Day 6

Custom exception

- SUN/ORACLE has given readymade exception classes for exceptional conditions which are already known to JVM
- e.g
    1. NullPointerException
    2. NumberFormatException
- JVM can not understand exceptional conditions/situations occurs in business logic. If we want to handle such situations then we should define Custom/User defined exception.
- If we want to define checked exception class then we should extend class from java.lang.Exception class and If we want to define unchecked exception class then we should extend class from java.lang.RuntimeException class
- Example : Custom Checked Exception

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

- Example : Custom UnChecked Exception

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

Abstract class and method

- abstract is keyword in java.
- According to clients requirement, if implementation of super class method is logically 100% incomplete then we should declare such method abstract.
- Abstract method do not contain body.
- If we declare any method abstract then it is mandatory to declare that class abstract.
- We can not instantiate abstract class but we can create reference of abstract class.
- Without declaring method abstract, we can declare class abstract. In other words, abstract class can contain:

1. Abstract method
2. Concrete method(static or non static)
3. Field
4. Constructor

- Example:
    1. java.lang.Number
    2. java.lang.Enum

3. java.util.Dictionary
4. java.util.Calendar

- It is mandatory to override abstract in sub class otherwise sub class can be considered as abstract.

```
abstract class A
{
    public abstract void f1();
}
abstract class B extends A //OK
{   }
class C extends A
{
    @Override
    public void f1( )   //OK
    {   }
}
```

- If implementation of a method is logically 100% complete then we should declare such method final.
- Final method inherit into sub class but we can not override it into sub class.
- We can declare overriden method final.
- e.g
    1. getClass()
    2. wait()
    3. notify()
    4. notifyAll()
- If implementation of a class is logically 100% complete then we should declare such class final.
- We can not extend final class i.e we can not create sub class of final class.
- Example
    1. java.lang.System
    2. java.lang.String, StringBuffer, StringBuilder
    3. All Wrapper classes
    4. java.lang.Math
    5. java.util.Scanner
- instanceof is operator in java which returns boolean value.
- If we want to check inheritance at runtime then we can use instanceof operator

```
private static void acceptRecord(Shape shape)
{
    if( shape instanceof Rectangle )
    {
        Rectangle rect = (Rectangle) shape;     //downcasting
    }
    else
    {
        Circle c = (Circle) shape;      //downcasting
    }
}
```

String Handling

- In C/C++, string is array of character which ends with '\0' character.
- In java String is collection of character object which do not ends with '\0' character.

```
//String str = "SunBeam";
//StringBuffer str = new StringBuffer("SunBeam");
StringBuilder str = new StringBuilder("SunBeam");
char ch = str.charAt(str.length()); //StringIndexOutOfBoundsException
```

- If we want to manipulate string then we can use following classes

1. java.lang.String
2. java.lang.StringBuffer
3. java.lang.StringBuilder
4. java.util.StringTokenizer
5. java.util.regex.Pattern
6. java.util.regex.Matcher

java.lang.String

- It is a final class declared in java.lang package.
- It implements Serializable, Comparable and CharSequence inteface.
    - Serializable is a marker/tagging interface.
    - Comparable is a interface havin "compareTo()" method
    - Following are abstract methods of CharSequence interface
        1. char charAt(int index)
        2. int length()
        3. CharSequence subSequence(int start, int end)
- String is not a built-in / primitive type. It is a class hence considered as reference type.
- We can create instance of String with and without new operator.

```
String str = new String("SunBeam Karad");
```

- "new String("SunBeam Karad")" -> String instance.
- String instance get space on heap section.

```
String str = "KDac";
```

is equivalent to

```
char[] arr = new char[ ]{ 'K','D','a','c'};
String str = new String( arr );
```

- "KDac" -> String literal.
- String lietrals get space on String literal pool / String pool.
- In java, String objects are constant/immutable. It means that, if we try to modify state of String instance then JVM create new instance of it.
- "public String concat(String str)" is a method of String class which is used to concat two strings.

```
String s1 = "Pre";
String s2 = "cat";
String s3 =  s1.concat(s2); //Precat
```

```
String str = "SunBeam";
str = str.concat( 12345 );      //Not OK
```

- If we want to concat state of instance of any primitive as well non primitive to String then we should use "+" operator.

```
String str = "SunBeam";
//str = str + 123;      //OK
str = str + new Date();//OK
```

- Constructor's of String

1. public String()

```
String str = new String( );
```

2. public String(char[] value)

```
char[] data = { 'A','B', 'C'};
String str = new String( data );
```

3. public String(byte[] bytes)

```
byte[] bs = { 65, 66, 67 };
String str = new String( bs );
```

4. public String(String original)

```
String s1 = "Karad";
String s2 = new String( s1 );
```

5. public String(StringBuffer buffer)

```
StringBuffer sb = new StringBuffer("Sandeep");
String str = new String( sb );
```

6. public String(StringBuilder builder)

```
StringBuilder sb = new StringBuilder("SunBeam");
String str = new String( sb );
```

- Methods of String Class

1. public char charAt(int index)
2. public int codePointAt(int index)
3. public int compareToIgnoreCase(String str)
4. public String concat(String str)
5. public boolean contains(CharSequence s)
6. public boolean startsWith(String prefix)
7. public boolean endsWith(String suffix)
8. public boolean equalsIgnoreCase(String anotherString)
9. public static String format(String format, Object... args)
10. public byte[] getBytes()
11. public String intern()
12. public boolean isEmpty()
13. public int indexOf(String str)
14. public int lastIndexOf(String str)
15. public int length()
16. public boolean matches(String regex)
17. public String[] split(String regex)
18. public String substring(int beginIndex)
19. public char[] toCharArray()
20. public String toLowerCase()
21. public String toUpperCase()
22. public String trim()
23. public static String valueOf(Object obj)

```
public static void main(String[] args)
{
```

```
    String str = "SunBeam Infotech Pune";
    //String subStr =  str.substring(8);        //Infotech Pune
    String subStr =  str.substring(8, 16);      //Infotech
    System.out.println(subStr);
}
```

```
public static void main(String[] args)
{
    String regex = " ";
    String str = "SunBeam Infotech Pune";
    String[] words = str.split(regex);
    for (String word : words)
        System.out.println(word);
}
```

String twisters

```
String s1 = new String("SunBeam");
String s2 = new String("SunBeam");
if( s1 == s2 )
    System.out.println("Equal");
else
    System.out.println("Not Equal");
//Output : Not Equal
```

```
String s1 = new String("SunBeam");
String s2 = new String("SunBeam");
if( s1.equals(s2) )
    System.out.println("Equal");
else
    System.out.println("Not Equal");
//Output : Equal
```

```
String s1 = new String("SunBeam");
String s2 = "SunBeam";
if( s1 == s2 )
    System.out.println("Equal");
else
    System.out.println("Not Equal");
//Output : Not Equal
```

```
String s1 = new String("SunBeam");
String s2 = "SunBeam";
if( s1.equals(s2) )
    System.out.println("Equal");
else
    System.out.println("Not Equal");
//Output : Equal
```

```
String s1 = "SunBeam";
String s2 = "SunBeam";
if( s1 == s2 )
    System.out.println("Equal");
else
    System.out.println("Not Equal");
//Output : Equal
```

```
String s1 = "SunBeam";
String s2 = "SunBeam";
if( s1.equals(s2))
    System.out.println("Equal");
else
    System.out.println("Not Equal");
//Output : Equal
```

```
String s1 = "Sun"+"Beam";       //SunBeam
String s2 = "SunBeam";
if( s1 == s2)
    System.out.println("Equal");
else
    System.out.println("Not Equal");
//Output : Equal
```

```
String str1 = "SunBeam";
String str = "Sun";
String str2 = str + "Beam";
if( str1 == str2 )
    System.out.println("Equal");
else
    System.out.println("Not Equal");
//Output : Not Equal
```

```
String str1 = "SunBeam";
String str = "Sun";
String str2 = (str + "Beam").intern();
if( str1 == str2 )
    System.out.println("Equal");
else
    System.out.println("Not Equal");
//Output : Equal
```

- Enumeration is interface declared in java.util package.
- Methods of Enumeration:
    1. boolean hasMoreElements()
    2. E nextElement()
- On the basis of delimeter, if we want to split tokens/string then we should use StringTokenizer class.
- It is declared in java.util package.
- Constructors of StringTokenizer

1. public StringTokenizer(String str)
2. public StringTokenizer(String str, String delim)
3. public StringTokenizer(String str, String delim, boolean returnDelims);

- Methods of StringTokenizer

1. public int countTokens()
2. public boolean hasMoreTokens()
3. public String nextToken()

```
String str = "SunBeam Infocom Karad";
StringTokenizer stk = new StringTokenizer(str);
System.out.println(stk.countTokens());  //3
```

```
String str = "SunBeam Infocom Karad";
StringTokenizer stk = new StringTokenizer(str);
String token = "";
while( stk.hasMoreElements() )
{
    token = (String) stk.nextElement();
    System.out.println(token);
}
```java
String str = "SunBeam Infocom Karad";
StringTokenizer stk = new StringTokenizer(str);
String token = "";
```
```

```
while( stk.hasMoreTokens() )
{
    token = stk.nextToken();
    System.out.println(token);
}
```

```
String str = "www.gmail.com";
String delim = ".";
StringTokenizer stk = new StringTokenizer(str,delim);
String token = "";
while( stk.hasMoreTokens() )
{
    token = stk.nextToken();
    System.out.println(token);
}
```

```
String str = "ab+bc*cd-de/ef";
String delim = "+*-/";
StringTokenizer stk = new StringTokenizer(str,delim);
String token = "";
while( stk.hasMoreTokens() )
{
    token = stk.nextToken();
    System.out.println(token);
}
```

StringBuffer and StringBuilder

- Both are final classes declared in java.lang package.
- Both are used to create mutable string object/instance
- If we want to create instance StringBuffer and StringBuilder then it is mandatory to use new operator.
- equals() and hashCode() method is not overriden in these classes.

```
StringBuffer sb1 = new StringBuffer("DAC");
StringBuffer sb2 = new StringBuffer("DAC");
if( sb1 == sb2 )
    System.out.println("Equal");
else
    System.out.println("Not Equal");
//Output : Not Equal
```

```
StringBuffer sb1 = new StringBuffer("DAC");
StringBuffer sb2 = new StringBuffer("DAC");
if( sb1.equals(sb2) )
```

```
    System.out.println("Equal");
else
    System.out.println("Not Equal");
//Output : Not Equal
```

```
StringBuilder sb1 = new StringBuilder("KDAC");
StringBuilder sb2 = new StringBuilder("KDAC");
if( sb1 == sb2 )
    System.out.println("Equal");
else
    System.out.println("Not Equal");
//Output : Not Equal
```

```
StringBuilder sb1 = new StringBuilder("KDAC");
StringBuilder sb2 = new StringBuilder("KDAC");
if( sb1.equals(sb2) )
    System.out.println("Equal");
else
    System.out.println("Not Equal");
//Output : Not Equal
```

- Methods of StringBuffer and StringBuilder

1. public StringBuffer append(String str)
2. public char charAt(int index)
3. public StringBuffer delete(int start, int end)
4. public int length()
5. public StringBuffer reverse()
6. public String substring(int start)

Pattern and Matcher

- If we want to validate String then we should use regular expression.
- If we want to use regular expression to validate String then we should use Pattern and Matcher class.
- these classes are declared in java.util.regex package.

```
EMAIL_PATTERN = "^[_A-Za-z0-9-\\+]+(\\.[_A-Za-z0-9-]+)*@[A-Za-z0-9-]+(\\.
[A-Za-z0-9]+)*(\\.[A-Za-z]{2,})$";

PHONE_PATTERN = "\\d{10}";

NAME_PATTEREN = "[a-zA-Z][a-zA-Z ]*";
```

Nested class

- If we define class inside scope of another class then it is called nested class.

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

- Access modifier of top level can be either package level private or public only . We can use any access modifier on nested class.
- Using nested class we can achieve encapsulation.
- There are 2 types of nested class:
    1. Inner class
    2. Static nested class.

Inner Class

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

```
class Outer
{
    class Inner
    {   }
}
```

- Instantiation of Top level class

```
Outer out = new Outer();
```

- Instantiation of Inner class(Method - I)

```
Outer out = new Outer();
Outer.Inner in = out.new Inner();
```

- Instantiation of Inner class(Method - II)

```
Outer.Inner in = new Outer().new Inner();
```

- Using instance, we can access members of inner class inside method of top level class.
- Without instance, we can access members of top level class inside method of non static nested class / inner class.

```
class Outer
{
        private int num1 = 10;
        class Inner
        {
                private int num1 = 20;
                public void print( )
                {
                        int num1 = 30;
                        System.out.println("Num1        :
"+Outer.this.num1);             //10
                        System.out.println("Num1        :
"+this.num1);   //20
                        System.out.println("Num1        :       "+num1);
//30
                }
        }
}
public class Program
{
        public static void main(String[] args)
        {
                Outer.Inner in = new Outer().new Inner();
                in.print();
        }
}
```

Static nested class

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

```
class Outer
{
    static class Inner
    {   }
}
```

- Instantiation of Top level class

```
Outer out = new Outer();
```

- Instantiation of static nested class

```
Outer.Inner in = new Outer.Inner();
```

- Using instance, we can access members of static nested class inside method of top level class.
- We can access static members of top level class inside method of static nested class directly. But If we want to access non static members of top level class inside method of static nested class then we must use instance of top level class.

Local Class

- If we define class Inside method then it is called local class.
- In java local class is also called as method local inner class.
- We can not create reference / instance of local class outside method.
- Types of local class
    1. Method local inner class.
    2. Method local anonymous inner class.

Method local inner class.

```
public class Program //Program.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 "Complex [real=" + real + ", imag="
+ imag + "]";
                        }
                }
                Complex c1 = new Complex();
                System.out.println(c1.toString());
        }
}
```

Method local anonymous inner class.

- In java, we can define class without name. Such class anonymous class.

- We can create anonymous class inside method only hence it is also called method local anonymous inner class.

```
Object obj = new Object() //Program$1.class
             {
                 String message = "Hello";
                 @Override
                 public String toString()
                 {
                     return message;
                 }
             };
System.out.println(obj.toString());
```