String Handling:

String is a collection of characters with in the double quotes.

Based on the character we can classify String into three types.

1) Numerical String
2) Character String
3) Special character String.

Numerical String:

If we are placing digits within the double quotes is called Numerical String.

Ex: String s = “123”;

Character String:

If we are placing character with in the double quotes is called Character String.

Ex: String s = “jagadesh”;

Special Characters String:

If we are placing any special characters with in the double quotes is called Special Character String.

Ex: String s = “$%^&*”

From above information whatever the character we are writing within the double quotes that are comes under String.

We can write one or more character or combination of characters.(Digit, characters, special characters) within the double quotes.

String s = “R”;

String s = “123abce%^&*”;

Java introducing String is an predefine class, which is available in java.lang package.

We can use String as a data type also.

Objects are classified into two types.

1) Immutable

2) Mutable.

Immutable Objects:

The value of an object has not changeable within the same memory; instead of changing it creates new object is called immutable.

Ex: All Wrapper classes (Byte, Short, Integer, Long, Float, Double, Character, Boolean), java.lang.String .

Mutable:

The value of an Object has been changed with in the memory is called mutable object.

Ex: java.lang.StringBuffer, java.lang. StringBuilder

If we want to check any two object in java we following ways.

1) By using “==” operator

2) By using equals().

1) “==” operator is always checks hashCode of an objects.

2) equals() of java.lang.Object class is also checks hashCode of an objects.

If we want to checks the content of objects then we can go for override the method within the String class. Java software developer has already override equals () in java.lang.String class.

The equals () of String class, is going to be check the content of an object.

In the java.lang.StringBuffer class equals() is not override.

Difference between

1) String s = “ram” and

2) String s1 = new String(“ram”);

In the first statement JVM will create only one object in String Constant Pool area.

In the second statement JVM will create two objects.

a) One is in Heap Area

b) Second is in String Constant Pool.

In the second statement reference (s1) is always points Heap Area object only.

Before creating object in String Constant Pool, JVM checks is there any object existed with “ram” content or not if not then it will create “ram” in SCP also. If object is existed JVM won’t create any object in SCP, using the existed object only.

If any object is not having any external reference, those objects are easily garbage (de-allocate).

But in SCP object some time don’t have any external reference, these are not garbage.

The objects which are available in SCP are strong objects.

String Constant Pool:

If we want to use the same objects repeated times, then place those

Object’s within the SCP only.

With the help of SCP we will get bellow advantages.

1) Memory utilization

2) Performance increase.

All the objects having same content then JVM is not create new object, JVM uses existed object memory only.

If anyone objects is doing updating, that updated (new) value is not affected to remaining objects, that updated value effected to only one object.

That JVM is not change the content of existed memory; instead it will create new memory with updated value.

SCP objects are very strong objects.

For each and every string we have memory in SCP.

intern (): If we want interact with the SCP data, then we have one method that intern().

```
public class CollectionDemo {
     public static void main(String[] args){
          String s = "ram";
          System.out.println("s: 
"+System.identityHashCode(s));
          s = "ram"+"chandra";
          System.out.println("s: 
"+System.identityHashCode(s));
          StringBuffer sb1 = new StringBuffer("ram");
          System.out.println("sb1: 
"+System.identityHashCode(sb1));
          sb1.append("chandra");
          System.out.println("sb1: 
"+System.identityHashCode(sb1));
     }
}
```

```
public class StringDemo {

     public static void main(String[] args) {

          String s1 = "ram";

          String s2 = new String("ram");

          System.out.println(s1==s2);

          System.out.println(s1.equals(s2));

          String s1 = "ram";

          String s2 = "ram123";

          s1.concat("123");

          System.out.println(s1);

          s1=s1.concat("123");

          System.out.println(s1);

          System.out.println(s1==s2);
```

```
s1="ram"+"123";
System.out.println(s1==s2);


String s1 = "ram123";
String s2 = new String("ram");
System.out.println(s1==s2);
s2.concat("123");
System.out.println(s2);
s2=s2.concat("123");
System.out.println(s2);
System.out.println(s1==s2);
s2="ram"+"123";
System.out.println(s2==s1);
String s1 = "CoreJava";
String s2 = new String("AdvancedJava");
String s3 = "AdvancedJava";
String s4 = s2.intern();
System.out.println(s3==s4);
String s1 = "corejava";
String s2 = "COREJAVA";
String s3 = s1.toUpperCase();
System.out.println(s2==s3);
```

```
String s4 = s3.toUpperCase();
System.out.println(s3==s4);
System.out.println(s2==s4);
String s1 = "ram";
String s2 = "Ram";
System.out.println(s1==s2);
System.out.println(s1.equals(s2));
System.out.println(s1.equalsIgnoreCase(s2));
String s1 = new String ("java is an oopl");
String s2 = new String ("java is an oopl");
System.out.println(s1==s2);
String s3 = "java is an oopl";
System.out.println(s1==s3);
String s4 = "java is an oopl";
System.out.println(s3==s4);
String s5 = "java is "+"an oopl";
System.out.println(s4==s5);
String s6 = "java is ";
String s7 = s6+"an oopl";
System.out.println(s4==s7);
final String s8 = "java is ";
String s9 = s8+"an oopl";
```

```
        System.out.println(s4==s9);
    }
}
```

Note: If we are doing modification on string by using reference(either SCP reference or heap area reference) new object will be created in the heap area.

String s1 = “ram”

s1.concat(“123”);(new object is available in heap area)

If we are doing modification on string by using content then new object will be created in the SCP.

S1=”ram”+”123”;(new object is created in the scp).

Note: With the help of intern() we can interact with unreferenced SCP data.

```
public class StringDemo { //extends String {
    public static void main(String[] args) {
        String s = "internationalization";
        System.out.println(s.length());
        String s1="A";
        String s2 = "a";
        System.out.println(s1.compareTo(s2));
        System.out.println(s2.compareTo(s1));
        StringBuffer sb1 = new StringBuffer("a");
StringBuffer sb2 = new StringBuffer("b");
//System.out.println(sb1.compareTo(sb2));
s=s.concat("--->java");
System.out.println(s);
System.out.println(s1.equals(s2));
System.out.println(s.startsWith("inter"));
System.out.println(s.endsWith("inter"));
System.out.println(s1.equalsIgnoreCase(s2));
byte b[] = s1.getBytes();
System.out.println(b[0]);
int i = 100;
int j = 200;
String s3 = String.valueOf(i);
String s4 = String.valueOf(j);
String s5 = String.valueOf('d');
System.out.println(i+j);
System.out.println(s3+s4+s5);
System.out.println(s.lastIndexOf("n"));
System.out.println(s.lastIndexOf("nation"));
System.out.println(s.indexOf('t'));
System.out.println(s.indexOf('n'));
String s6 = "he";
```

```
        System.out.println(s6.replace('e', 'i'));

        System.out.println(s.replaceAll("nation", "india"));

        String s7 = "java is an oopl";

        String s8[] = s7.split(" ");

        for(String s9: s8){

            System.out.println(s9);

        }

        String s10 = "10,20,30,40,50";

        int total = 0;

        String s11[] = s10.split(",");

        for(String s12: s11){

            System.out.println(s12);

            total = total+Integer.parseInt(s12);

        }

        System.out.println(total);


        //we cannot convert string value to character value

        //we dont have Character.parseChar().

    }

}public class StringHandling {
    public static void main(String[] args) {
        String s1 = "ram chandra";
        System.out.println("Number of characters: "
                                +s1.length());
        System.out.println("checking for data existed or not: "
                                +s1.isEmpty());
        String s2 = "";
        System.out.println("checking for data existed or not: "
                                     +s2.isEmpty());
        //System.out.println("character at specific index
position: "+s2.charAt(4));
        System.out.println("character at specific index position:
"
                                     +s1.charAt(4));
System.out.println("character ascii value at specific index
position: "+
                                     s1.codePointAt(4));
System.out.println("character ascii value at previous index" +
        " of specific index position: "+s1.codePointBefore(4));
        char[] c = new char[10];
        String s3 = "internationalization";//20characters
        s3.getChars(5,11,c,0);
        for(int i=0;i<c.length;i++){
            System.out.println(c[i]);
        }
        System.out.println("=============");
        byte[] b = new byte[10];
        s3.~~getBytes~~(5,11,b,4);
        for(int i=0;i<b.length;i++){
            System.out.println(b[i]);
        }
        System.out.println("==================");
        b= s3.getBytes();
        for(int i=0;i<b.length;i++){
            System.out.println(b[i]);
        }
        System.out.println("================");
        System.out.println("ram".equals("Ram"));
        System.out.println("ram".equalsIgnoreCase("Ram"));

        System.out.println("ram".compareTo("Ram"));//114-82
```

```
        System.out.println("ram".compareTo("ramchandra"));//0-
    number of characters(0-7==>-7)

        System.out.println("ram".compareToIgnoreCase("Ram"));//
    114-82
            System.out.println(s3.startsWith("on"));
            System.out.println(s3.startsWith("in"));
            System.out.println(s3.endsWith("on"));
            System.out.println(s3.endsWith("in"));
            System.out.println("AB".hashCode());

            System.out.println(s3.indexOf('n'));
            System.out.println(s3.indexOf('n',11));
            System.out.println(s3.indexOf("on"));
            System.out.println(s3.indexOf("on", 10));

            System.out.println(s3.lastIndexOf('n'));
            System.out.println(s3.lastIndexOf("in"));
            System.out.println("*****"+s3.lastIndexOf('n', 8));
            System.out.println(s3.substring(5));
            System.out.println(s3.substring(5, 11));

            s1.concat(" nit");
            System.out.println(s1);
            s1= s1.concat(" nit");
            System.out.println(s1);

            System.out.println("nitn".replace('n', 'k'));
            System.out.println("ram chandra
    ram".replaceFirst("ram", "java"));
            System.out.println("ram chandra
    ram".replaceAll("ram", "java"));

            String s4 = "naresh i technologies";
            String[] s5 = s4.split(" ");
            for(int i=0;i<s5.length;i++){
                System.out.println(s5[i]);
```

```
		}
		char[] c3 = new char[10];
		c3 = "ram".toCharArray();
		for(int i=0;i<c3.length;i++){
			System.out.println(c3[i]);
		}
		String s6 = "     varun kirn ram sam yaane    ";
		System.out.println("s6: "+s6);
		String s7= s6.trim();
		System.out.println("s7: "+s7);


		System.out.println("ram".toUpperCase());
		System.out.println("VARUN".toLowerCase());
	}
```

Difference between String and StringBuilder and StringBuffer

String:

- Immutable
- Java 1.0
- Not synchnoized
- Concat()
- compareTo()
- Not ThreadSafe
- Higher performance
- Not good at result
- Implements java.lang.Comparable
- No reverse()
- No capacity()

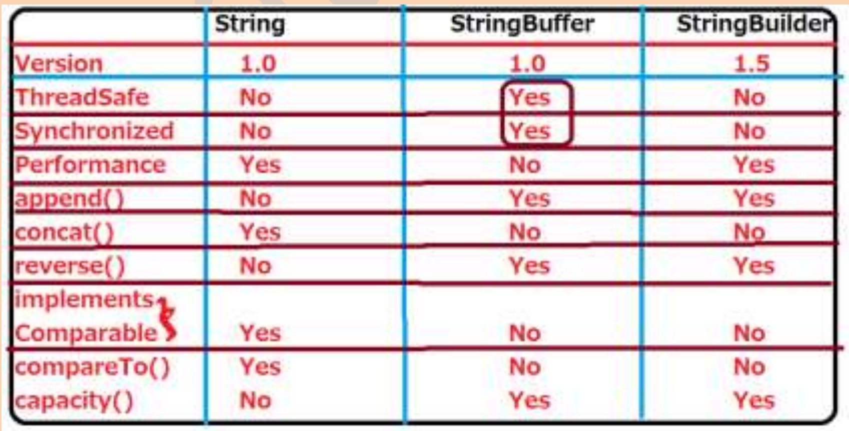
StringBuffer:

- mutable

- Java 1.0
- synchnoized
- append
- NO compareTo()
- ThreadSafe
- low performance
- good at result
- not Implements java.lang.Comparable
- reverse()
- capacity()

StringBuilder:

- mutable
- Java 1.5
- Not synchnoized
- append
- NO compareTo()
- Not ThreadSafe
- Higher performance
- Not good at result
- Not Implements java.lang.Comparable
- reverse()
- capacity()



	String	StringBuffer	StringBuilder
Version	1.0	1.0	1.5
ThreadSafe	No	Yes	No
Synchronized	No	Yes	No
Performance	Yes	No	Yes
append()	No	Yes	Yes
concat()	Yes	No	No
reverse()	No	Yes	Yes
implements Comparable	Yes	No	No
compareTo()	Yes	No	No
capacity()	No	Yes	Yes

```
public class StringHandling {
    public static void main(String[] args) {
        String s = "ram";
        String s1 = new String();
        System.out.println(s);
        System.out.println(s1);
        String s2 = new String("ramchandra");
        System.out.println(s2);
        String s3 = new String("");
        System.out.println(s3+"***");
        byte b[] = {65,66,67,68,69,70,71,72};
        String s4 = new String(b);
        System.out.println(s4);
        String s5 = new String(b,2,4);
        System.out.println(s5);
        char c[] = {'a','b','c','d','e','f'};
        String s6 = new String(c);
        System.out.println(s6);
        String s7 = new String(c,1,3);
        System.out.println(s7);
        StringBuffer sb  = new StringBuffer("standard");
        String s8 = new String(sb);
```

```
                System.out.println(s8);

                StringBuilder sb1 = new StringBuilder("Edition");

                String s9 = new String(sb1);

                System.out.println(s9);

        }

}
```

Note: If we want to use the compareTo() , both objects must be homogeneous and both objects must be implements java.lang.Comparable.

```
public class StringHandling {
        public static void main(String[] args) {
                String s1 = "internationalization";
                System.out.println("length: "+s1.length());
                System.out.println(s1.charAt(5));
                System.out.println(s1.codePointAt(5));
                System.out.println(s1.codePointBefore(5));
                System.out.println(s1.concat(" java"));
                System.out.println("ram".compareTo("Ram"));//r-
R=114-82
                System.out.println("Ram".compareTo("ram"));
                System.out.println("Ram".compareTo("Ram"));
                System.out.println("Ram".compareTo("RamChandra"));

        System.out.println("ram".compareToIgnoreCase("Ram"));
                System.out.println(s1.endsWith("m"));
                System.out.println(s1.endsWith("on"));
                System.out.println("ram".equals("Ram"));
                System.out.println("ram".equalsIgnoreCase("Ram"));
                System.out.println(s1.indexOf('n'));
                System.out.println(s1.indexOf("nation"));
                System.out.println(s1.lastIndexOf('n'));
System.out.println(s1.indexOf("on"));
System.out.println(s1.lastIndexOf("on"));
System.out.println(s1.indexOf('n',6));
System.out.println(s1.indexOf("on",10));
System.out.println(s1.isEmpty());
//String s2 =" ";
String s2 ="";
System.out.println(s2.isEmpty());
String s3 = "AB";
System.out.println(s3.hashCode());
System.out.println(System.identityHashCode(s3));

char a[] = new char[10];
String s4 = "nation";
a=s4.toCharArray();
for(char a1: a){
	System.out.println(a1);
}
System.out.println("VARUN".toLowerCase());
System.out.println(s4.toUpperCase());
String s5 = "    kit nit ramchandra java      ";
System.out.println("s5: "+s5);
String s6 = s5;
System.out.println("s6: "+s6);
String s7 = s5.trim();
System.out.println("s7: "+s7);
String s8 = "ram sam nit kit vikram";
String s9[] = s8.split(" ");
System.out.println(s8);
for(String s10: s9){
	System.out.println(s10);
}
String s10="naresh i technologies";
String s11 = s10.replace('i', 'I');
System.out.println(s11);
String s12 = s10.replaceAll("naresh", "karthik");
System.out.println(s12);
```

```
        }
}
```

JVM ARCHITECTURE:

Virtual Machine:

VM is an software/Application which will provide environment or memory areas for executing the programs.

JVM: JVM is an software/application which will provide environment or five individual identical memory areas for executing the java programs.

We have two types of JVM

1) Client JVM

2) Server JVM.

JVM is only provides abstraction.

JRE is provides implementation for that abstraction.

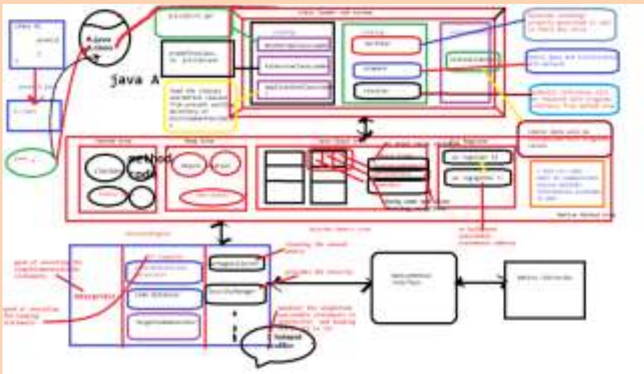
We have different JVM'S for different OS.

That means we have different JRE software for different Operating Systems.

If we want to execute the java program, first we should interact with the JVM.

With the help of "java" command we communicate with JVM. "java" keyword is always followed by class name.

syntax: java Class_Name



Java is internally communicate the with JVM, then JVM will come into the picture, first JVM will read class name and JVM control will go to secondary memory, that class byte code will be loaded from secondary memory to primary memory.

JVM internally uses ClassLoaderSubSystem to load the .class files from secondary memory to primary memory.

In the ClassLoaderSubSystem, we have three separate phases. Those are

1) Loading phase.

2) Linking phase.

3) Initialization phase.

Loading Phase: In this phase the byte code (predefine and user define class) will be loaded from secondary memory to primary with the help of bellow classloaders.

In this phase we have three ClassLoaderSubSystems.

1. ApplicationClassLoaderSubSystem
2. ExtensiveClassLoaderSubSystem
3. BootstrapClassLoaderSubSystem.

ApplicationClassLoaderSubSystem:

It will loads user define class files from current directly and environment variable path.

ExtensiveClassLoaderSubSystem:

It will loads predefine class files from jre\lib\ext folder.

BootstrapClassLoaderSubSystem:

It will load predefine class files from jre\lib\rt.jar file.

In the above three phases class is not available, then we will get one error cannot find or load main class <class_name>.

If available that .class file will loaded from secondary memory to primary memory and handover to linking phase.

Linking:

In this phase the loaded byte code will be checked by verifier.

In this phase we have three components.

1. Verifier
2. Preparer
3. Resolver.

Verifier will check whether the byte code is properly organized or not, is there any virus and hacking code or not. If yes verifier will give verifier error, if not that byte code will be handover to preparer.

Preparer will provide the default values to static variables in loading phase.

Resolver: it will convert symbolic reference into original references.

After this phase code will be handover to initializer.

Initialization: all default values of static data will be replaced with original or actual data.

Runtime Memory Areas:

JVM provides 5 runtime memory areas.

1) Method Area

2) Heap Area

3) Java Stack Area

4) PC Registers

5) Native Method Area

Method Area: In this area all class data is stored. that means all the static data is available in the method area.

Heap Area: in this area all object data is stored. that means all the non-static data is available in heap area.

All the objects are created in the heap area only.

Java Stack Area: Every thread have its own stack.

These stack will be interact with local data/method level data. Java stack have different slots, those slots are called stack frames.

Java Stack Frame will convert into three parts.

1) Local Variable Storage Area:

In this all local variables are stored.

2) Operand Stack:

in this phase all operations and calculations will be happens.

3) FrameData:

If method contains any exceptions, those exceptions will be thrown by frame data only.

PC Registers:

Every thread have its own PC Registers, these registers will hold the next execution statements address.

Native method area:

If java wants communicate with c and c++ code, that code will be available in Native method area.

If we interact with c and c++ code, we need libraries, those will be given by Native Library, that library will be handover by Native Method Interface to executable engine.

Execution Engine:

Internally JVM uses two translators to convert byte code to executable code.

1. Interpreter
2. JIT Compiler

Interpreter is good at execute the single time execution statements.

JIT compiler is good at execute the looping statements.

But these two are unable to find out behavior of statements.

In this one special component come into picture that is "profiler"

First profiler will identify the weather statements are single time or looping statements.

If single time execution statement then those statements handover to interpreter otherwise handover to JIT compiler.

Java is a high performance language, the reason java internally uses two translators to covert our byte code to executable. So java applications are executed with in the less time. If the application executed within the less time automatically the performance will increasers.

In this executable engine we have some other special components.

Those are Garbage Collector and Security manager.

Wrapper classes:

Representing the primitive data in the format of object is called wrapper class.

Wrapper class are wraps the primitive data in the format of object.

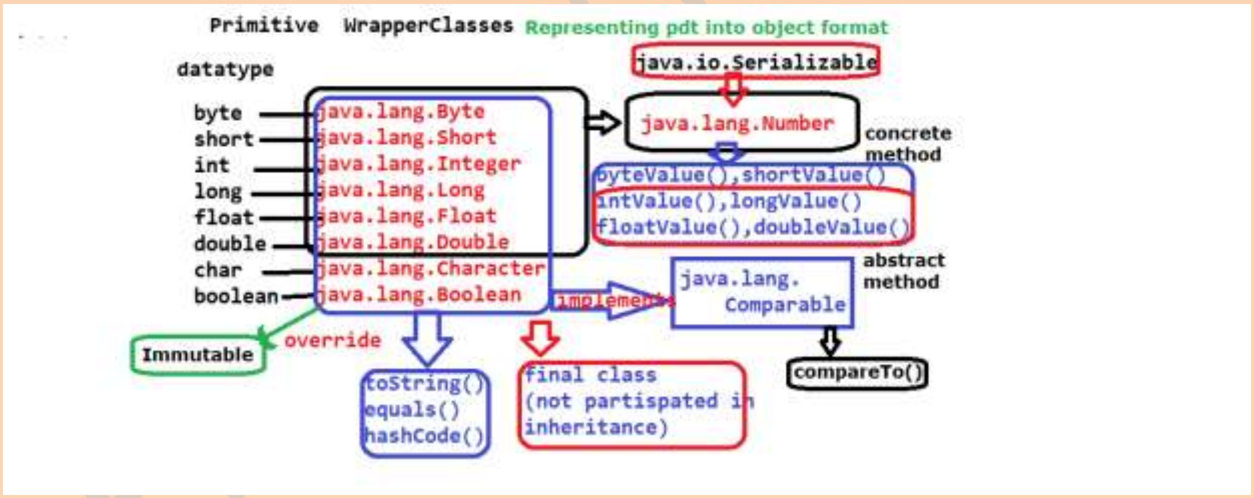
When we create an object to a wrapper class, it contains a field/area and in this filed, we can store a primitive data type.

Whenever we send the data throughout the networks we should always prefer to send the data in the form of object.

If we want use any predefine logic (method), we cannot call on top of primitive variable, but we can call on top of object.

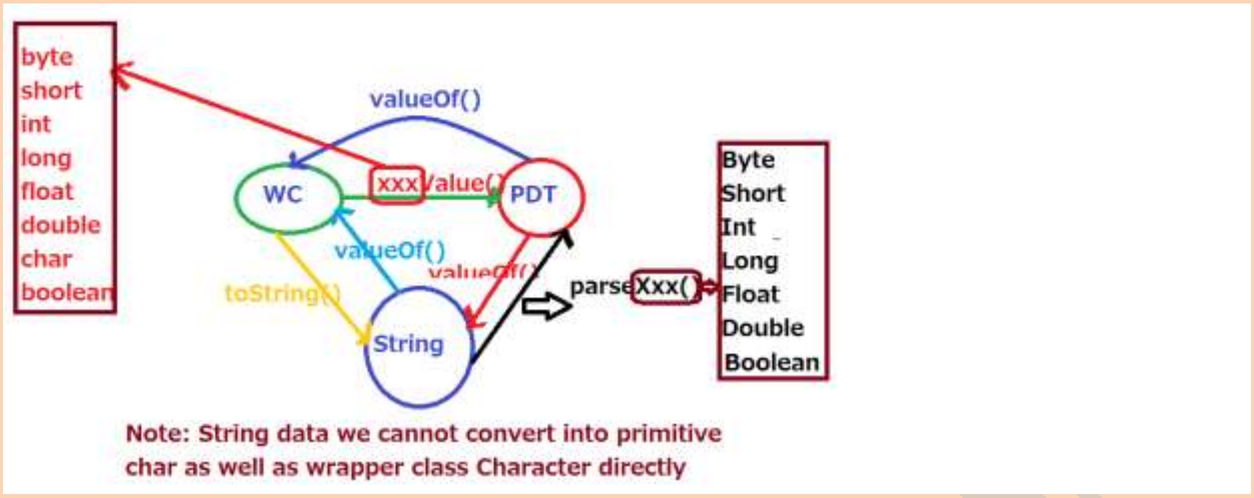
From above two conditions we need Wrapper class.

Primitive	wrapper class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean



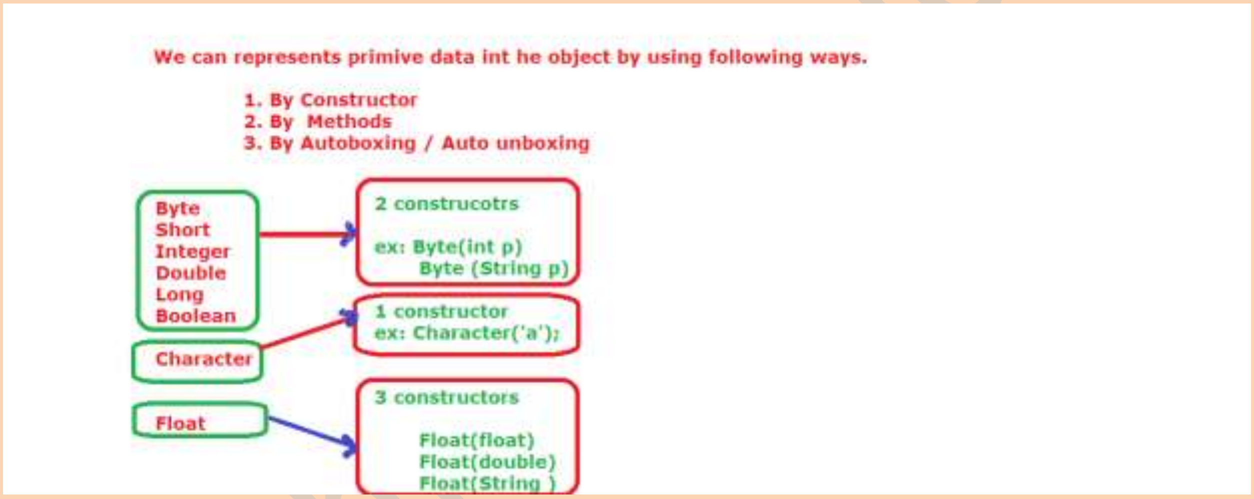
We can convert primitive data into wrapper class object in two ways.

1. By using constructors.
2. By using Methods.



Note: String data we cannot convert into primitive char as well as wrapper class Character directly

Constructors in Wrappers class:


We can represents primive data int he object by using following ways.

1. By Constructor
2. By Methods
3. By Autoboxing / Auto unboxing

1) java.lang.Byte:

    This class contains two constructors.

        Byte (byte value)--10

        Byte (String value)”10”

2) java.lang.Short

    This class contains two constructors.

        Short (short value)

        Short (String value)

3) java.lang.Integer

This class contains two constructors.

Integer (int value)

Integer (String value)

4) java.lang.Long

This class contains two Constructors

Long (long value)

Long (String value)

5) java.lang.Float

This class contains three constructors.

Float (float value)

Float (double value)

Float (String value)

6) java.lang.Double

This class contains two constructors.

Double (double value)

Double (String value)

7) java.lang.Character

This class contains only one constructor

Character (char value)

8) java.lang.Boolean

This class contains two constructors

Boolean (Boolean value)

Boolean (String value)

<u>Java.lang.Number:</u>

It is an abstract class.

It is the super class for Byte, Short, Integer, Long, Float, Double.

This class contains 4 abstract methods and two concrete methods.

The above 4 abstract methods are implemented in their sub classes (6)

It implements java.io.Serializable

Note:

All wrapper classes are implements Serializable and Comparable interfaces.

All the wrapper classes are immutable objects.

Note: All the wrapper class override the hashCode(), toString(), equals().

We can use '+' operator on top of wrapper class reference variables but not normal objects (references).

```
public class WrapperDemo {

        public static void main(String[] args) {

                byte b = 100;

                Byte a1 = new Byte(b);

                Byte a2 = new Byte("127");
```

```
            System.out.println(a1);
            System.out.println(a2);
            Short a3 = new Short((short)120);
            Integer a4 = new Integer(125);
            Long a5 = new Long(120);
            Float a6 = new Float("12.34f");
            System.out.println(a6);
            /*Long a7 = new Long("120l");
            System.out.println(a7);*/
            Double a8 = new Double(23.34d);
            System.out.println(a8);
            Double a9 = new Double("23.34d");
            System.out.println(a9);
            Character a10 = new Character('a');
            System.out.println(a10);
            Boolean a11 = new Boolean("false");
            System.out.println(a11);
            Boolean a12 = new Boolean("true1");
            System.out.println(a12);
        }
    }
```

```
public class WrapperDemo {


        public static void main(String[] args) {
                //Primitive data type to Wrapperclass type
                Byte a1  = Byte.valueOf((byte)100);
                Short a2 = Short.valueOf((short)122);
                Integer a3 = Integer.valueOf(100);
                Long a4 = Long.valueOf(123l);
                Float a5 = Float.valueOf(12.34f);
                Double a6 = Double.valueOf(12.34);
                Character a7 = Character.valueOf('a');
                Boolean a8 = Boolean.valueOf(false);
                System.out.println(a1+"...."+a2+"...."+a3+

        "...."+a4+"...."+a5+"...."+a6+"...."+a7+"...."+a8);


//Wrapper class data to prmitive datatype
                byte b1 = a1.byteValue();
                short b2 = a2.shortValue();
                int b3 = a3.intValue();
                long b4 = a4.longValue();
                float b5 = a5.floatValue();
        double b6 = a6.doubleValue();
        char b7 = a7.charValue();
        boolean b8 = a8.booleanValue();
                System.out.println(b1+"...."+b2+"...."+b3+

        "...."+b4+"...."+b5+"...."+b6+"..."+a7+"...."+a8);
//String data type to Wrapperclass type
                Byte a1  = Byte.valueOf("100");
                Short a2 = Short.valueOf("122");
                Integer a3 = Integer.valueOf("100");
                Long a4 = Long.valueOf("123l");//here l is not deleted
                //Long a4 = Long.valueOf("123");
                Float a5 = Float.valueOf("12.34f");
                Double a6 = Double.valueOf("12.34d");
                //Character a7 = Character.valueOf("a");
                Boolean a8 = Boolean.valueOf("false");
                System.out.println(a1+"...."+a2+"...."+a3+
                        "...."+a4+"...."+a5+"...."+a6+"...."+a8);


        }
}
```

```
public class WrapperDemo {
        public static void main(String[] args) {
                //String data type to Wrapperclass type
                                Byte a1  = Byte.valueOf("100");
                                Short a2 = Short.valueOf("122");
                                Integer a3 = Integer.valueOf("100");
                        //Long a4 = Long.valueOf("123l");//here l is not
deleted
                                Long a4 = Long.valueOf("123");
                                Float a5 = Float.valueOf("12.34f");
                                Double a6 = Double.valueOf("12.34d");
                                //Character a7 = Character.valueOf("a");
                                Boolean a8 = Boolean.valueOf("false");
                                System.out.println(a1+"...."+a2+"...."+a3+

        "...."+a4+"...."+a5+"...."+a6+"...."+a8);
                                //wrapper class to string
                                String b1 = a1.toString();
                                String b2 = a2.toString();
                                String b3 = a3.toString();
                                String b4 = a4.toString();
                                String b5 = a5.toString();
                String b6 = a6.toString();
                Character c = 'a';
                String b7 = c.toString();
                String b8 = a8.toString();
                System.out.println(b1+"...."+b2+"...."+b3+

        "...."+b4+"...."+b5+"...."+b6+"...."+b7+"...."+b8);
                System.out.println(b1+b2);
                //string data to primitive datatype
                byte c1 = Byte.parseByte("100");
                short c2 = Short.parseShort("234");
                int c3 = Integer.parseInt("122");
                long c4 = Long.parseLong("222");
                float c5 = Float.parseFloat("6666.88f");
                double c6 = Double.parseDouble("23.34");
                //char c7 = Character.parseChar("a");
                boolean c8 = Boolean.parseBoolean("true");
                boolean c9 = Boolean.parseBoolean("true1");
                System.out.println(c1+"...."+c2+"...."+c3+

        "...."+c4+"...."+c5+"...."+c6+"...."+c8+"...."+c9);
                //pdt to string
                String d1 = String.valueOf(127);
```

```
String d2 = String.valueOf(127);
String d3 = String.valueOf(127);
String d4 = String.valueOf(127l);
String d5 = String.valueOf(127.89f);
String d6 = String.valueOf(127.78d);
String d7 = String.valueOf('a');
String d8 = String.valueOf(false);
System.out.println(d1+"...."+d2+"...."+d3+

"...."+d4+"...."+d5+"...."+d6+"...."+d7+"...."+d8);
System.out.println(d1+d8);
//System.out.println(127+false);


String e1 = Byte.toString((byte)100);
String e2 = Short.toString((short)200);
String e3 = Integer.toString(100);
String e4 = Long.toString(100l);
String e5 = Float.toString(223.34f);
String e6 = Double.toString(34.45d);
String e7 = Character.toString('a');
String e8 = Boolean.toString(false);
System.out.println(e1+"...."+e2+"...."+e3+
        "...."+e4+"...."+e5+"...."+e6+"...."+e7+"...."+e8);
                        System.out.println(e5+e6);
            }
}
```

Auto Boxing: converting or representing primitive data to object data directly is called auto boxing.

```
    ex: Integer i= 10;
        Boolean b = false;
```

Auto Unboxing: converting or representing wrapper class to primitive data type data directly is called auto unboxing.

```
    ex:
        int i = new Integer(100);
        boolean b = new Boolean(100);
```

Above statements valid from java 1.5 version.

Packages:

Package is a collection/group of classes, interfaces, enums, and sub-packages.

Some of the packages are given by java software and some of the packages are given by compiler.

The packages which are given by java s/w are called predefine packages.

The packages which are given by compiler or created by the compiler are called user define packages.

Package is a special java folder.

By using "package" keyword we can develop our own packages.

Syntax:

```
package packagename;
```

Example:

```
package p1;
```

Package is a second section of our program, first section is comment section.

Package statement is always first statement in java program.

Package statement is an optional statement but in the real time every ".java" file having package keyword.

The compilation and execution of the package program is different from normal program.

<u>Compilation phase:</u>

If we want to compile the java package related program we must use below option.

Syntax: `javac  -d  .  FileName.java`

•Here "-d" option will specify the folder creation. We are giving information to compiler to create one folder with the package name called nit.

•Here dot (“.”) operation will specify, current directory that means in nit folder and place the “dot class” file in that nit package.

Execution:

We should use the package name before the class name.

Syntax: java packagename.classname

Creating package in current working directory:

```
package nit;
class A{
     public static void main(String...args){
       System.out.println("this is package               program");
     }
}
```

Filename: A.java

compilation: javac -d . A.java

execution: java nit.A

Creating package in some other directory:

```
package google;
class B{
     public static void main(String...args){
       System.out.println("this is package program");
```

```
    }
}
```

FileName: B.java

Compilation: javac -d otherdirectorypath filename.java

Ex: javac -d E:\cj B.java

Execution: other directory packages can be execute in two ways

1) shift out from current working directoy to package existing directory.

```
C:\Users\lenovo\desktop\jse(7to9)> cd E:\cj(enter)

E:\cj> java nit.B
```

2) setting the classpath

```
C:\Users\lenovo\desktop\jse(7to9)>

set classpath=.;E:\cj;

java nit.B
```

Creating sub pakcages:

```
package nit.cj.ram;

class C{

public static void main(String...args){

System.out.println("sub packages");

}

}
```

Filename: C.java

Compilation: javac -d . C.java

Execution: java nit.cj.ram.C

<u>Access Modifiers:</u>

Accessibility modifiers will provide permission to developer to use the variable, methods, class, interface, enum, abstract class.

In java, we have four types of accessibility modifiers, they are

- private.
- default. (package-private)
- protected.
- public

private:

private data can be access only with in the class level. We cannot acces out side of the class with in the same .java file, with in the same package or within the outside package.

default:

If we are not declared any access modifier in front of the variable, that is comes under default data.

From java 1.8 onwards we can default keyword in front of the methods of interface.

default data can be accessible with in the same class, with in the same .java file, with in the same package, but not accessible within the outside package class.

protected: protected can be access within the same class, with in the same .java file, with in the same package, but not accessible within the outside package class.

protected data some time acting as default data and sometimes as an public.

If other package class subclass of this class and data is static type, then protected data can be access outside of the package also.

Note:

Packages are provides security to both default and protected data. Packages provide 100% security to default data and protected data. But some time packages are not provide security to protected data.

public: public data can be access anywhere in the project.

```
package ram;

public class A {

        private int a = 111;

        int b = 222;

        protected int c = 333;
        protected static int d = 444;
        public int e = 555;
        public static void main(String[] args) {
                A obj = new A();
                System.out.println(obj.a);
                System.out.println(obj.b);
                System.out.println(obj.c);
                System.out.println(obj.e);
        }
}
```

Filename: A.java

Compilation: javac -d . A.java

Execution: java ram.A

```
package ram;
public class B {
        public static void main(String[] args) {
                A obj = new A();
                //System.out.println(obj.a);
                System.out.println(obj.b);
                System.out.println(obj.c);
```

```
            System.out.println(obj.e);
        }
}
```

Filename: B.java

Compilation: javac -d . B.java

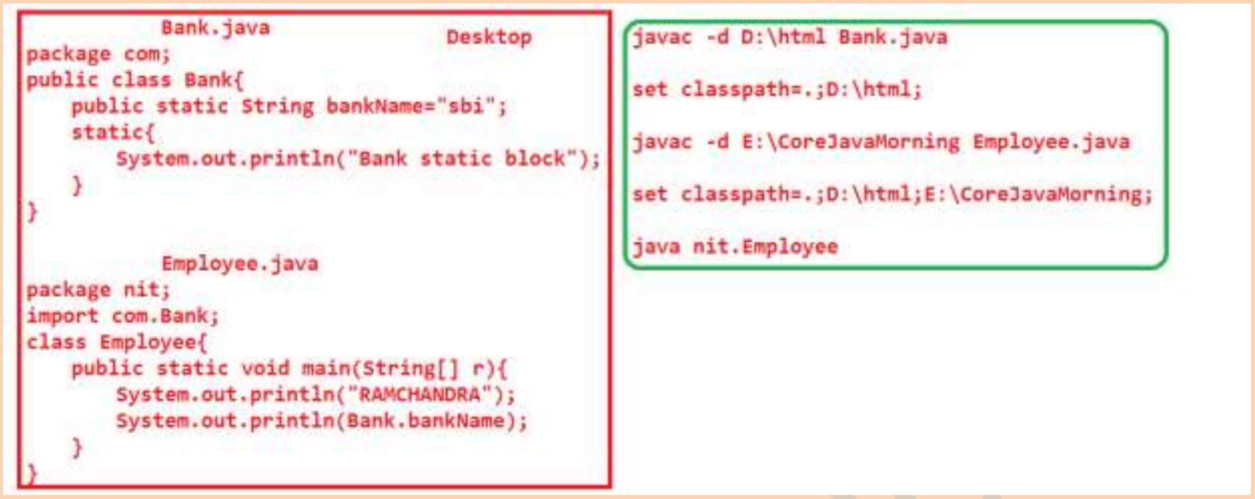
Execution: java ram.B

```
package sam;
import ram.A;
public class C extends A{
        public static void main(String[] args) {
            A obj = new A();
            //System.out.println(obj.a);
            //System.out.println(obj.b);
            //System.out.println(obj.c);
            System.out.println(obj.d);
            System.out.println(obj.e);


        }
}
```

Filename: C.java

Compilation: javac -d . C.java

Execution: java sam.C



```
        Bank.java                    Desktop
package com;
public class Bank{
    public static String bankName="sbi";
    static{
        System.out.println("Bank static block");
    }
}

        Employee.java
package nit;
import com.Bank;
class Employee{
    public static void main(String[] r){
        System.out.println("RAMCHANDRA");
        System.out.println(Bank.bankName);
    }
}
```

```
javac -d D:\html Bank.java

set classpath=.;D:\html;

javac -d E:\CoreJavaMorning Employee.java

set classpath=.;D:\html;E:\CoreJavaMorning;

java nit.Employee
```

import statements

It is programming elements, which is useful for recognize the class,interfaces,eneum,annotation by compiler which are not available in java.lang package.

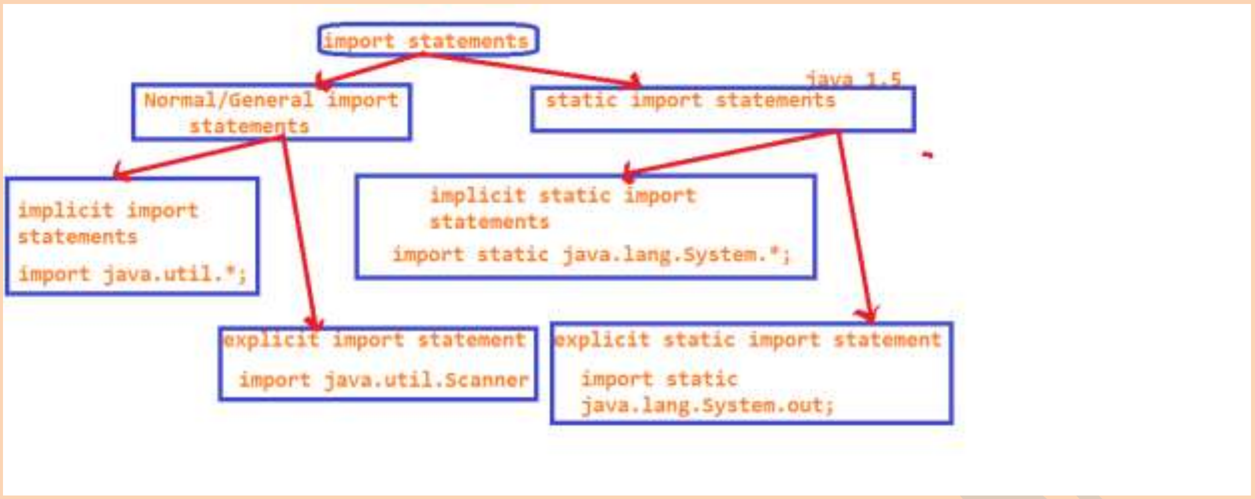
These are two types.

1. static import statements.
2. non-static/General import statement

General import statement can loads only classes, interface, abstract class, annotations, enum but not its members directly.(fields, methods).

Static import statements import the static fields, static methods. These variables and members must be public.

It is introduced in java 1.5 version.


import statements
Normal/General import statements
static import statements
java 1.5
implicit import statements
import java.util.*;
implicit static import statements
import static java.lang.System.*;
explicit import statement
import java.util.Scanner
explicit static import statement
import static java.lang.System.out;

Note:

Fully Qualified Name= packagename+classname/interfacename/enumname/abstractclass/annotationname;

```
package sam;

/*import static java.lang.System.out;

import static java.lang.System.err;

import static java.lang.System.currentTimeMillis;

import static java.lang.System.identityHashCode;

import static java.lang.System.exit;*/

import static java.lang.System.*;

class B{


}

public class C{
```

```
public static void main(String[] args) {
        System.out.println("static out filed");
        out.println("with out system class");
        err.println("err field");
        System.out.println(System.currentTimeMillis());
        out.println(currentTimeMillis());
        B obj = new B();
        out.println(System.identityHashCode(obj));
        out.println(identityHashCode(obj));
        exit(0);
        System.out.println("program terminated in the previous
statement");
        }
}
```

Difference between #include and import statement?

#include will copy the code from library to program. so internally the code size will be increased.

import statement will give only response to java program.

Difference between import java.util.ArrayList and java.util.*;

In the first import statement we can access only ArrayList and its super class and interface functionalities.

In the second import statement we can access all classes and interface and abstract class related to util package.

Java archive: (jar)

It is one compressed file in java standard edition.

It contains java standard edition classes.

It can reduce file size.

With the help jar we can execute the programs also.

Web archive: (war).

It is also one compressed file in advanced java.

It contains the following files.

.servlet related classes

.jsp file

.html files

.jpeg files

.xml files

.properties files

Enterprise archive: (ear)

It is also one compressed file in enterprise edition.

This file contains the following files.

.servlet related classes

.jsp

.html

.jpeg

.xml files

.properties files and enterprise java beans classes.

Creating .jar file and add .class file:

```
public class A{
public static void main(String... args){
	System.out.println("jar file");
}
}
```

```
javac A.java
```

We are getting A.class file

```
jar -cf one.jar A.class
```

Delete the A.class file

Set the one.jar file in the class path then execute the program

```
java A
```

output: jar file.

```
public class B{
public static void main(String... args){
        System.out.println("updating jar file");
    }
    }
    javac B.java
```

We are getting B. class file

add B.class file to one.jar

```
jar -uf one.jar B.class

jar -tf one.jar

META-INF/MANIFIEST.MF
A.class
B.class

java A
java B
```

Executing the java program with help of javaw command.

```
java.io.*;
class D{
```

```
public static void main(String...ram)throws IOException{
FileOutputStream fos = new FileOutputStream("ram1.text");
PrintStream ps = new PrintStream(fos);
ps.write('d');
ps.write('e');
}
}
```

```
javac D.java
javaw D
```

creating executable jar.

1) Write one package program.

```
package suji;
public class A{
public static void main(String...ram){
    System.out.println("executablejar");
}
}
```

filename A.java

2) Compile our package program with below syntax.

```
javac -d . filename.java
javac -d . A.java
```

3) Take one notepad write below code in that.

```
Main-Class: suji.A
```

4) Save above notepad with MANIFEST.MF

5) Add MANIFEST.MF file with the help of bellow code.

```
jar -cvfm ud.jar MANIFEST.MF suji
```

6) Type bellow in the command prompt

```
java -jar ud.jar.
```

Note: class (A) must be public

In the MANIFEST.MF file M,C must be capital letter after ':' we need give one space and after packagename.classname we need to click on enter button.

ud.jar name is userdefine.

Executing the java program by using batch file.

1) write one program

```
package  rabi;

public class  A{

    public static void main(String...ram){

    System.out.println("batchfile");

    }

}
```

filename A.java

2) take one notepad

write bellow code in that.

```
javac -d . A.java

java rabi.A

pause
```

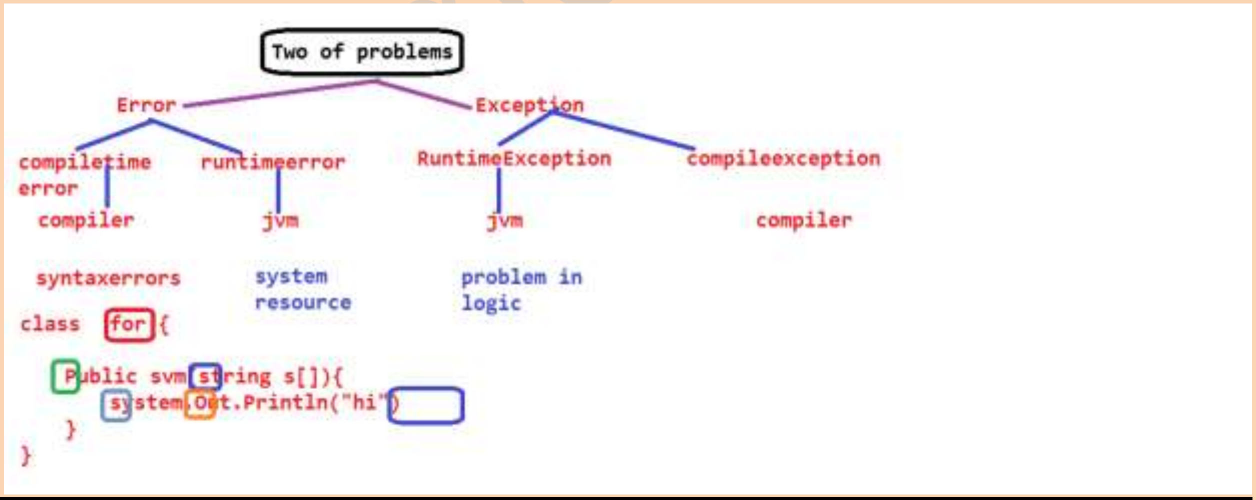
And save with filename.bat

ex: run.bat

3) double click on run.bat

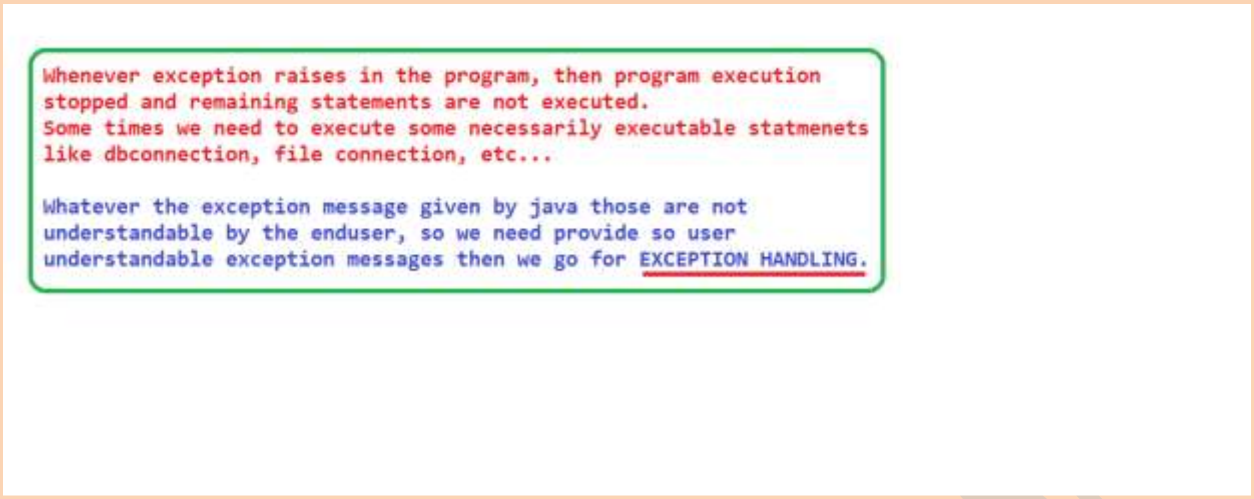
Automatically command prompt will be open and compile the program and executing the program and display the output.

Exception Handling:



```
class for{
  Public svm(string s[]){
    system.Out.Println("hi")
  }
}
```

Exception: Exception is an event. It can be raised by the JVM at runtime due to problem of logic.



Whenever exception raises in the program, then program execution stopped and remaining statements are not executed.
Some times we need to execute some necessarily executable statmenets like dbconnection, file connection, etc...

Whatever the exception message given by java those are not understandable by the enduser, so we need provide so user understandable exception messages then we go for EXCEPTION HANDLING.

Some of logical problems are:

```
class MM{
        public static void main(String[] s){
                        String s1 = s[0];
                        String s2 = s[1];
                System.out.println(s1+s2);
        }
}
```

javac MM.java

valid execution statements:

```
        java MM 123 234
        java MM ram sam
        java MM "%^&*(" "@#$%^"
```

Exception raised execution statements:

```
        java MM (click on enter)

java.lang.ArrayIndexOutOfBoundsException--0

        java MM 123 (enter)

java.lang.ArrayIndexOutOfBoundsException--1

class MM{

        public static void main(String[] s){

        String s1 = s[0];

        String s2 = s[1];
        System.out.println(s1+s2);

        int i = Integer.parseInt(s1);

        int j = Integer.parseInt(s2);

        System.out.println(i+j);

        }

}

        javac MM.java

        java MM 123 234

        o/P: 123234

             357

        java MM ram sam

        o/p: ramsan

java.lang.NumberFormatException
```

```
class MM{
        int a =111;
        void m1();
        public static void main(String... ram){
                System.out.println("Main Method");
                MM obj = new MM();
                obj = null;
                //System.out.println(obj.a);
                obj.m1();
        }
}
```

Note:

On top of null reference, we cannot call any variables, methods.

If we call then we will get java.lang.NullPointerException.

```
class MM{
        public static void main(String... ram){
        System.out.println("Main Method");
        //Object o = new String();//upcasting
        Object o = new Object();
        String s1 = (String)o;//downcasting
        }
}
```

java MM → java.lang.ClassCastException

```
class MM{
        public static void main(String... ram){
        System.out.println("Main Method");
                int a [] = new int[-6];
        }
}
```

Array dimensions must be positive or zero not negative.

If we mention size in negative mode then we will get

java.lang.NegativeArraySizeException.

Before doing the down casting, first we should do up casting otherwise we will get java.lang.ClassCastException

```
class MM extends Thread{
        public void run(){
                System.out.println("this is run method");
        }
        public static void main(String[] s){
                MM obj = new MM();
                obj.start();
                obj.start();
        }
}
javac MM.java
java MM
this is run method
java.lang.IllegalThreadStateException
class MM {
        public static void main(String[] s){
                int a = 10/0;
        }
}
javac MM.java
java MM
java.lang.ArithmeticException

class MM {
        int a =111;
        public static void main(String[] s){
                MM obj = new MM();
                obj=null;
                System.out.println(obj.a);
        }
}
javac MM.java
java MM
java.lang.NullPointerException

class MM extends Thread{
        public void run(){
                System.out.println("this is run method");
        }
        public static void main(String[] s){
                MM obj = new MM();
                obj.setPriority(11);//legal priorities from 1 to 10
                obj.start();

        }
}
javac MM.java
java MM
java.lang.IllegalArgumentException
class MM {
        public static void main(String[] s1){
```

```
        Object o = new Object();
        String s = (String)o;
    }
}
javac MM.java
java MM
class MM{
    public static void main(String... ram){
        System.out.println("Main Method");
        String s = "internationalization";//i18n
        System.out.println(s.charAt(20));
    }
}
```

In above program, we dont have 20th index position in string s.

But we tried to print the values thatwhy we are getting

java.lang.StringIndexOutOfBoundsException

java.lang.ClassCastException

```
float a = 10/0.0f-->infinity
float b = -10/0.0f--> -infinity
float c  = 0/0.0f--->NaN
float d = 0./0; -->NaN
```

<u>Error:</u> Error is an event which can be raised by jvm at runtime due to problem of lackof system resource.

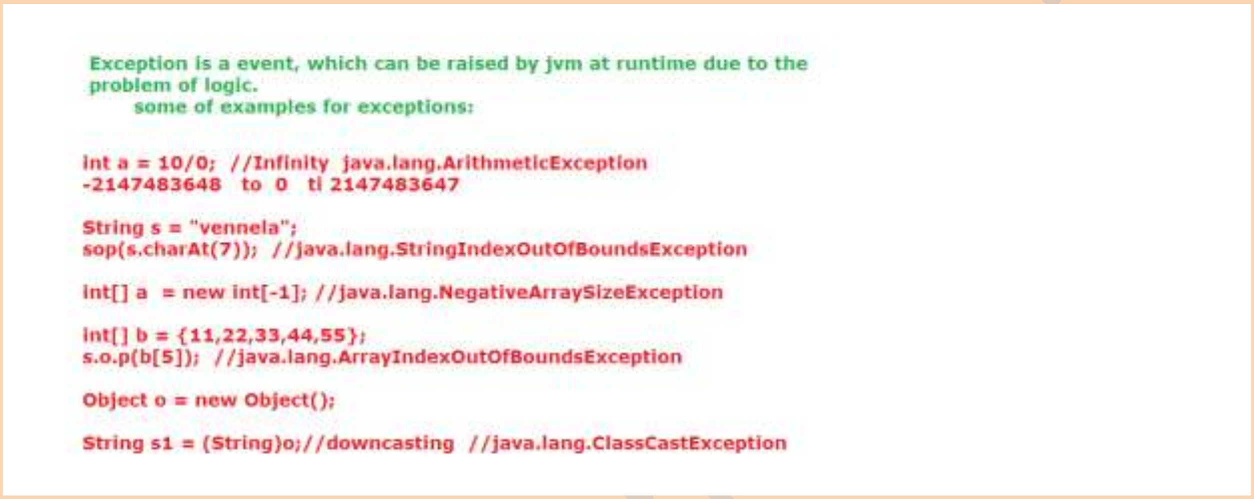
```
class MM {

        MM obj = new MM();

        public static void main(String[] s1){

            MM obj = new MM();


        }

}


javac MM.java

java MM

java.lang.StackOverflowError
```

Note:

JVM version always greater than or equal to compiler version, otherwise we will get one runtime error that is:

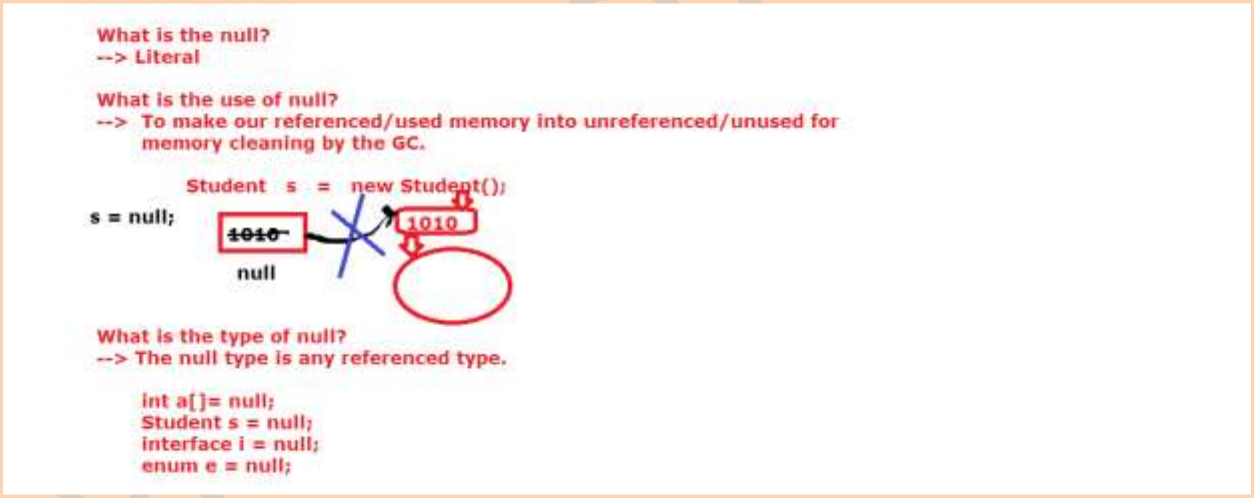
    UnsupportedClassVersionError

```
class MM {

        static int a = 10/0;

    public static void main(String[] s1){



    }
}

javac MM.java

java MM

java.lang.ExceptionInInitilizerError
```



Exception is a event, which can be raised by jvm at runtime due to the problem of logic.
some of examples for exceptions:

```
int a = 10/0;  //Infinity  java.lang.ArithmeticException
-2147483648   to  0   ti 2147483647

String s = "vennela";
sop(s.charAt(7));  //java.lang.StringIndexOutOfBoundsException

int[] a  = new int[-1]; //java.lang.NegativeArraySizeException

int[] b = {11,22,33,44,55};
s.o.p(b[5]);  //java.lang.ArrayIndexOutOfBoundsException

Object o = new Object();

String s1 = (String)o;//downcasting  //java.lang.ClassCastException
```



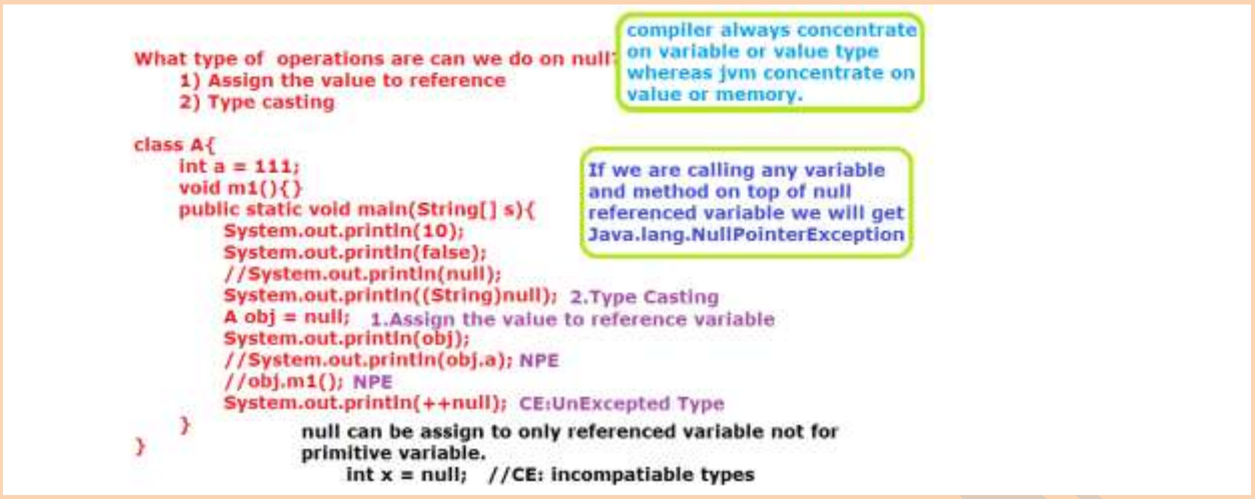
What is the null?
--> Literal

What is the use of null?
--> To make our referenced/used memory into unreferenced/unused for memory cleaning by the GC.

Student s = new Student();
s = null;
~~1010~~
null
1010

What is the type of null?
--> The null type is any referenced type.

```
int a[]= null;
Student s = null;
interface i = null;
enum e = null;
```



What type of operations are can we do on null?
1) Assign the value to reference
2) Type casting

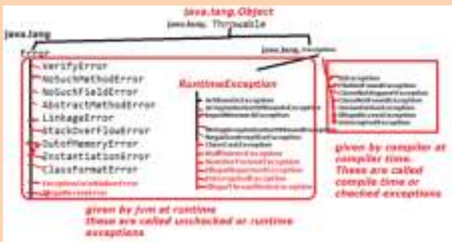
compiler always concentrate on variable or value type whereas jvm concentrate on value or memory.

```
class A{
    int a = 111;
    void m1(){}
    public static void main(String[] s){
        System.out.println(10);
        System.out.println(false);
        //System.out.println(null);
        System.out.println((String)null);  2.Type Casting
        A obj = null;  1.Assign the value to reference variable
        System.out.println(obj);
        //System.out.println(obj.a); NPE
        //obj.m1(); NPE
        System.out.println(++null);  CE:UnExcepted Type
    }
}
```

If we are calling any variable and method on top of null referenced variable we will get Java.lang.NullPointerException

null can be assign to only referenced variable not for primitive variable.
int x = null;   //CE: incompatiable types

Exception Hierarchy:



Use of Exception Handling:

1) Whenever exception is raised internally JVM provides some predefine exception message.

   Those messages are not understand by the end user.

   If we want to provide user friendly message then we can go for exception handling.

2) Whenever exception is raised automatically the remaining statements are not executed.

Program abnormally terminated (without giving information to programmer).

So some necessary executable statements are not executed.

If we want to execute all necessary statements then we can go for exception handling.

We can achieve exception handling mechanism with the help of following keywords

those are

1. try
2. catch
3. finally
4. throw
5. throws
6. assert

try : In this block we should always write exception raised statements.

If we don't know which statements is raises exception then write all the statements in try block.

catch: In this block we should hold exception object which was raised in the try block.

This block execution is depends upon the try block.

If try block not raised exception then catch block will not be executed.

If try block raised exception then catch block will be executed.

Note: **catch block always holds exception objects not normal objects.**

finally: This block is not depends upon try block result.

If try block is raised the exception or not, the finally block always executed.

This block is used write the statements which are necessary to execute.

Like database connection, file closing.....

This block is always executed.

When finally block not executed?

If programmer write System. exit (0), that means if we are explicitly stop the program then finally block not executed.

If JVM execution suddenly stopped then finally block not executed.

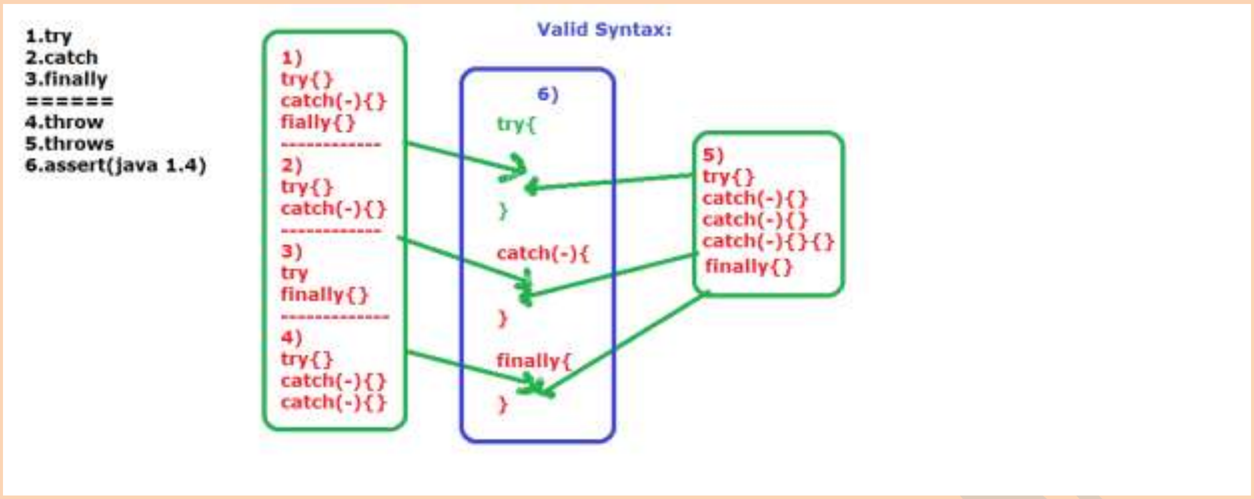
Valid combinations between try and catch and finally blocks:

(1)

```
try{
}
```

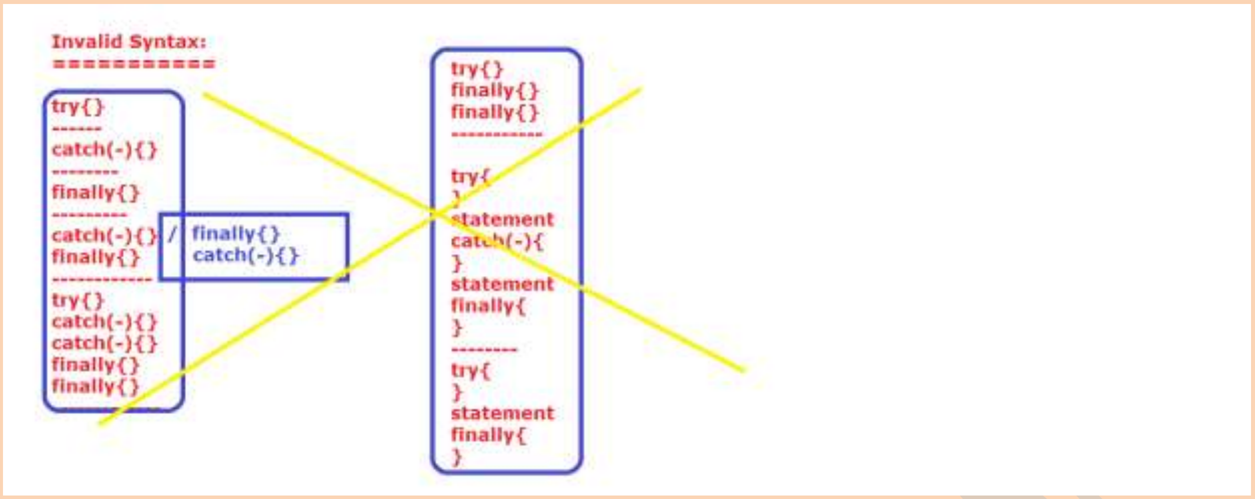
```
    catch(-){
    }
    finally{
    }
    -----------------
(2)
try{
}
finally{
}
-------------
(3)
try{
}
catch(-){
}
--------------
(4)
try{
}
catch(){
}
```

```
catch(){
}
catch(){
}
finally{
}
--------------
(5)
try{
}
catch(){
}
catch(){
}
catch(){
}
--------------
(6)
try{
        try{
        }
        catch(-){
        }
        finally{
        }
}
catch(-){
        try{
        }
        catch(-){
        }
        finally{
        }
}
finally{
        try{
        }
        catch(-){
        }
        finally{
        }
}
```



Invalid statements:

```
try{

}

finaly{

}

catch{

}

************

try{

}

**********

catch(){

}

**********

finally{
}
********

try{
}
finally{
}
finally{
}
***********
try{
}System.out.println("we cannot write");
catch(-){
}System.out.println("we cannot write");
finally{
}
***********
try{
}
System.out.println("we can not write");
finally{
}
```



Invalid Syntax:
===========

```
try{}
------
catch(-){}
--------
finally{}
---------
catch(-){} / finally{}
finally{}    catch(-){}
-----------
try{}
catch(-){}
catch(-){}
finally{}
finally{}
```

```
try{}
finally{}
finally{}
-----------
try{
}
statement
catch(-){
}
statement
finally{
}
--------
try{
}
statement
finally{
}
```

Internal flow of Exception:

Whenever exception is raised in the program JVM will do the following things.

1. JVM will check behavior/characteristic of an exception.

    int a = 10/0;

2. Based on characteristic of an exception, JVM will select one appropriate exception class.

    java.lang.ArithmeticException

3. After selecting the class, JVM will create an object for that selected class.

    ArithmeticException ae= new ArithmeticException();

4. While creating an object for that JVM will add description of an exception.

like "/ by zero".

5. Finally JVM will handover to catch block or print on the console.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at collection.MapDemo.main(MapDemo.java:8)
```

-->catch block parameter type must be required three types of functionalities, those are

1.Object functionalities

2.Throwable functionalities

3.Exception functionalities

We have three ways(Methods) to print exception messages.

1. printStackTrace().
2. toString()
3. getMessage().

1.printStackTrace():

It will print exception message in the following format

```
        java.lang.ArithmeticException: / by zero
   at exception.ExceptionDemo.main(ExceptionDemo.java:8)
```

It will print predefine exception class fully qualified name, description of an exception fully qualified user define class name, method name, .java file name, line number, package name.

Exception class
Description of an Exception
method name
userdefine class
line number

2. toString();

java.lang.ArithmeticException: / by zero

It will print predefine exception class fully qualified name, description of an exception

3. getMessage();

/ by zero

It will print only description of exception

```
public class ExceptionDemo {
	public static void main(String[] args) {
		//int a = 10/0;
		//System.out.println("these statements are not executed");
		try{
```

```
                int a = 10/0;
            }
            catch(Exception e){
                e.printStackTrace();
                System.out.println(e.toString());
                System.out.println(e.getMessage());
            }
            System.out.println("These statements are executed");
        }
    }

    public class ExceptionDemo {
        public static void main(String[] args) {
            try{
                int a = 10/0;
            }
            catch(Exception e){
                System.out.println("DONT ENTER ZERO AS AN
                DENOMINATOR");
            }
        }
    }
```

In the first program we are execute the all the statements

in this program we did give user understandable exception message.

```
public class ExceptionDemo {
	public static void main(String[] args) {
		try{
			int a = 10/0;
			System.out.println("not executed");
		}
		//System.oyut.println("we cannot write the statement
		//	b/w try and catch")
		catch(Exception e){
		System.out.println("DONT ENTER ZERO AS AN 
DENOMINATOR");
		//System.exit(0);
		//in the above line we did explacitly stop the program
		}
		//System.oyut.println("we cannot write the
		//statement b/w finally and catch")
		finally{
			System.out.println("this block is always 
executed");
```

```
                }
                System.out.println("we can write the statements " +
                        "after the finally block");
        }
}
public class ExceptionDemo {
        public static void main(String[] args) {
                try{
                        //int a = 10/0;
                        //Object o = new Object();
                        //String s = (String)o;
                        //int a [] = {10,20,30};
                        //System.out.println(a[3]);
                        //int a[] = new int[-1];
                         Object[] s = new Integer[4];
                         s[0] = 4.4;
                }
                //catch(Exception e){}
                catch(ArithmeticException e){
                        System.out.println("dont enter zero as an 
denominator");
                }
        catch(ClassCastException e){
            System.out.println("Super class memory we can
not" +
                    " place into subclass reference ");
        }
        catch(ArrayIndexOutOfBoundsException e){
            System.out.println("we dont have any element in
3rd " +
                    "index position");
        }
        catch(NegativeArraySizeException ne){
            System.out.println("we can not create array" +
                    " with negative values");
        }
        catch(Exception e){
            System.out.println("we can not place double value
in " +
                    "integer array");
        }
        finally{
            System.out.println("this block is always
executed");
        }
        }
}
```

If we are writing more than one catch blocks, java.lang.Exception class type parameter must be placed in last catch block.

The reason is whatever the exception is raised in the catch block that is always matched with Exception class type. (upcasting).

So the remaining catch blocks will not be executed.

```
public class ExceptionDemo2 {
public static void main(String[] args) {
        try{
        System.out.println("outer try block");
        try{
                System.out.println("inner try block");
                int a= 10/0;
        }
        /*catch(Exception e){

                System.out.println("inner catch block");
                System.out.println("dont enter zero as deono");
        }*/
        finally{
                System.out.println("inner finally block");
        }
        }
catch(Exception e){
        System.out.println("outer catch block");
}
finally{
        System.out.println("outer finally block");
}
}
}
```

Note: If any exception is available in inner try block first jvm check inner catch block.

If available then inner catch will be execute. If not available or not matched then JVM checks outer catch block.

If available then outer catch block will be executed, if not available or parameter type exception class is not supports raised exception class (not matched) then raised exception class object handover to JVM.

```
public class ExceptionDemo2 {
                public static void main(String[] args) {
                try{
                        System.out.println("outer try block");
                        try{
                                System.out.println("inner try block");
                                int a= 10/0;
                        }
                        /*catch(Exception e){
                                System.out.println("inner catch block");
                                System.out.println("dont enter zero as
deono");
                        }*/
                        finally{
                                System.out.println("inner finally block");
                        }
                }catch(Exception e){
                        System.out.println("outer catch block");
                }
                finally{
                        System.out.println("outer finally block");
                }
        }
}
```

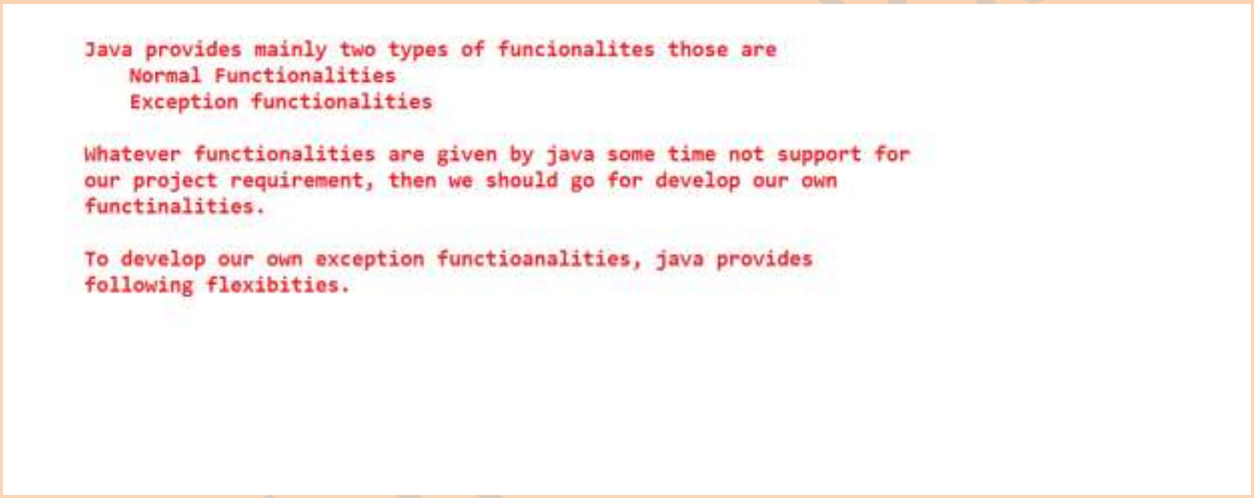
Note:

finally block not only override the return value but also override the exception.

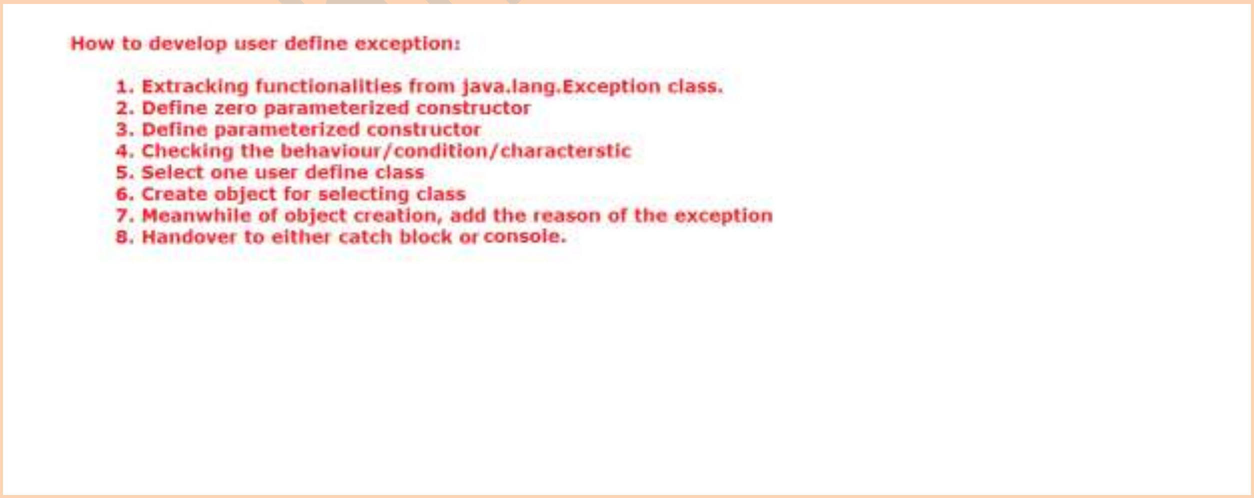
```
package exception;
class Student{
```

```
int sid = 111;
String sname = "ram";
        void m1()throws Throwable{
                Student s1 = this;
                System.out.println(s1.sid);
                System.out.println(s1.sname);
                s1 = null;
                try{
                System.out.println(s1.sid);//NullPointerException
                }catch(Exception e){
                        //throw e;
                        throw e.initCause
                (new ClassCastException("we are " +     "calling
                variable on null reference"));
                }
        }
}

public class ChinedException {
        public static void main(String[] args)  {
                Student s = new Student();
                try {
```

```
                s.m1();
            } catch (Throwable e) {
                System.out.println(e.getCause());
            }
        }
    }
```



```
Java provides mainly two types of funcionalites those are
    Normal Functionalities
    Exception functionalities

Whatever functionalities are given by java some time not support for
our project requirement, then we should go for develop our own
functinalities.

To develop our own exception functioanalities, java provides
following flexibities.
```



How to develop user define exception:

1. Extracking functionalities from java.lang.Exception class.
2. Define zero parameterized constructor
3. Define parameterized constructor
4. Checking the behaviour/condition/characterstic
5. Select one user define class
6. Create object for selecting class
7. Meanwhile of object creation, add the reason of the exception
8. Handover to either catch block or console.

```
package exception;
```

```
class MyException extends Exception{
        MyException(String msg){
                super(msg);
        }
        MyException(){
        }
void getAge(int age)throws MyException{
        try{
                if(age >= 18){
                System.out.println("This person is elegible for voting");
        }
        else
                throw new MyException("This person is not elegible for
voting");
        }catch(Exception e){
                throw e;
        }
        }
}
public class UserDefineExceptionDemo {
        public static void main(String[] args)throws MyException {
                MyException me = new MyException();
                me.getAge(18);

                /*try{

                    me.getAge(17);

                }catch(Exception e){

                }*/

        }

}
```

If we want to develop user define exceptions, programmer should do the following things.

1) Programmer should take one user define class.

Ex: class MyException{

```
    }
```

2) Creating an object for that class.

new MyException("This person is not eligible for voting");

3) If we want to add exception message to object we need some logic, that logic is available in java.lang.Exception .

4) If we want to forward our control and exception message from sub class to super class we need bellow s yntax.

```
ex: MyException(String msg){

        super(msg);

      }
```

5) If exception is predefine, then JVM will handover that exception object from try block to catch block.

Here exception is user define, so we should give the information to JVM about our exception object.

If we want handover the object from try block to catch block we need "throw" keyword.

With the help of throw keyword we can give information to JVM about our object that is make our object as an exception object.

And also with the help of throw keyword we can send exception object from called method to calling method.

If any method having throw keyword in its catch block that method must be use throws keyword.

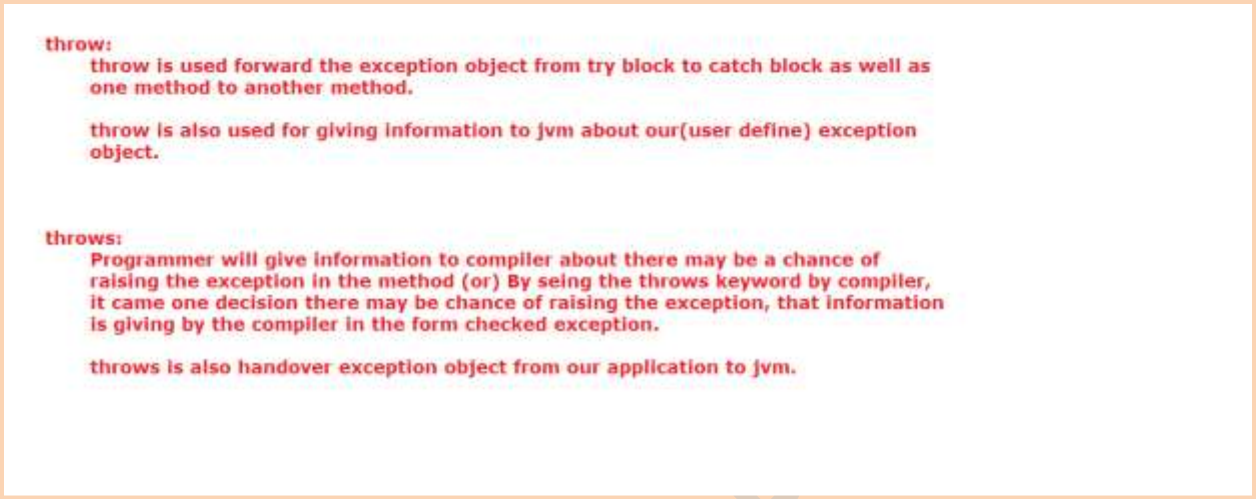
throw: "throw" keyword is used to forward the exception object from one method to another method within the application explicitly.

throws: If any compile time exception raised in our program, then we need to handle those exceptions. If we are not interest to handle, then we need handover those exceptions to JVM. "throws" is used to forward the exception object from java application to JVM application then we can go for "throws" keyword.

With the help of throws developer provide some information to compiler that is there may be chance of raising exception in our method.

Again with the help of throws keyword compiler will give information to programmer as a compiler time exception.

If are using any method, which is raised exception, in that time compiler will throw compile time/checked exception.



throw:

throw is used forward the exception object from try block to catch block as well as one method to another method.

throw is also used for giving information to jvm about our(user define) exception object.

throws:

Programmer will give information to compiler about there may be a chance of raising the exception in the method (or) By seing the throws keyword by compiler, it came one decision there may be chance of raising the exception, that information is giving by the compiler in the form checked exception.

throws is also handover exception object from our application to jvm.

We can handle checked exception in the following two ways.

1. try and catch block
2. throws keyword.

```
public class ExceptionDemo {

    static int m1(){

        try{

            System.out.println("try block");

            //int a = 10/0;

            return 111;

        }

        catch(Exception e){

            System.out.println("catch block");

            //return 222;
```

```
		}
		finally{
			System.out.println("finally block");
			//return 333;
		}
		System.out.println("unable to write");
		return 444;
	}
	public static void main(String[] args) {
		System.out.println(m1());
	}
}
```

Note:

If try and catch and finally blocks having return statement, we can’t write any statements after the finally block.

try and catch block return statement values always replaced with finally block return statement value.

If try and catch block not having any return statement, after the finally block we can write some other statements (optional) and also we should write return statement (mandatory) also.

Additional points:

If any method having return statement:

1. We should write return statement in try and catch block.

2. If try and catch block having return statement, we can’t return statement after the catch block.

3. If try only having only return statement we need to return statement after the catch block.

4. If catch block having only return statement, we need to return statement after the catch block.

5. try and catch block return statements values always override by the finally block return statement value.

```
package nit.cj.sb;

import java.util.Scanner;

class InvalidAgeException extends java.lang.Exception{
	InvalidAgeException(String message){
		super(message);
	}
	InvalidAgeException(){
	}
	public void getAge(int age)throws Exception {
		try{
			if(age >= 18){
				System.out.println("you are elegible for
voting");
			}
			else {
				throw new InvalidAgeException("invalid
age");
			}
		}
		catch(Exception e){
			//e.printStackTrace();
			throw e;
		}
	}
```

```
}
public class ExceptionDemo {
	public static void main(String[] args)
	{//throws Exception{
		InvalidAgeException iae = new InvalidAgeException();
		Scanner scan = new Scanner(System.in);
		System.out.println("enter your age");
		byte age = scan.nextByte();
		try{
			iae.getAge(age);
		}catch(Exception e){
			//e.printStackTrace();
			System.out.println("Sorry! You Dont have age 
greater than 17");

		}
	}
}
```

Why can’t we write Object type parameter in catch block?

A) Catch block always executing meanwhile of exception raises in the try block.
B) Try block always throwing execption objects, which are directly or indirectly subclass of Throwable class.
C)So catch block parameter type required both Object class functionalaites and Throwable class functionalities.
D) If we are writing
catch(Object o){
}
Parameter ‘o’ only hava Object class functionalities but doesnot Throwable functionalities.

E) **Thts why we can not write Object type parameter in catch block.**

In which scenario finally block not executed?

A) **If programmer writing System.exit(0) then automatically jvm functionalities will stop, so finally block not executed**

```
public class Demo {

	public static void main(String[] args) throws Exception{
		try{
			System.out.println("try block");
			int a = 10/0;
			System.exit(0);
		}
		catch(Exception e){
			System.out.println("catch block");
			System.exit(0);
		}
		finally{
			System.out.println("finally block");
		}
	}
}
```

B) Jvm suddenly crash.

Exception Chaining:

```
public class ExceptionDemo {

	int a = 111;

	static void m1()throws Throwable{
		try{
			ExceptionDemo ed = new ExceptionDemo();
			System.out.println(ed.a);
			ed = null;
			System.out.println(ed.a);
		}
		catch(Exception e){
		throw e.initCause(new ArithmeticException("we are 
calling" +
					" variable a on top of null reference"));
		}
	}
	public static void main(String[] args) {
		int a = 10/0;
		try {
			m1();
		} catch (Throwable e) {
			//e.printStackTrace();
			System.out.println(e.getCause());
		}
	}
}
```

One Exception class provide the information like cause of exception of another exception class is called exception chaining.

If we want to add exception message to existed object we have one method that is "initCause()".

If we want read the exception message, we have one more method that is "getCause()".

Exceptions can be classified as follows.

<u>Fully checked exceptions:</u>

Compiler will check class, and all it subclasses is called FCE. All compile time exceptions are comes under fully checked exceptions only.

<u>Partially checked exceptions:</u>

Compiler will check class and not check all its subclass is called PCE.

Ex: java.lang.Exception,

java.lang.Throwable

If method having return type and having try and catch blocks,we need to write return statement with value in both blocks.

If method having return type and having try and catch and finally blocks we no need to write return statement with values in try and catch blocks. Directly we can write return statement with value in finally block.

If try and catch having return statements then finally block return statement always overrides the try and catch block return statement.

If finally block having return statement we can’t write any other statement after the finally block.

We can write some statements after finally block in the following conditions.

Only try block having return statement (or)

Only catch block having return statement.

If no block having return statement.

```
package exception;

public class ExceptionDemo {

	public static void main(String[] args){

		try{

			System.out.println("outer try block");

			try{

				System.out.println("inner try block");

				int a[] ={1,2,3};

				System.out.println(a[3]);

			}catch(ArithmeticException e){

				System.out.println("inner catch block");

			}
```

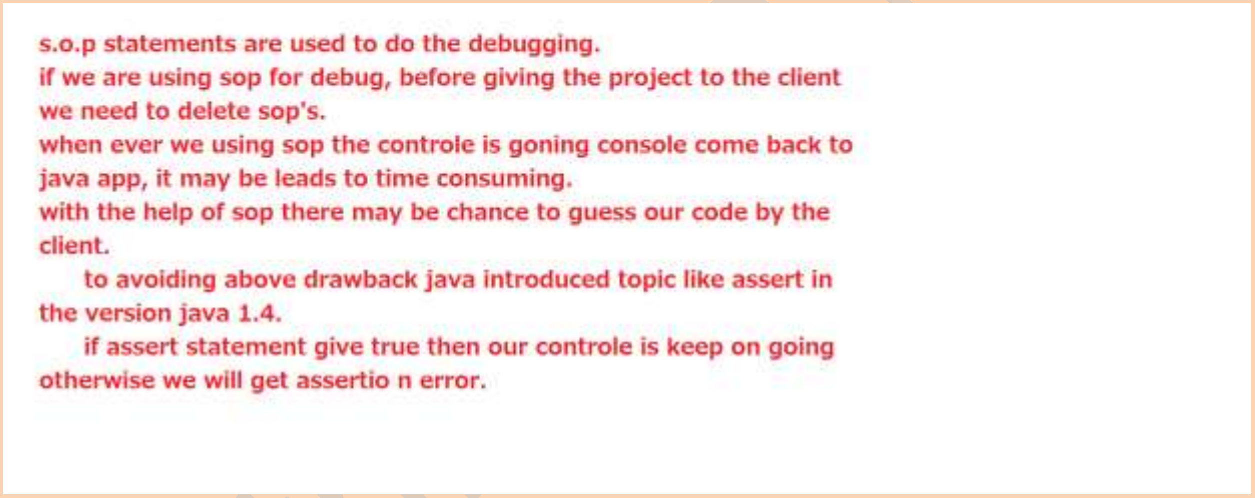
```
                finally{
                        System.out.println("inner finally block");
                }
        }catch(NullPointerException e){
                System.out.println("outer catch block");
        }
        finally{
                System.out.println("outer finally block");
        }

    System.out.println("***********************************
*****");
        try{
                System.out.println("outer try block");
                try{
                        System.out.println("inner try block");
                        System.out.println("ram".charAt(3));
                }
                finally{
                        System.out.println("inner finally block");
                }
        }catch(Exception e){
                System.out.println("outer catch block");
```

```
}
finally{
        System.out.println("outer finally block");
}
System.out.println("************************");
try{
        System.out.println("outer try block");
        try{
                System.out.println("inner try block");
                int a[] = new int[-1];
        }catch(Exception e){
                System.out.println("inner catch block");
        }
        finally{
                System.out.println("inner finally block");
        }
}catch(Exception e){
        System.out.println("outer catch block");
}
finally{
        System.out.println("outer finally block");
}
```

```
System.out.println("********************************");
    try{
        System.out.println("outer try block");
        int a = 10/0;
        try{
            System.out.println("inner try block");
        }catch(Exception e){
            System.out.println("inner catch block");
        }
        finally{
            System.out.println("inner finally block");
        }
    }catch(Exception e){
        System.out.println("outer catch block");
    }
    finally{
        System.out.println("outer finally block");
    }
    System.out.println("***************");
    try{
        System.out.println("outer try block");
```

```
                try{
                        System.out.println("inner try block");
                }catch(Exception e){
                        System.out.println("inner catch block");
                }
                finally{
                        System.out.println("inner finally block");
                }
        }catch(Exception e){
                System.out.println("outer catch block");
        }
        finally{
                System.out.println("outer finally block");
        }

    }
}
static void m2(){
        try{
                System.out.println("try block");
        }
        catch(Exception e){
                System.out.println("catch block");
        }
        finally {
                System.out.println("finally block");
        }
        System.out.println("we can write");//valid
    }
```

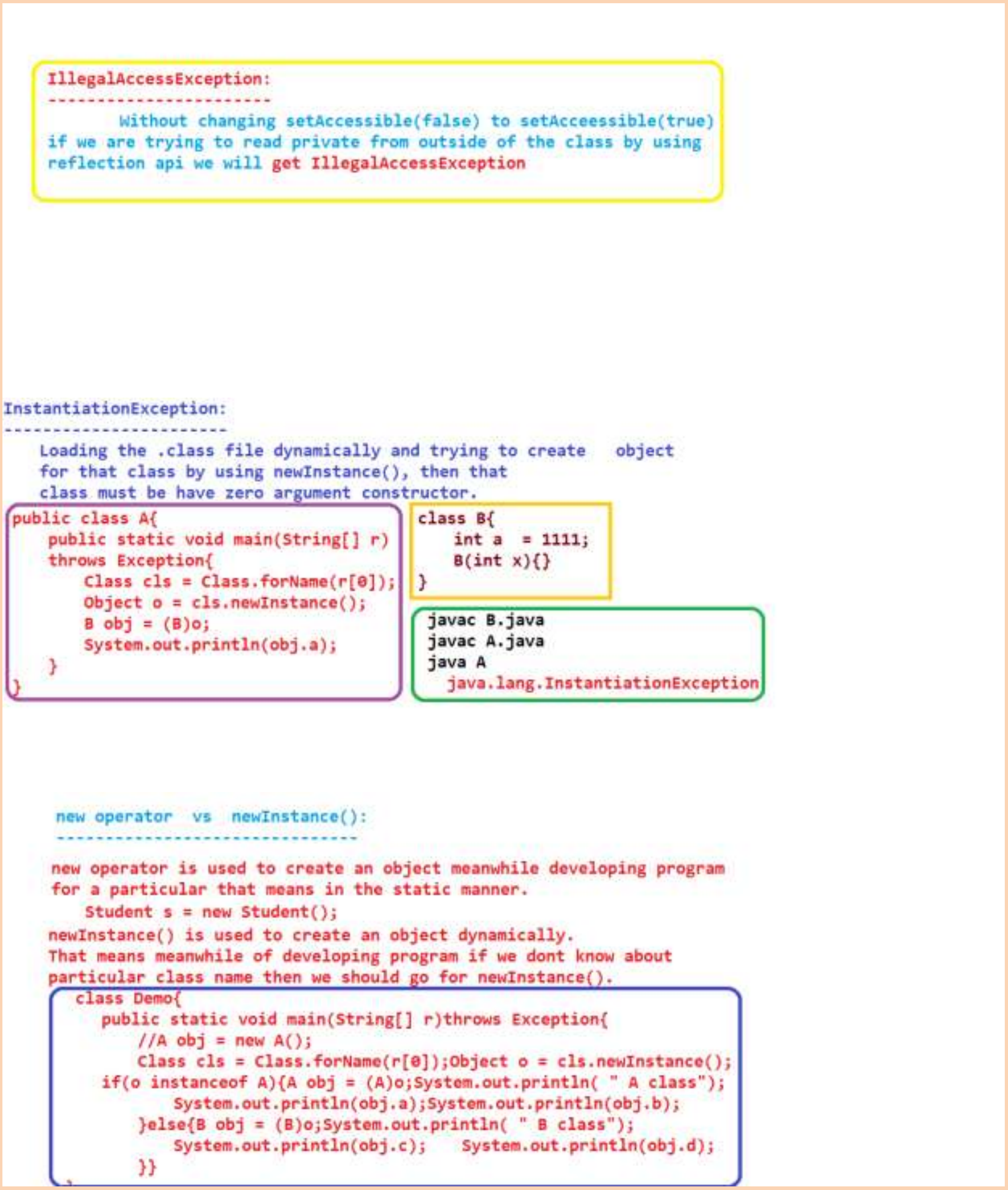
****



s.o.p statements are used to do the debugging.
if we are using sop for debug, before giving the project to the client we need to delete sop's.
when ever we using sop the controle is goning console come back to java app, it may be leads to time consuming.
with the help of sop there may be chance to guess our code by the client.

to avoiding above drawback java introduced topic like assert in the version java 1.4.

if assert statement give true then our controle is keep on going otherwise we will get assertio n error.



```
IllegalAccessException:
-----------------------
       Without changing setAccessible(false) to setAcceeessible(true)
if we are trying to read private from outside of the class by using
reflection api we will get IllegalAccessException
```

```
InstantiationException:
-----------------------
   Loading the .class file dynamically and trying to create   object
   for that class by using newInstance(), then that
   class must be have zero argument constructor.
```

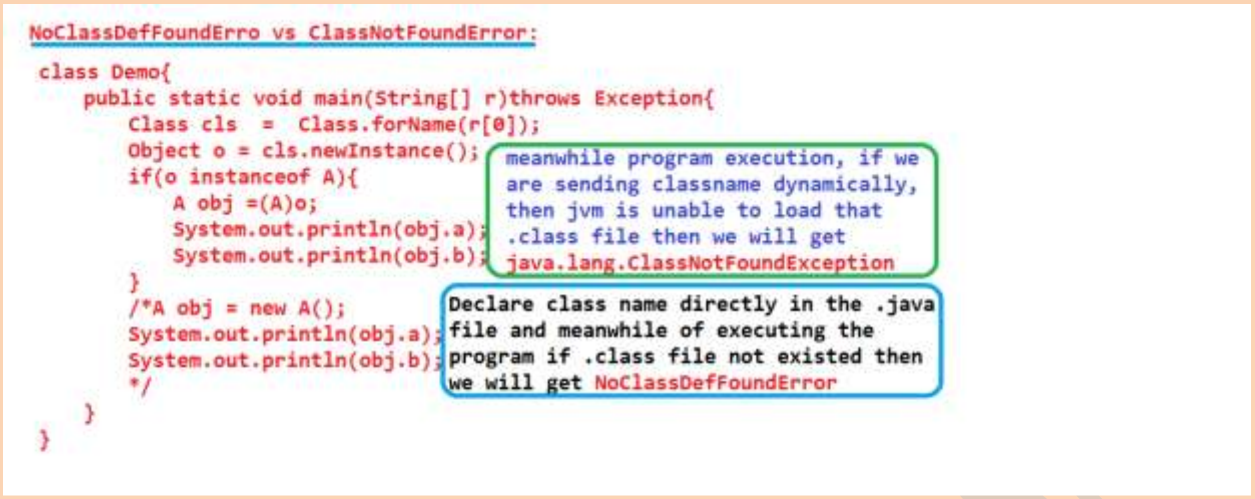
```
public class A{
    public static void main(String[] r)
    throws Exception{
        Class cls = Class.forName(r[0]);
        Object o = cls.newInstance();
        B obj = (B)o;
        System.out.println(obj.a);
    }
}
```

```
class B{
    int a  = 1111;
    B(int x){}
}
```

```
javac B.java
javac A.java
java A
  java.lang.InstantiationException
```

```
new operator  vs  newInstance():
-------------------------------
new operator is used to create an object meanwhile developing program
for a particular that means in the static manner.
   Student s = new Student();
newInstance() is used to create an object dynamically.
That means meanwhile of developing program if we dont know about
particular class name then we should go for newInstance().
```

```
  class Demo{
     public static void main(String[] r)throws Exception{
         //A obj = new A();
         Class cls = Class.forName(r[0]);Object o = cls.newInstance();
     if(o instanceof A){A obj = (A)o;System.out.println( " A class");
             System.out.println(obj.a);System.out.println(obj.b);
         }else{B obj = (B)o;System.out.println( " B class");
             System.out.println(obj.c);    System.out.println(obj.d);
         }}
```



NoClassDefFoundErro vs ClassNotFoundError:

```
class Demo{
    public static void main(String[] r)throws Exception{
        Class cls  =  Class.forName(r[0]);
        Object o = cls.newInstance();
        if(o instanceof A){
            A obj =(A)o;
            System.out.println(obj.a);
            System.out.println(obj.b);
        }
        /*A obj = new A();
        System.out.println(obj.a);
        System.out.println(obj.b);
        */
    }
}
```

meanwhile program execution, if we are sending classname dynamically, then jvm is unable to load that .class file then we will get java.lang.ClassNotFoundException

Declare class name directly in the .java file and meanwhile of executing the program if .class file not existed then we will get NoClassDefFoundError

```
import java.util.Scanner;

class ExceptionDemo{

        public static void main(String[] s){

                try{

                System.out.println("try block");

                Scanner scan = new Scanner(System.in);

                System.out.println("enter array size");

                int size = scan.nextInt();

                int [] a = new int[size];//NASE

                System.out.println("enter value for c");

                int c = scan.nextInt();

                int b = 10/c;

                System.out.println("enter index for character");

                int index = scan.nextInt();
                System.out.println("ram".charAt(index));//SIOOBE
                int [] c1= {11,22,33};
                System.out.println(c1[3]);//AIOOBE
                }

                catch(NegativeArraySizeException ae){
                    System.out.println("catch block");
                    System.out.println("dont give negative values");
                }
                catch(ArithmeticException e){
                    System.out.println("catch block");
                System.out.println("dont enter zero as denominator");
                }
                catch(StringIndexOutOfBoundsException e){
                    System.out.println("catch block");
System.out.println("there is no character give appropriate index
number");
                }
                catch(Exception e){
                    System.out.println("catch block-exception");
                }
                finally{
```

```
                    System.out.println("finally block");
                }
        }
}
class MethodOverloading{
        void m1(Exception x){
                System.out.println("Exception-m1()");
        }
        void m1(ArithmeticException y){
                System.out.println("ArithmeticExcepiton-m1()");
        }
        public static void main(String[] s){
                    System.out.println("MethodOverloading class main
method");
                MethodOverloading mo = new MethodOverloading();
                //mo.m1(10/0);
                int a = 10/0;
                mo.m1(a);
        }
}
```

CTError: m1(int x) not existed

===
=============================

```
class MethodOverloading{
        void m1(Exception x){
                System.out.println("Exception-m1()");
        }
        void m1(ArithmeticException y){
                System.out.println("ArithmeticExcepiton-m1()");
        }
        void m1(int x){
                System.out.println("int-m1()");
        }

        public static void main(String[] s){
                        System.out.println("MethodOverloading class main
method");
                MethodOverloading mo = new MethodOverloading();
                //mo.m1(10/0);
                int a = 10/0;
                mo.m1(a);
        }
}
RuntimeException: java.lang.ArithmeticException
========================================
============================
class MethodOverloading{
        void m1(Exception x){
                System.out.println("Exception-m1()");
        }
        void m1(ArithmeticException y){
                System.out.println("ArithmeticExcepiton-m1()");
        }
        void m1(int x){
                System.out.println("int-m1()");
        }

        public static void main(String[] s){
                        System.out.println("MethodOverloading class main
method");
                MethodOverloading mo = new MethodOverloading();

                mo.m1(10/0);
        }
}
```

RuntimeException: java.lang.ArithmeticException

java.lang.Object:

It is the super class for all predefines and user defined classes in java either directly or indirectly.

```
class A{
}
```

If we write class in the above manner java internally converts into as a sub class of java.lang.Object class.

```
javac A.java

javap A

class A extends java.lang.Object{
	A(){
		super();
	}
}
```

java.lang.Object class having bellow methods.

1) getClass()

2) toString()

3) equals()

4) hashCode()

5) clone()

6) notify()

7) notifyAll()

8) wait()

9) wait(-)

10) wait(-,-)

11) finalize()

Among the 11 methods, we can override 5 methods only.

Remaining methods are not participated in the method override concept.

Below methods are participated in the method override concept.

1) toString()
2) eauals()
3) hashCode()
4) clone()
5) finalize()

toString():

```
	public String toString(){
return getClass().getName() + "@"+Integer.toHexString(hashCode());
	}
```

In the above syntax : getClass() method will create java.lang.Class reference with specific class bytecode.

getName() is the non-static method in java.lang.Class, with help of this method we will get class name and package information of an loaded class.

i.e: System.out.println(new Student());

s.o.p method internally calls toString() method.

This method(getClass().getName()) will print the output like "packagename.Student".

"@" --> whatever the code which is available in double quotes as it is printed on the console.

hashCode() method always return one unique identification number of memory in integer(decimal number system) format.

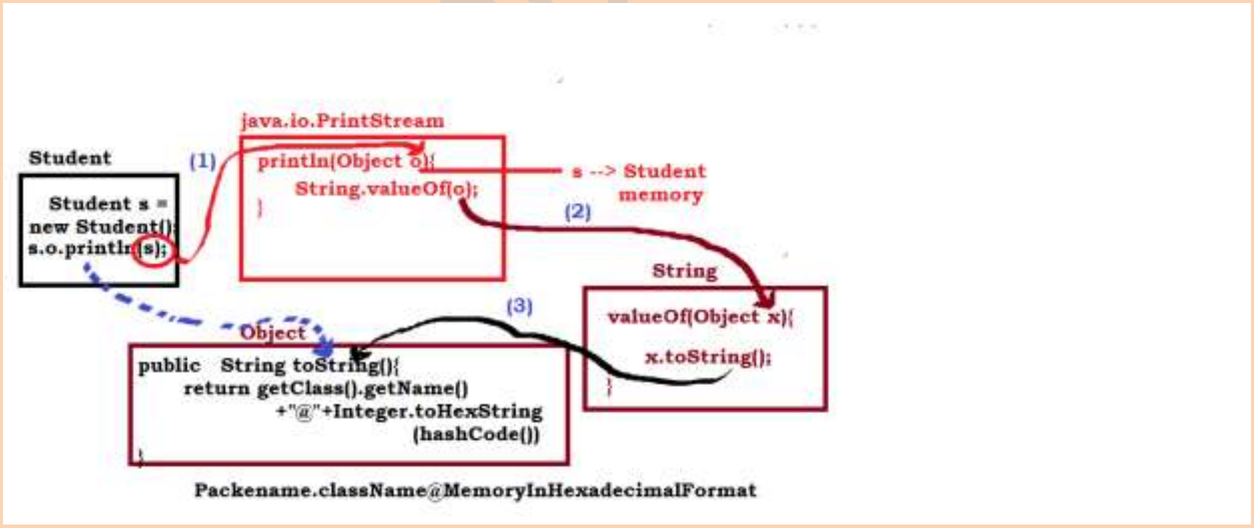
This integer number will converted into hexa decimal format with the help of toHexString() of java.lang.Integer class.

Ex: Assume if hashCode() return 159--> this 159 will converted into hexadecimal format like "9f".

finally we will get one output as

"packename.Student@9f"

Note: Java avoids pointers concept with the help of toString().



package object;

```
class Employee{
        int eid;
        String ename;
        double esal;
        byte eage;
        Employee(int eid, String ename, double esal, byte eage){
                this.eid = eid;
                this.ename = ename;
                this.esal = esal;
                this.eage = eage;
        }
        public String toString(){
             System.out.println("controle comes to toString()");
//return getClass().getName()+"@"+Integer.toHexString
                                              (hashCode());
        return eid+" "+ename+" "+esal+" "+eage;
        }
}
public class ToStringDemo {
        public static void main(String[] args) {
                Employee e1 = new
        Employee(101,"ram",15000,(byte)28);
                System.out.println(e1);
        }
}
```

hashCode():

This method provides a unique identification number of memory.

Whenever we use new keyword, JVM blindly provides new memory, that means new hashcode.

JVM is not check content to produce the hashcode.

JVM is always provides a new hashcode for different object.

If we want to provide hashCode based on content then we should go for override hashCode ().

"==" and equals () of Object class always calls JVM provides hashCode (). So if want call the user define hashCode () definitely we should override equals () of Object class.

equals (): This method is always checks hashCode() of objects, if hashCodes of any two objects are different then equals() method returns false, otherwise returns true.

If we want to call our own hashCode () from equals (), we need to override the equals () in our class.

We can know the identification number of memory by using two ways.

1) hashCode ():

This method first preference gives to our own hashCode(), if not available Object class hashCode()(JVM provide memory)

2) identityHashCode();

This is static method of System class, it is always provides jvm provide hashcode(identification number of memory)

```
package object;

class Student{

	int sid;

	String sname;

	Student(int sid, String sname){

		this.sid = sid;

		this.sname = sname;

	}

	static int i=0;

	@Override

	public int hashCode(){

		i++;

		return sid+sname.hashCode();

	}

	public boolean equals(Object o){

		if(o instanceof Student){

			Student s = (Student)o;

			if(this.sid == s.sid &&
this.sname.equals(s.sname)){

				return true;

			}
```

```
                else return false;
            }
            else return false;
            /*if(this.hashCode() == s.hashCode()){
                return true;
            }
            return false;*/
        }
}
public class HashCodeDemo {
        public static void main(String[] args) {
            Student s1 = new Student(101,"ram");
            Student s2 = new Student(102,"sam");
            System.out.println(s1==s2);
            Student s3 = new Student(101,"ram");
            System.out.println(s1==s3);
            System.out.println("s1: "+s1.hashCode());
            System.out.println("s2: "+s2.hashCode());
            System.out.println("s3: "+s3.hashCode());
            System.out.println(Student.i);
            System.out.println("********************");
            System.out.println("jvm s1:
"+System.identityHashCode(s1));
            System.out.println("jvm s2:
"+System.identityHashCode(s2));
            System.out.println("jvm s3:
"+System.identityHashCode(s3));
            System.out.println("********************");
            System.out.println(s1.equals(s2));
            System.out.println(s1.equals(s3));
      }
}
```

cloning:

Creating new object with updated data is called cloning.

In the cloning, we will get new object with new memory.

For doing cloning java provides one predefine method called clone(), this is protected method in java.lang.Object class.

clone() will throws one checked/compile time exception i.e "java.lang.CloneNotSupportedException"

If any class data is partispated in the cloning that class must be implements java.lang.Cloneable interface.

java.lang.Cloneable is an marker interface.

(An interface which doesn’t contains any methods)

Whenever we doing the cloning, non-static data are not going to be loaded.

(Creating new object with new memory with updated data without duplicate code is called cloning).

```
package object;

class Address implements Cloneable{

	String cityName = "hyderabad";

	String stateName = "telangana";

}

public class Student implements Cloneable{

	int sid = 111;

	String sname = "ram";

	Address addr = new Address();

	public static void main(String[] args) throws

	CloneNotSupportedException{

		Student s1 = new Student();

		Student s2 = (Student)s1.clone();

		System.out.println(s1.sid);

		System.out.println(s1.sname);

		System.out.println(s1.addr.cityName);

		System.out.println(s1.addr.stateName);

		System.out.println("************");
System.out.println(s2.sid);
System.out.println(s2.sname);
System.out.println(s2.addr.cityName);
System.out.println(s2.addr.stateName);
System.out.println("************");
s1.sid = 222;
s1.sname = "sam";
s1.addr.cityName = "vizag";
s1.addr.stateName = "AndharaPradesh";
System.out.println(s1.sid);
System.out.println(s1.sname);
System.out.println(s1.addr.cityName);
System.out.println(s1.addr.stateName);
System.out.println("************");
System.out.println(s2.sid);
System.out.println(s2.sname);
System.out.println(s2.addr.cityName);
System.out.println(s2.addr.stateName);
System.out.println("************");
s2.sid = 333;
s2.sname = "suji";
s2.addr.cityName = "sklm";
```

```
            s2.addr.stateName = "AP";
            System.out.println(s1.sid);
            System.out.println(s1.sname);
            System.out.println(s1.addr.cityName);
            System.out.println(s1.addr.stateName);
            System.out.println("************");
            System.out.println(s2.sid);
            System.out.println(s2.sname);
            System.out.println(s2.addr.cityName);
            System.out.println(s2.addr.stateName);
            System.out.println("************");
      }
}
```

Cloning is two types

1. Shallow cloning
2. Deep cloning.

Shallow cloning:

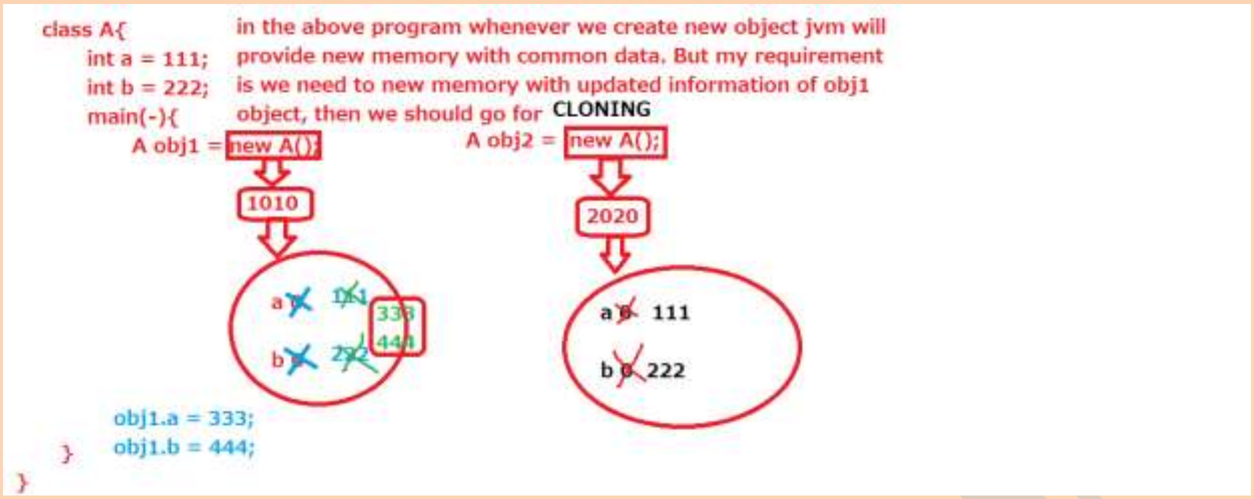
If we are doing updating on cloned object or existed object those changes will be effects to other objects is called shallow cloning.

Ex: Bellow program

```
public class A implements Cloneable{
    int a = 111;
    int b = 222;
    public static void main(String[] args)
    throws CloneNotSupportedException{
        A obj1 = new A();
        System.out.println("obj1: "+obj1.a+"...."+obj1.b);
        obj1.a = 333;
        obj1.b = 444;
        System.out.println("obj1: "+obj1.a+"...."+obj1.b);
        A obj3 = (A)o;//down casting
        System.out.println("obj3: "+obj3.a+"...."+obj3.b);
    }

}
```

```
package clonning;
class Address{
    String cityName = "Hyderabad";
    String stateName = "Telangana";
}
public class Student implements Cloneable{
    int sid = 101;
    String sname = "ram";
    Address addr = new Address();
    public static void main(String[] args)
    throws CloneNotSupportedException{
        Student s1 = new Student();

        System.out.println(s1.sid);
        System.out.println(s1.sname);
        System.out.println(s1.addr.cityName);
        System.out.println(s1.addr.stateName);

        s1.sid = 201;
        s1.sname = "yaane";
        s1.addr.cityName = "vizag";
        s1.addr.stateName = "AP";

        System.out.println("=================");
```

```
            System.out.println(s1.sid);
            System.out.println(s1.sname);
            System.out.println(s1.addr.cityName);
            System.out.println(s1.addr.stateName);

            System.out.println("=================");
            Student s2 = new Student();

            System.out.println(s2.sid);
            System.out.println(s2.sname);
            System.out.println(s2.addr.cityName);
            System.out.println(s2.addr.stateName);

            System.out.println("=================");

            Object o = s1.clone();
            Student s3 = (Student)o;

            System.out.println(s3.sid);
            System.out.println(s3.sname);
            System.out.println(s3.addr.cityName);
            System.out.println(s3.addr.stateName);

            System.out.println("==============");
            s3.addr.cityName = "chennai";
            System.out.println(s3.addr.cityName);
            System.out.println("==============");
            System.out.println(s1.addr.cityName);

        }
    }
```



class A{
int a = 111;
int b = 222;
main(-){
A obj1 = new A();
1010
a 111 333
b 222 444
obj1.a = 333;
obj1.b = 444;
}
}
in the above program whenever we create new object jvm will provide new memory with common data. But my requirement is we need to new memory with updated information of obj1 object, then we should go for CLONING
A obj2 = new A();
2020
a 0 111
b 0 222

Deep cloning:

If we are doing updating on cloned object those changes will not be effected to existed object is called deep cloning.

Ex: Bellow program.

```
class Address implements Cloneable{
     String cityName ="Hyderabad";
     String stateName = "Telangana";
     public Address clone()throws CloneNotSupportedException{
          return (Address)super.clone();
     }
}
public class Student implements Cloneable{
     int sid = 101;
     String sname = "suji";
     Address addr = new Address();

     @Override
     public Student clone()throws CloneNotSupportedException{
          Student s3 = (Student)super.clone();
          this.addr = this.addr.clone();
          return s3;
     }

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

```
    throws CloneNotSupportedException{
         Student s1 = new Student();
         System.out.println("s1: "+s1.sid+".."+s1.sname+".."

    +s1.addr.cityName+"..."+s1.addr.stateName);
         System.out.println("============================");

         s1.sid=201;
         s1.sname="ram";
         s1.addr.cityName="vizag";
         s1.addr.stateName="AP";
         System.out.println("s1: "+s1.sid+".."+s1.sname+".."

    +s1.addr.cityName+"..."+s1.addr.stateName);
         Student s2 = s1.clone();
         System.out.println("============================");
         System.out.println("s2: "+s2.sid+".."+s2.sname+".."

    +s2.addr.cityName+"..."+s2.addr.stateName);
         System.out.println("============================");
         s1.addr.cityName="chennai";
         s1.addr.stateName="tn";
         System.out.println("s1: "+s1.sid+".."+s1.sname+".."

    +s1.addr.cityName+"..."+s1.addr.stateName);
         System.out.println("============================");
         System.out.println("s2: "+s2.sid+".."+s2.sname+".."

    +s2.addr.cityName+"..."+s2.addr.stateName);

    }

}
```