08/11/2017

* Packages-

> Reusability of one project related files to another project.

> It is defined as the collection of class files for developing a project in single folder specification.

> It is used for placing the duplicate files within the same location.

> It is used for specifying the scope of class and its members by using access modifiers.

=> Packages are classified into two types-

1) User defined packages / Custom packages

2) Predefined packages / Built in packages

NOTE-

Custom packages should never starts with java because predefined packages starts with java.

ex-

import java.xxxx.xxxx → invalid (i.e predefined only)

import com.java.xxxx → valid (can be used at 2nd)

2) Build in Packages-

These packages are provided by java vendors and it contains somany predefined classes and each classes contains so many predefined methods, variables etc. which are known as API (Application Programming Interface)

-> So, there are predefined API's for Multitasking, Network, error handling etc.

-> Predefined package folder is located at the JDK software location as a compressed file named "src"

-> Every built in packages starts with either;

1) java -> used for developing standalone application

2) javax -> used for developing enterprise java application (i.e Web application)

* Java packages are as below -

=> Packages.

- java.lang
- java.io
- java.util
- java.lang.reflect
- java.nio (from java07)
- java.applet
- java.awt
- java.network
- java.math
- java.rmi
- java.sql
- javax.sql
- javax.swings
- etc.

=> Classes of java.lang package

Object	Maths	Character	Runtime Exception
String	Number	Boolean	Thread
StringBuffer	Byte	Float	ThreadGroup
StringBuilder	Short	Double	etc.
System	Integer	Void	
Runtime	Long	Exception	

=> Interfaces of java.lang package -

- Cloneable
- Comparable
- CharSequence
- Runnable
- etc.

* java.lang -

This package is the default package (imported package) in every java application. So, without importing java.lang package we can call the classes of lang package directly.

NOTE -

If we provide import statement explicitely for lang package then we will get error.

error → the import statement for java.lang package is never used.

① Object Class-

It is the default super class for any type of classes without using "extends" keywords.

ex-

Class Hello{} -> no extends keyword. i.e its direct SuperClass

Class A extends Hello{} -> extends keyword i.e Object will become indirect super class.

NOTE-

All the member of SuperClass can be used directly with any class reference since from any super class, the member will be inherited to sub class.

=> To know predefined members of any class or members of class we can use the below syntax.

javap <fully Qualified Class Name>

javap java.lang.Object -> valid

javap Object -> invalid statement
i.e- error- no class located

=> The Object class doesn't have any Super Class.

* Members of java.lang.Object class-

i) public final native Class getClass();

native show(); -> native method implementation (i.e body of the method)

native methods; -> method/body will be provided by JVM from Low Level Language and native stack interface or stack library.

NOTE-

If we declare method with native keyword then we should not provide the body of method.

class -> keyword used for defining / declaring the class

Class -> predefined class in lang package (i.e java.lang.Class)

ex-

```
public class Lab {
    public static void main (String[] args){
        System.out.println ("Main Started");
        Hello h = null;
        System.out.println (h.x);
//      System.out.println (h.y); -> invalid
        h = new Hello();
    }
}

Class Hello{
    static int x=90;
    static{
        System.out.println ("SB -> Hello");
    }
    {
        System.out.println ("IB -> Hello");
    }
}
```

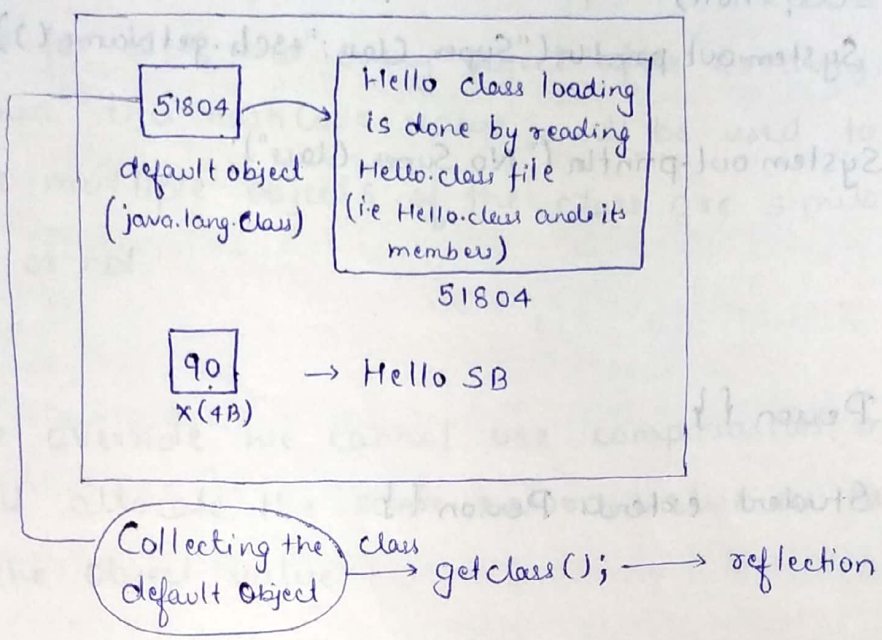
* Object and reference creation in JVM-Memory.

```
Lab Test{
    public static void main (String arg[]){
        Hello h=null;
        System.out.println(h.x);
    }

Class Hello{
    static int x=90;
    static {System.out.println ("Hello SB");}
    { System.out.println("Hello IB");}
}
```

NOTE-

getclass, we cannot override as its a final value

JVM Memory



=> getclass is used for collecting the default object which is created by the JVM internally at the time of class loading of every class to get the information about the class and its internal member either for predefined class or custom class.

=> The default Object is of java.lang.Class so the return type of this method is class return type.

=> This method cannot be overridden in sub class because of final declaration and getclass implementation inside the object class is fixed implementation

ex-

```
public class Lab443{
    public static void main (String[] arg){
    System.out.println("Main Started");

    String str = "SD";
    Student stu = new Student();
    Object obj = new Object();

    Class cls = str.getClass();      // -> Class-java.lang.String SuperClass -> Object
//  Class cls = stu.getClass();      // -> Class- Student SuperClass -> Person
//  Class cls = obj.getClass();      // -> Class-java.lang.Object SuperClass -> NoSC

    Class scls = cls.getSuperClass();
```

```
System.out.println ("Class Name :"+ cls.getName ());
if (scls!=null)
    System.out.println ("Super Class :"+scls.getName ());
else
    System.out.println ("No Super Class");
}
}

Class Person {}

class Student extends Person {}
```

In the above code we can check for the class and the super class of the members (i.e obj. class or variables)

for.

```
String str ="SD";
```

Class is → java.lang.String

Super Class → java.lang.Object

for

```
Student stu = new Student();
```

Class is → Student

Super Class → Person

for

```
Object obj=new Object();
```

Class is → java.lang.Object

Super Class → No Super Class. (i.e Object is already a Super Class)

ii) public native int hashCode() -

It is a unique numerical representation which is allocated for each and every object after creating, by the JVM, then this hashCode value will be used to compare when the multiple objects of the class are similar are duplicate or not.

* NOTE-

If we override we cannot use comparasion method. i.e. it will allocate the same return value as new value for all the object value.

Overriding can happen but there will be some flaws.

=> There are two types of implementation of hash code method.

1) Object class implementation -

Used for comparing multiple object of the class

2) Over ridden hash code implementation -

Will not be used for comparing the multiple object of the class.

ex-

```
class Student{
    int sid;
    long phone;
    Student(int sid, long phone){
        this.sid = sid;
        this.phone = phone;
    }
```

Main class

=> next page.

```
public class Lab444{
    public static void main (String[] arg){
        Student st1 = new Student (99, 31903290);
        Student st2 = new Student (99, 31903290);
        Student st3 = new Student (88, 65799999);
        Student st4 = st1;

        System.out.println ("Student 1:"+ st1.hashCode());
        System.out.println ("Student 2:"+ st2.hashCode());
        System.out.println ("Student 3:"+ st3.hashCode());
        System.out.println ("Student 4:"+ st4.hashCode());

        System.out.println (st1 == st2);
        System.out.println (st1 == st3);
        System.out.println (st1 == st4);
        System.out.println (st3 == st4);
    }
}
```

O/P

```
-> Student 1 : 366712642
   Student 2 : 1829164700
   Student 3 : 2018699554
   Student 4 : 366712642
```

→ return type will always be integral.

```
false -> memory comparasion (st1, st2)
false -> memory comparasion (st1, st3)
true  -> memory comparasion (st1, st4)
false -> memory comparasion (st3, st4)
```

ex- Overrinding hashCode();

```
public class Lab445{
    public static void main (String[] arg){
        Employee em1 = new Employee(99, 31903290);
        Employee em2 = new Employee(99, 31903290);
        Employee em3 = new Employee(88, 65799999);
        Employee em4 = em1;
    System.out.println ("Employee 1:"+ em1.hashCode());
    System.out.println ("Employee 2:"+ em2.hashCode());
    System.out.println ("Employee 3:"+ em3.hashCode());
    System.out.println ("Employee 4:"+ em4.hashCode());

    System.out.println (em1 == em2);
    System.out.println (em 1 == em3);
    System.out.println (em 1 == em4);
    System.out.println (em 3 == em4)
  }
}
class Employee{
    int eid;
    long phone;
    Employee (int eid, long phone){
        this.eid= eid;
        this.phone = phone;
    }
    public int hashCode(){
        return (int)(phone | eid);
    }
```

} → overriding the hashCode method.

O/p

```
Main Started
Employee 1: 31903355          true
Employee 2: 31903355          false
Employee 3: 65800063
Employee 4: 31903355
false
false
```

iii) public String toString();

09/11/2017

It is predefined method of Object class which is used for hiding the memory address of reference which is created for each and every object and reference variable is storing the address of actual value's memory location. So when we display the reference variable of any class object with SOP statement then internally JVM is calling the toString() implementation of Object class

i.e.-

<fully Qualified Class Name> @ < Hexadecimal of HashCode >

NOTE-

In java we cannot predict the address of any variable or any element in java application.

ex-

```
public class Lab446{
    public static void main (String[] arg){
        Student st1 = null;
        System.out.println (st1);                    -> null
        st1 = new Student (99,"Sri");
        System.out.println (st1);                    -> JLC.Student@15db9742
        String str = new String ("Java Learning Center");  -> already overridden (String)
        System.out.println (str);                    -> Java Learning Center
        Integer ref = new Integer (123);             -> already overridden (Integer)
        System.out.println (ref);                    -> 123.
    }
}

Class Student {
    int sid;
    String name;
    Student (int sid, String name){
        this.sid = sid;
        this.name = name;
    }
}
```

O/P

```
null
JLC.Student@15db9742
Java Learning Center
123
```

ex. Manually overridding toString()

```
public class Lab448{
    public static void main (String[] args){
    Student st1 = new Student (99, "Sri");
    Student st2 = new Student (88, "Nivas");
    System.out.println (st1);
    System.out.println (st2);
    }
}

class Student{
    int sid;
    String name;
    Student (int sid, String name){
        this.sid = sid;
        this.name = name;
    }
    public String toString (){
        return sid + "\t" + name;
    }
}
```

→ overridding syntax to return actual value to reference call. st1 & st2 insted of memory address call.

O/P

```
99    Sri
88    Nivas
```

iv) public boolean equals (Object obj) -

It is used for comparing actual contents of two reference variable

=> Object equal() implementation -

```
public boolean equal (Object obj){
    return (this == obj);
}
```

=> equal() has two implementation -

1) Object equal() implementation which is used for comparing address of reference variable but not its content because internally inside the Object class it is providing implementation with == operator since the actual member of the custom classes is not known by JVM.

2) Overridden equal() implementation can be provided within the custom class to compare the actual contents of the Object as per the class member.

ex -

```
public class Lab449{
    public static void main (String [] args){
        String st1 = new String ("Sri");
        String st2 = new String ("Sri");
        String st3 = new String ("Manish");
        System.out.println (st1 == st2);
        System.out.println (st1 == st3);
        System.out.println (st1.equals(st2));
        System.out.println (st1.equals (st3));
    }
}
```

-> already overridden String. so value can be provided directly

-> it will compare the address, not content

-> it will compare the actual content

O/P

false	true
false	false

ex- Manually overridding boolean equals()

```
public class Lab451 {
    public static void main (String[] arg){
        Student st1 = new Student ( 99,"Sri");
        Student st2 = new Student ( 99,"Sri");
        Student st3 = new Student (88,"Manish");
        Student st4 = st1;

        System.out.println ("using == operator");
        System.out.println ( st1 == st2);
        System.out.println (st1 == st3);
        System.out.println (st1 == st4);
        System.out.println ( st2 == st3);

        System.out.println ("using equals()");
        System.out.println (st1.equals (st2));
        System.out.println (st1.equals (st3));
        System.out.println (st1.equals (st4));
        System.out.println (st2.equals (st3);
    }
}
Class Student {

    int sid;
    String sname;
    Student (int sid,String sname){
        this.sid = sid;
        this.sname = sname;
    }
    public boolean equals (Object obj){
not <// if (obj instanceof Student){
mandatory.
but
can be
used.
            Student st = (Student) obj;
            return this.sid= st.sid && this.sname.equals (st.name);
//          }
//          return false;
        }
    }
```

O/P-

false	true
false	false
true	true
false	false

Q→ Collect the customer details regarding Customer ID, Name, Mobile Number, Mail ID, and count of product and display the details of all customers and find out which one are duplicate customer details with its actual content.

Ans

```
public class Details {
    public static void main (String [] arg){
     Customer C1 = new Customer (101,"Ram", 9900112233L,"ram@xyz.com", 9);
     Customer C2 = new Customer (102,"Shyam", 9900334455L,"Shyam@xyz.com",7);
     Customer C3 = new Customer (101,"Ram", 9900112233L,"ram@xyz.com",9);

     System.out.println ("*** Customer Details ***");
     System.out.println (c1);
     System.out.println (C2);
     System.out.println (C3);

     System.out.println ("*** Duplicate Customer Details ***");
     if (c1.equals (C2)) {
          System.out.println ("C1 & C3 are Duplicate Customers");
     } else if (c1.equals(c3)){
          System.out.println ("C1 & C3 are Duplicate Customers");
     } else if ( c 2.equals (C3)){
          System.out.println ("C2 & C3 are Duplicate Customers");
     } else {
          System.out.println ("No Duplicate Customer found");
     }
    }
}
```

```
class Customer {
    int cid;
    String cname;
    long phone;
    String mid;
    int count;
    Customer (int cid, String cname, long phone, String mid, int count){
        this.cid = cid;
        this.cname = cname;
        this.phone = phone;
        this.mid = mid;
        this.count = count;
    }

  public String toString(){
      return cid+"\t"+cname+"\t"+phone+"\t"+mid+"\t"+count;
  }

  public boolean equals (Object obj){
      Customer c = (Customer) obj;
      return this.cid == c.cid && this.cname.equals(c.cname)
          && this.phone == c.phone && this.mid.equals(c.mid)
          && this.count == c.count;
    }
}
```

v) Cloning:-

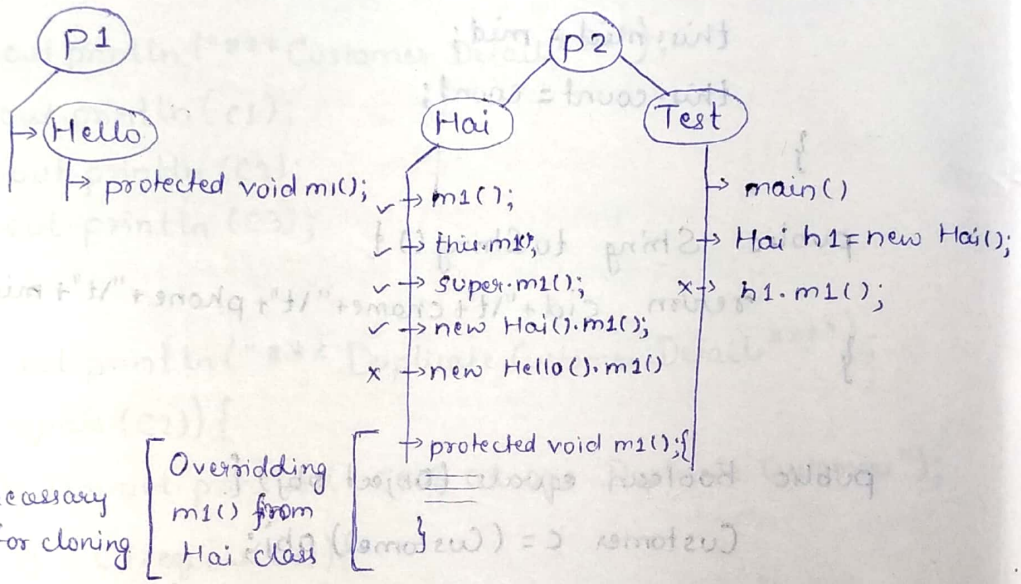
10/11/2017

Cloning is the process of creating copy of actual object data with current values only

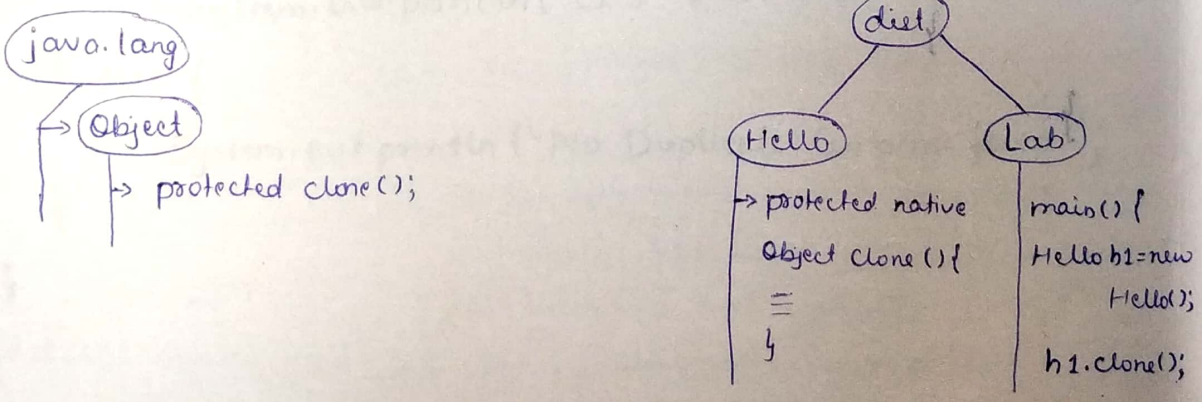
To perform cloning we need to use clone method of Object class.

Syntax-

```
protected native Object clone();
```

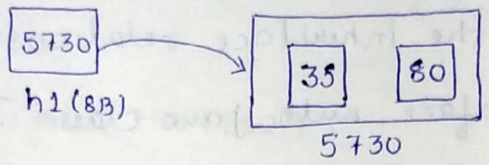


Similarly.

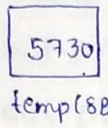


ex

Hello h1 = new Hello (35,80);



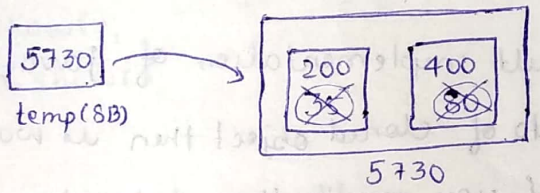
Hello h1 = temp;



temp reference is created and is pointed to the same actual object

temp.x = 200;
temp.y = 400;

So, the actual value will be changed and no cloning is happening.



NOTE-

1) The protected method of one class can be accessed any where in the same package and we can access in the current sub class of different package in following ways: i.e - directly, this, super and current subclass object. But if we access the protected member with the super class Object or other sub class object then there is no visibility.

But if we want to access the protected method in the other sub class of different package then we need to override the protected method in current sub class then the override method will become member to the different package and we can access any where in the different package.

2) By default any java class Object is not eligible for Cloning. To make the Java Objects eligible for cloning we need to provide the inheritance relationship with Cloneable Marker Interface with java class. In case if java Objects are not sub type of Cloneable Interface then it will throw error.

Close Not Supported Exception.

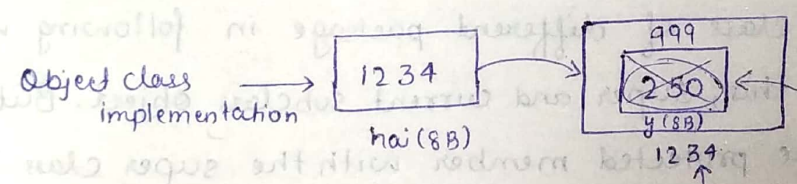
* Types of Cloning-

1) Shallow Cloning
2) Deep Cloning.

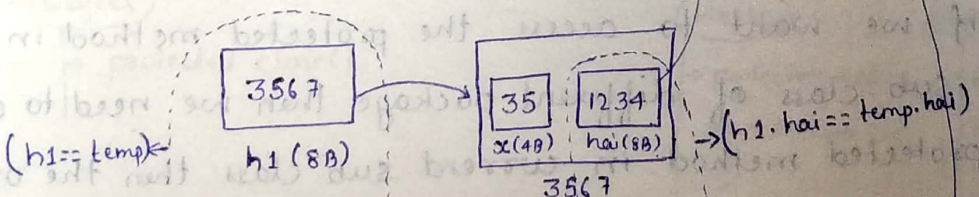
1) Shallow Cloning-

This is default implementation of java.lang.Object. If you modify the data of cloned object then it won't affect the actual object data. But if you modify the data of member Object then it will affect other object also.

Hai hai = new Hai();



Hello h1 = new Hello(35, hai);

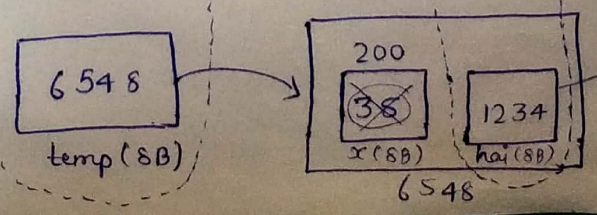


h1.hai = temp.hai;

Hello temp = (Hello)h1.clone();

S.O.P (h1 == temp); → false , S.O.P(h1.hai == temp.hai); → true

temp.x = 200;
temp.y = 999;



its original value will get changed

ex-

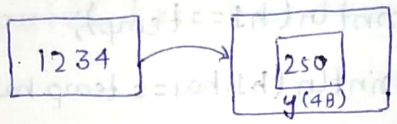
```
public class Lab452 {
    public static void main (String[] args) throws CloneNotSupport
                                                    Exception {
        Hai hai = new Hai (250);
        Hello h1 = new Hello (35, hai);
        Hello temp = (Hello) h1.clone();
        h1.hai = temp.hai;
        System.out.println(h1 == temp);
        System.out.println (h1.hai == temp.hai);
        temp.y = 999;
        h1.show();
        temp.show();
        temp.hai.x = 200;
        h1.show();
        temp.show();
    }
}

class Hai {
    int x;
    Hai(int x){
        this.x = x;
    }
}

class Hello implements Cloneable {
    int y;
    Hai hai;
    Hello (int y, Hai hai){
        this.y = y;
        this.hai = hai;
    }
    void show(){
        System.out.println ("Hello => y:" +y);
        System.out.println (" Hai => x: " hai.x);
    }
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}
```

2) Deep Cloning-

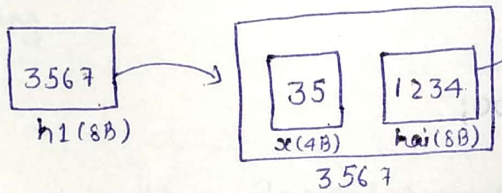
There is no default implementation of deep cloning (i.e. custom implementation is required.

If we modify the data of cloned object then it won't affect the actual object but if you modify the data of the members object then it also will not affect the other object.

Hai hai = new Hai();



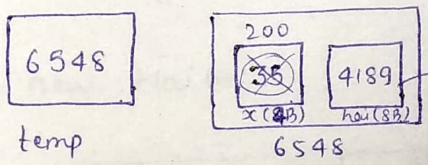
Hello h1 = new Hello(35, hai)



Hello temp = (Hello)h1.clone();

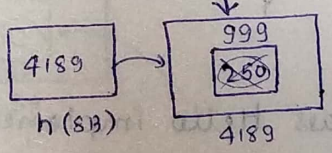
→ typecasting (downcastion)

Super class to subclass assignment.



temp.x = 200;

temp.hai.y = 999;



h1 == temp

h1.hai == temp.hai

instanceof-

We need to use instanceof to check if it's a subtype of Clonable or not.

ex-

```
public class Lab453{
    public static void main(String[] arg) throws CloneNotSupportedException{
        Hai hai = new Hai(250);
        Hello h1 = new Hello(35, hai);
        Hello temp= ~~new~~ (Hello)h1.clone();
        h1.hai = temp.hai;
        System.out.println(h1 == temp);
        System.out.println(h1.hai == temp.hai);
        h1.show();
        temp.show();
        temp.y = 999;
        h1.show();
        temp.show();
        temp.hai.x = 200;
        h1.show();
        temp.show();
    }
}
class Hai{
    int x;
    Hai(int x){
        this.x = x;
    }
}
class Hello implements Cloneable{
    int y;
    Hai hai;
    Hello(int y, Hai hai){
        this.y = y;
        this.hai = hai;
    }
    void show(){
        System.out.println("Hello=>y:"+y);
        System.out.println("Hai=>x:"+hai.x);
    }
```

```
public Object clone() throws CloneNotSupportedException{
    if (this instanceof Cloneable){
        Hai hai = new Hai(this.hai.x);
        Hello h = new Hello(this.y, hai);
        return h;
    } else
        throw new CloneNotSupportedException(getClass().getName())
    }
}
```

* Memory Management-

JAVA has a strong memory management

1) JVM will request OS to provide some amount of memory to run java application.
2) O.S will allocate some specified amount of memory for the JVM based on RAM size.

3) By this initialization of memory, JVM will initialize with certain amount of memory and JVM memory will be created
4) It will convert byte code to machine code and the memory will be allocated as per the components and execute the application.

Syntax.

to get memory information (in KB representation)

```
Runtime rt = Runtime.getRuntime();
System.out.println("Initial Memory:"+rt.totalMemory());
System.out.println("Maximum Memory:"+rt.maxMemory());
```

=> the memory size which will display is acquired automatically as per the application requirement.

To explicitely define the memory, we need to provide the memory size during runtime argument

To explicitely increase size of memory allocation for JVM-

-Xms → Used to increase the Initial Memory of JVM

-Xmx → Used to increase the Maximum Memory of JVM

=> Writing code during runtime argument-

It has to be provided during execution, not compilation

Syntax-

```
java -Xms512m -Xmx1024m Lab ↵
```

NOTE-

Runtime Class is a predefined class included in lang package.

=> Once the JVM memory is initialized, then it is classified into multiple runtime data area.

1) Heap Memory.
2) Method Area/Class Heap
3) Stack Memory
4) PC Register

Steps-

1) JVM → OS
2) OS → JVM
3) JVM will start
4) Byte code → Native code
5) Component memory allocation.

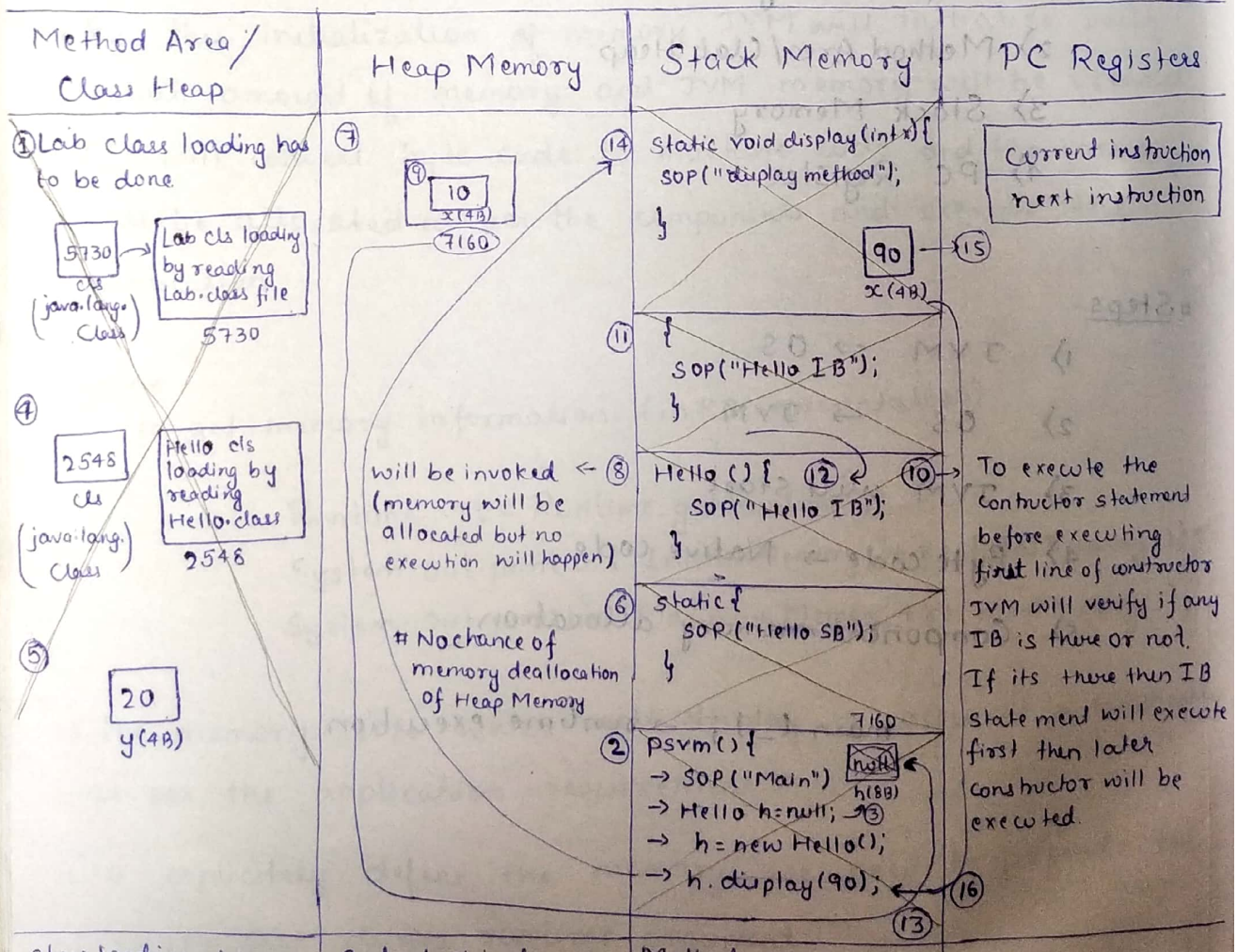
main (_); → runtime execution.

ex-

```
class Hello {
    int x=10
    Static int y= 20;

    Static {
      System.out.println("Hello SB");
    }
    {
      System.out.println("Hello IB");
    }

    void show () { }

    Static void display (int x) { }

    Class A { }
}
```

```
class Lab{
    public static void main
                (String[] arg){
    System.out.println("Main");
    Hello h= null;
    h= new Hello();
    h.display (90);
    }
}
```

JVM Memory Representation



Class loading, default object, static variable	Actual object and Instance variable	Methods, constructors, Blocks, local variables, Parameters.	

* Garbage Collector-

15/11/2017

Inside Heap memory each and every object is allocating memory but there are 2 types of object which are allocating memory.

1) Used Object

2) Unused Object.

=> Whenever memory crysis occures inside the heap memory then JVM will search for unused objects, if there are any list of unused objects then those unused object memory will be cleaned by Garbage Collector. and if in case there are no unused object then it will throw error message that "out of memory error".

=> Always GC will invoke on unused object but not on used objects and it will invoke whenever the memory crysis occures or else it not invoke to clean memory. So, it a service thread.

1) Whenever the memory crysis occurs in the Application then JVM will call the garbage collector automatically by using

```
System.gc();
```

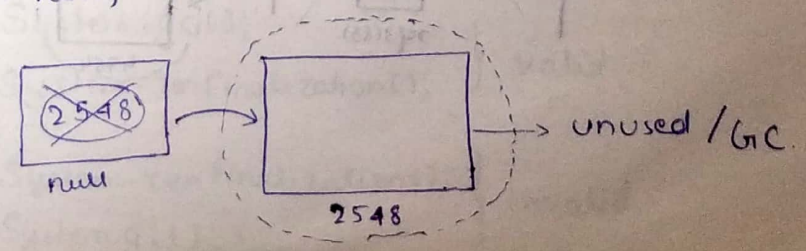
or

```
Runtime rt = Runtime.getRuntime();
rt.gc();
```

2) JVM will prepare the list of unused objects in the application by following cases:-

◎ Case 1-

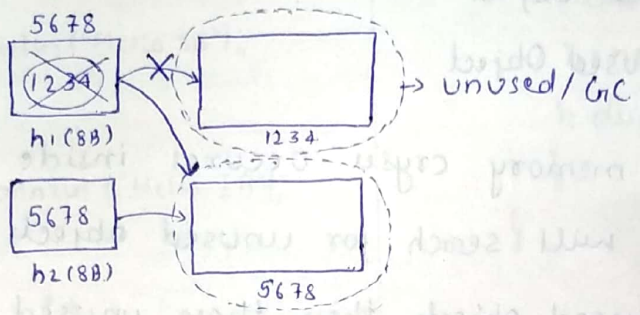
```
Hello h = new Hello();
h = null;
```



ⓑ Case 2-

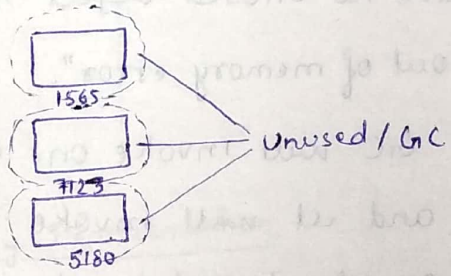
```
Hello h1 = new Hello();
Hello h2 = new Hello();
```

h1 = h2 ; → memory address assignment



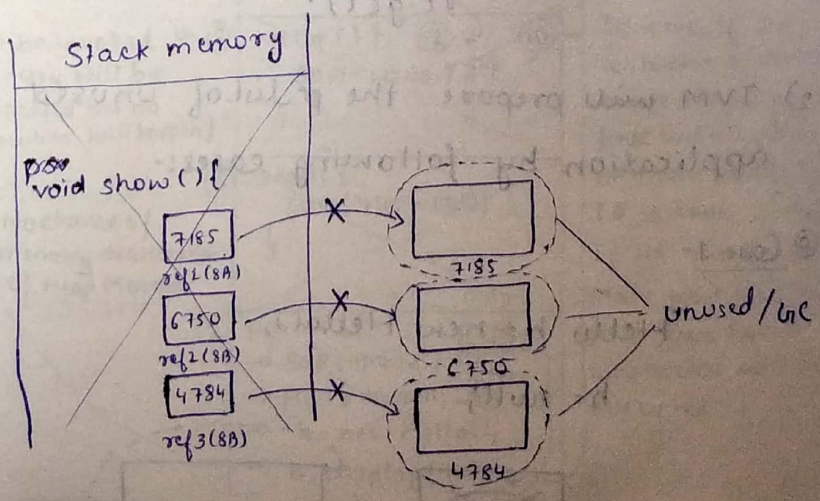
ⓒ Case 3-

```
new Hello();
new Hello();
new Hello();
```



ⓓ Case 4-

```
void show(){
    Hello ref1 = new Hello();
    Hello ref2 = new Hello();
    Hello ref3 = new Hello();
```



Scenario 1 -

Whatever the list of unused objects are not connected with any resources like DB connection, IO connection, then GC whill invoke to clean the Memory of unused Object

Scenario 2 -

Whatever the list of unused objects are connected with any resources like DB connection, IO connection, Network connection then Garbage Collector reuse to clean the Memory of unused Object.

⇒ Before invoking the GC, JVM will call System.runFinalization(); to invoke the finalize() of Object class which w used for closing the connection with resources and after that GC will invoke to clean the memory of unused object.

⇒ But there is no predefined implementation in finalize() of Object class so we need to override the finalize() in each of and provide the logic for Resource cleanup code.

NOTE -

1) GC is used for ~~correctin~~ cleaning the memory of unused object and finalize() task is to close the connection with resources.

2) After invoking GC only System.runFinalization() has to be invoked automatically by JVM or explicitely user can call

3) There is no use of calling System.runFinalization() before invoking GC because there is no list of unused object prepared by JVM and it dosn't know on which object it has to call finalize()

```
System.gc();
System.runFinalization();   } valid

System.runFinalization();
System.gc();                } Invalid.
```

* String Class-

String is defined as collection of characters or character Array

=> There is no chance of providing sub class for String class since it is declared as final.

=> String class property cannot be inherited to any other class

=> String Objects are immutable Objects. The content of the string Object cannot be modified once it is created in the allocated memory.

=> In case if you try to modify the content of the String Object then it will do modification in different memory location.

=> Every java class is mutable - i.e modification can happen

ex-

Custom implementation of immutable class.

```
final class Hello{
    private final int x;
    private final int y;
        Hello (int x, int y){
            this.x=x;
            this.y=y;
        }
    public String toString(){
        return x+"\t"+y;
    }
}
```

ex-

String class Implementation.

```
public final class String{
    private final char[] value;
}
```

scope type variable name

NOTE-

1) By default any custom java class Object are mutable Object and we can do the modification of data within the allocated memory itself but the String class Objects are declared as immutable Object by customizing the class and its members. Because of the Immutable nature String class object can be created in two ways

a) With new Operator

b) Without new Operator.

2) Internally whatever the content we are providing for String reference, reference variable, it is storing in the form of character array representation. So, inside the String class there is a Instance variable of character array type with name as value and it is declared as private final scope.

3) We can create the String class object in two ways with the help of one specified component of memory that is String constant pool (SCP) which is created inside the heap memory and SCP stores all the String Literals.

16/11/2017

ⓐ Without new Operator-

```
String str1 = "JLC";
String str2 = "JLC";
str1 == str2; -> true
```

1) It allocates 8B of memory for the reference variable and assign null value.

2) It will verify whether the String Literal is present in SCP or not.

3) If String Literal is present in SCP then it will ignore or if String Literal is not present in SCP then JVM will create the SCP Object for the String Literal.

4) When the String Object is creating without new operator then the SCP object address will be assigned for String reference variable.

ⓑ With new Operator-

```
String st1 = new String("JLC");
String st2 = new String("JLC");
```

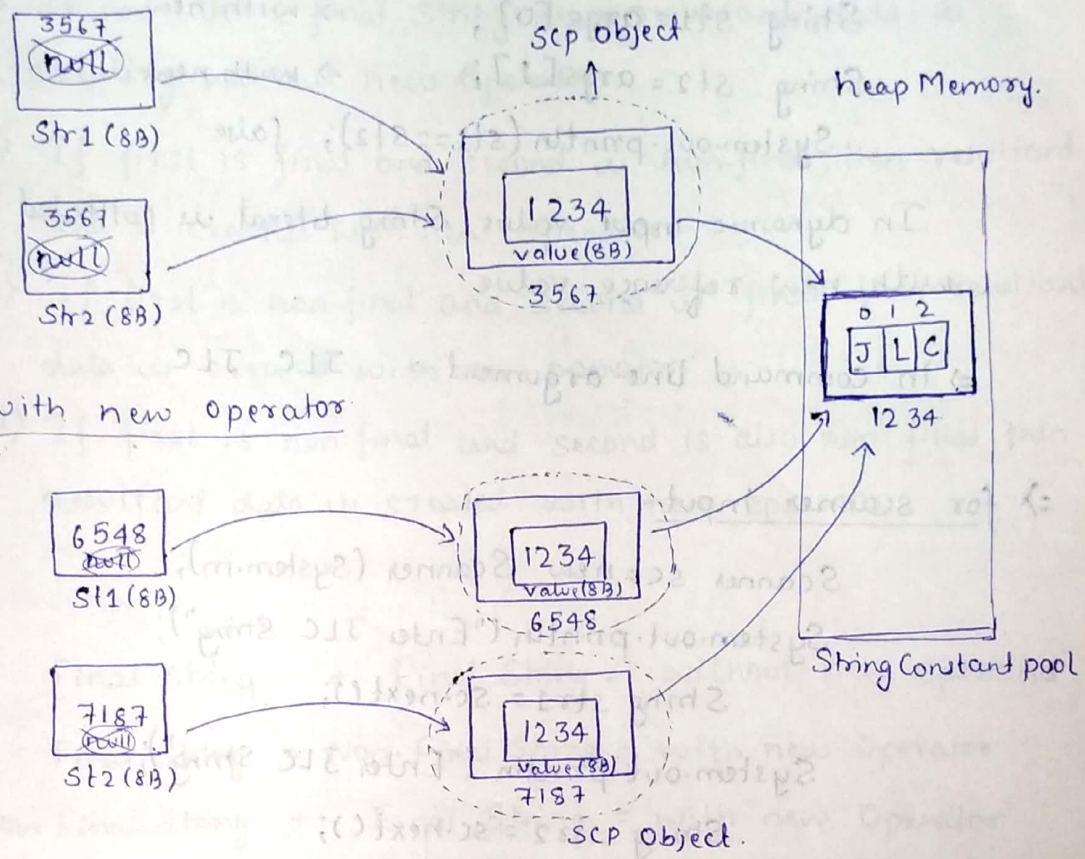
1) It allocates 8B of memory for the reference variable and assign null value

2) It will verify whether the String Literal is present in SCP or not.

3) If String Literal is present in SCP then it will ignore or if String Literal is not present in SCP then JVM will create the SCP object for String Literal.

4) When the String Object is created with new Operator then a seperate Object will be created outside the SCP and that Object address will be assigned to the reference variable.

without new operator



* intern()

It is newly defined method in String class which is used for taking the address of internal literal address.

normal comparasion => St1 == Str1 ; -> it will compare outer memory address
i.e 6548 == 3567 ; -> false.

intern() comparasion => St1.intern() == St2.intern() -> it will compare internal memory address
i.e SCP Object memory address assigned.
i.e 1234 == 1234 ; -> true.

intern() comparasion => St1.intern() == Str1 ; -> this is also a valid syntax

dynamic input value-

=> for command line argument

```
String st1 = args[0];        -> with new
String st2 = args[1];        -> with new
System.out.println(st1==st2); false
```

In dynamic input value String literal is collected always with new reference value.

=> In command line argument -> JLC JLC

=> for scanner input-

```
Scanner sc = new Scanner(System.in);
System.out.println("Enter JLC String");
String str1 = sc.next();
System.out.println("Enter JLC String");
String str2 = sc.next();

System.out.println(str1);
System.out.println(str2);
System.out.println(str1==str2); -> false
System.out.println(str1.intern()==str2.intern()); -> true
```

Note-

1) Any dynamic input value which we are collecting at the time of execution which is of String type then by default the String content will return with new Operator only.

final String ← String st2 = "JLC";
final String ← String st3 = "INDIA";
to make final

String st4 = st2 + st3; → with new operator i.e diff address

String st5 = "JLC" + "INDIA"; → without new operator i.e same address

2) In case of concatination of two String Object with the + operator then following rules needs to be followed.

a) If we take 2 final String then resultant data is creating without new Operator

b) If first is final and second is non-final, then resultant data is created with new operator.

c) If first is non-final and second is final, then resultant data is created with new operator.

d) If first is non-final and second is also non-final then resultant data is created with new operator.

Final String + Final String = without new operator

Final String + Non-Final String = with new Operator

Non Final String + Final String = with new Operator

Non Final String + Non-Final String = with new Operator

* <u>concat()</u>-

It is a newly defined method in String class which is used for concatinating the two String Object. This method always return the resultant String with new operator only either for final String or non-final String.

The priority of concatination operator will be higher than any operator.

```
int a=90; String b=120;
System.out.println("Result is:"+a==a);  -> invalid.
System.out.println("Result is:"+(a==a);  -> valid -> true.
System.out.println("Result is:"+b==b);  -> valid -> false.
```

* equals();

It is an overridden method in String class from Object class and it is used for comparing the actual content of String Object as per the case sensitivity

* equalsIgnoreCase();

It is a newly defined String class method which is used for comparing the actual content of String Object by ignoring case sensitivity.

* public int compareTo(String)-

This is an abstract method of comparable interface. It is overridden in the String class and it is used for comparing which String Object is maximum and which String Object is minimum and returning numeric value by the ASCII difference among the character.

System.out.println("ABC".compareTo("ADF"));

internally ABC - ADF

A-A = 0 65-65=0

B-D = -2 66-68=-2 -> it will stop here no match

System.out.println("SRI".compareTo("SRI")); -> 0

System.out.println("SRI".compareTo("ADG");

SRI - ADG

S-A → 83-65 -> 18 -> will stop here

System.out.println("SRI".compareTo("SRINIVAS"); -> -5

↳ extra 5 character

System.out.println("SRINIVAS".compareTo("SRI"); -> 5

extra 5 character

17/11/2017

* Compare To Ignore Case ();

It is a newly added method in String class which is used for comparing two Strings Object to find out which is maximum or minimum by ignoring case sensitivity.

* public static void value () {} :

It is a newly defined method which is used for assigning any type of value to the string reference variable either premitive or reference.

```
int x = 99;
String str1= x;      -> invalid
String str2= (String)x;  -> invalid
String str3= String.valueOf (x);  -> valid

Hello h1= null;
String st1= String.valueOf (h1);  -> valid
System.out.println (st1);  -> valid

String str4 = String.valueOf (null);  -> invalid -> null pointer error.

String str5 = String.valueOf (999);  -> valid.
```

* toUpperCase () & toLowerCase ();

```
String str = "JlcInDia";
System.out.println (str);  -> JlcInDia
System.out.println (str.toUpperCase());  -> JLCINDIA
System.out.println (str.toLowerCase());  -> jlcindia
```

* length ():

```
System.out.println ("JLCINDIA");  -> JLCINDIA
System.out.println (str.length());  -> 8
```

↳ it will count including space character also.

* trim()-

It is a newly defined method used for deleting the extra space character at the starting and ending position of String content but not in middle position.

```
String str = " ";
System.out.println(str.length());  -> 0
```

* isEmpty()-

Used to check whether String Literal is empty or not and returns boolean value.

```
System.out.println(str.isEmpty());  -> true
```

* StartsWith (String str) -

It is a newly defined method which is used for verifying the String content whether specific word is starting or not and this method is also overloaded method for verifying specified String or word at the specified index position but not the starting index.

```
String str = "Welcome to Bangalore";
System.out.println(str.startsWith("Welcome"));
```

↓ the ASCII value will be matched, if correct then true or else will return false (also case sensitive)

```
System.out.println(str.startsWith("Welcome", 3));
```

↓ It's overloaded method

↓ index check i.e: specified position

* endsWith()-

It is also a newly defined method which is used for comparing content at the ending position of the String and it is used for always checking or verifying at the index position only.

```
System.out.println (str.endsWith ("Bangalore"));  -> true
System.out.println (str.endsWith ("BLORE"));  -> false
System.out.println (str.endsWith (""));  -> true
```

↳ empty String Literal allowed with return value as true.

There is no overloaded method for endsWith().

* For replacing the content of the String we need to use three methods as follows-

a) replace (char srcchar, char destchar)-

It is used for replacing one character with another character in String content on all occurance.

b) replaceFirst (String srcword, String destword)-

It is used for replacing one word with another word in the String content on first occurance only.

c) replaceAll (String srcword, String destword)-

It is used for replacing one word with another word in the String content on all occurance

ex-

```
String str = "Hi, This is JLC. It is providing training on JAVA
                and the popular language is JAVA";
System.out.println (str.replace ('J', '*'));
System.out.println (str.replaceFirst ("JAVA", "HADOOP"));
System.out.println (str.replaceAll ("JAVA", "HADOOP"));
```

String:

String is defined as collection of character or char Array

```
public final class String {
        private final char[] value;
}
```

* indexOf() and lastIndexOf() -

```
String str = "Hi, This is JLC. It is providing traning on
        JAVA and the popular language is JAVA";

String str1 = "JLCINDIA";
              0 1 2 3 4 5 6 7
```

→ indexing starts with 0

=> System.out.println(str1.indexOf('A')); → 8 (7) — it will return index value.

=> System.out.println(str1.indexOf('a')); → -1 — there is no matching so -1 is giving

=> System.out.println(str.indexOf('J')); → 11 — it will check for J at first occurance

=> System.out.println(str.indexOf('J', 12)); → 43 — if we have to check for next occurance

=> System.out.println(str.indexOf('J', 44)); → 77 — then we need to give previous occurance +1 (i.e 11+1=>12)

=> System.out.println(str.indexOf('J', 78)); -1 → No more occurance

=> System.out.println(str.indexOf("JAVA")); → 43

=> System.out.println(str.indexOf("JAVA", 44)); → 77

=> System.out.println(str.lastIndexOf("JAVA")); → 77

=> System.out.println(str.lastIndexOf("JAVA", 76)); → 43

=> System.out.println(str.lastIndexOf("JAVA", 42)); → similar as for next indexing here we need to check for previous so, index - 1

77 - 1 = 76

43 - 1 = 42

These are newly defined methods in String class which are used for getting the index position of specified character. or specified word in the Total content of String.

NOTE -

indexOf () = it will start the search from starting index of the String.

lastIndexOf() = It will start the search from ending index of the string.

These methods are used for finding out the number of occurance for the specified word or specified character in the String.

* substring () -

```
String str = "JLCINDIA";
             0 1 2 3 4 5 6 7

System.out.println (str.substring (3)); -> INDIA

System.out.println (str.substring (0,2)); -> JL   -> last index is not included
System.out.println (str.substring (0,3)); -> JLC  if we are defining the range.

System.out.println (str.substring (3,6)); -> IND

System.out.println (str.substring (5,8)); -> DIA

System.out.println (str.substring (5,9)); -> invalid -> String index out of bound exception
```

NOTE -

SubString index position should be in range.

```
for (int i=0; i< str.length (); i++){
    System.out.println (str [i]);      -> Invalid
}
```

it can be only taken for array but String is not an array.

* VVI

So. `for (int i=0; i< str.length(); i++){`

```
    System.out.println (i+"\t"+ str.charAt(i));  -> valid syntax
}
```

20/11/2017

charAt (int index) -

This method is used for returning the specified character at specified index position and it is also used for displaying total content of the String as character by character with corresponding index position.

```
String str = "JLCINDIA";
System.out.println(str.charAt(1));  -> L
System.out.println(str.charAt(5));  -> D.
for (i=0; i< str.length; i++){
    System.out.println(i+"\t"+ str.charAt(i));
}.
```

* Conversions.

=> String to charArray & String to Byte Array-

=> Manual implementation (String to char Array).

```
class Lab{
    public static void main (String[] args){
        String str = "JLCINDIA";
        int len = str.length();
        char ch[]= new char [len];
        System.out.println("** String to Array **");
        for (int i=0; i<length; i++){
            ch[i]= str.charAt(i);
        }
        for (int i=0; i<ch.length; i++){
            System.out.println(i+"\t"+ch[i]);
        }.
    }
}
```

=> Predefined implementation -

```
char ch[] = str.toCharArray();
```

=> Manual implementation (String to Byte)-

```
class Lab {
    public static void main (String [] org){
        String str = "JLCINDIA";
        int len = str.length();
        byte barr[] = new byte [len];
        System.out.println("**String to Byte**");
        for (int i=0; i<barr.length; i++){
            char c = str.charAt (i);
            byte b = (byte)c;
            barr [i] = b;
        }
        for (int i=0; i<barr.length; i++){
            System.out.println( i+"\t"+ barr[i]);
        }
    }
}
```

=> predefined implementation-

```
byte barr[] = Str.getBytes();
```

* Reverse conversion-

=> charArray to String and byteArray to String-

```
String (char value[]);
String (byte value[]);
```

ex. `char ch[] = {'A','B','C','1','2','3','a','b','c'};`

`byte barr[] = {97, 98, 99, 65, 66, 67, 48, 49}`

```
String st1 = new String(ch);
String st2 = new String(barr);
```

=> for index counting

```
String st3 = new String (ch, 0, 6);   -> ABC123
String st4 = new String (barr, 3, 5); -> ABC
```

* Split Method-

It is a newly defined method in String class which is used for making the multiple parts of the String content based on specified String literals (i.e. delimeter content) on all Occurences.

ex-

String str = "I am in Bangalore and trained on JAVA. The popular platform is JAVA developer. My profession is JAVA developer in Capgemini"

```
String msc[]= str.split("JAVA"); //or ("\\.");
for (int i=0; i< msc.length; i++){
    System.out.println(msc[i]);
}
```

for . we need to use escape sequence character.

O/P → I am in Bangalore and trained on
. The popular platform is
developer. My profession is
developer in Capgemini

=> Termination on specific occurance-

split (String str, int limit);

It is an overloaded method which is used to stop the splitting of the String content on the specified limit value. But limit should be within the range of occurance

String msc[] = str.split("JAVA", 2);

O/P
I am in Bangalore and trained on
. The popular platform is JAVA developer. My .....

NOTE-

The limit value should be within the range.
If the limit value is less than or equal to 0 and greater than number of occurances then the splitting will be normal.

* matches (String regExp) -

It is the newly defined method in String class which is used for comparing the total content of the String is matching the specified pattern or not. If its matching, it will return true or else false.

The patterns should be provided by using regular expression

[ ] = collection of characters.

* = zero or more character.

^ = starting character of String.

ex-

```
class Lab {
    public static void main (String [] arg){
        String exp1 = "[A-Z]";
        String exp2 = "[A-Z]*";
        String exp3 = "[A-Za-z0-9]*";

        System.out.println ("H". matches (exp1)); -> True
        System.out.println ("h". matches (exp1)); -> False.
        System.out.println ("ABC". matches (exp2)); -> True
        System.out.println ("abc". matches (exp2)); -> False
        System.out.println ("123abcABC". matches (exp3)); -> True.
        System.out.println ("123**abc$$ABC". matches (exp3)); -> False.
```

=> for defining starting character-

```
String exp4 = "^[A-Z][A-Za-z0-9]*";
System.out.println ("123abcABC". matches (exp4)); -> False
System.out.println ("A123abcABC". matches (exp4)); -> True
```

* Note-

If no /blankspace is there then it will return true.

* regionMatches() -

It is newly defined method in String class which is used for comparing the specific content of the String in matching another String specific region. If it is matching, it will return true else false.

ex-

```
Class Lab{
    public static void main (String[] arg){
        String exp1 = "my id is sri@gmail.com";
        String exp2 = "www.gmail.com is the url to access the google account";
        String exp3 = "my id is sri@GMAIL.COM";
        System.out.println(exp1.regionMatches(0, exp2,0,5));  -> false
        System.out.println(exp1.regionMatches(13,exp2,4,5));  -> true
        System.out.println(exp3.regionMatches(true,13,exp2,4,5); -> true
```

Syntax-

```
exp3.regionMatches(true, 13, exp2, 4, 5);
```

- exp3 → Source string object
- true → consider case sensitivity
- 13 → marker of source expression
- exp2 → destination expression
- 4 → Starting index of destination expression
- 5 → count of character from starting index

⇒ To escape case sensitivity True can be used

⇒ But to force case sensitivity False can be used or can be left blank.

* Limitations of String Object-

String Object is an immutable object because of this nature, once we assign the content that is fixed, if we try to do any modification then always it will go for different memory allocation so because of this, mainly in online application if multiple members of classes are declared as String then user is going to provide input value then as many multiple modification done by the user is equal to multiple memory allocation which is memory wastage. So, String class cannot be taken wherever appending of data or data modification operations are there. So, this is a draw back of String class.

To overcome the drawback of String. JAVA vendor has provided some new class in the lang package with same behaviour of String class and the nature of the class is mutable. The class is named as "String Buffer Class".

* String Buffer Class Concept-

It is based on the concept "Capacity".

Capacity -> how many character it can hold

It's based on threadsafe mechanism. i.e multitasking cannot be supported.

=> The default capacity is 16, if the character increases by 16 then it will increase as new capacity.

new capacity = (Old capacity * 2) + 2

=> drawback -

Synchronized nature.

i.e. multitasking not allowed.

* String builder class concept-

Non-synchronized nature
i.e. support for multitasking

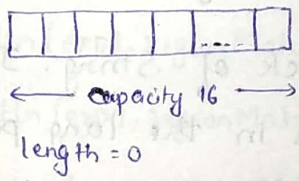
=> String builder can be created by only one way.
→ with new operator

* Constructor concept to provide the string builder object-

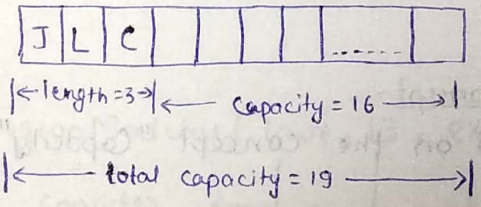
1) StringBuilder ();
2) String Builder (String str);
3) String Builder (int capacity);

} constructor overloading

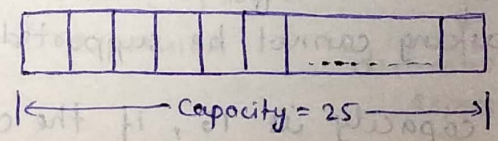
1) StringBuilder sb = new StringBuilder();



2) String Builder sb = new StringBuilder ("JLC");



3) StringBuilder sb = new StringBuilder (25);



sb.capacity (); → to check for capacity
sb.length () ; → to check for length

NOTE-

There is no inheritance between String Builder and String for comparassion between both.

sb1.contentEquals (sb2); for comparasion between String builder classes ~~only~~ and String both.

=> As, String Builder class equals is not overridden so, content cannot be compared directly. To do so, we first have to convert String Builder to String.

So,

```
String st1 = sb1.toString();
String st2 = sb2.toString();

st1.equals(st2);  -> true
```

=> hashcode() is also not overridden.

=> compareTo() cannot be used directly because no inheritance we need to convert StringBuilder to String before compare.

```
String st1 = sb1.toString();
String st2 = sb2.toString();

st1.compareTo(st2);
```

if the o/p is

+ve -> first is max i.e - st1.

-ve -> second is max i.e - st2.

* <u>ensureCapacity(int cap)</u> -

This is newly defined method in StringBuilder class which is used for verifying whether the specified capacity is there or not. It will do the verification in following ways-

1) If the new capacity parameter i/p value is greater than the current capacity and next generating capacity then String builder object will be assigned for new capacity value or else new capacity is greater than current capacity and it is less than next generating capacity then it is using to assign next generating capacity value for the StringBuilder object.

2) If the new capacity is less than current capacity and less than next generating capacity then StringBuilder object capacity value we need to change.

* setlength()-

This method is used to do modification for length of the String Builder either to increase or decrease. If you want to decrease length of String Builder then extra character will be deleated and if you want to increase the length of string builder then extra count of character will be added as default value of the character.

* insert()-

It is a newly defined method in String builder class which is used for appending extra data in the middle position of the string content or otherwise we can append the data last position by default by using append().

* Wrapper Class-

These are predefined class which are provided for each and every premitive datatype and in the scenario of where reference is required, we cannot provide premitive values and where premitive is required, we cannot provide the reference.

These wrapper class are mainly used in collection framework APIs.

⇒ Totally 9 wrapper classes are provided.

boolean → Boolean
byte → Byte
short → Short
int → Integer
long → Long
char → Character
float → Float
double → Double
void → Void

=> Hierarchy-

Object → super class

- Boolean
- Character
- Void

} → direct sub class

Number

- Byte
- Short
- Integer
- Long
- Float
- Double

} → Object is indirect super class

=> With the help of wrapper class we can convert one type to another. There are 6 type conversions.

1) primitive to String
2) String to primitive
3) primitive to wrapper Object
4) wrapper Object to primitive
5) String to wrapper Object
6) Wrapper Object to String.

=> We can also convert one base to another by below methods

=> toOctalString()

=> toHexaDecimalString()

=> toBinaryString()

=> To represent size, ranges for each and every premitive datatype there are pre defined constants.

→ SIZE

→ MAX_VALUE

→ MIN_VALUE

ex.

```
System.out.println (Byte.SIZE);
System.out.println (Byte.MAX_VALUE);
System.out.println (Byte.MIN_VALUE);
```

These 3 are applicable for all datatypes.

```
System.out.println (Integer.toOctalString (a));
System.out.println (Integer.toHexString (a));
System.out.println (Integer.toBinaryString (a));
```

⇒ There are 3 classes which can be used only for Float and Double.

```
System.out.println (Double.POSITIVE_INFINITY);
System.out.println (Double.NEGATIVE_INFINITY);
System.out.println (Double.NaN);
```

Ques- I have a medical shop in that customer will come and take some tablets while on billing system I am collecting the customer details ie Customer Name, Mobile, Email, address (city), and product name. Once the bill is completed, one copy should be placed with me and another copy will be given to customer For this scenario, write a code to get merchand copy and customer copy.

Code-

```
public class Billing {
    public static void main (String[] arg) throws CloneNotSupportedException {
    CustomerDetails mc = new CustomerDetails ("Ram Mohan", 9900771133L,
                                "ram@xyz.com", "Bangalore",
                                "Soradon");

    CustomerDetails cc = (CustomerDetails) mc.clone();
```

```
System.out.println("*** Master Copy ***");
System.out.println(mc);
System.out.println("*** Customer Copy ***");
System.out.println(cc);
mc.cadd = "Mumbai";
cc.pname = "Disprin";
System.out.println("*** Modified Master Copy ***"+mc);
System.out.println("*** Modified Customer Copy ***"+cc);
}
}

class CustomerDetail implements Cloneable {
String cname, cmail, cadd, pname;
long cmob;
CustomerDetail (String cname, long cmob, String cmail,
        String cadd, String pname){
    this.cname = cname;
    this.cmob = cmob;
    this.cmail = cmail;
    this.cadd = cadd;
    this.pname = pname;
}
public String toString(){
    return "Name:"+cname+"\n"+"Mobile:"+cmob+"\n"+
        "Email:"+cmail+"\n"+"City:"+cadd+"\n"+
        "Product Name:"+pname;
}

public Object clone() throws CloneNotSupportedException{
    return super.clone();
}
}
```

⇒ Conversions-

1) Primitive to String-

a) By using String concatination operator.

```
int x = 100;
String st1 = " " + x;
String st2 = x+" ";
```

b) By using valueOf() in String class

```
String st3 = String.valueOf(x);
```

c) By using toString of corresponding wrapper class

```
String st4 = Integer.toString(x);
```

2) String to Premitive-

```
String str = "123";
byte b = Byte.parseInt("100");
byte b = Byte.parseInt(str);
byte b = Byte.parsInt("400");  -> invalid. out of range.
                                  Number format exception.
int x = Integer.parseInt(str);
```

for boolean-

```
boolean b = Boolean.parseBoolean("False");  -> case sensitivity is ignored. if in " "
```

for character-

```
String str = "SRINIVAS";
Char ch1 = Str.charAt(0);  -> S
Char ch2 = Str.charAt(3);  -> N
```

NOTE-

If the input value is mismatched while conversion then it will throw Number format exception.

3) Primitive to Wrapper Object - (Boxing mechanism) -

a) By using predefined constructor with primitive value as parameter

```
int x = 99;
Integer in = new Integer (x);
System.out.println (in);
```

b) By using predefined "value of method" in each and every wrapper class

```
Integer in = Integer.valueOf (x)
System.out.println (in);
```

4) Wrapper Object to primitive - (Unboxing mechanism) -

```
Float f1 = new Float (3456.3454f);
float f = f1.floatValue();
int x = f1.intValue();
long L = f1.longValue();
```

if

```
Integer in = new Integer (250);
int x = in.intValue();      -> valid
byte b = in.byteValue();    -> valid -> type casting will be done automatically
                                        -> O/P = -6
```

for Boolean -

```
Boolean b1 = new Boolean (false);
boolean b = b1.booleanValue();
System.out.println (b);
```

for Character -

```
Character ch1 = new Character ('A');
Char c = ch1.charValue();
System.out.println (c);
```

3) Primitive to Wrapper Object - (Boxing mechanism) -

a) By using predefined constructor with primitive value as parameter.

```
int x = 99;
Integer in = new Integer (x);
System.out.println (in);
```

b) By using predefined "value of method" in each and every wrapper class

```
Integer in = Integer.valueOf (x)
System.out.println (in);
```

4) Wrapper Object to primitive - (Unboxing mechanism) -

```
Float f1 = new Float (3456.3454f);
float f = f1.floatValue();
int x = f1.intValue();
long L = f1.longValue();
```

if

```
Integer in = new Integer (250);
int x = in.intValue();      -> valid
byte b = in.byteValue();    -> valid -> type casting will be done automatically
                                        -> O/P = -6
```

for Boolean -

```
Boolean bl = new Boolean (false);
boolean b = bl.booleanValue();
System.out.println (b);
```

for Character -

```
Character ch1 = new Character ('A');
Char c = ch1.charValue();
System.out.println (c);
```

5) String to Wrapper Object -

For this conversion, there must and should be Inheritance relation.

a) By using the predefined constructor with String as the parameter in each and every wrapper class except character

```
String str = "123";
Integer in = new Integer(str);

String str = "ABC";
Character ch = new Character(str.charAt(0));
```

b) By using valueOf() in corresponding Wrapper classes.

```
String str = "123";
Integer in = new Integer(str);
System.out.println(in);

String st1 = "400";
Byte b1 = new Byte(st1);
System.out.println(b1); → invalid, st1 is out of range
                          for byte datatype

String st2 = "ABC";
Character ch = new Character(st2.charAt(0));
System.out.println(ch);

String st3 = "SRI";
Boolean b2 = new Boolean(st3);
System.out.println(b2); → false.  if st3 = "True" then
                                  o/p → true
```

6) Wrapper Object to String-

a) By using toString() which is overridden in every wrapper class. (i.e- returns Actual content).

```
Integer in = new Integer (345);
String st1 = in.toString();
System.out.println (st1);   -> 345
```

b) String class valueOf() can be used.

```
String st2 = String.valueOf(in);   -> 345
```

* Boxing and Unboxing-

before java 5

=> boxing-

```
int x = 120;
Integer in = new Integer (x);
Integer in = x ;  -> invalid.
```

=> unboxing-

```
Integer in1 = new Integer(999);
int x = in1.intValue();
int x = in1 ;  -> invalid
```

after java 5

=> boxing.

both are valid i.e- Autoboxing.

both are valid i.e- Autoboxing. un

NOTE-

After java 5 AutoBoxing is implemented, and auto unboxing and no constructor definition and no predefined method are required.

ex-

=> Boxing

```
Integer int1 = new Integer (90);
Integer int2 = new Integer (90);
System.out.println (int1 == int2);   -> false
```

=> Autoboxing

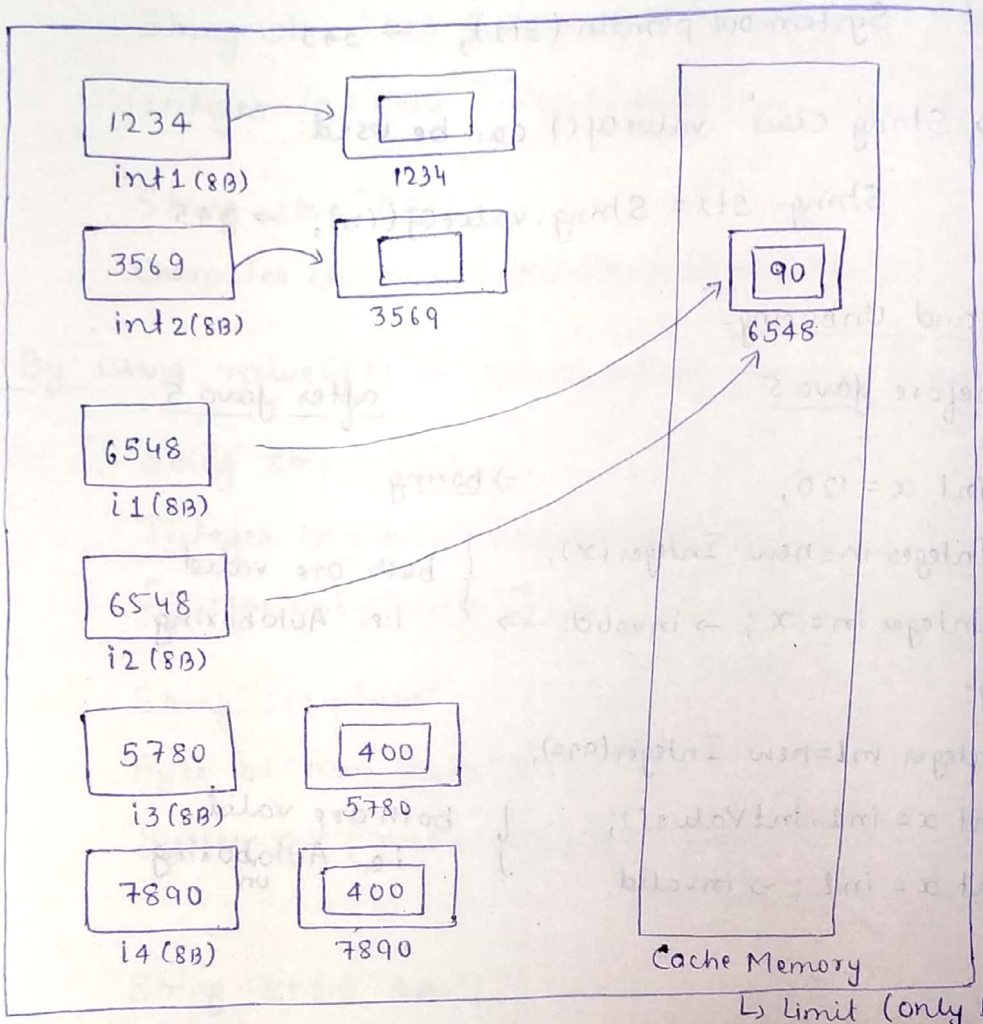
```
Integer i1 = 90;
Integer i2 = 90;
System.out.println (i1 == i2);   -> true
```

```
Integer i3 = 400;
Integer i4 = 400;
System.out.println(i3 == i4);
```

} out of byte range

-> false

=> Memory representation-



↳ Limit (only byte range) 127 to -128

NOTE-

=> for floating and double it will always create a newly allocated memory because of exponential notation.

=> for integer we can modify the cache memory limit.

java -D java.lang.Integer.IntegerCache.high=650 Lab

Autoboxing and type casting is possible with.

```
Object obj = 120;

internally { // Integer in = 120;
             // Object obj = in;

Number num = 450L;

internally { // Long l = 450L;
             // Number num = l;

internally { int x = 100;
             long l = x;

internally { float f = 34.56f;
             double d = f;
```

Autoboxing and type casting is not possible with-

```
Long l1 = 350;

internally { // Integer in = 350;
             // Long l1 = in;
```

→ invalid → no inheritance relation between Integer & Long.

→ This is by default integer value

```
Double d1 = 46.78f;

internally { // Float f1 = 45.78f;
             // Double d = f1;
```

no inheritance relation between both.

24/11/2017.

* System-

System is a final class which is used to get the information about the System and its properties.

System class constructor is declared as the private. It is not possible to create the Object of the class.

There is no necesity to create the Object of System class because all the members are declared as static member and we can access within the class name only.

At the JVM startup, it is provided some reference variable for i/p and o/p device connections in System class.

=> There are 3 Stream classes-

1) public static final Input Stream in; => Keyboard
2) public static final Print Stream out; => Console -> print all data
3) public static final Print Stream err; => Console -> print all err msgs.

NOTE-

"out" is the reference variable of Print Stream class.

=> To display date and time using System class-

```
class Lab {
    public static void main (String [] arg) {
        long time = System.currentTimeMillis();      -> in millisecond format
        System.out.println(time);
        Date dt = new Date (time);      -> in standard format
        System.out.println (dt);
    }
}
```

S.O.P (System.getProperty ("OS.name")); → gets OS name details

S.O.P (System.getProperty ("JLCWebsite")); → gets website details

System.setProperty ("OS.name", "LINUX"); → it will temperorly change the OS name as LINUX.

System.setProperty ("JLCWebsite", "www.jlcindia.com"); → will set property

Object cannot be created for System class as it is private.

```
System.out.println ("Main Started");
System.out.println ("Enter some i/p data");
int asc = System.in.read();
System.out.println ((char)asc);
System.out.println ("Hello Guys");
System.err.println ("Invalid User Name");
```
→ To print error messages.

To change the i/p device and o/p device the below code can be used.

```
public static void setIn (java.io.InputStream);
public static void setOut (java.io.PrintStream);
public static void setErr (java.io.PrintStream);
```

To display the output other than console.

```
{ System.out.println ("Hello Guys by using out");
  System.err.println ("Error msg1 by using err");
  FileOutputStream fos = new FileOutputStream ("G:\\JA-B11\\
                                LangPackage\\Welcome.txt");
    PrintStream ps = new PrintStream (fos);
    System.setOut (ps);
  // System.setErr (ps);
    for (String st:args) {
            System.out.println (st);
        }
    System.err.println ("Error msg 2 by using err");
}
```

* Runtime -

It is also a final class. It is used for getting the information about current running JVM and its memory management. It is also used for executing the additional process locally.

=> Runtime class behaviour is following one design pattern i.e - Single Ton Class Design Pattern

=> Single Ton Class means the class which is restricted for creating only one Object of Class.

=> Runtime class is declared as private constructor.

To terminate JVM abnormally -

System.exit(0); is used

This method is declared in both system class and runtime class and it is used for terminating JVM abnormally at any point of statement. Normally JVM will execute the application from starting line of main method to ending line of main method. which is known as normal termination.

=> Runtime memory checking.

```
public class Lab{
    public static void main (String[] arg){
        Runtime rt = null;
    //  rt = new Runtime();
        System.out.println(Runtime.getRuntime());
        rt = Runtime.getRuntime();
        System.out.println("AP:" + rt.availableProcessors());
        System.out.println("MM:" + rt.maxMemory());
        System.out.println("TM:"+rt.totalMemory());
        System.out.println("FM:"+rt.freeMemory());
    }
}
```

Runtime argument => java -Xmx1024m -Xms512m Lab {on cmd}.

ex- To open any application from code-

```
public class Lab {
    public static void main (String arg[]) throws Exception {
        Runtime rt = Runtime.getRuntime();
        Process p = rt.exec ("mspaint");

// OPEN Welcome.txt FILE in NOTEPAD
        String cmds[] = {"notepad", "D:\\Welcome.txt"};
        Process p2 = rt.exec(cmds);

// OPEN jlc website with FIREFOX
        String cmds1 [] = {"C:\\Program Files\\Mozilla Firefox\\firefox.exe",
                                    http://www.jlcindia.com"};
        Process p3 = rt.exec (cmds1);
        System.out.println ("Press Enter to close all application");
        System.in.read();
        p.destroy();
        p2.destroy();
        p3.destroy();
        System.out.println ("Main Completed");
    }
}
```

* Math Class-

It is a final class which is used to perform the arithmetic operation, trignometric functions or logarathmic operation etc and there are so many predefined methods in math class. which is used.

Math class constructor is private.

```
System.out.println (Math.max(354, 21));  -> 354
System.out.println (Math.min (354, 21));  -> 21
System.out.println (Math.pow (3, 2));  -> 9
System.out.println(Math.sqrt(3));  -> 1.732
System.out.println (Math.cbrt (27));  -> 3
System.out.println (Math.round (344.677));  -> 345  (decimal dependent)
System.out.println ( Math.ceil (34.058));  -> 35  (next number)
System.out.println (Math.floor( 34.998));  -> 34  (remove decimal)
```

ex- To generate random number-

```
Math.random();
```

Used for security purposes.

ex-

```
public class Lab {
    public static void main (String [] args){
    for (int i=0; i<10; i++) {
        System.out.println (Math.random());
    }
    System.out.println ("\n*****");
    for (int i=0; i<10; i++){
        System.out.println (10 * Math.random());
    }
  }
}
```

01/12/2017

* Big Integer Class-

It is available in java.math package and it is the sub class of java.lang.number

It is used for exceeding the range of long datatype when we need to collect the value only in numerical part.

ex-

```
BigInteger bin1 = new BigInteger("9223720368547758 07");
System.out.println(bin1);  -> valid.

BigInteger bin2 = new BigInteger(" *1000* ");  -> invalid
                                                  strings not
                                                  allowed.
bin1 = bin1.add(bin2);
System.out.println(bin1);  -> error.
```

*

ex-

```
BigInteger bin3 = new BigInteger("10");   binary 1010
System.out.println(bin3.bitCount());  -> no of ones in binary
                                          = 2
System.out.println(bin3.bitLength());  -> total no. of bit length
                                          = 4
```

ex-

```
double d = 1.1797693134862315 7E308;
System.out.println(d);
d = d+100;
System.out.println(d);  -> The o/p value will be same
                           because we can not say in
                           fractional part. (i.e no 100 addition)
                           for this we can use BigDecimal();
```

* Big Decimal-

It is used for exceeding the range of double datatypes when we need to collect the values in fractional part of values.

ex-

```
BigDecimal bd1 = new BigDecimal(1.1797693134 8623157E308);
BigDecimal bd2 = new BigDecimal(1000);
bd1 = bd1.add(bd2);
System.out.println(bd1);  -> valid
System.out.println(bd1.POSITIVE.INFINITY);  -> error
                                               Number format
                                               exception.
```

* Design Patterns-

It is a solution to the recurring problem in the specified context. It is of 2 types

1) GOF design pattern (Gang of four)
2) Domain specific design pattern.

1) GOF- They are common for all the domain patterns.

=> list of GOF= singleton, Iterator, Builder, Flyweight, Strategy etc

2) Domain specific design pattern. It changes from domain to domain

eg: J2EE design pattern
Dot net design pattern.

=> lists = Intercepting filter, Frontcontroller etc.

* Singleton Design Pattern-

Any java class which is restricted for only one instance Object for jvm is known as SingleTon class.

=> Runtime class in lang pack is SingleTon pattern.

=> Steps for custom SingleTon class-

a) Define any type of constructor of the class as the private scope so that no one can create the object of the class outside

b) Declare one reference variable of same class and create only one object of that class and assign to reference variable and declare it as the private final static.

c) Declare one public static method that returns the single instance of the class.

d) Whenever we need to clone the SingleTon class object then inside the class defination we need to override the clone() by returning this keyword.

ex-

```
class Lab {
    public static void main (String[] arg){
        User U1 = User.getUser();
        System.out.println(U1);
        System.out.println(User.getUser());
    }
}

class User {
    private static User INS = new User();
    private User() {}
    static User getUser(){
        if (INS == null)
            INS = new User();
        return INS;
    }
}
```

* N-ton design pattern-

Any java class has to restrict for N numbers of Object for the class description is known as N-ton Design pattern.

=> Steps for N-ton design pattern-

a) Define the constructor as private scope.

b) Declare one or two or three reference variable of the class and create objects. ~~Static method~~

c) In public static method give condition and then return each object according to condition.

d) For cloning overriding clone() by returning this keyword.

ex-

```
class Lab {
    public static void main (String[] arg) {
        for (int i=0; i<10; i++) {
            System.out.println (User.getUser ());
        }
    }
}

class User {
    private static User INS1 = new User();
    private static User INS2 = new User();
    private User() {}
    public static User getUser () {
        if (Math.random() > 0.5)
            return INS1;
        else
            return INS2;
    }
}
```

implementation

* Marker interface-

Interface declared without any member is known as Marker interface or tagged interface or Empty interface. It is used for providing as the Rule for performing some process/ operation etc. logically by JVM.

=> Marker interface cannot be used to achieve 100% abstraction principle or cannot achive multiple inheritance in java.

> Cloneable marker interface-

To check the eligibility of cloning process for java Object.

> Custom marker interface-

JLC Cloneable

Voting --> min age >=18.

ex-

```
class Vprocess {
    boolean Voting (User u){
        if (u instanceof MINAGE18)
            return true;
        else
            return false;
    }
}
interface MINAGE18 {}

class User extends Vprocess {}

// class User extends Vprocess implements MINAGE18 {}
```

NOTE-

The candidates having ages ≥18 in their birth certificate only can implement MINAGE18.

ex-

```
public class Lab {
    public static void main (String[] arg) throws
                                CloneNotSupportedException {
    Hello h1 = new Hello (10, 20);
        System.out.println (h1);
    Hello h2 = h1.myclone ();
        System.out.println (h2);
        System.out.println (h1==h2);
    }
}

interface JLCCloneable {}
```

P.T.O →

```
Class Hello implements JLCCloneable {
    int x;
    int y;
      Hello (int x, int y){
        this.x = x;
        this.y = y;
      }
    public String toString(){
        return (x+"\t"+y);
    }
    public Hello myclone() throws CloneNotSupportedException {
      if (~~instan~~ this instanceof JLCCloneable)
        return new Hello(x,y);
      else
        throw new CloneNotSupportedException("Implement
                                              Marker Interface");
      }
    }.
```

* <u>Immutable Class</u>-

The content of the Object while creating is fixed and if we try to modify then it will allocate in new memory location but there is no chance of modifying the content.

=> <u>Steps to define custom immutable class.</u>

a) Define the class as final.

b) Define the variable as private and final

c) Define the getter method to access the values.

d) Don't define setter method.

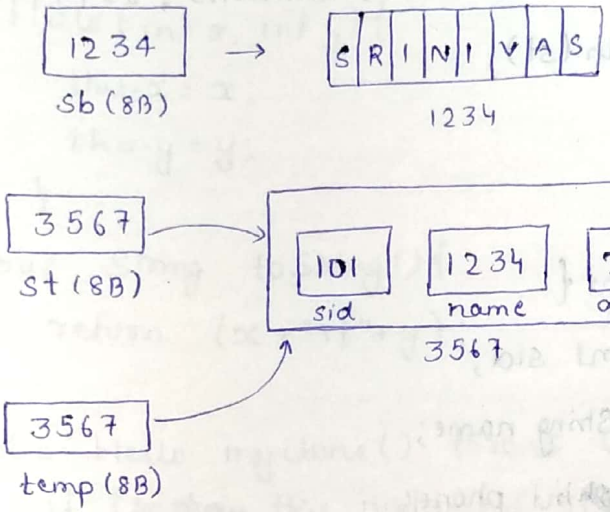
e) If any mutable object available as member then
   > while initializing or accessing the mutable object, create the Object and use that.

ex-

```
public class Lab {
  public static void main (String[] arg) {
    Student st = new Student (99, "SRINIVAS", 65799999);
    System.out.println (st);
  }
}

final class Student {
    private final int sid;
    private final String name;
    private final int phone;
      public Student (int sid, String name, int phone) {
        this.sid = sid;
        this.name = name;
        this.phone = phone;
      }
    public int getsid() {
        return sid;
    }
    public String getname () {
        return name;
    }
    public int getphone () {
        return phone;
    public String toString () {
        return "SID:"+ sid + "\t"+ "Name:"+ name + "\t"+ "Phone"+ phone;
    }
}
```

* If any mutable member or class is present-

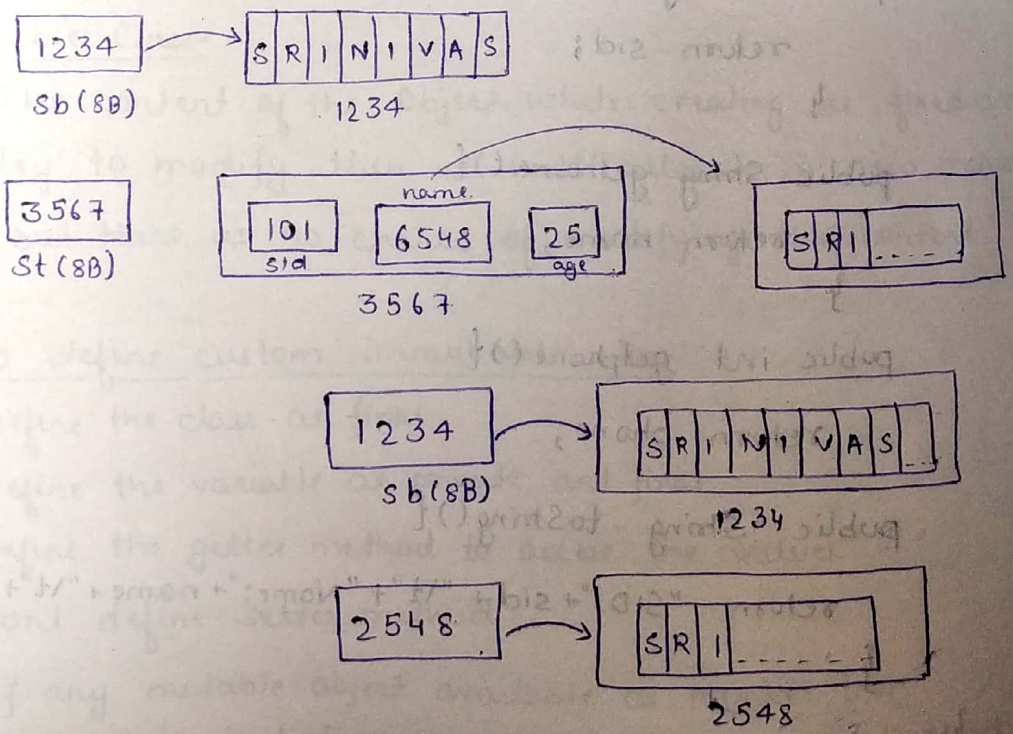
In case of StringBuilder class → Mutable



Sb.append ("NIVAS"); ⊗ this.name = name; → invalid

⊗ public StringBuilder getname() {
return name; → invalid
}

⇒ To make immutable in case of StringBuilder class-



Sb.append ("NIVAS");

× StringBuilder sb1 = s1.getName();
Sb1. append ("NIVAS");

this.name = new StringBuilder (name);
public StringBuilder getname() {
return new StringBuilder (name);
}

ex- When mutable members are there-

```
public class Lab {
  public static void main (String[] arg) {
    StringBuilder sb = new StringBuilder ("SRI");
    Student st = new Student (101, sb, 25);
    Student temp = st;
    System.out.println (st);
    sb.append ("NIVAS");
    System.out.println (st);
    StringBuilder sb1 = st.getname();
    sb1.append ("NIVAS");
    System.out.println (st);
    System.out.println (st == temp);
    System.out.println ("**********");
    StringBuilder ref = new StringBuilder ("SRINIVAS");
    st = new Student (101, ref, 25);
    System.out.println (st);
    System.out.println (st == temp);
  }
}

final class Student {
  private final int sid;
  private final StringBuilder name;
  private final int age;
  public Student (int sid, StringBuilder name, int age) {
      this.sid = sid;
      this.name = new StringBuilder (name);
      this.age = age;
  }
  public int getsid() {
      return sid;
  }
  public StringBuilder getname() {
      return new StringBuilder (name);
  }
```

P.T.O
→

```
public int getage(){
    return age;
  }

  public String toString(){
    return "SID:"+ sid +"\t"+"Name:"+name+"\t"+"Age:"+age;
  }
}
```

Output-

```
SID:101        NAME:SRI        AGE:25
SID:101        NAME:SRI        AGE:25
SID:101        NAME:SRI        AGE:25
TRUE
* * * * * * * * * * *
SID:101        NAME:SRINIVAS   AGE:25
FALSE.
```