OOPJ Notes Day-10 Session-1 Date: 08/05/2023

Shallow copy and Deep copy

Shallow Copy

```
class Emp implements Cloneable
{
    int EmpId;
    String EmpName;
    public Emp(int empId, String empName) {
        super();
        EmpId = empId;
        EmpName = empName;
    }
    @Override
    public String toString() {
        return "Emp [EmpId=" + EmpId + ", EmpName=" + EmpName + "]";
    }

}

public class DemoOfClone {

    public static void main(String[] args) {

        Emp e1=new Emp(1001, "ABC");

        System.out.println("Emp e1:   "+e1.toString());

        Emp e2=e1;
        System.out.println("Emp e2:    "+e2.toString());

        e1.EmpName="xyz";

        System.out.println("Emp e1:   "+e1.toString());
        System.out.println("Emp e2:    "+e2.toString());

    }

}
```

Functional interface

- A functional interface is who have exactly one abstarct method and can have any number of default method.

```
-interface A  //Functional Interface
```

```
{
    void get(); //abstract public void get();

    default void set()
    {
        System.out.println("Am Default def of set() of Interface A");
    }
}

class IntA implements A
{
    public void get()   //Defination of get() of Interface A
    {
        System.out.println("Im get of Interface A implented in class IntA");
    }
}

public class FunInt {

    public static void main(String[] args) {

        IntA a=new IntA();
        a.get();    //Calling of get() by instance of IntA

        a.set();   //default deffination of Default method is fetched

    }

}
```

Exception propagation

- When ever there is chain of methods who are finally calling the exception generator. Then the exception is handled in master caller.

```
class Exp
{
    void M()
    {
        int a=20;
        int b=0;
        int c=a/b;
    }
    void N()
    {
        M();
    }
    void P()
    {
        try
        {
            N();
        }
        catch(ArithmeticException ex)
        {
            System.out.println("Exception Handled Here in P");
        }
    }
}

public class ExepProp {

    public static void main(String[] args) {

        Exp e1=new Exp();

        e1.P();

    }

}
```

Upcasting in Collection (Classes instances to Interfaces refernces)

- In Collection at every level at top there is an interface like List, Set and Map.
- So while declaring a collection class to use we can have reference variable of there parent inteface. ### List Travsering using Enumeration
  1. Enumeration is interface declared in java.util package.
  2. It was introduced in JDK 1.0.
  3. Methods of Enumeration I/F:
     - boolean hasMoreElements()
     - E nextElement()
  4. Using Enumeration we can traverse limited collections. For Example: Vector, Hashtable etc.
  5. Using Enumeration, we can traverse collection only forward direction. During traversing we can not add, set or remove elements from underlying collection.
- Note: Kinldy make a best use of Itrator, ListItrator, Enumaration, Comparator, Comprarable
- Kindly make use of Remove method in arraylist ### Working with Collection: Set Family or Set Interface Hierarchy

- Set interface and its hierarchy
  1. It is sub interface of java.util.Collection interface.
  2. HashSet, LinkedHashSet, TreeSet, EnumSet are Set collections.
  3. Set Collections do not contain duplicate elements.
  4. This interface is a member of the Java Collections Framework.
  5. It is introduced in JDK 1.2
  6. Method names of Collection and Set interface are same. No new method is added in Set interafce.
- Difference between Set and List
  1. ArrayList, Vector, LinkedList are List Collections and HashSet, LinkedHashSet, TreeSet are Set collections
  2. List collections can contain duplicate elements but Set collections do not contain duplicate elements.
  3. List collections can contain null elements but not all Set collections contain null elements.
  4. We can traverse elements of List collection using ListIterator as well as Iterator but we can we can traverse elements of Set collection using Iterator only.
  5. All List collections are sequential collections but we can not give gurantee of order of elements in Set collection.
- TreeSet with primitive and non primitive elements 1. It is a Set collection. 2. It can not contain duplicate elements. 3. It can not contain null elements. 4. It contains data in sorted order. 5. TreeSet implementation is based on TreeMap<K,V>. 6. It is unsynchronized collection. Using Collections.synchronizedSortedSet() method we can make it synchronized. 7. This class is a member of the Java Collections Framework. 8. It is introduced in JDK 1.2
- Note: If we want to use TreeSet to store elements of non final type then non final type should implement Comparable interface.
- Points to remember in TreeSet
  1. How to create instance of TreeSet:
  2. How to add elements in TreeSet
  3. Can we add duplicate elemnts in TreeSet?: No
  4. Can we add 'null' in TreeSet?: No
  5. Can we convert TreeSet to ArrayList "'java public class CollTreeSetDemo {

  public static void main(String[] args) {

```
TreeSet<Integer> ts1=new TreeSet<Integer>();

ts1.add(10);
ts1.add(45);
ts1.add(50);
ts1.add(55);
ts1.add(90);
ts1.add(9);
```

```
for(Integer in: ts1)
{
    System.out.println(in);
}

ArrayList<Integer> al=new ArrayList<Integer>();
System.out.println("ArrayList Print");
al.addAll(ts1);
for(Integer in: al)
{
    System.out.println(in);
}
}
```

} “‘ - Searching Concept and its type - Linear search with pros and cons. - Binary search with pros and cons. - Searching problem with array and need of Hashing - Hash function, HashCode, Slot - Collision and Collision resolution techniques - Separate Chaining - What is the need to override equals and hashCode method? - Reading Assigment of above listed. - HashSet and LinkedHasSet - HashSet does not store elements in sorting manner just like treeset