Fragile Base Class Problem

- If we make changes in the method of super class then it is necessary to recompile that class and all its sub classes. It is called fragile base class problem.

Interface

- Set of rules is called specification/standard.
- It is a java language feature that is used to define specifications for the sub classes.
- Interface help us:
    1. to develop/build trust realtionship between service provider and service consumer.
    2. to minimize vendor dependancy(to achive loose coupling)
- It is reference type.
- interface is keyword in java.
- Interface can contain:
    1. Nested interface
    2. Constants(Final Fields)
    3. Abstract Method
    4. Default Method
    5. Static Method
- We can declare fields inside interface. Interface fields are implicitly considered as public static and final.

```
interface A
{
    int number = 10;
    //public static final int number = 10;
}
```

- Interface methods are by default considered as public and abstract.

```
interface A
{
    void print();
    //public abstract void print();
}
```

- We can not define constructor inside interface.
- We can not instantiate interface but we can create reference of interface.
- Using "implements" keyword, we can define rules in sub class.

```
interface A
{
    int number = 10;
    //public static final int number = 10;
```

```
    void print();
    //public abstract void print();
}
class B implements A
{
    @Override
    public void print()
    {
        System.out.println("Inside B.print");
    }
}
public class Program
{
    public static void main(String[] args)
    {
        A a = new B();
        a.print();
    }
}
```

- It is mandatory to override abstract methods of interface otherwise sub class can be considered as abstract.

Interface Inheritance:

- In inheritance, if super type and sub type is interface then it is called interface inheritance.

Interface Implementation Inheritance:

- In inheritance, if super type is interface and sub type is class then it is called interface Implementation inheritance.

Implementation Inheritance

- In inheritance, if super type and sub type is class then it is called implementation inheritance.
- I1, I2, I3 -> Interfaces
- C1, C2, C3 -> Classes

1. I2 implements I1; //Not OK
2. I2 extends I1; // OK
3. I3 extends I1, I2; // OK
4. I1 extends C1; //Not OK
5. C1 extends I1; //Not OK
6. C1 implements I1; //OK
7. C1 implements I1, I2; //OK
8. C2 implements C1; //Not OK
9. C2 extends C1; //OK
10. C3 extends C1, C2; //Not OK

11. C2 extends C1 implements I1, I2; //OK

- If interfaces having method with same name then sub class can override it only once.

When to use abstract class and interface?

- If "is-a" realationship exist between super type & sub type and if we want to provide same method design in all the sub classes then we should declare super type abstract.
- Using abstract class, we can group objects/instances of related types together.

```
Shape[] arr = new Shape[ 3 ];
arr[ 0 ] = new Rectangle();
arr[ 1 ] = new Circle();
arr[ 2 ] = new Triangle();
```

- Abstract class can extends only one class(abstract/concrete).
- We can write constructor inside abstract class.
- Abstract class may/may not contain abstract method.
- If "is-a" realationship is not exist between super type & sub type and if we want to provide same method design in all the sub classes then we should declare super type interface.
- Using interface, we can group instances of unrelated type together.

```
Printable[] arr = new Printable[ 3 ];
arr[ 0 ] = new Complex();
arr[ 1 ] = new Date();
arr[ 2 ] = new Point();
```

- Interface can extends more than one interfaces.
- We can not define constructor inside interface.
- Interface methods are by default abstract.

Adapter class

- Abstract helper class, which allows us to override some of the methods of interface is called Adapter class.

```
interface A
{
    void f1();
    void f2();
    void f3();
}
abstract class B implements A //Adpater class
```

```
{
    @Override
    public void f1() {  }
    @Override
    public void f2() {  }
    @Override
    public void f3() {  }
}
```

Comparable and Comparator implementation

- If we want to sort array of value type using Arrays.sort() then sort() method implicitly use "Dual-Pivot Quicksort" algorithm.
- Comparable is interface declared in java.lang package.
- "int compareTo(T other)" is a method of Comparable interface.
- If we want to sort array of instances of same type then reference type must implement Comparable interface.
- compareTo() method returns integer value:
    - Returns a negative integer, zero, or a positive integer as current object is less than, equal to, or greater than the specified object.
- If we use Arrays.sort() method to sort array of instances of reference type then sort() method implicitly use "iterative mergesort" algorithm.
- Comparator is interface declared in java.util package.
- "int compare(T o1,T o2)" is a method of Comparator interface.
- If we want to sort array of instances of same type as well different type then we should use Comparator interface.
- compare method returns integer value.
    - Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.
- If any class implements Comparable interface then it is considered as sortable.
- All the wrapper classes implements Comparable interface.