OBJECT ORIENTED PROGRAMMING IN JAVA (OOPs)

OOPs

- The class is at the core of Java. It is the logical construct upon which the entire Java language is built because it defines the shape and nature of an object. As such, the class forms the basis for object-oriented programming in Java. Any concept you wish to implement in a Java program must be encapsulated within a class.

OOPs

- **Structured programming**:
  Algorithms + Data Structures = Programs

- **OOP reverses the order**:
  puts the **data first**, then looks at the **algorithms** to operate on the data.

Class

- A class is the template or blueprint from which objects are made. Perhaps the most important thing to understand about a class is that it defines a new data type. Once defined, this new type can be used to create objects of that type. Thus, a class is a template for an object, and an object is an instance of a class. Because an object is an instance of a class, you will often see the two words object and instance used interchangeably.
- A class is declared by using **class keyword.**

Class

- **Syntax**:

```
class classname {
    type instance-variable1;
    type instance-variable2;
    // ...
    type instance-variableN;

    type methodname1(parameter-list) {
      // body of method
    }
    type methodname2(parameter-list) {
      // body of method
    }
    // ...
    type methodnameN(parameter-list) {
      // body of method
    }
}
```

Class

- **Example**:

```
public class Box {
    private double width;
    private double height;
    private double depth;
}
```

As stated, a class defines a new type of data. In this case, the new data type is called **Box.** You will use this name to declare objects of type Box. **It is important to remember that** a class declaration only creates a template, **it does not create an actual object.** Thus, the preceding code does not cause any objects of type Box to come into existence.

Class

- To actually create a **Box object,** you will use a statement like the following:

```
Box mybox = new Box(); // create a Box object called mybox
```

After this statement executes, **mybox** will be an **instance of Box.** Thus, it will have “physical” reality.

To assign the width variable of mybox the value 100

mybox.width = 100;

Class

- Internal details of ***new*** operator:

  the ***new operator*** dynamically **allocates memory for an object**. It has this general form:

class-var = new *classname* ( );

Statement	Effect
Box mybox;	null mybox
mybox = new Box();	mybox → Width, Height, Depth Box object

Class

- Here, class-var is a variable of the class type being created. The class name is the name of the class that is being instantiated. The class name followed by parentheses specifies the constructor for the class. A constructor defines what occurs when an object of a class is created. Constructors are an important part of all classes and have many significant attributes. Most real-world classes explicitly define their own constructors within their class definition. However, if no explicit constructor is specified, then Java will automatically supply a default constructor. This is the case with **Box.**

Objects

- Objects represents the states and behaviors of class.
- The three key characteristics of objects are:
  - **object's behavior**—*What can you do with this object, or what methods can you apply to it?*
  - **object's state**—*How does the object react when you invoke those methods?*
  - **object's identity**—*How is the object distinguished from others that may have the same behavior and state?*

Objects

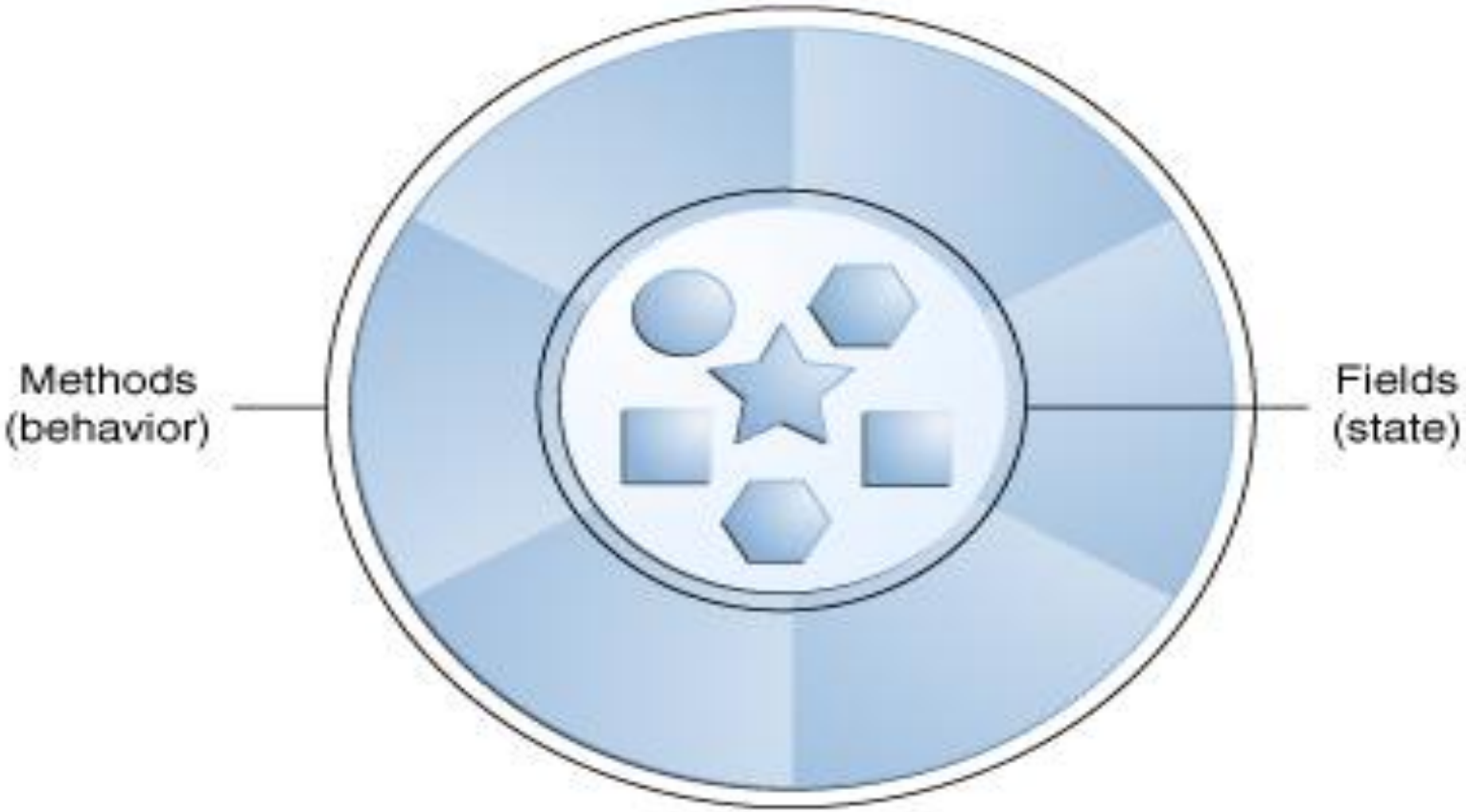


Fig. : A software object.

Objects

- **Syntax**:
  - **<class-name> ref-var = new <class-name>**

Example:

for class **Dog:**

Dog myDog = new Dog();

Constructor

- Constructor in java is a special type of method that is used to initialize the object.
- Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor.

Rules for creating java constructor

- There are basically **two rules** defined for the constructor.

1. Constructor name must be **same as its class name** .

2. Constructor must have no explicit **return type** .

Types of java constructors

- There are two types of constructors:

  1. **Default** constructor (**no-arg constructor**)

  2. **Parameterized** constructor

-

Default Constructor

- A constructor having **no parameter** is known as default constructor.

- **Syntax :**

  <class_name>(){ }

Example:

Package Explorer

- FIrstProgram
- HelloWorldDialog
- PackageExample
  - src
    - org
      - constr
        - Course.java
      - course
      - multiclassex
      - pkgpvt
      - stud
      - typeconversion
      - Course.java
  - JRE System Library [jdk1.7.

Course.java

```java
package org.constr;

public class Course {

    private String course;

    public Course() {
        course = "JAVA";
    }

    private void showCourse() {
        System.out.println("Course is: " + course);
    }

    public static void main(String[] args) {

        Course course = new Course(); //Constructor called first.

        course.showCourse(); //Course is: JAVA
    }

}
```

Console | @ Javadoc | Problems | Declaration

```
<terminated> Course (2) [Java Application] C:\Program Files\Java\jdk1.7.0_51\bin\javaw.exe (Apr 23, 2015, 9:12:
Course is: JAVA
```

Q) What is the purpose of default constructor?

- Default constructor provides the **default values to the object like 0, null** etc. depending on the data type.

Parameterized Constructor

- A constructor **having parameters** is known as parameterized constructor.
- Parameterized constructor is used **to provide different values to the distinct objects**.

Example:

Package Explorer

- FIrstProgram
- HelloWorldDialog
- PackageExample
  - src
    - org
      - constr
        - Area.java
        - Course.java
      - course
      - multiclassex
      - pkgpvt
      - stud
      - typeconversion
      - Course.java
  - JRE System Library [jdk1.7.

Area.java

```java
package org.constr;

public class Area {

    private int length;
    private int breadth;

    public Area(int length, int breadth) {
        this.length = length;
        this.breadth = breadth;
    }

    public void calculateArea() {
        int area = length * breadth;
        System.out.println("Area is:" + area);
    }

    public static void main(String[] args) {
        Area area = new Area(5, 4);
        area.calculateArea();
    }

}
```

Console | @ Javadoc | Problems | Declaration

```
<terminated> Area (1) [Java Application] C:\Program Files\Java\jdk1.7.0_51\bin\ja
Area is:20
```

Constructor Overloading in Java

- Constructor overloading is a technique in which a class can have any number of constructors that differ in parameter lists.
- The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Example:

Package Explorer

- FIrstProgram
- HelloWorldDialog
- PackageExample
  - src
    - org
      - constr
        - Area.java
        - Course.java
        - Student.java
      - course
      - multiclassex
      - pkgpvt
      - stud
      - typeconversion
      - Course.java
  - JRE System Library [jdk1.7.

Student.java

```java
package org.constr;

public class Student {
    private int id;
    private String name;
    private int age;

    public Student(int i, String n) {
        id = i;
        name = n;
    }

    public Student(int i, String n, int a) {
        id = i;
        name = n;
        age = a;
    }

    public void display() {
        System.out.println(id + " " + name + " " + age);
    }

    public static void main(String args[]) {

        Student s1 = new Student(101, "Rajesh");
        s1.display();

        Student s2 = new Student(102, "Aryan", 15);
        s2.display();
    }

}
```

Console | @ Javadoc | Problems | Declaration

```
<terminated> Student (2) [Java Application] C:\Program Files\Java\jdk1.7.0_51\bin\javaw.exe (Ap
101 Rajesh 0
102 Aryan 15
```

Difference between constructor and method in java

Java Constructor	Java Method
Constructor is used to initialize the state of an object.	Method is used to expose behaviour of an object.
Constructor must not have return type.	Method must have return type.
Constructor is invoked implicitly.	Method is invoked explicitly.
The java compiler provides a default constructor if you don't have any constructor.	Method is not provided by compiler in any case.

Q) Does constructor return any value?

- yes, that is current class instance (You cannot use return type yet it returns a value).
-

Q)Can constructor perform other tasks instead of initialization?

Yes, like object creation, starting a thread, calling method etc. You can perform any operation in the constructor as you perform in the method.

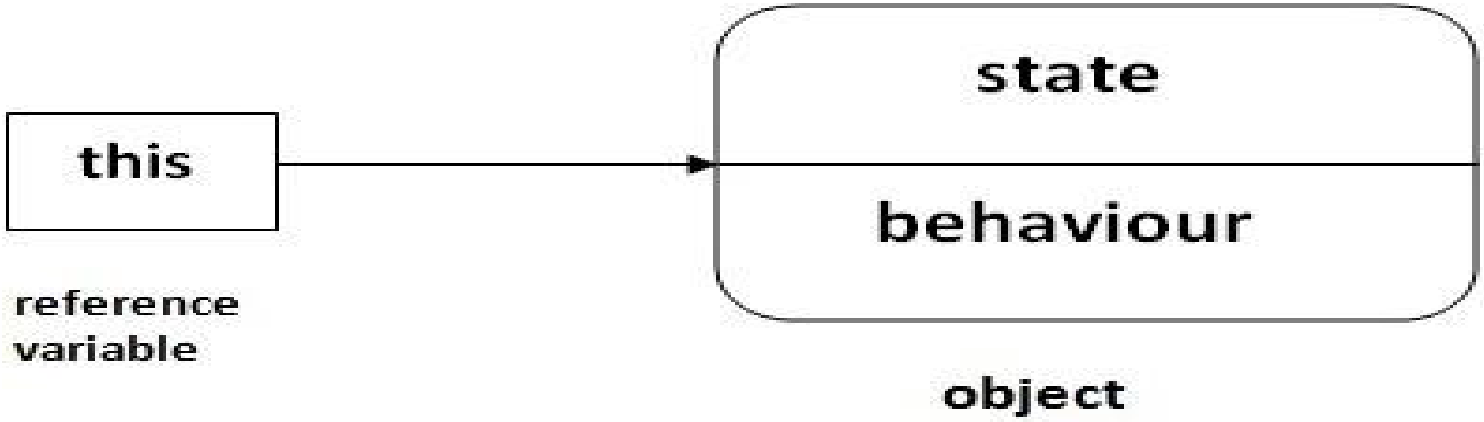
‘this’ keyword in java

- There can be a lot of usage of **java this keyword.**
- In java, ‘**this**’ keyword **represents the current object.**

Usage of java this keyword

- This keyword can be use to:

1. refer current class instance variable.
2. invoke current class constructor.
3. invoke current class method (implicitly)



The this keyword can be used torefer current class instance variable.

- If there is **ambiguity between the instance variable and parameter**, this keyword resolves the problem of ambiguity.

Example:

```
1.  class Student10{
2.          int id;
3.          String name;
4.
5.          Student10(int id, String name){
6.                  id = id;
7.                  name = name;
8.          }
9.          void display(){System.out.println(id+" "+name);}
10.
11.         public static void main(String args[]){
12.         Student10 s1 = new Student10(111,"Karan");
13.         Student10 s2 = new Student10(321,"Aryan");
14.         s1.display();
15.         s2.display();
16.         }
17. }
```

Output:

```
0 null
0 null
```

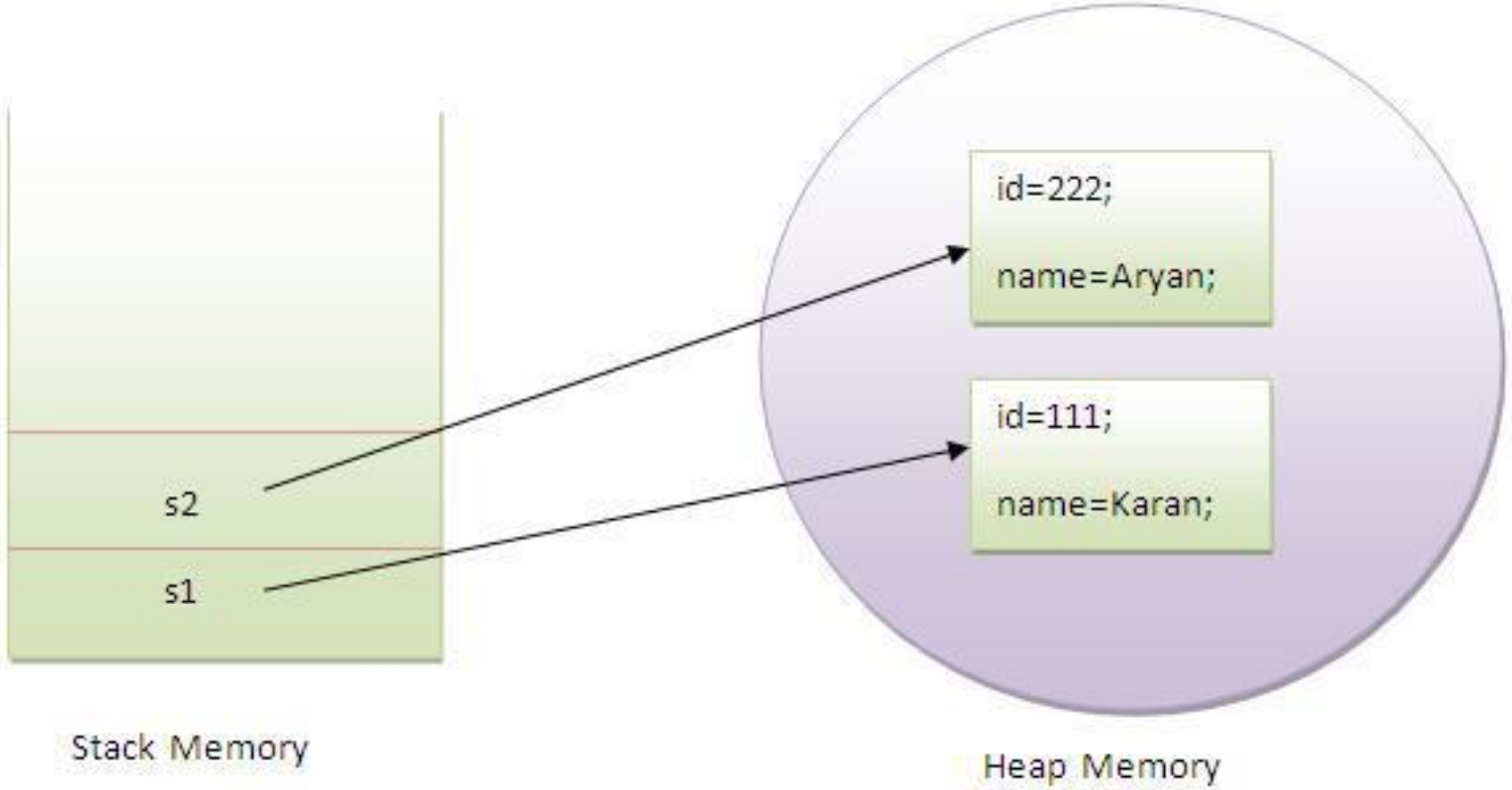
- In the above example, parameter (**formal arguments**) and **instance variables** are same that is why we are using this keyword to distinguish between local variable and instance variable.

Example

```
1. //example of this keyword
2. class Student11{
3. int id;
4. String name;
5.
6. Student11(int id,String name){
7. this.id = id;
8. this.name = name;
9. }
10. void display(){System.out.println(id+" "+name);}
11. public static void main(String args[]){
12. Student11 s1 = new Student11(111,"Karan");
13. Student11 s2 = new Student11(222,"Aryan");
14. s1.display();
15. s2.display();
16. }
17. }
```

Output:

```
111 Karan
222 Aryan
```


id=222;
name=Aryan;
id=111;
name=Karan;
s2
s1
Stack Memory
Heap Memory

Program where this keyword is not required

- If local variables(formal arguments) and instance variables are different, there is no need to use this keyword like in the following program:

Example:

```
class Student12{
int id;
String name;

Student12(int i, String n){
id = i;
name = n;
}
void display(){System.out.println(id+" "+name);}
public static void main(String args[]){
Student12 e1 = new Student12(111,"karan");
Student12 e2 = new Student12(222,"Aryan");
e1.display();
e2.display();
}
}
```

Output:

```
111 Karan
222 Aryan
```

this() can be used to invoked current class constructor.

- The this() constructor call can be used to invoke the current class constructor (constructor chaining). This approach is better if you have many constructors in the class and want to reuse that constructor.

Example:

```
1. //Program of this() constructor call (constructor chaining)
3. class Student13{
4. int id;
5. String name;
6. Student13(){System.out.println("default constructor is invoked");}
7.
8. Student13(int id, String name){
9. this ();//it is used to invoked current class constructor.
10. this.id = id;
11. this.name = name;
12. }
13. void display(){System.out.println(id+" "+name);}
14.
15. public static void main(String args[]){
16. Student13 e1 = new Student13(111,"karan");
17. Student13 e2 = new Student13(222,"Aryan");
18. e1.display();
19. e2.display();
20. }
21. }
```

Output:

```
default constructor is invoked
default constructor is invoked
111 Karan
222 Aryan
```

Rule: Call to this() must be the first statement in constructor.

```
class Student15{
int id;
String name;
Student15(){System.out.println("default constructor is invoked");}

Student15(int id, String name){
id = id;
name = name;
this ();//must be the first statement
}
void display(){System.out.println(id+" "+name);}

public static void main(String args[]){
Student15 e1 = new Student15(111,"karan");
Student15 e2 = new Student15(222,"Aryan");
e1.display();
e2.display();
}
}
```

Output: Compile Time Error

The this keyword can be used to invoke current class method (implicitly).

- We may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method.
- Let's see the example :

```
class A{

void m(){}

void n(){
m();
}

public static void main(String args[]){

 new A().n();

}}
```


```
class A{

void m(){}

void n(){
this.m();
}

public static void main(String args[]){

 new A().n();

}}
```

Let’s Prove this keyword

- Let's prove that this keyword **refers to the current class instance variable**. In this program, we are printing the reference variable and this, output of both variables are same.

PrintThisKeyword.java

```java
package oop;

public class PrintThisKeyword {

    public void printThis() {
        System.out.println(this); // prints same reference ID
    }

    public static void main(String[] args) {
        PrintThisKeyword obj = new PrintThisKeyword();
        obj.printThis();

        System.out.println(obj); // prints the reference ID

    }

}
```

1

2

Console | Markers | Properties | Servers | Data Source Explorer | Snippets

```
<terminated> PrintThisKeyword [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (S
oop.PrintThisKeyword@6276e1db
oop.PrintThisKeyword@6276e1db
```

Java static keyword

- The static keyword in java is used for **memory management mainly**. We can apply java static keyword **with variables, methods, blocks and nested class.** The static keyword belongs to the class than instance of the class.

The static can be:

1. **variable** (also known as class variable)
2. **method** (also known as class method)
3. **block**
4. **nested** class

Example of static variable

Java static property is shared to all objects

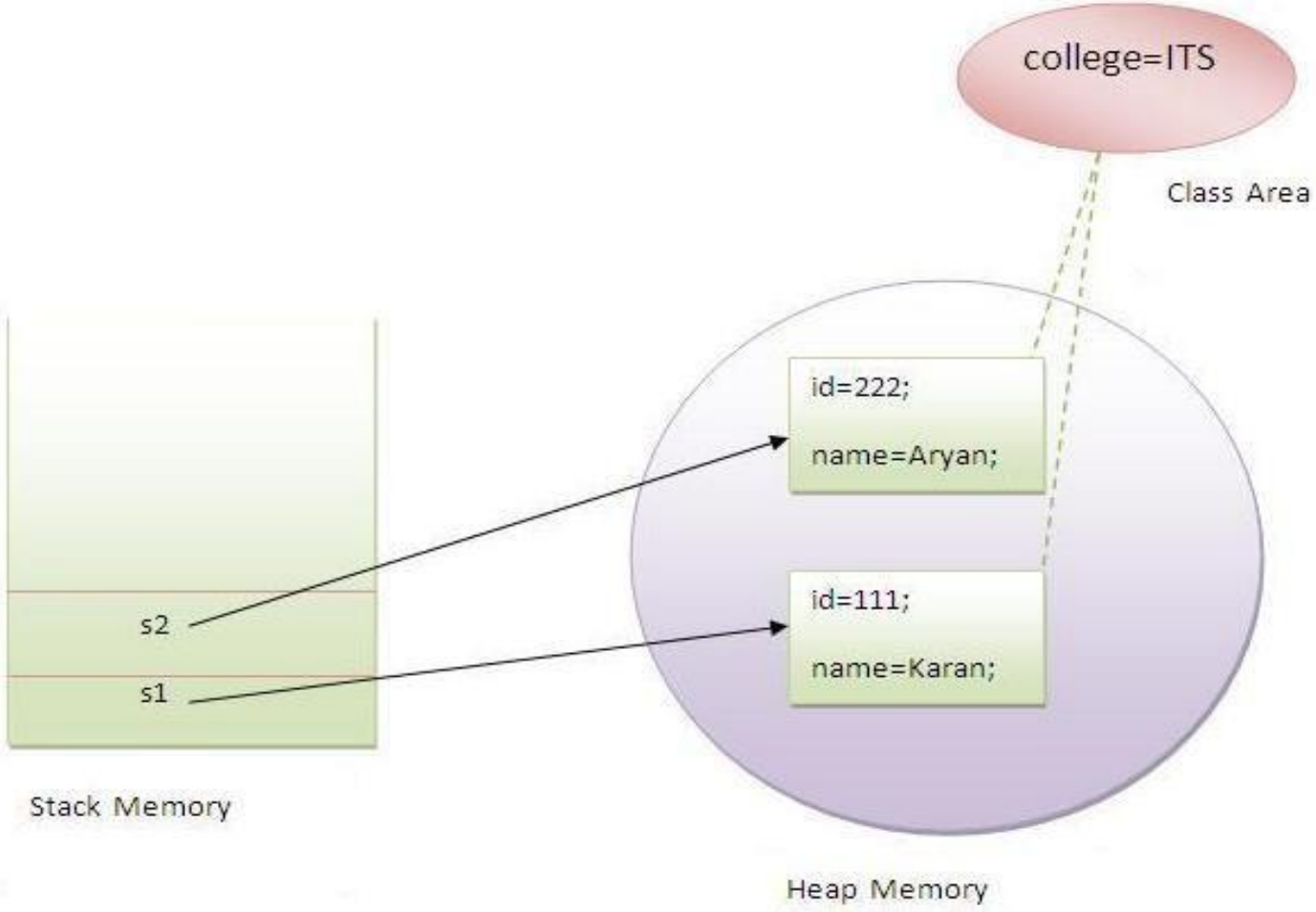
StaticVariable.java

```java
public class StaticVariable {
    int rollno;
    String name;
    static String college = "ITS";

    StaticVariable(int r, String n) {
        rollno = r;
        name = n;
    }

    void display() {
        System.out.println(rollno + " " + name + " " + college + "\n");
    }

    public static void main(String args[]) {

        StaticVariable s1 = new StaticVariable(111, "Karan");
        StaticVariable s2 = new StaticVariable(222, "Aryan");

        s1.display();
        s2.display();
    }

}
```

Console | Markers | Properties | Servers | Data Source Explorer | Snippets

```
<terminated> StaticVariable [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Sep 1, 2015, 9:42:34 AM
111 Karan ITS

222 Aryan ITS
```

Example:


college=ITS
Class Area
id=222;
name=Aryan;
id=111;
name=Karan;
s2
s1
Stack Memory
Heap Memory

Program of counter with/without static variable

CounterWithoutStatic.java

```java
package oop;

public class CounterWithoutStatic {
    int count = 0;// will get memory when instance is created

    CounterWithoutStatic() {
        count++;
    }

    private void showCount() {
        System.out.println(count);
    }

    public static void main(String args[]) {

        new CounterWithoutStatic().showCount();
        new CounterWithoutStatic().showCount();
        new CounterWithoutStatic().showCount();

    }

}
```

CounterWithStatic.java

```java
package oop;

public class CounterWithStatic {
    static int count = 0;// will get memory only once and retain its value

    CounterWithStatic() {
        count++;
    }

    private void showCount() {
        System.out.println(count);
    }

    public static void main(String args[]) {

        new CounterWithStatic().showCount();
        new CounterWithStatic().showCount();
        new CounterWithStatic().showCount();

    }

}
```

Console | Markers | Properties | Servers | Data Source Explorer | Snippe

```
<terminated> CounterWithoutStatic [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\jav
1
1
1
```

Console | Markers | Properties | Servers | Data Source Explorer

```
<terminated> CounterWithStatic [Java Application] C:\Program Files\Java\jdk1.7.0_79\b
1
2
3
```

Without Static Variable: Variable will **get memory when object is created**.

With Static Variable: Variable will **get memory when class is loaded and retain its value**.

Java static method

- If we apply static keyword with any method, it is known as static method.
  - A ***static method belongs to the class rather than object of a class.***
  - A static method can be invoked without the need for creating an instance of a class.
  - static method can access static data member and can change the value of it.

Example:

ChangeStaticValue.java

```java
public class ChangeStaticValue {
    int rollno;
    String name;
    static String college = "TRICHANDRA";

    static void change() {
        college = "ASCOL";
    }

    ChangeStaticValue(int r, String n) {
        rollno = r;
        name = n;
    }

    void display() {
        System.out.println(rollno + " " + name + " " + college + "\n");
    }

    public static void main(String args[]) {

        ChangeStaticValue s1 = new ChangeStaticValue(111, "Amrit");
        s1.display();

        ChangeStaticValue.change(); //college:ASCOL
        ChangeStaticValue s2 = new ChangeStaticValue(222, "Bindu");
        ChangeStaticValue s3 = new ChangeStaticValue(333, "Champ");

        s1.display();
        s2.display();
        s3.display();
    }

}
```

Console | Markers | Properties | Servers | Data Source Explorer | Snippets

```
<terminated> ChangeStaticValue [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Sep 1, 2015, 9:33:39 AM)
111 Amrit TRICHANDRA

111 Amrit ASCOL

222 Bindu ASCOL

333 Champ ASCOL
```

S1

S2

S3

Some Restrictions for static method

1. The static method **can not use non static data member or call non-static method directly**.

2. this and **super** cannot be used in static context.

Java static block

- Is used to initialize the static data member.
- It is executed before main method at the time of class loading.

StaticBlock.java

```
public class StaticBlock {
    static {
        System.out.println("static block is invoked");
    }

    public static void main(String args[]) {
        System.out.println("Hello from main");
    }

}
```

1

2

Console | Markers | Properties | Servers | Data Source Explorer | Snipp

```
<terminated> StaticBlock [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (S
static block is invoked
Hello from main
```

Q) Can we execute a program without main() method?

- Yes, one of the way is static block but in previous version of JDK not in JDK 1.7.

```
class A3{
    static{
            System.out.println("static block is invoked");
             System.exit(0);
             }
}
```

Output: static block is invoked (if not JDK7)

Output:Error: Main method not found in class A3, please define the main method as: public static void main(String[] args)

Encapsulation

- Encapsulation in Java is a mechanism of **wrapping the data (variables) and code acting on the data (methods) together as a single unit** also known as

  data binding

- In encapsulation **the variables of a class will be hidden from other classes**, and can be accessed only through the methods of their current class, therefore it is **also known as data hiding**.

Encapsulation

- To achieve encapsulation in Java:
  - **Declare the variables** of a class as **private**.
  - **Provide public setter and getter methods** to modify and view the variables values.

Example:

Inheritance in Java

- Inheritance in java is a mechanism in which one object **acquires all the properties and behaviors of parent object**.

- The idea behind inheritance in java is that you can create new classes that are built upon existing classes.

- When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.

Inheritance in Java

- Inheritance represents the **IS-A relationship,** also known as ***parent-child relationship.***

Why use inheritance in java ?

- For **Method Overriding** (so runtime polymorphism can be achieved).

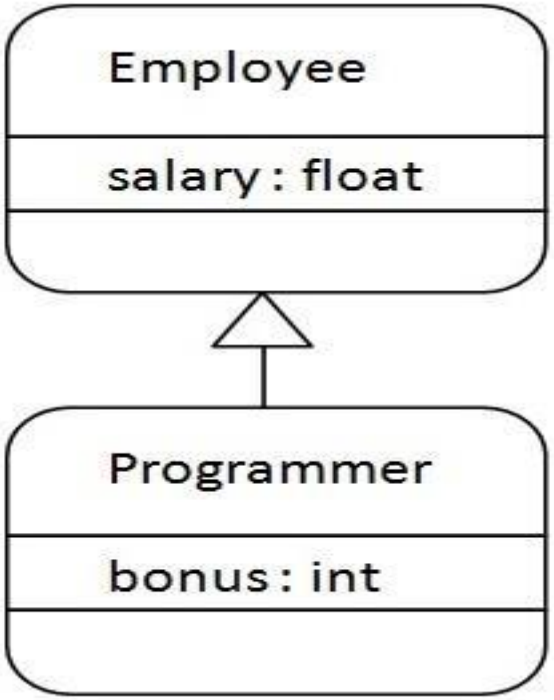
- For **Code Reusability**.

Syntax of Java Inheritance

```
class  <Subclass-name> extends <Superclass-name> {
    //methods and fields
}
```

- The **extends keyword** indicates that you are making a new class that derives from an existing class**.**
- In the terminology of Java, a class **that is inherited is called a super class**. The new class is called a **subclass**.

Understanding the simple example of inheritance



	Class	Package	Subclass	World
public	y	y	y	y
protected	y	y	y	n
no modifier	y	y	n	n
private	y	n	n	n

y: accessible
n: not accessible

- As displayed in the above figure, **Programmer is the subclass** and **Employee** is the **super-class**.
- Relationship between two classes is **Programmer IS-A Employee.**
- It means that Programmer is a type of Employee**.**

Example:

```
1. class Employee{
2. float salary=40000;
3. }

4. class Programmer extends Employee{
5.        int bonus=10000;
6.        public static void main(String args[]){
7.        Programmer p=new Programmer();
8.        System.out.println("Programmer salary is:"+p.salary);
9. System.out.println("Bonus of Programmer is:"+p.bonus);
10. }
11. }
```

Output:

Programmer salary is: 40000.0
Bonus of programmer is:10000

In the above example, Programmer object can access the field of own class as well as of Employee class **i.e. code reusability**.

Example:

ect Explorer

- JavaStud [JavaStud master]
  - src
    - control
    - oop
    - oop.inheritence
      - Car.java
      - Vehicle.java
    - org
    - org.constr
    - org.course
    - org.multiclassex
    - org.pkgpvt
    - org.stud
    - org.typeconversion
    - str
  - JRE System Library [jdk1.7.0_
  - README.md
- nb-h-c
- nb-h-pm-w
- PackageExample

Car.java

```java
package oop.inheritence;

public class Car extends Vehicle {
    private int cc;
    private int gears;

    public void attributesCar() {
        // The subclass refers to the members of the superclass

1       // System.out.println("Color of Car : " + color); //ERROR:private
        // field:color

2       System.out.println("Speed of Car : " + super.speed); // super.speed or
                                                              // speed

        System.out.println("Size of Car : " + size);

        System.out.println("CC of Car : " + cc);
        System.out.println("No of gears of Car : " + gears);

3       super.attributes(); // WE CAN USE Super in any non static method.
    }

    public static void main(String[] args) {

        Car c1 = new Car();

4       // c1.color = "Blue"; //ERROR: private field:color

        c1.speed = 200;
        c1.size = 22;
        c1.cc = 1000;
        c1.gears = 5;

5       c1.attributes();

        // super.attributes(); //ERROR: Cannot use super in a static context

6       c1.attributesCar();

    }
}
```

Vehicle.java

```java
package oop.inheritence;

public class Vehicle {

    private String color;
    public int speed;
    protected int size;

    protected void attributes() { //public or protected.
        System.out.println("Color : " + color);
        System.out.println("Speed : " + speed);
        System.out.println("Size : " + size);
    }

}
```

Console | Markers | Properties | Servers

```
<terminated> Car [Java Application] C:\Program Files\Java\
Color : null
Speed : 200
Size : 22
Speed of Car : 200
Size of Car : 22
CC of Car : 1000
No of gears of Car : 5
Color : null
Speed : 200
Size : 22
```

Vehicle

Car

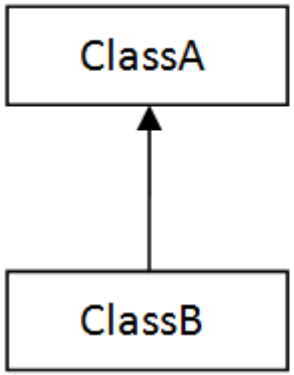
Types of Inheritance in java

- On the basis of class, there can be three types of inheritance in java:
  - **single**,
  - **multilevel** and
  - **hierarchical**.

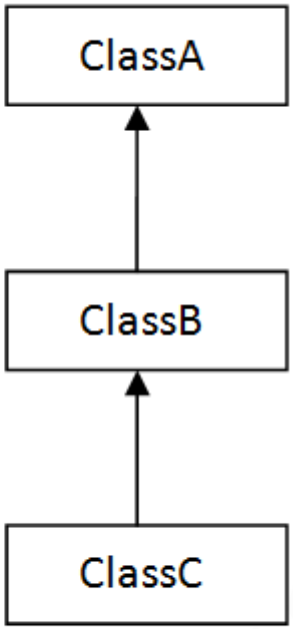
➢ **Note**: Multiple inheritance is not supported in java through class.

➢ In java programming, **multiple** and **hybrid** inheritance is **supported through interface only**. We will learn about interfaces later.

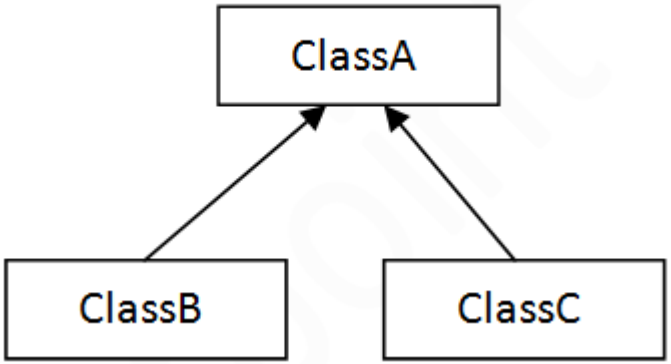
Types of Inheritance in java



1) Single



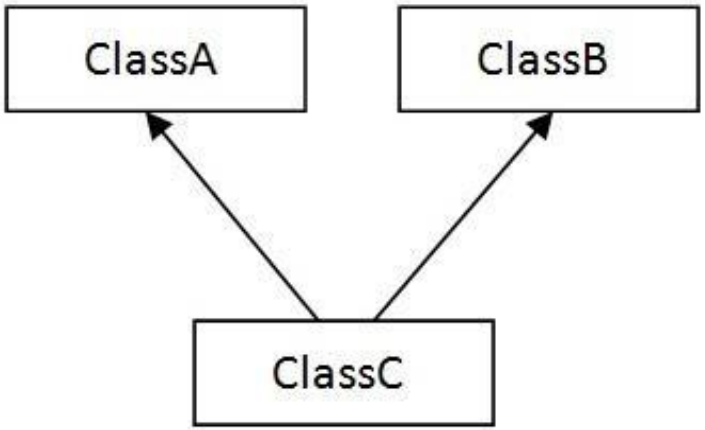
2) Multilevel



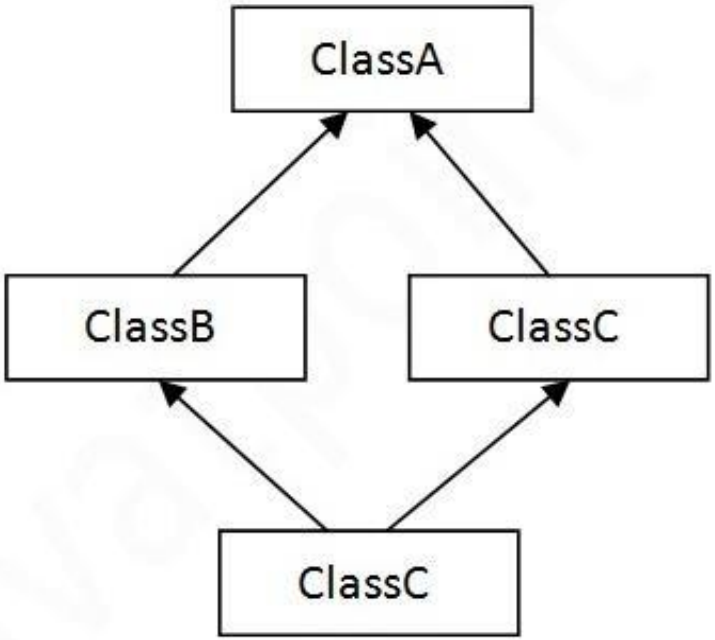
3) Hierarchical

When a class extends multiple classes i.e. known as multiple inheritance.

For Example:



4) Multiple



5) Hybrid

Q) Why multiple inheritance is not supported in java?

- To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

❖Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. If A and B classes have same method and you call it from child class object, there will be ambiguity to call method of A or B class.

❖Since compile time errors are better than runtime errors, java renders compile time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error now.

Example:

```
class A{
void msg(){System.out.println("Hello");}
}

        class B{
        void msg(){System.out.println("Welcome");}
        }
class C extends A,B{//suppose if it were

Public Static void main(String args[]){
        C obj=new C();
        obj.msg();//Now which msg() method would be invoked?
                }
        }
```

Compile Time Error

Method Overloading in Java

- If a class have multiple methods by **same name but different parameters**, it is known as **Method Overloading.**

- If we have to perform only one operation, having same name of the methods increases the readability of the program.

- Suppose you have to perform **addition of the given numbers** but **there can be any number of arguments**, if you write the method such as **a(int,int) for two parameters**, and **b(int,int,int) for three parameters**
- then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs. So, we perform method overloading to figure out the program quickly.

Different ways to overload the method

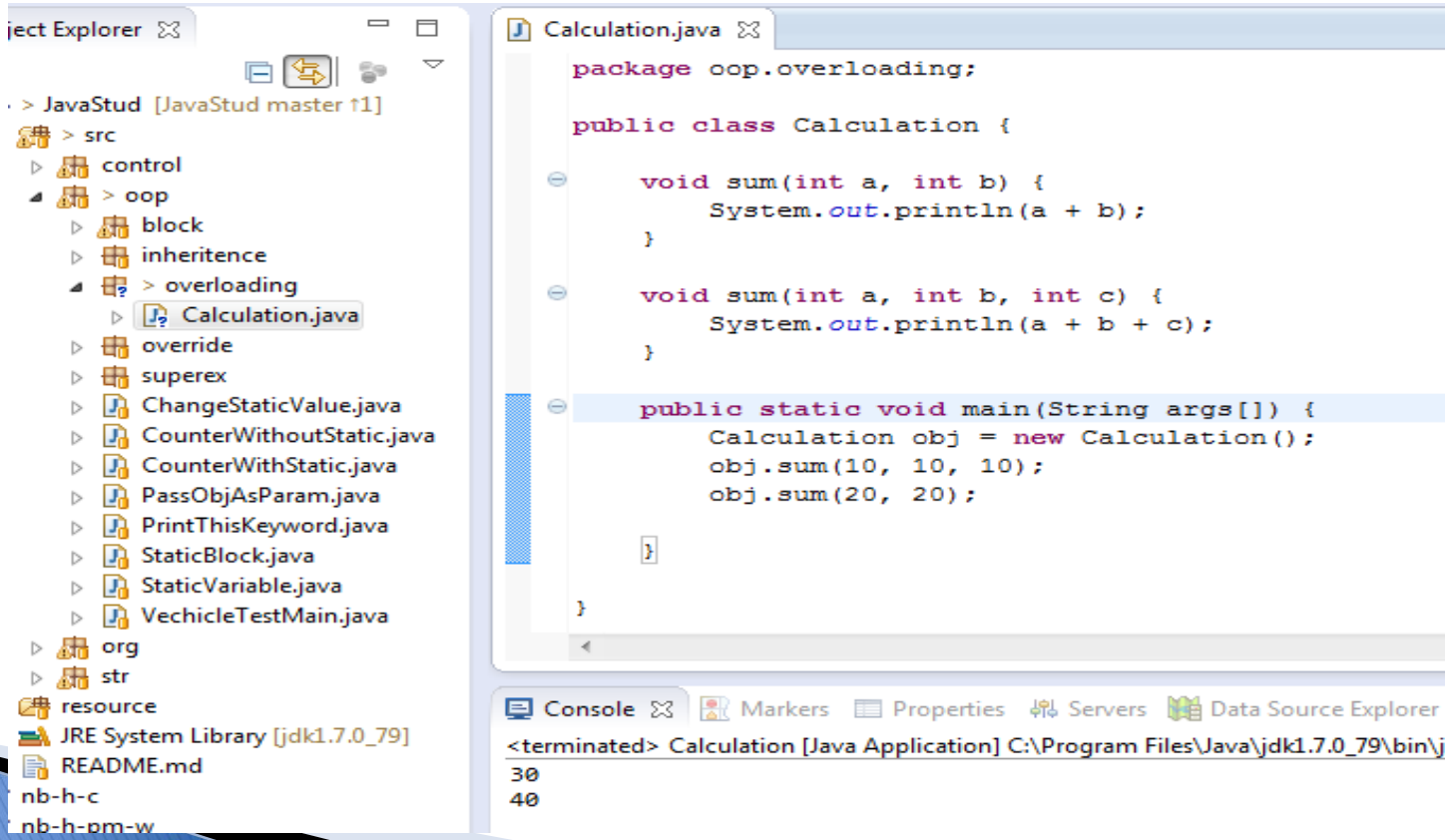
1. By changing **number of arguments**
2. By changing the **data type**

- **Method Overloading is not possible** by changing the return type **of the method.**

Example :

- **Method Overloading by changing the no. of arguments**

In this example, we have created two overloaded methods, first sum method performs addition of two numbers and second sum method performs addition of three numbers.



Example:

- **Method Overloading by changing data type of argument**

In this example, we have created two overloaded methods that **differs in data type**. The first **sum method** receives two **integer arguments** and second sum method receives **two double arguments**.