |
| 3 | public poll() | This method retrieves and removes the head of this queue, or returns null if this queue is empty. |
| 4 | public peek() | This method retrieves, but does not remove, the head of this queue, or returns null if this queue is empty. |
| 5 | iterator() | Returns an iterator over the elements in this queue. |
| 6 | booleancontains(Object o) | This method returns true if this queue contains the specified element |

Java program to demonstrate the working of PriorityQueue

**import** java.util.\*;

**class** PriorityQueueDemo{

**public** **static** **void** main(String args[]){

PriorityQueue<String> queue=**new** PriorityQueue<String>();

queue.add("Amit");

queue.add("Vijay");

queue.add("Karan");

queue.add("Jai");

queue.add("Rahul");

System.out.println("head:"+queue.element());

System.out.println("head:"+queue.peek());

System.out.println("iterating the queue elements:");

Iterator itr=queue.iterator();

**while**(itr.hasNext()){

System.out.println(itr.next());

}

queue.remove();

queue.poll();

System.out.println("after removing two elements:");

Iterator<String> itr2=queue.iterator();

**while**(itr2.hasNext()){

System.out.println(itr2.next());

}

}

}

Output:head:Amit

 head:Amit

 iterating the queue elements:

 Amit

 Jai

 Karan

 Vijay

 Rahul

 after removing two elements:

 Karan

 Rahul

 Vijay

**10. Write a java program to perform and, or and xor operation using BitSet collection class.**

The BitSet class creates a special type of array that holds bit values. The BitSet array can increase in size as needed. This makes it similar to a vector of bits. This is a legacy class

**Source Code:**

import java.util.BitSet;

public class BitSetDemo {

 public static void main(String args[]) {

 BitSet bits1 = new BitSet(16);

 BitSet bits2 = new BitSet(16);

 // set some bits

 for(int i = 0; i < 16; i++) {

 if((i % 2) == 0) bits1.set(i);

 if((i % 5) != 0) bits2.set(i);

 }

 System.out.println("Initial pattern in bits1: ");

 System.out.println(bits1);

 System.out.println("\nInitial pattern in bits2: ");

 System.out.println(bits2);

 // AND bits

 bits2.and(bits1);

 System.out.println("\nbits2 AND bits1: ");

 System.out.println(bits2);

 // OR bits

 bits2.or(bits1);

 System.out.println("\nbits2 OR bits1: ");

 System.out.println(bits2);

 // XOR bits

 bits2.xor(bits1);

 System.out.println("\nbits2 XOR bits1: ");

 System.out.println(bits2);

 }

}