Day 16

Cloneable Implementation

- If we want to create new instance from existing instance then we should use clone() method.
- clone() is non final method of java.lang.Object class.
- Syntax: protected native Object clone() throws CloneNotSupportedException
- Inside clone() method, if we want to create shallow copy of current instance then we should use "super.clone()"
- Cloneable is marker interface declared in java.lang package.
- Without implementing, Cloneable interface, if we try to create clone of instance then clone() methods throws CloneNotSupportedException.

Marker Interface

- An interface which do not contain any member is called marker interface.
- It is also called as tagging interface.
- Main purpose of marker interface is to generate metadata for JVM.
- Example:
    1. java.lang.Cloneable
    2. java.util.EventListener
    3. java.util.RandomAccess
    4. java.io.Serializable
    5. java.rmi.Remote

Iterable & Iterator implementation

- Iterable is interface declared in java.lang package.
- "Iterator iterator()" is abstract method of java.lang.Iterable interface.
- It is a factory method for Iterator.
- Iterator is interface declared in java.util package.
- Abstract methods of java.util.Iterator interface:
    1. boolean hasNext()
    2. E next()
- Using foreach loop we can traverse elements of array and instance whose type implements java.lang.Iterable interface.
- If class implements Iterable interface then it is considered as travserible for "for-each" loop.
- In C++, if we use any instance as a pointer then it is called smart pointer.
- Iterator is smart pointer that is used to traverse collection.

Nested Class

- We can define class inside scope of another class. It is called nested class.

```
//Top Level class
class Outer //Outer.class
{
    //Nested class
    class Inner //Outer$Inner.class
    {

    }
}
```

- We can declrae top level class either package level private or public only.
- We can use any access modifier on nested class.
- During inheritance, nested class inherit into sub class.
- Using nested class, we can achive encapsulation.
- Types of nested class
    1. Non static nested class / Inner class
    2. Static nested class

Inner class

- Non static nested class is also called as inner class.
- If implementation of nested class depends on implementation of top level class then we should declare nested class non static.
- Note : For Simplicity, consider non static nested class as a non static method of a class.

```
class Outer
{
    class Inner
    {

    }
}
public class Program
{
    public static void main(String[] args)
    {
        Outer out = new Outer();
        //Outer.Inner in = out.new Inner();
        Outer.Inner in = new Outer().new Inner();
    }
}
```

- Inside Top-Level class we can declare static as well as non static members but inside non static nested class / Inner class we can declare only non static members.

Static Nested Class

- If we declare nested class static then it is called as static nested class.
- If implementation of nested class do not depends on implementation of top level class then we should declare nested class static.
- Note : For Simplicity, consider static nested class as a static method of a class.

```
class Outer
{
    static class Inner
    {

    }
}
public class Program
{
    public static void main(String[] args)
    {
        Outer out = new Outer();
        Outer.Inner in = new Outer.Inner();
    }
}
```

- We can declare static and non static members inside top level class as well as static nested class.

Local Class

- In java, we can define class inside method. It is called local class.
- We can not create reference and instance of local class outside method.
- Types
    1. Method Local Inner class
    2. Method Local Annonymous Inner Class

Method Local Inner class

```
public class Program//Pogram.class
{
    public static void main(String[] args)
    {
        class Complex   //Program$1Complex.class
        {
            private int real = 10;
            private int imag = 20;
            @Override
            public String toString()
```

```
            {
                return this.real+" "+this.imag;
            }
        }
        Complex c1 = new Complex();
        System.out.println(c1.toString());
    }
}
```

Method Local Annonymous Inner class

```
public static void main(String[] args)
{
    Object obj = new Object() //Program$1.class
    {
        private String str = "Hello";
        @Override
        public String toString()
        {
            return str;
        }
    };
    System.out.println(obj.toString());
}
```

```
public static void main(String[] args)
{
    Printable p = new Printable()
    {
        @Override
        public void print()
        {
            System.out.println("Hello");
        }
    };
    p.print();//DMD
}
```

- If interface contains only one abstract method then such interface is called functional interface.
- Functional interface can contain multiple default method as well as multiple static methods but it can contain only one abstract method.
- @FunctionalInterface is annotation which is used to check whether interface is FunctionalInterface or not.
- java.util.function package contains library defined functional interfaces.
    1. Predicate
        - boolean test(T t)

2. Supplier
    - T get()
3. Consumer
    - void accept(T t)
4. Function
    - R apply(T t)

- if we want to implement functional interface then we should use lambda expression or method reference.