Day 7

Exception Handling

try with resource

```
try( Scanner sc = new Scanner(System.in))
{
    System.out.print("Number    :       ");
    int number = sc.nextInt();
    System.out.println("Number  :       "+number);
}
```

- If we use resource with try then it is not necessary to write finally block. In this case, JVM implicitly gives call to the close function.

throws clause

- If we want to delegate exception(checked/unchecked) from one method to another method then we should use throws clause.

```
public static void printRecord( ) throws InterruptedException
{
    for( int count = 1; count <= 10; ++ count )
    {
        System.out.println("Count: "+count);
        Thread.sleep(250);
    }
}
public static void main(String[] args)
{
    try
    {
        Program.printRecord();
    }
    catch (InterruptedException e)
    {
        e.printStackTrace();
    }
}
```

Custom Exception

- JVM can understand exceptional conditions that is occurred in business logic. If we want to handle such situations then we should write custom exception class.

- If we want to define custom unchecked exception class then we should extend the class from java.lang.RuntimeException class.

```
class StackOverflowException extends RuntimeException
{   }
```

- If we want to define custom checked exception class then we should extend the class from java.lang.Exception class.

```
class StackOverflowException extends Exception
{   }
```

Exception Chaining

- Generally exceptions are handled by throwing new type of exception. It is called exception chaining
- If we want trace any application then we should use exception chaining.

Bug

- If runtime error gets generated due to application developer's mistake then it is considered as bug.
- Example:
    1. NullPointerException
    2. ArrayIndexOutOfBoundsException
    3. ClassCastException
- We should not provide try catch block to handle bug rather we should find out cause of the bug.

Exception

- If runtime error gets generated due to end users mistake then it is considered as Exception.
- Example:
    1. ClassNotFoundException
    2. FileNotFoundException
- We should provide try catch block to handle exception

Error

- If runtime error gets generated due to enviromental condition then it is considered as error.

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() {      }
}
```

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.