Java

Module 3

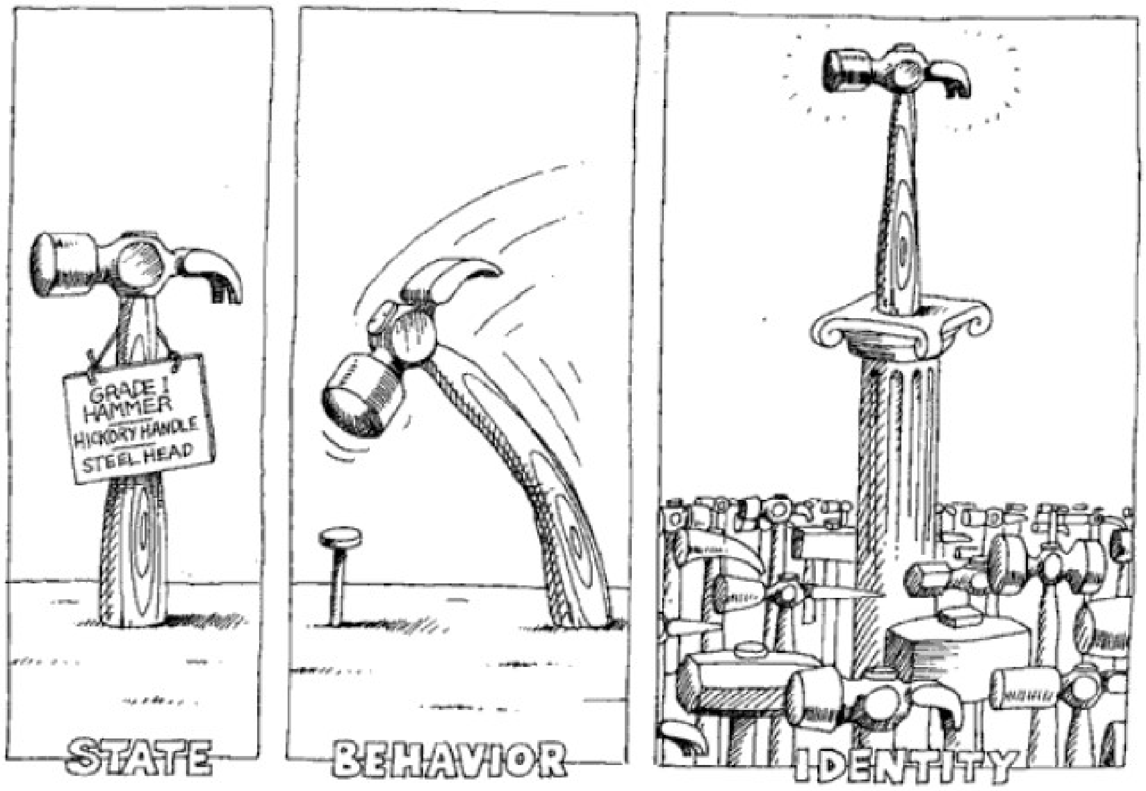
Object Oriented Programming

Author
Srinivas Dande

My JAVA LEARNING CENTER

Everything in the world is an object.

- **Grady Booch - defined the object as follows:**



An object has state, exhibits some weii-defined behavior, and has a unique identity.

An object has state, behavior, and identity; the structure and behavior of similar objects are defined in their common class; the terms instance and object are interchangeable.

```
class Stack
{
        int top;
        String name[];     }  State or Data members or Fields or Properties

        push()
        pop()              }  Behavior or Member functions or Methods
}
```

3.1 OOPS Concepts

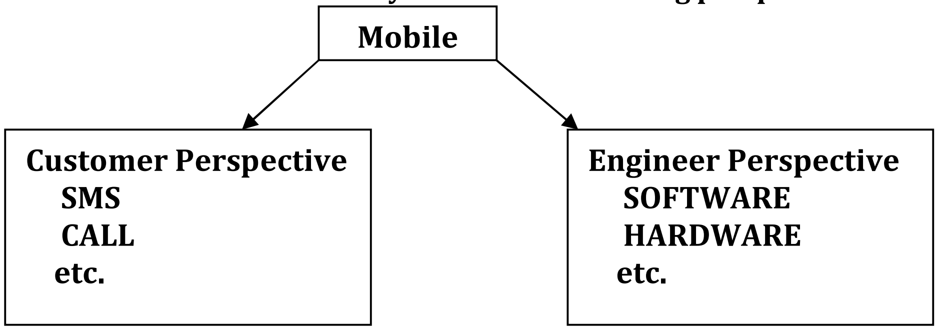
- There are four major Object Oriented principles.
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism

3.1.1 Abstraction

- Abstraction is the process of providing necessary properties and operations of an object.
- When you are describing an object then you need to consider various perspectives.

  Ex:

  To describe a Mobile you can take following perspectives:



- When we are describing a Mobile for Customer then we need to provide only necessary properties and operations which are required for Customer by hiding some of the properties and operations which is required for Engineer.

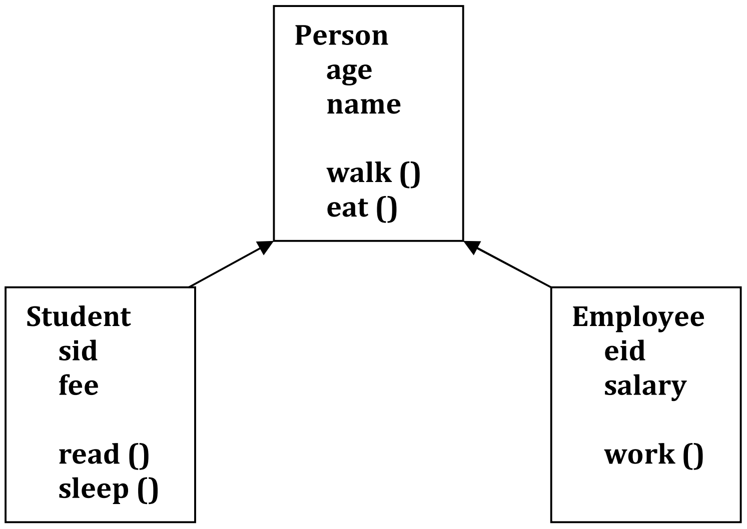
3.1.2 Encapsulation

- Encapsulation is a process of describing the state and behavior of an object in a class.

```
class Person
{
        int age;        } State or Data members or Fields or Properties
        String name;

        walk ()  } Behavior or Member functions or Methods
        eat ()
}
```

3.1.3 Inheritance

- Inheritance is the process of writing new class by inheriting commonly used state and behavior of existing class.



- Existing Class is called as Super class or Base class or Parent Class.
- Newly defined Class is called as Sub class or Derived class or Child class.

3.1.4 Polymorphism

- Polymorphism (One name - Many forms)
- Polymorphism is the ability of an object to behave differently at different situations.

Ex:

1. Power button has polymorphic behavior
   a. When you press once, it will switch on the device.
   b. When you press it again device will be switched off.
2. + Operator in java has polymorphic behavior
   a. When operands are numeric then it will perform arithmetic addition.
   b. When the operands are of String type then it will perform String Concatenation.

3.2. Classes and Objects

Class

- Class is a prototype or template or pattern or blueprint of object's State and Behavior.
- Class contains logical description of real world entity in terms of State and Behavior.
- Class Definition tells the JVM how to create Objects for that Class.

Class = State + Behavior

- .

Syntax:

```
[modifiers] class <ClassName>
{
// Members of class
}
```

```
class Customer{
   int cid;
   String cname;
   long phone;
   void show(){
  ...
   }
}
```

Object

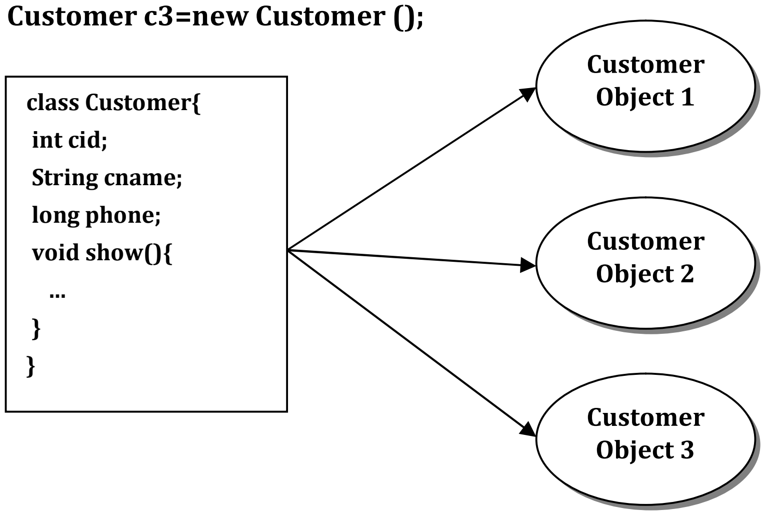
- Object is the physical representation of class.
- Object is the run-time entity because memory will be allocated for an object while executing the program as per class description.
- Memory for object will be allocated in the HEAP MEMORY.
- It is also called as instance of the class so both object and instance are interchangeable.

Syntax:

```
<ClassName> <refVarName> =new <ClassName> ();
```

Ex:

```
Customer c1=new Customer ();
Customer c2=new Customer ();
Customer c3=new Customer ();
```



```
class Customer{
 int cid;
 String cname;
 long phone;
 void show(){
   ...
 }
}
```

Lab248.java

```
class Hello {
int a;
int b=20;

void show(){
System.out.println("a : "+ a);
System.out.println("b : "+ b);
}

}

class Lab248 {
public static void main(String args[]){

Hello h1=new Hello();
h1.show();

h1.a=10;
h1.show();

Hello h2=new Hello();
h2.show();

h2.a=100;
h2.b=200;
h2.show();

}
}
```

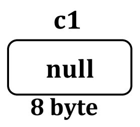
Lab249.java

```
class Customer {
int cid;
String cname;
String email;
long phone;

void show(){
System.out.println(cid+"\t"+cname+"\t"+email+"\t"
+phone);
}

}

class Lab249 {
public static void main(String args[]){

Customer  cust1 = new Customer();
cust1.show();

cust1.cid=101;
cust1.cname="Sri";
cust1.email="sri@myjlc.com";
cust1.phone=123456;
cust1.show();

Customer  cust2 = new Customer();
cust2.show();

cust2.cid=102;
cust2.cname="Vas";
cust2.email="vas@myjlc.com";
cust2.phone=56789;
cust2.show();
}
}
```

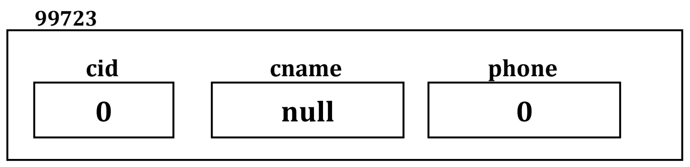
What is happening inside the JVM when you create an object:

Customer c1=new Customer ();

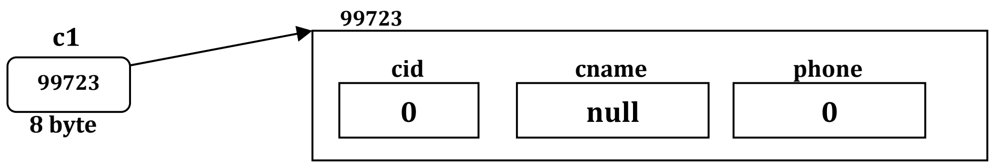
1) **Allocates 8 bytes of memory for the reference variable and initializes with null value.**



2) **Creates an object i.e. Allocates the memory for instance variables of the class and initializes with the default values.**



3) **Address of newly created object will be assigned to the reference variable.**



3.3. Members of Class

Following members can be defined in a class:

1) **Variables**
2) **Blocks**
3) **Constructors**
4) **Methods**
5) **Inner Classes**

3.3.1. Variables

- **Variable is the container which holds user data.**
- **Memory will be allocated for the variable while executing the program.**
- **Value of the variable can be changed any number of times during the program execution.**

Types of Variables

There are two types of variables based on data type used to declare the variable.

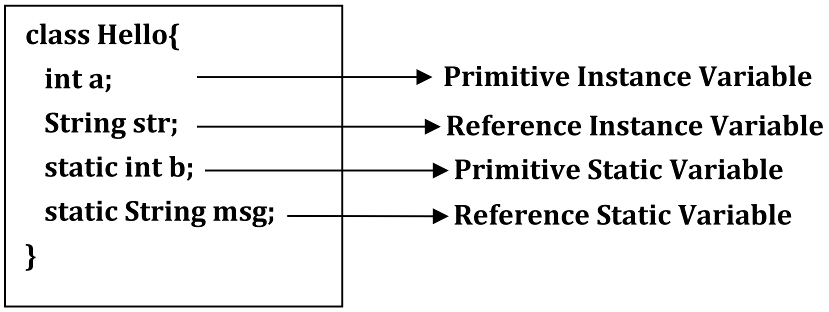
1) **Primitive Variables**
2) **Reference Variables**

Types of Variables

There are two types of variables which can be declared inside the class directly.

1) **Instance Variables**
2) **Static Variables**

- **Both Instance Variables and Static Variables can be Primitive or Reference.**



```
class Hello{
  int a;
  String str;
  static int b;
  static String msg;
}
```

Primitive Instance Variable
Reference Instance Variable
Primitive Static Variable
Reference Static Variable

3.3.1.1. Instance Variables

- Variables declared in the class without static modifier are called as instance variables.
- Instance variables are also called as non-static variables.
- Instance Variables belongs to Instance or Object. So Memory will be allocated for instance variables at the time of creating an object.
- Multiple copies of memory will be allocated for multiple objects.

- <u>Accessing Instance Variables.</u>

```
class Hello{
   int a;
}
```

a) Hello h = new Hello(); h.a	Allowed Only one way to access the Instance.
b) Hello h = null; h.a	Not Allowed
c) Hello.a	Not Allowed

3.3.1.2 Static Variables

- Variables declared in the class using static modifier are called as static variables.
- Static variables are also called as Class variables.
- Static Variables belongs class. So Memory will be allocated for static variables at the time of loading the class.
- Only One copy of memory will be allocated for multiple objects.

- <u>Accessing Static Variables.</u>

```
class Hello{
     static int b;
}
```

a) Hello h = new Hello(); h.b	Allowed
b) Hello h = null; h.b	Allowed
c) Hello.b	Allowed

Lab250.java

```
class Hello{

int a=10;
static int b=20;

void m1(){
System.out.println(a);
System.out.println(b);
}

static void m2(){
//System.out.println(a);
System.out.println(b);
}
}

class Lab250 {
public static void main(String args[]){
Hello.m2();
}
}
```

Lab251.java

```
class Hello{

int a=10;
static int b=20;

void m1(){
System.out.println(a);
System.out.println(b);
}

static void m2(){
//System.out.println(a);
System.out.println(b);
}
}

class Lab251 {
public static void main(String args[]){
Hello h=new Hello();
h.m1();
h.m2();
}
}
```

Lab252.java

```
class Hello{
int a;
}

class Lab252 {
public static void main(String args[]){
Hello h=new Hello();
System.out.println(h.a);
h.a=99;
System.out.println(h.a);
}
}
```

Lab253.java

```
class Hello{
static int b;
}

class Lab253 {
public static void main(String args[]){
Hello h=new Hello();
System.out.println(h.b);
h.b=99;
System.out.println(h.b);
}
}
```

Lab254.java

```
class Hello{
int a;
}

class Lab254 {
public static void main(String args[]){
Hello h1=new Hello();
Hello h2=new Hello();

System.out.println(h1.a+"\t"+h2.a);
h1.a=99;
System.out.println(h1.a+"\t"+h2.a);
}
}
```

Lab255.java

```
class Hello{
static int b;
}

class Lab255 {
public static void main(String args[]){
Hello h1=new Hello();
Hello h2=new Hello();

System.out.println(h1.b+"\t"+h2.b);
h1.b=99;
System.out.println(h1.b+"\t"+h2.b);
}
}
```

Lab256.java

```
class Hello{
int a;
}

class Lab256 {
public static void main(String args[]){
new Hello().a=99;
System.out.println(new Hello().a);
}
}
```

Lab257.java

```
class Hello{
static int b;
}

class Lab257 {
public static void main(String args[]){
new Hello().b=99;
System.out.println(new Hello().b);
}
}
```

Lab258.java

```
class Hello{
int a;
}

class Lab258 {
public static void main(String args[]){
Hello h=null;
System.out.println(h.a);
}
}
```

Lab259.java

```
class Hello{
static int b;
}

class Lab259 {
public static void main(String args[]){
Hello h=null;
System.out.println(h.b);
}
}
```

Lab260.java

```
class Hello{
int a;
}

class Lab260 {
public static void main(String args[]){
System.out.println(Hello.a);
}
}
```

Lab261.java

```
class Hello{
static int b;
}

class Lab261{
public static void main(String args[]){
System.out.println(Hello.b);
}
}
```

Lab262.java

```
class Lab262{
int a;
public static void main(String args[]){
System.out.println(a);
}
}
```

Lab263.java

```
class Lab263{
static int b;
public static void main(String args[]){
System.out.println(b);
}
}
```

3.3.2 Blocks

- Block is a set of instructions defined in curly braces.
- Block doesn't have any name so you can't invoke the block explicitly.
- Block will be invoked by the JVM automatically.
- Blocks defined inside the class directly as a member are called as Initialization Blocks.
- There are two types of initialization blocks:
  - Instance Initialization Block (IIB)
  - Static Initialization Block (SIB)

3.3.2.1 Instance Initialization Blocks

- Block defined inside the class directly without static modifier is called as Instance Initialization Block.
- Instance Initialization Block can be called shortly as Instance Bloks
- Instance Bloks will be invoked by the JVM automatically at the time of Creating the Object.

```
class  Hello{

 {
   System.out.println("I am Instance Block");
 }

}
```

3.3.2.2 Static Initialization Blocks

- Block defined inside the class directly with static modifier is called as Static Initialization Block.
- Static Initialization Block can be called shortly as Static Blocks
- Static Blocks will be invoked by the JVM automatically at the time of Loading the Class.

Ex:

```
class  Hello{

 static{
   System.out.println("I am SIB");
 }

}
```

Lab264.java

```
class Lab264{
public static void main(String args[]){
int a;
a=10;
System.out.println(a);
}
}
```

Lab265.java

```
class Hello{
int a;
a=10;
System.out.println(a);
}

class Lab265{
public static void main(String args[]){
Hello h=new Hello();
}
}
```

Lab266.java

```
class Hello{
int a;

{
System.out.println("I am I.B ");
a=10;
System.out.println(a);
}

}

class Lab266{
public static void main(String args[]){
Hello h=new Hello();
}
}
```

Lab267.java

```
class Hello{
static int b;

static{
System.out.println("I am S.B ");
b=20;
System.out.println(b);
}

}

class Lab267{
public static void main(String args[]){
System.out.println("In Main : b = "+Hello.b);
}
}
```

Lab268.java

```
class Hello{
int a;
static int b;
{
System.out.println("I am I.B ");
System.out.println(a);
a=10;
System.out.println(a);
}
static{
System.out.println("I am S.B ");
System.out.println(b);
b=20;
System.out.println(b);
}
}
class Lab268{
```

Lab269.java

```
class Hello{
int a;
static int b;
{
System.out.println("I am I.B ");
System.out.println(a);
a=10;
System.out.println(a);
}
static{
System.out.println("I am S.B ");
System.out.println(b);
b=20;
System.out.println(b);
}
}
class Lab269{
```

```
public static void main(String args[]){
System.out.println("In Main : b = "+Hello.b);
}
}
```

```
public static void main(String args[]){
Hello h=new Hello();
}
}
```

Lab270.java

```
class Hello{

{
System.out.println("I am I.B ");
}

static{
System.out.println("I am S.B ");
}

}
class Lab270{
public static void main(String args[]){
Hello h=null;
}
}
```

Lab271.java

```
class Lab271{
static{
System.out.println("I am S.B ");
}

public static void main(String args[]){
System.out.println("I am main() ");
}
}
```

Lab272.java

```
class Hello{
{
System.out.println("I am I.B ");
}
static{
System.out.println("I am S.B ");
}
}

class Lab272{
public static void main(String args[]){
Hello h1=new Hello();
Hello h2=new Hello();
Hello h3=new Hello();
}
}
```

Lab273.java

```
class Hello{
{
System.out.println("I am I.B-1 ");
}
{
System.out.println("I am I.B-2 ");
}
static{
System.out.println("I am S.B-1 ");
}
static{
System.out.println("I am S.B-2 ");
}
}

class Lab273{
public static void main(String args[]){
Hello h=new Hello();
}
}
```

3.3.2.3 Local Variables

- Variables can be declared at two places:
  - Inside the class directly as a member
  - Inside the members of the class
- Variables declared inside the class members like methods, constructors and blocks are called as LOCAL VARIABLES.

Lab274.java

```
class Hello{
int a=10;
static int b=20;
{
System.out.println("I am I.B ");
int c=30;
System.out.println(a);
System.out.println(b);
System.out.println(c);
}
}

class Lab274{
public static void main(String args[]){
Hello h=new Hello();
}
}
```

Lab275.java

```
class Hello{
int a=10;
static int b=20;
{
System.out.println("I am I.B ");
int c;
System.out.println(a);
System.out.println(b);
System.out.println(c);
}
}

class Lab275{
public static void main(String args[]){
Hello h=new Hello();
}
}
```

Lab276.java

```
class Hello{
static int b=20;
static {
System.out.println("I am S.B ");
final int c=30;
System.out.println(b);
System.out.println(c);
}
}

class Lab276{
public static void main(String args[]){
Hello h=new Hello();
}
}
```

Lab277.java

```
class Hello{
static int b=20;
static {
System.out.println("I am S.B ");
static int c=30;
System.out.println(b);
System.out.println(c);
}
}

class Lab277{
public static void main(String args[]){
Hello h=new Hello();
}
}
```

Lab278.java

```
class Hello{
void m1() {
System.out.println("I am m1() ");
int a=10;
System.out.println(a);
}

void m2(){
System.out.println("I am m2()");
String a="Hello Guys";
System.out.println(a);
}
}
class Lab278{
public static void main(String args[]){
Hello h=new Hello();
h.m1();
h.m2();
}
}
```

Lab279.java

```
class Hello{
void m1() {
System.out.println("I am m1() ");
int a=10;
System.out.println(a);
}

void m2(){
System.out.println("I am m2()");
System.out.println(a);
}

}
class Lab279{
public static void main(String args[]){
Hello h=new Hello();
h.m1();
h.m2();
}
}
```

Lab280.java

```
class Hello{

int a=99;

void m1() {
System.out.println("I am m1() ");
int a=10;
System.out.println(a);
}

void m2(){
System.out.println("I am m2()");
System.out.println(a);
}

}

class Lab280{
public static void main(String args[]){
Hello h=new Hello();
h.m1();
h.m2();
}
}
```

Lab281.java

```
class Hello{

int a=10;
static int b=20;

void m1() {
int c=30;
}

}

class Lab281{
public static void main(String args[]){
Hello h=new Hello();
System.out.println(h.a);
System.out.println(h.b);
System.out.println(h.c);
}
}
```

3.3.2.4 Local Blocks

- Block can be defined at two places:
  - Inside the class directly as a member
  - Inside the members of the class
- Blocks defined inside the class members like methods, blocks, constructors are called as LOCAL BLOCKS.
- Local blocks will be executed only when the enclosing class member is executed.

Lab282.java

```
class Hello{
void m1() {
int c=30;
{
System.out.println("I am Local Block");
System.out.println(c);
}
}
}
class Lab282{
public static void main(String args[]){
Hello h=new Hello();
}
}
```

Lab283.java

```
class Hello{
void m1() {
int c=30;
{
System.out.println("I am Local Block");
System.out.println(c);
}
}
}
class Lab283{
public static void main(String args[]){
Hello h=new Hello();
h.m1();
}
}
```

Lab284.java

```
class Hello{
void m1() {
int a=10;
{
System.out.println("I am LB-1");
int b=20;
System.out.println(a);
System.out.println(b);
//System.out.println(c);
}
{
System.out.println("I am LB-2");
int c=30;
System.out.println(a);
//System.out.println(b);
System.out.println(c);
}
}
}
```

```
class Lab284{
public static void main(String args[]){
Hello h=new Hello();
h.m1();
}
}
```

Lab285.java	Lab286.java

class Lab285{
public static void main(String args[]){
System.out.println("Main Begin");
int a=10;
{
int a=20;
System.out.println(a);
}
System.out.println("Main End");
}
}
	class Lab286{
	public static void main(String args[]){
	System.out.println("Main Begin");
	{
	int a=20;
	System.out.println(a);
	}
	int a=10;
	System.out.println("Main End");
	}
	}

SUMMARY

Instance Variables:

1) Multiple copies of memory will be allocated for multiple objects.
2) Instance variable is related to instance or object of the class so when you change data of one object then that will not affect data of other objects.
3) Instance variables must be accessed by reference variable which contains object of the class.
4) Following are the ways to access instance variable:
   a) Using Reference variable that contains object reference

      Hello h = new Hello ();

      h.a=90;   // VALID

   b) Directly using Object reference

      new Hello().a=90; // VALID (Not Recommendable because no reusability)
5) When you try to access instance variable with null reference then java.lang.NullPointerException will be thrown at runtime.

   Hello h=null;

   h.a=90;
6) When you try to access instance variable with Class name then error message will be given at compile time called "non-static variable a cannot be referenced from a static context"

   Hello.a=90;

Static Variables:

7) Only one copy of memory will be allocated for static variable for multiple objects and single copy of memory will be accessed by all the objects.
8) Static variable is related to class so when you change data of static variable then modified data will be accessed by all objects.
9) Static variables can be accessed with class name directly and object is not required but you can access static variable with object also.
10) Following are the ways to access static variable:
    a) With class name

        Hello.b=90;

    b) Using the reference variable which contains null

        Hello h = null;
        h.b=90;

    c) Using the reference variable which contains object address

        Hello h = new Hello ();
        h.b=90;

    d) Directly using Object reference

        new Hello().b=90;

11) When you refer static variable with other then class name then compiler will change it with class name.

    a) Hello h = null;

        h.b=90;   -> Changes to Hello.b=90;

    b) Hello h = new Hello ();

        h.b=90;   -> Changes to Hello.b=90;

    c) new Hello().b=90;   -> Changes to Hello.b=90;

Initialization Blocks:

12) **Class can contain only five class members; you can't place any other Java Statements directly.**

    Ex:

```
class Hello{
   int a;    //VALID
   a=10;   // INVALID
}
```

```
class Hai{
int b=10;                      // VALID
System.out.println(b);   // INVALID
}
```

13) **IIB will be invoked by the JVM automatically after initializing the object.**
14) **SIB will be invoked by the JVM automatically after initializing the class.**
15) **Instance Variable must be initialized within instance block only.**
16) **Static Variable should be initialized within Static block only but there is a chance to initialize static variables within instance block also.**
17) **Instance block will be executed every time when object is created.**
18) **Static block will be executed only once when class is loaded.**
19) **Class will be loaded only once when you or JVM uses the class members first time.**
20) **Instance variables cannot be accessed from static context directly but can be accessed with object reference.**

Local Variables

21) **JVM will not initialize Local variables with default value; it must be initialized by you before
    using.**
22) **Scope of Local variable is within the class member where it is declared.**
23) **Local variable is not a member of a class so it cannot be accessed using class name or object.**
24) **Local variables cannot be static. Only final is allowed for local variables.**
25) **Memory for the Local Variables will be allocated in the STACK MEMORY.**

Assignment #7

Q1) What are OOPS concepts?

Q2) What is an Abstraction?

Q3) What is Encapsulation?

Q4) What is Inheritance?

Q5) What is Polymorphism?

Q6) Explain State and Behavior of class?

Q7) What is Class?

Q8) What is the syntax to define class?

Q9) What is an Object?

Q10) What is the syntax to create Object?

Q11) What is happening inside the JVM when you create an object?

Q12) What are the valid members of class?

Q13) How to access the members of the class?

Q14) What is the use of .(dot) Operator?

Q15) How many types of variables can be declared inside the class directly?

Q16) What is an Instance Variable?

Q17) What is a non-static variable?

Q18) When memory will be allocated for Instance variables?

Q19) How many times memory will be allocated for Instance variables?

Q20) What are ways available to access Instance Variables?

Q21) What is Static Variable?

Q22) What is a Class variable?

Q23) When memory will be allocated for Static variables?

Q24) How many times memory will be allocated for Static variables?

Q25) What are ways available to access Static Variables?

Q26) What is Block?

Q27) How many types of blocks can be written inside the class directly?

Q28) What is Initialization Block?

Q29) How many types of blocks can be written inside the class directly?

Q30) What is Instance Initialization Block?

Q31) When Instance Initialization Block will be executed?

Q32) How many times Instance Block will be executed?

Q33) Can I write multiple Instance Block inside the Class?

Q34) What is Static Initialization Block?

Q35) When Static Initialization Block will be executed?

Q36) How many times Static Initialization Block will be executed?

Q36) Can I write multiple Static Block inside the Class?

Q37) Can I access Instance Variables from IIB?

Q38) Can I access Static Variables from IIB?

Q39) Can I access Instance Variables from SIB?

Q40) Can I access Static Variables from SIB?

Q41) Why Instance Variables are not allowed to access from Static Block?

Q42) What is Local Variables?

Q43) Is Local Variable a member of Class?

Q44) Can I access Local Variables with object reference?

Q45) Can I access Local Variables with Class Name?

Q46) Can I access Local Variables with null reference?

Q47) Can I access Local Variable of Instance Block from Static Block?

Q48) What is the scope of Local Variables?

Q49) What are the modifiers allowed for Local Variables?

Q50) What is Local Block?

Q51) When will be the Local Block gets executed?

Q52) Can I write multiple Local Block inside the Members of Class?

Q53) Is Local Block a member of Class?

Practice Test #7

Q.No	Question	Options	Answer

1	class Test1{
	public static void main(String args[]){
	Student st=new Student();
	System.out.println(st.sname);
	} }
	class Student{
	String sname;
	}
		A) null
		B) 0
		C) NullPointerException
		D) Compile Time Error	

2	class Test2{
	public static void main(String args[]){
	Student st=new Student();
	st.sname="Sri";
	System.out.println(st.sname);
	} }
	class Student{
	String sname;
	}
		A) Sri
		B) null
		C) NullPointerException
		D) Compile Time Error	

3	class Test3{
	public static void main(String args[]){
	Student st1=new Student();
	Student st2=new Student();
	st1.sname="Sri";
	System.out.println(st1.sname+"\t"+st2.sname);
	} }
	class Student{
	static String sname;
	}
		A) null null
		B) Sri Sri
		C) Runtime Exception
		D) Compile Time Error	

4	class Test4{
	public static void main(String args[]){
	Student st=null;
	st.name="Nivas";
	System.out.println(st.name);
	} }
	class Student{
	String name;
	}
		A) null
		B) Nivas
		C) Runtime Exception
		D) Compile Time Error	

5	class Test5{
	public static void main(String args[]){
	Student st=null;
	st.name="Nivas";
	System.out.println(st.name);
	}
	}
	class Student{
	static String name;
	}
		A) null
		B) Nivas
		C) Runtime Exception
		D) Compile Time Error	

6	class Test6{
	public static void main(String args[]){
	System.out.println(new Student().name="Sri");
	System.out.println(new Student().name);
	}
	}
	class Student{
	String name;
	}
		A) Garbage Value
		null
		B) Sri
		null
		C) Sri
		Sri
		D) Runtime Exception
		E) Compile Time Error	

7	class Test7{
	public static void main(String args[]){
	System.out.println(new Student().name="Sri");
	System.out.println(new Student().name);
	}
	}
	class Student{
	static String name;
	}
		A) Garbage Value
		null
		B) Sri
		null
		C) Sri
		Sri
		D) Runtime Exception
		E) Compile Time Error	

8	class Test8{
	public static void main(String args[]){
	String email=null;
	email="sri@jlc.com";
	System.out.println(email);
	}
	}
		A) null
		B) sri@jlc.com
		C) Runtime Exception
		D) Compile Time Error	

9	class Test9{
	static String email=null;
	email="sri@jlc.com";
	public static void main(String args[]){
	System.out.println(email);
	}
	}
		A) null
		B) sri@jlc.com
		C) Runtime Exception
		D) Compile Time Error	

10	class Test10{
	static String email=null;
	{
	email="sri@jlc.com";
	}
	public static void main(String args[]){
	System.out.println(email);
	}
	}
		A) null
		B) sri@jlc.com
		C) Runtime Exception
		D) Compile Time Error	

11	class Test11{
	public static void main(String args[]){
	System.out.println(email);
	}
	static String email=null;
	static{
	email="sri@jlc.com";
	}
	}
		A) null
		B) sri@jlc.com
		C) Runtime Exception
		D) Compile Time Error	

12	class Test12{
	public static void main(String args[]){
	System.out.println(email);
	new Test();
	System.out.println(email);
	}
	static String email=null;
	{
	email="sri@jlc.com";
	}
	}
		A) null
		null
		B) null
		sri@jlc.com
		C) sri@jlc.com
		sri@jlc.com
		D) sri@jlc.com
		null
		E) Runtime Exception
		F) Compile Time Error	

13	class Test13{
	public static void main(String args[]){
	Student st=null;
	System.out.println("Main");
	}
	}
	
	class Student{
	static{
	System.out.println("St Block");
	}
	}
		A) St Block
		Main
		B) St Block
		C) Main
		D) Runtime Error
		E) Compile Time Error	

14	class Test14{
	public static void main(String args[]){
	new Student();
	new Student();
	System.out.println("Main");
	}
	}
	
	class Student{
	static{
	System.out.println("Block");
	}
	}
		A) Block
		Block
		Main
		B) Block
		Main
		C) Main
		D) Block
		E) Runtime Error
		F) Compile Time Error	

15	class Test15{
	public static void main(String args[]){
	new Student();
	new Student();
	System.out.println("Main");
	}
	}
	
	class Student{
	{
	System.out.println("Block");
	}
	}
		A) Block
		Block
		Main
		B) Block
		Main
		C) Main
		D) Block
		E) Runtime Error
		F) Compile Time Error	

16

```
class Test16{
{
System.out.println("BLOCK1");
}
public static void main(String args[]){
System.out.println("Main");
}
}
```

A) BLOCK1
   Main
B) BLOCK1
C) Main
D) Runtime Error
E) Compile Time Error

17

```
class Test17{
static{
System.out.println("BLOCK1");
}
public static void main(String args[]){
System.out.println("Main");
}
}
```

A) BLOCK1
   Main
B) BLOCK1
C) Main
D) Runtime Error
E) Compile Time Error

18

```
class Test18{
public static void main(String args[]){
System.out.println("Main");
}
static{
System.out.println("BLOCK1");
}
}
```

A) BLOCK1
   Main
B) BLOCK1
C) Main
D) Runtime Error
E) Compile Time Error

19

```
class Test19{
static final float f1;
public static void main(String args[]){
System.out.println(f1);
}
}
```

A) 0.0
B) Garbage Value
C) Runtime Error
D) Compile Time Error

20	class Test20{
	static final float f1;
	{
	f1=90;
	}
	
	public static void main(String args[]){
	System.out.println(f1);
	}
	}
		A) 0.0
		B) 90.0
		C) Garbage Value
		D) Runtime Error
		E) Compile Time Error	

21	class Test21{
	public static void main(String args[]){
	Student st=new Student();
	System.out.println(st.fee);
	}
	}
	
	class Student{
	final float fee;
	}
		A) 0.0
		B) Garbage Value
		C) Runtime Error
		D) Compile Time Error	

22	class Test22{
	public static void main(String args[]){
	Student st=new Student();
	System.out.println(st.fee);
	}
	}
	
	class Student{
	final float fee;
	{
	fee=1000;
	}
	}
		A) 0.0
		B) 1000.0
		C) Runtime Error
		D) Compile Time Error	

23	class Test23{
	public static void main(String args[]){
	Student st=new Student();
	System.out.println(st.fee);
	}
	}
	
	class Student{
	final float fee;
	{
	fee=1000;
	}
	{
	fee=2000;
	}
	}
		A) 0.0
		B) 1000.0
		C) 2000.0
		D) Runtime Error
		E) Compile Time Error	

24	class Test24{
	public static void main(String args[]){
	int ab=90;
	System.out.println(ab);
	{
	System.out.println(ab);
	}
	}
	}
		A) 0
		0
		B) 90
		0
		C) 90
		90
		D) Compile Time Error	

25	class Test25{
	public static void main(String args[]){
	{
	int ab=90;
	System.out.println(ab);
	}
	System.out.println(ab);
	}
	}
		A) 0
		0
		B) 90
		0
		C) 90
		90
		D) Compile Time Error	

26	class Test26{
	public static void main(String args[]){
	{
	int ab=90;
	System.out.println(ab);
	}
	
	String ab="AB";
	System.out.println(ab);
	}
	}
		A) 0
		null
		B) 90
		AB
		C) 90
		90
		D) Compile Time Error	

27	class Test27{
	public static void main(String args[]){
	{
	int ab=90;
	 {
	  System.out.println(ab);
	 }
	}
	
	String ab="AB";
	System.out.println(ab);
	}
	}
		A) 0
		null
		B) 90
		AB
		C) 90
		90
		D) Compile Time Error	

28	class Test28{
	public static void main(String args[]){
	int ab=90;
	System.out.println(ab);
	
	{
	String ab="AB";
	System.out.println(ab);
	}
	
	}
	}
		A) 0
		null
		B) 90
		AB
		C) 90
		90
		D) Compile Time Error	

29	class Test29{
	static int ab=90;
	static{
	int ab=10;
	System.out.println(ab);
	}
	public static void main(String args[]){
	System.out.println(ab);
	}
	}
		A) 10
		90
		B) 10
		C) 90
		D) 10
		10
		E) 90
		90
		F) Compile Time Error	

30	class Test30{
	static int ab=90;
	static{
	ab=10;
	System.out.println(ab);
	}
	public static void main(String args[]){
	System.out.println(ab);
	}
	}
		A) 10
		90
		B) 10
		C) 90
		D) 10
		10
		E) 90
		90
		F) Compile Time Error	

31	class Test31{
	static int ab=90;
	{
	ab=10;
	System.out.println(ab);
	}
	public static void main(String args[]){
	System.out.println(ab);
	}
	}
		A) 10
		90
		B) 10
		C) 90
		D) 10
		10
		E) 90
		90
		F) Compile Time Error	

32	class Test32{
	public static void main(String args[]){
	Hello h=new Hello();
	h.show();
	}
	}
	class Hello{
	void show(){
	this=null;
	} }
		A) null
		B) 0
		C) Runtime Error
		D) Compile Time Error	

33	class Test33{
	public static void main(String args[]){
	Hello h=new Hello();
	h.show();
	} }
	class Hello{
	int a;
	void show(){
	System.out.println(Hello.this.a);
	}
	}
		A) null
		B) 0
		C) Runtime Error
		D) Compile Time Error	

34	class Test34{
	public static void main(String args[]){
	System.out.println(Hello.a);
	}
	}
	class Hello{
	static int a=10;
	static Hello h=new Hello();
	{
	System.out.println("IB");
	}
	static{
	System.out.println("SB");
	}
	}
		A) IB
		SB
		10
		B) SB
		IB
		10
		C) 10
		SB
		IB
		D) Runtime Error
		E) Compile Time Error	