Assignment - 03.

6

12/08/24

Name :- K. Hanees
Regno :- 1923240 84

Collections of Java as follows :-

1. ArrayList :- An ArrayList is resizable array implemen

```
import java.util.*;
class ArrayListex{
 public static void main(String[] args){
  Arraylist <String> List=new ArrayList<>();
   List.add("Apple");
   List.add ("Banana");
   List.add ("cherry");
  System.out.println(List);
 }
}
```

Output:-

```
[Apple, Banana, Cherry]
```

LinkedList:-

A LinkedList is a doubly- Linked List implem of List interface

program:-

```
import java.util.*;
class LinkedListex{
 public static void main (String args[]){
   LinkedList <String> List =new LinkedList<>();
    List.add(" Apple");
    List. add (" cherry");
  }
 }
```

utput:-

```
[Apple, cherry]
```

Assignment - 03. 6

Name:- K.Hance
Regno:- 192324084

12/08/24

Collections of Java as follows:-

1. ArrayList:- An Arraylist is resizable array impleme

```
import Java.util.*;
Class ArrayListex{
 public static Void main(String[] args){
  Arraylist <String> List=new ArrayList<>();
   List.add("Apple");
   List.add ("Banana");
   List.add ("cherry");
  System.out.println(List);
 }
}
```

output:-

```
[Apple, Banana, cherry]
```

LinkedList:-

A LinkedList is a doubly- Linked List imple of List interface

program:-

```
import Java.util.*;
class LinkedListex{
 public Static Void main (String args[]){
   LinkedList <String> List =new LinkedList<>();
    List.add(" Apple");
   List. add (" cherry");
  }
 }
```

utput:-

```
[Apple, cherry]
```

3. Hashset:-

A Hashset is a Set implementation that uses a hashtable for storage

Code:-

```
import Java.util.*;
class Hashexd
  public static void main (String args[]) {
      Hashset<String> set = new Hashset<>();
       Set.add ("Apple");
       Set.add (" Icecream");
     System.out.println (set);
  }
 }
```

Output:-

```
   [Apple, Icecream]
```

reeSet

A Treeset is a Set implementation tha

ree for storage.

ode:-

```
import Java.util.*;
lass Treesetex{
  public static void main (String args[]) {
      Treeset<String> set = new Treeset<>();
        Set.add ("Apple");
       Set. add ("Banana");
       Set. add ("cherry");
   System.out.println (set);
  }
  }
```

3. Hashset:-

A Hash set is a set implementation that uses a hashtable for storage

Code:-

```
import Java.util.*;
class Hashex{
  public static void main (String args[]) {
      Hashset<String> set = new Hashset<>();
       set.add ("Apple");
       set.add (" Icecream");
     System.out.println (set);
  }
  }
```

Output:

```
    [Apple, Icecream]
```

reeSet

A Treeset is a set implementation tha
ree for storage.

de:-

```
import Java.util.*;
lass Treesetex{
  public static void main (String args[]) {
     Treeset<String> set = new Treeset<>();
        set.add ("Apple");
       set.add ("Banana");
       set.add ("cherry");
   System.out.println (set);
   }
   }
```

output:-

[Apple, Banana, cherry].

5. Hashmap:- a map implementation that uses a has
table for storage.

```
import Java.util.*;
class Hashmapez{
    public Static void main(String args[]){
     Hashmap<String,Integer> map = new Hashmape
      map.put("Apple", 1);
      map.put("Banana", 2);
     map. put ("Cherry",3);
   System.out.println(map);
  }
 }
```

output:-

```
{Apple = 1, Banana = 2, Cherry = 3}.
```

5. TreeMap:-

A 'TreeMap' is a map implementation that
tree for storage

Code:-

```
import Java.util.*;
class Treemapex{
  public Static void main(String args[]){
    TreeMap<String, Integer> map = new TreeM
      map.put("Apple", 1);
      map. put ("Banana",2);
      map. put ("cherry",3);
   System.out.println(map);
  }
}
```

output:-

```
[Apple, Banana, cherry].
```

5. Hashmap:- a map implementation that uses a has table for storage.

```
import Java.util.*;
class Hashmapex{
    public static void main(String args[]){
    Hashmap<String,Integer> map=new Hashmape
     map.put("Apple", 1);
     map.put("Banana",2);
    map. put ("Cherry",3);
   System.out.println(map);
  }
 }
```

outputs:-

```
{Apple = 1, Banana = 2, cherry = 3}.
```

5. TreeMap:-

A 'TreeMap' is a map implementation that tree for storage

Code:-

```
import Java.util.*;
class Treemapex{
 public static void main(String args[]){
  TreeMap<string, Integer map = new TreeM
    map.put("Apple", 1);
    map.put("Banana",2);
    map. put ("cherry",3);
  System.out.println(map);
 }
}
```

Output:-
{Apple=1, Banana=2, cherry=3}

7. LinkedHashset

A Linkedhashset is a set implementation that uses a hashtable and Linked List for storage.

Code:-

```
import Java.util.*;
class LinkedHashsetex{
  Public static void main(String args[]){
    LinkedHashset<String>set = new LinkedHashset<>();
    Set.add("Apple");
    Set.add("Banana");
    Set.add("cherry");
  System.out.println(set);
 }
}
```

Output:-

```
[Apple, Banana, cherry]
```

Priority Queue:-

A priority Queue is a Queue implementation t
Orders elements Based on their natural or
Or a custom Comparator.

Code:-

```
Import Java.util.*;
class PriorityQueueex{
  Public static Void main(String[] args){
   PriorityQueue<String> Queue = new Priority Qu
```

Output:-

```
{Apple=1, Banana=2, cherry=3}
```

7. LinkedHashset

A Linkedhashset is a set implementation that uses a hashtable and Linked List for storage.

Code:-

```
import Java.util.*;
class LinkedHashsetex{
  Public Static void main (String args[]){
   LinkedHashset<String> set = new LinkedHashset<>();
   Set.add("Apple");
   Set.add("Banana");
   Set.add("cherry");
System.out.println(set);
}
}
```

output:-

```
[Apple, Banana, cherry]
```

Priority Queue:-

A PriorityQueue is a Queue implementation orders elements Based on their natural or or a custom Comparator.

Code:-

```
Import Java.util.*;
class PriorityQueueex{
 Public Static Void main(String[] args){
  PriorityQueue<String> Queue = new PriorityQu
```

```
    queue.add("Apple");
    queue.add("Banana");
    queue.add("cherry");
    System.out.println(queue);
  }
}
```

Output:-

```
[Apple, Banana, cherry].
```

9. Array Dequeue:-

An ArrayDequeue is adeque implementation uses an array for storage.

Code:-

```
import java.util.*;
class ArrayDequeue {
    public static void main(String args[]){
      ArrayDeque <String> deque= new ArrayDeque<>
       deque.add(" Apple");
       deque.add ("Banana");
       System.out.println(dequeue);
    }
  }
```

output:-

```
[Apple, Banana].
```

10. Stack:-

LIFO Implementation of the List Interface.

```
  queue.add("Apple");
  queue.add ("Banana");
  queue.add ("cherry");
  System.out.println(queue);
 }
}
```

output:-

```
[Apple, Banana, cherry].
```

Array Dequeue:-

An ArrayDequeue is adeque implementation uses an array for storage.

code:-

```
import java.util.*;
class ArrayDequee2{
  public static void main(String args[]){
    ArrayDeque<String> deque= new ArrayDeque<>
    deque.add(" Apple");
    deque.add ("Banana");
    System.out.println(dequeue);
  }
 }
```

output:

```
[Apple, Banana].
```

Stack:-

LIFO implementation of the List Interface.

```
import Java.util.*;
class Stackex{
public static void main(String args[]){
        Stack<String> stack = new Stack<>();
        stack.push("Apple");
        stack.push("Banana");
        stack.push("cherry");
      System.out.println(stack);
  }
}
```

output:-

```
[Apple, Banana, cherry].
```

11. Vector:-

A vector is a Synchronized implementation of the List interface.

code:-

```
import Java.util.*;
Class Vectorex{
  public static void main(String args[]){
  Vector<String> vector = new Vector<>();
      Vector.add("Apple");
      vector.add("Custard Apple");
    System.out.println(vector);
  }
}.
```

output:-

```
[Apple, custard Apple].
```

```
import Java. util.*;
class Stackex{
public static void main (String args[]){
        Stack<String> stack = new Stack<> ();
         stack.push("Apple");
         stack.push ("Banana");
        stack.push ("cherry");
      System.out. println(stack);
   }
  }
```

output:-

```
    [Apple, Banana, cherry].
```

11- Vector:-

A vector is a synchronized implementation of the List interface.

Code:-

```
import Java.util.*;
class Vectorex{
  public static void main (String args[]){
   Vector<String>vector = new Vector<>();
      Vector.add ("Apple");
     vector.add ("Custard Apple");
    System. out. println (vector);
  }
 }.
```

output:-

```
   [Apple, Custard Apple].
```