K. Rupesh Reddy
192324097
CSA0985

ASSIGNMENT- 3

Collections:

Java collections framework is a set of classes and interface in java that provides a unified architecture for representing and manuplation collections.

Types of collection:

1. Arraylist: An Arraylist is resizable array implementation.

```
import java.util.collections;
import java.util.*;
Class main {
    public static void main(String[] args) {
    Arraylist<String> list = new Arraylist<>();
        list.add("apple");
        list.add("carrot");
        list.add("Banana");
    System.out.println(list);
    collections.sort(list);
    System.out.println(list);
```

Output:

[Apple, Carrot, Banana]
[Apple, Banana, Carrot]

2. LinkedList: A LinkedList is a doubly linked list implementation of List interface.

```
import java.util.*;
class Linked{
public static void main(String[] args){
    LinkedList<String>List= new LinkedList<>();
    List.add("apple");
    List.add("Banana");
    System.out.println(List);}}
```

output:

[Apple, Banana]

3. HashSet: A Hashset is a set of implementation tat used a hash table for storage.

```
import java.util.*;
class Main {
  public static void main(String[] args) {
    HashSet<String> st= new HashSet<>();
    st.add("apple");
    st.add("Carrot");
    System.out.println(st);}}
```

output:

[Apple, Carrot]

4. TreeSet: A TreeSet is a set implementation that uses tree structured storage.

```
import java.util.*;
Class Main {
  public static void main(String [] args){
  TreeSet<String> set = new TreeSet<>();
    set.add("apple");
    set.add("Bamana");
    System.out.println(set); }}
```

Output:

[apple, Bamana]

5. HashMap: A Map implementation that uses a hash table for storage.

```
Class Main {
  public static void main(String[] args){
  HashMap<String> Map = new HashMap<>();
    map.put(1,"apple");
    map.put(2,"Bamana");
    System.out.println(map);
    System.out.println(map.keySet());
    System.out.println(map.values());}}
```

Output:

{1= apple, 2= Bamana}
[1, 2]
[apple, Bamana]

6. TreeMap: A TreeMap is a map implements that uses Tree for storage.

```
import java.util.*;
class Main{
public static void main (String[] args){
TreeMap<String> Map = new TreeMap<>();
    map.put(1, "apple");
    map.put(2, "Bomana");
    System.out.println(map);
    System.out.println(map.keySet());
    System.out.println(map.values());
```

Output:

{1= apple, 2= Bomana}
[1,2]
[apple, Bomana]

7. LinkedHashSet: A LinkedHashSet is a type of set that of maintains a doobly-linked list of its elements

```
import java.util.*;
class main{
public static void main(String[] args){
LinkedHashSet<String> set = new LinkedHashSet<>();
    set.add("apple");
    set.add("Bomana");
    set.add("cherry");
System.out.println(set);}}
```

8. Priority Queue: A priority queue is a queue implementation that orders elements based on their natural ordering or a custom comparator.

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

Output;

[Apple, Banana, cherry]

9. Array Dequeue; An ArrayDeque is a deque implementation that uses an array for storage.

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

Output;

[Apple, Banana]

10. Stack: LIFO implementation of the List interface

```
import java.util.*;
Class main {
  public Class Main (String[] args) {
    Stack<String> Stack = new Stack<>();
      Stack.push("Apple");
      Stack.push("Banana");
      System.out.println(Stack); }}
```

Output: [Apple, Banana]

11. Vector: A vector is a syncronized implementation of the List interface.

```
import java.util.*;
  Class vector {
  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]