



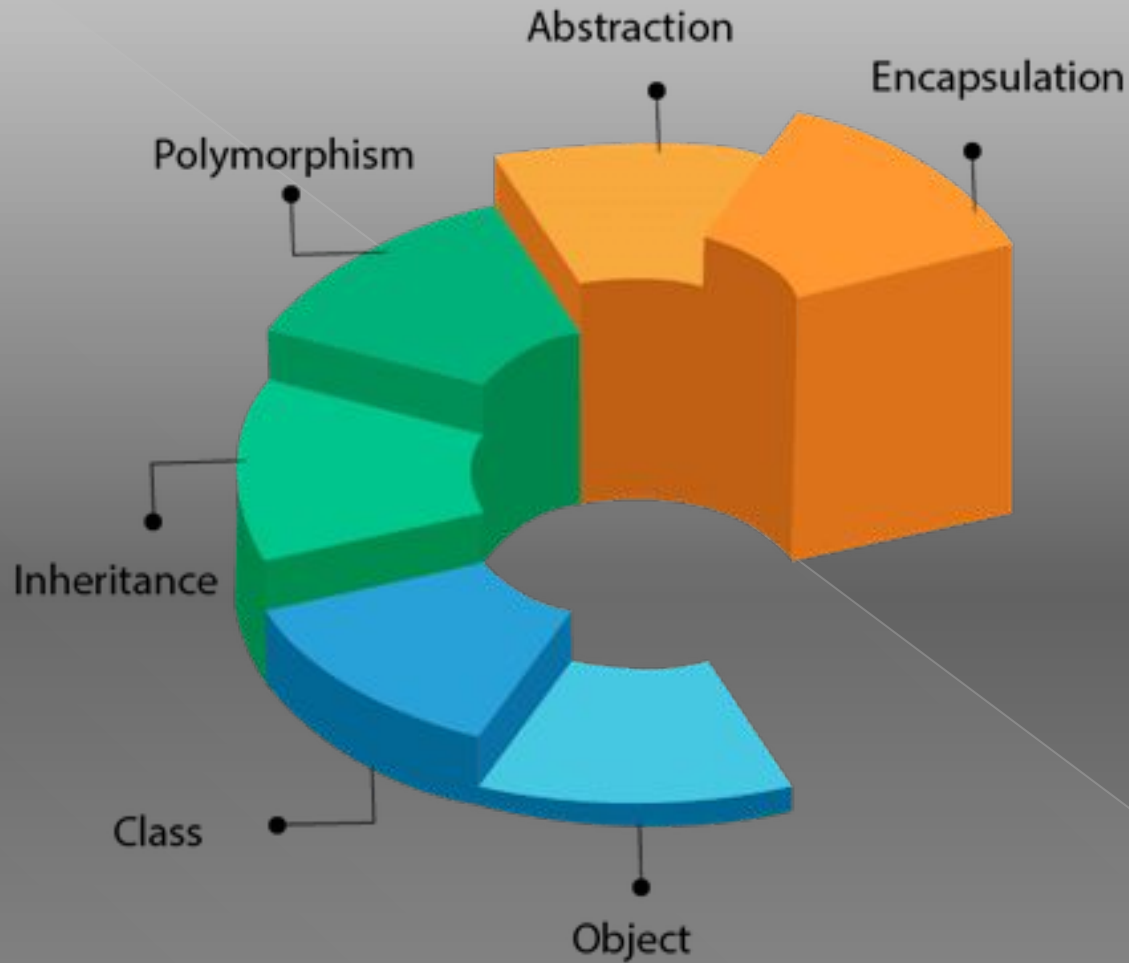
# DEPARTMENT OF BCA

**SUBJECT: JAVA PROGRAMMING**  
**SEMESTER: 2<sup>ND</sup> Semester BCA**

# Module 2

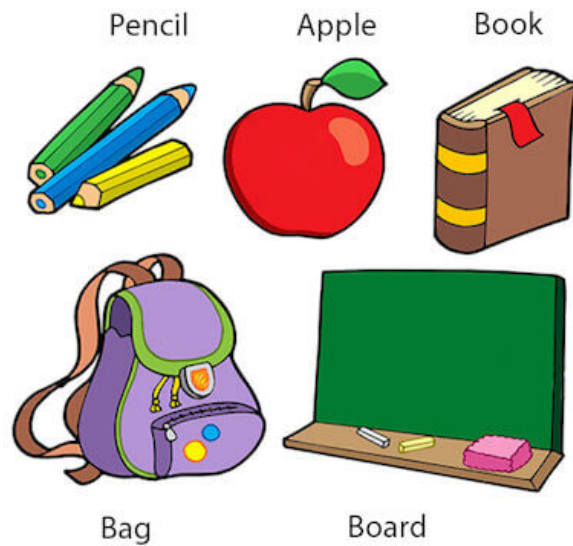
## CLASS FUNDAMENTALS

# OOPs (Object-Oriented Programming System)





### Objects: Real World Examples



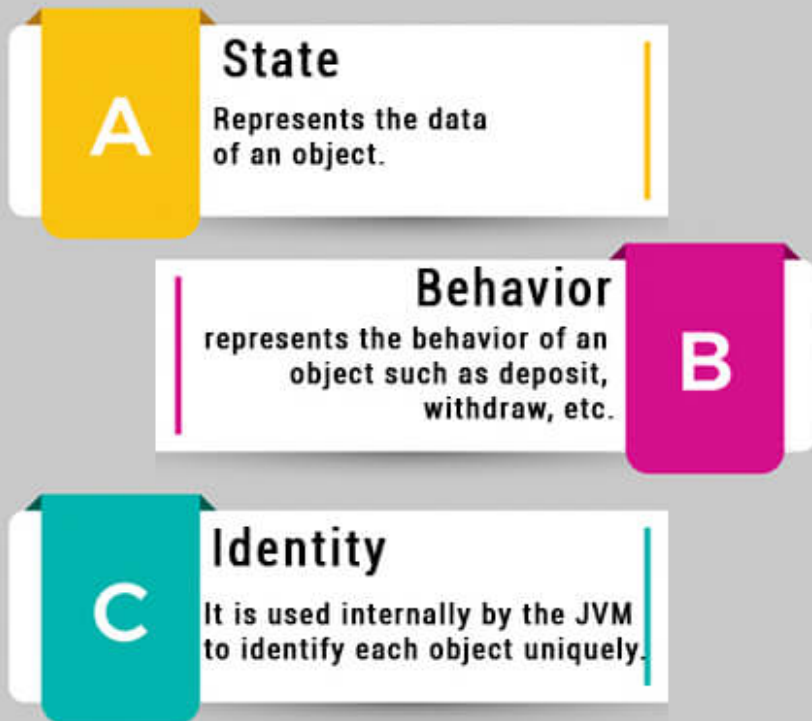
# Objects:

- An entity that has **state** and **behavior** is known as an object.

**e.g.,** chair, bike, marker, pen, table, car, etc.

- It can be physical or logical (tangible and intangible).
- The example of an intangible object is the banking system

## Characteristics of Object



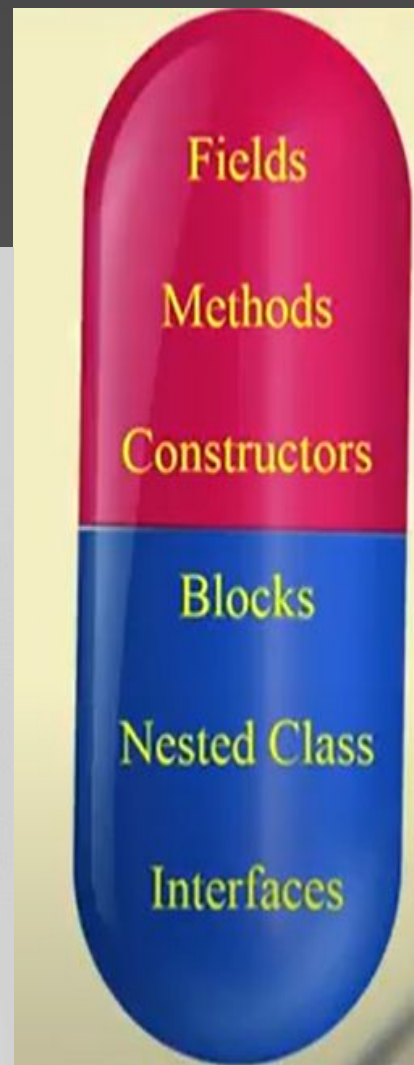
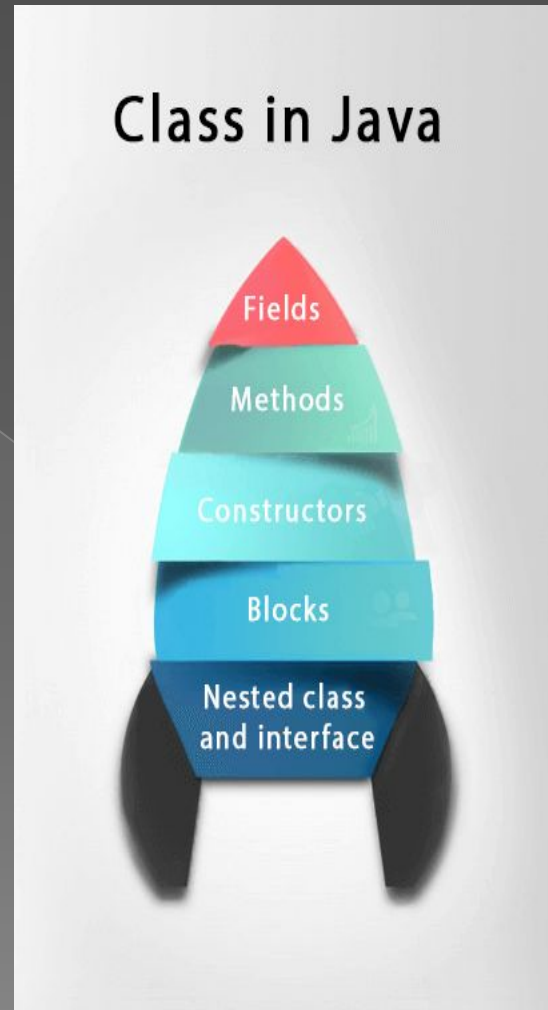
# CLASS CONCEPT IN JAVA

- The object is at the core of Java programming.
- Java provides the concept of class to build objects.
- A class defines the shape and working of an object.
- The concept of class is the logical construct upon which the entire Java language is built.

# What is a CLASS?

- ◉ A class is a group of objects, which have common properties.
- ◉ It is a template / blueprint from which objects are created.
- ◉ It is a logical entity.

A class in java can contain:



# General Structure of a CLASS

- It should start with the uppercase letter.
- It should be a noun such as Color, Button, System, Thread, etc.
- Use appropriate words, instead of acronyms.
- Example: -**

```
public class Employee
{
//code snippet
}
```

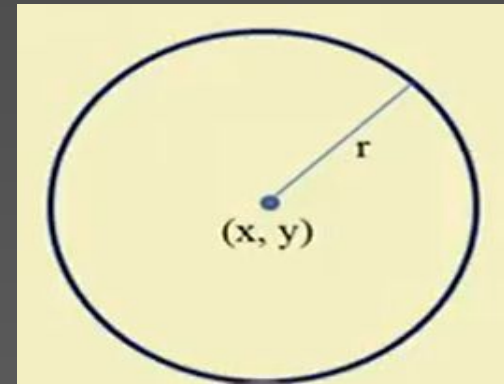
```
class <class-name>{
    <type> <variable 1>;
    <type> <variable 2>;
    <type> <variable 3>;
    ...
    <type> <variable n>;

    <type> <method 1>(<parameter-list 1>){
        // Body of the method 1
    }
    <type> <method 1>(<parameter-list 2>){
        // Body of the method 2
    }
    ...
    <type> <method 1>(<parameter-list m>){
        // Body of the method m
    }
}
```



# Java class- an example

```
class circle {  
    double x, y; // The coordinates of the center  
    double r;    // The radius  
}
```



## Adding methods to a circle class

```
class circle {  
    double x,y; // The coordinates of the center  
    double r;    // The radius  
  
    // Method that returns circumference  
    double circumference(){  
        return 2*3.14159*r;  
    }  
    // Method that returns area  
    double area(){  
        return (22/7)*r*r;  
    }  
}
```

# Declaring an object of type **circle** class

```
// A program that uses the circle class
// Call this file circledemo1.java
class Circle {
    double x,y; // The coordinates of the center
    double r;    // The radius

    // Method that returns circumference
    double circumference(){
        return 2*3.14159*r;
    }
    // Method that returns area
    double area(){
        return (22/7)*r*r;
    }
}
```

```
//The following class declare an object of type Circle
class CircleDemo1 {
    public static void main(String args[]){
        Circle c = new Circle();
        c.x = 0.0;
        c.y = 0.0;
        c.r = 5.0;
        System.out.println("Circumference" + c.circumference());
        System.out.println("Area" + c.area());
    }
}
```

JAIN UNIVERSITY BCA

# Ways to initialize object

There are 3 ways to initialize object in Java.

1. By reference variable
2. By method
3. By constructor

# Multiple objects declaring

**eg:** Rectangle r1=**new** Rectangle(),r2=**new** Rectangle();

//the following class declares multiple objects of type Circle

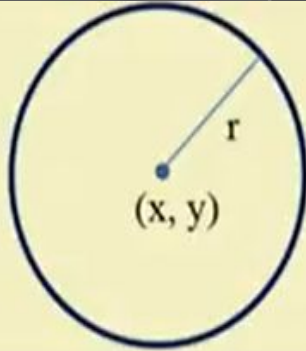
```
class CircleDemo
{
    public static void main(String args[])
    {
        Circle c1 = new Circle();
        Circle c2 = new Circle();
        // Initialize the circles
        c1.x = 3.0;
        c1.y = 4.0;
        c1.r = 5.0;
        c2.x = -4.0;
        c2.y = -8.0;
        c2.r = 10.0;

        System.out.println("Circumference Circle 1" + c1.circumference());
        System.out.println("Area Circle 1" + c1.area());
        System.out.println("Circumference Circle 2" + c2.circumference());
        System.out.println("Area Circle 2" + c2.area());
    }
}
```

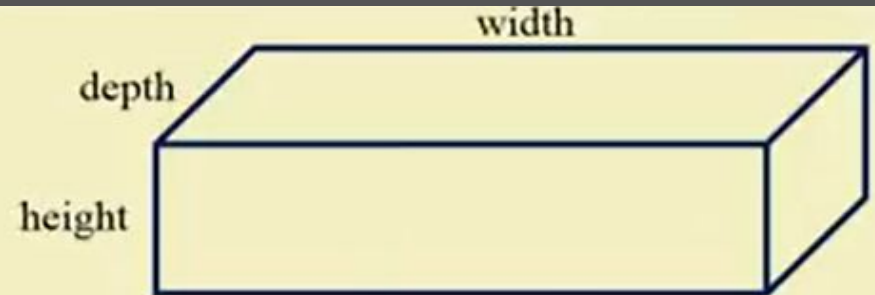
```
// A program that uses the circle class
// Call this file circledemo1.java
class Circle {
    double x,y; // The coordinates of the center
    double r;    // The radius

    // Method that returns circumference
    double circumference(){
        return 2*3.14159*r;
    }
    // Method that returns area
    double area(){
        return (22/7)*r*r;
    }
}
```

# Multiple classes declaration in a program



```
class Circle {  
    double x,y;  
    double r;  
    double circumference(){  
        return 2*3.14159*r;  
    }  
    double area(){  
        return (22/7)*r*r;  
    }  
}
```



```
class Box{  
    double width;  
    double height;  
    double depth;  
    double area(){  
        double a;  
        a = (width*height + height*depth + width*depth) * 2;  
        return a;  
    }  
    double volume(){  
        double v;  
        v = width*height*depth;  
        return v;  
    }  
}
```

# Main class containing multiple classes

```
// Declaring objects of type Circle and Box
class MulticlassDemo {
    public static void main(String args[]){
        Circle c = new Circle();
        Box b = new Box();
        // Initialize the circles
        c.x = 3.0; c.y = 4.0; c.r = 5.0;
        b.width = 3.0; b.height = 4.0; b.depth = 5.0;
        System.out.println("Circumference Circle" + c.circumference());
        System.out.println("Area Circle" + c.area());
        System.out.println("Area of Box" + b.area());
        System.out.println("Volume of Box" + b.volume());
    }
}
// Save this file as MulticlassDemo.java
```

# Important notes

1. There should be a class which contains a method `main()`. This class is called `main class`.
2. There should be only `one` main class.
3. The name of the program file should be same as the name of the main class followed by `.java` as an extension.
4. If there is no main class, then there should be compilation error.

**NOTE: Java program can not exist without main class**



# About main() Method

- Java program starts its execution from a method belongs to main class only.

```
1  import java.lang.*;
2
3  class Calculator
4  {
5      double i;
6      double x = Math.sqrt(i);
7  }
8
9  class MainMethodDemo
10 {
11     public static void main(String args[])
12     {
13         Calculator c = new Calculator();
14         c.i = 20;
15         System.out.println("Square root of " + c.i + "is" + c.x);
16     }
17 }
```

access modifier

return type

String class

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

keyword

method name

array of string objects



# About main() method...

Let us examine each statement step-by-step.

Import  
Statements

```
import java.lang.*;
```

Declaration  
of class

```
class Calculator{  
    double i;  
    double x = Math.sqrt(i);  
}
```

Declaration of  
main class

```
class Example{  
    public static void main(String args[]){  
        Calculator a = new Calculator();  
        a.i = 20;  
        System.out.println("Square root of "+a.i+" is "+a.x);  
    }  
}
```

# Significance of main class

- ⦿ Java program starts its execution from a method belongs to a class only.
- ⦿ The main() method is the starting point of the execution of the main thread.
- ⦿ If there are multiple classes, then the ambiguity is resolved by incorporating a main() method into only one special class called main class.
- ⦿ The name of the java program should be named after this class so that java interpreter unanimously choose that class to start its execution.

# Significance of Main() method continued...

- ◎ **public:** It is an access specifier. We should use a public keyword before the main() method so that JVM can identify the execution point of the program. If we use private, protected, and default before the main() method, it will not be visible to JVM.
- ◎ **static:** You can make a method static by using the keyword static. We should call the main() method without creating an object. Static methods are the method which invokes without creating the objects, so we do not need any object to call the main() method.
- ◎ **void:** In Java, every method has the return type. Void keyword acknowledges the compiler that main() method does not return any value.
- ◎ **main():** It is a default signature which is predefined in the JVM. It is called by JVM to execute a program line by line and end the execution after completion of this method. We can also overload the main() method.

# Significance of Main() method continued...

- ◉ **String args[]:** String is a class defined in java.lang API.
- ◉ The main() method also accepts some data from the user. It accepts a group of strings, which is called a string array. It is used to hold the command line arguments in the form of string values.
- ◉ Here, args[] is the array name, and it is of String type. It means that it can store a group of string.
- ◉ Remember, this array can also store a group of numbers but in the form of string only.
- ◉ Values passed to the main() method is called arguments. These arguments are stored into args[] array, so the name args[] is generally used for it.

# Significance of Main() method continued...

- What happens if the main() method is written without String args[]?

## ANSWER:

The program will compile, but not run, because JVM will not recognize the main() method. Remember JVM always looks for the main() method with a string type array as a parameter.

## Execution Process

- First, JVM executes the static block, then it executes static methods, and then it creates the object needed by the program. Finally, it executes the instance methods. JVM executes a static block on the highest priority basis. It means JVM first goes to static block even before it looks for the main() method in the program.

# Output from java program?

```
System.out.println("Square root of "+a.i+" is "+a.x);
```

- ⦿ **System** is a final class from the **java.lang** package.
- ⦿ **Out** is a class variable of type **PrintStream** declared in the **System** class.
- ⦿ **Println** is a method of the **PrintStream** class.
- ⦿ **a.i** and **a.x** represents the names of variables to be printed.
- ⦿ **+** is a concatenation operator, it is used to concatenate the string values.

# Variations of print statements

- ◉ **Println():** method in Java is also used to display a text on the console. This text is passed as the parameter to this method in the form of String. This method prints the text on the console and the cursor remains at the start of the next line at the console. The next printing takes place from next line
- ◉ **Print():** method in Java is used to display a text on the console. This text is passed as the parameter to this method in the form of String. This method prints the text on the console and the cursor remains at the end of the text at the console. The next printing takes place from just here.
- ◉ **Printf():** is used when you want to format your string. This will clean up any concatenation. For a simple example -> ("Hello, " + username + "! How are you?") could easily be cleaned up a bit by using ("Hello, %s! How are you?", username).

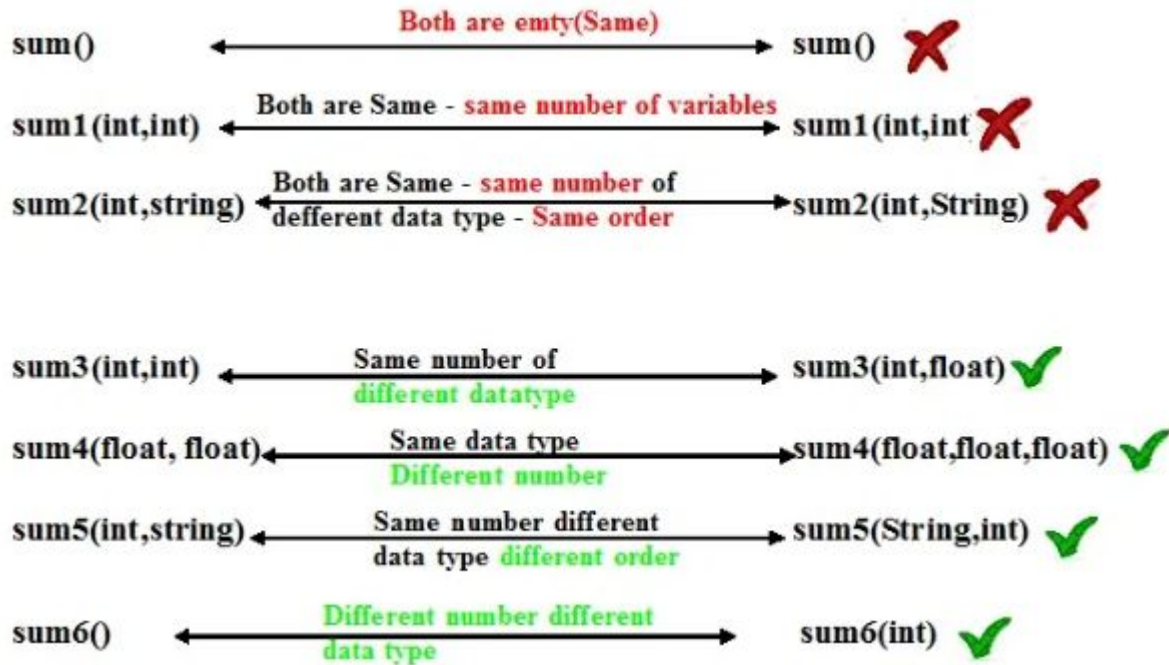
# Methods with Parameters

```
class circle {  
    double x,y;  
    double r;  
    double circumference(){  
        return 2*3.14159*r;  
    }  
    double area(){  
        return (22/7)*r*r;  
    }  
    void setCircle(double a, double b, double c){  
        x = a; // Set center x-coordinate  
        y = b; // Set center y-coordinate  
        r = c; // Set radius  
    }  
}
```

```
class CircleDemo3 {  
    public static void main(String args[]){  
        Circle c1 = new Circle();  
        Circle c2 = new Circle();  
        // Initialize the circles  
        c1.setCircle(3.0,4.0,5.0);  
        c2.setCircle(-4.0,8.0,10.0);  
        System.out.println("Circumference Circle 1" + c1.circumference());  
        System.out.println("Area of circle 1" + c1.area());  
        System.out.println("Circumference Circle 2" + c2.circumference());  
        System.out.println("Area of circle 2" + c2.area());  
    }  
}
```



# Method overloading



Image



Document

print()

# Method overloading

- ⦿ It is legal for a class to have two or more methods with the same name.
- ⦿ However, Java has to be able to uniquely associate the invocation of a method with its definition relying on the number and types of arguments.
- ⦿ Therefore the same-named methods must be distinguished
  - > by the number of arguments.
  - > by the types of arguments.
  - > Order of the arguments.
- ⦿ Overloading and inheritance are two ways to implement polymorphism

# Real time use of method overloading...

- It also gives you more **flexibility** while writing API, for example, you can look at the `println()` method. The job of this method is to print something and add a new line. Now, you can print any kind of data type e.g. `int`, `short`, `long`, `float`, `String` or `Object`.
- This is where overloading makes your job easy, a [`println\(\)`](#) is to print and method parameters will describe what to print. If overloading was not available, you would end up with clumsy APIs like `printString()`, `printObject()`, `printInteger()` etc. So, method overloading provides you flexibility while writing API.
- Searching for a vehicle either by vehicle number or owner name.

# Method overloading in real time

**Overloading:** Consider the basic human function of speaking!

Assume, you are supposed just perform the function of talking. Say, you have to tell the story of your day, to a total stranger. Your function will get over pretty quickly. Say, now you are telling the same to your beloved. You will go through more details as compared to the previous one. What has happened here, is, you have performed the same function, but based on the parameter, stranger/beloved, your way of implementing the function changed!

```
1  class You
2  {
3      void talk(Stranger obj)
4      {
5          sysout("Hi, my day was great!");
6      }
7      void talk(Beloved obj)
8      {
9          sysout("Hi, my day was great! You won't believe what happened today! Bl
10     }
11 }
```

# Using objects as parameters

- Consider the student details of 3<sup>rd</sup> semester. There are 5 different section.
- In these five sections we are supposed to find the toppers from each section.
- Example program to be written here.....  
.....  
.....  
.....

```
class Rectangle {  
    int length;  
    int width;  
  
    Rectangle(int l, int b) {  
        length = l;  
        width = b;  
    }  
  
    void area(Rectangle r1) {  
        int areaOfRectangle = r1.length * r1.width;  
        System.out.println("Area of Rectangle : "  
                             + areaOfRectangle);  
    }  
}  
  
class RectangleDemo {  
    public static void main(String args[]) {  
        Rectangle r1 = new Rectangle(10, 20);  
        r1.area(r1);  
    }  
}
```

# Using objects as parameters (programs to practice)

- ⦿ Addition of two complex numbers.
- ⦿ Addition of two different time periods.
- ⦿ Addition of two different weights given.
- ⦿ Calculate the fine for a particular vehicle for different rule breaking.

# Returning object

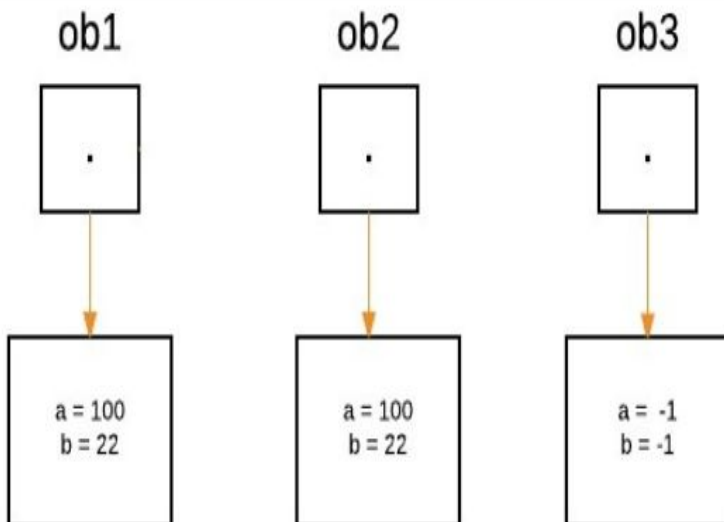
- ◉ When we pass a primitive type to a method, it is passed by value. But when we pass an object to a method, the situation changes dramatically, because objects are passed by what is effectively call-by-reference.
- ◉ While creating a variable of a class type, we only create a reference to an object. Thus, when we pass this reference to a method, the parameter that receives it will refer to the same object as that referred to by the argument.
- ◉ This effectively means that objects act as if they are passed to methods by use of call-by-reference.
- ◉ Changes to the object inside the method do reflect in the object used as an argument.

# Returning object – an example

## ◎ ..\Programs\Module2\ObjectPassDemo1.java

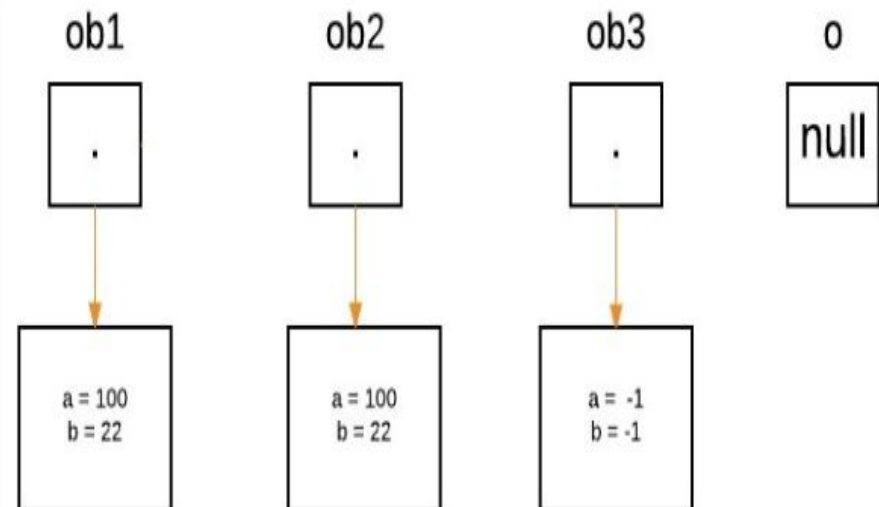
- Three objects 'ob1', 'ob2' and 'ob3' are created:

```
ObjectPassDemo ob1 = new ObjectPassDemo(100, 22);  
ObjectPassDemo ob2 = new ObjectPassDemo(100, 22);  
ObjectPassDemo ob3 = new ObjectPassDemo(-1, -1);
```



- From the method side, a reference of type Foo with a name a is declared and it's initially assigned to null.

```
boolean equalTo(ObjectPassDemo o);
```

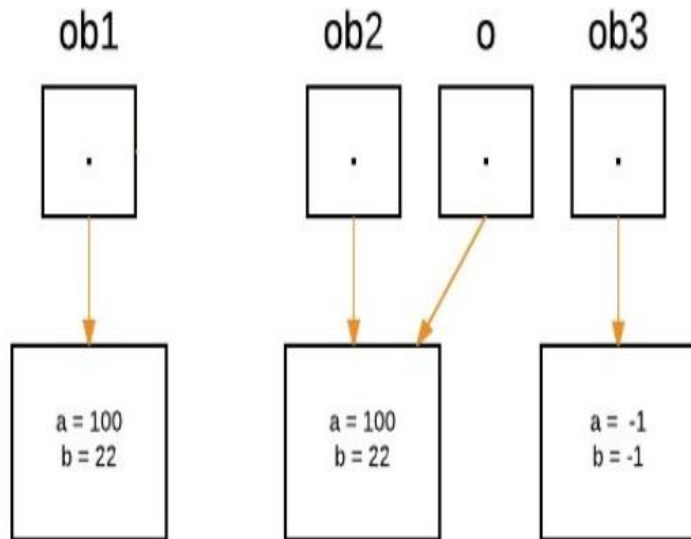




# Returning object – an example

- As we call the method `equalTo`, the reference 'o' will be assigned to the object which is passed as an argument, i.e. 'o' will refer to 'ob2' as following statement execute.

```
System.out.println("ob1 == ob2: " + ob1.equalTo(ob2));
```

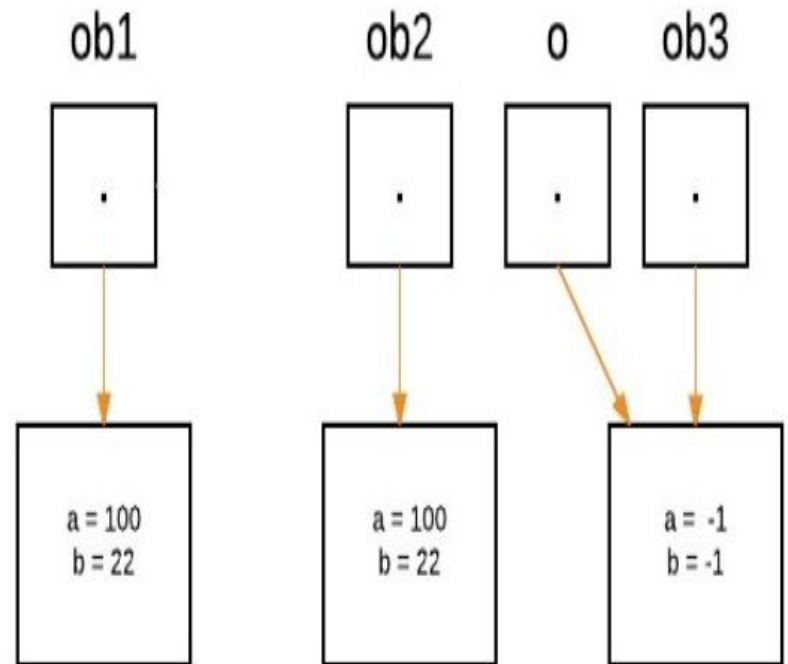


- Now as we can see, `equalTo` method is called on 'ob1', and 'o' is referring to 'ob2'. Since values of 'a' and 'b' are same for both the references, so `if(condition)` is true, so boolean true will be return.

```
if(o.a == a && o.b == b)
```

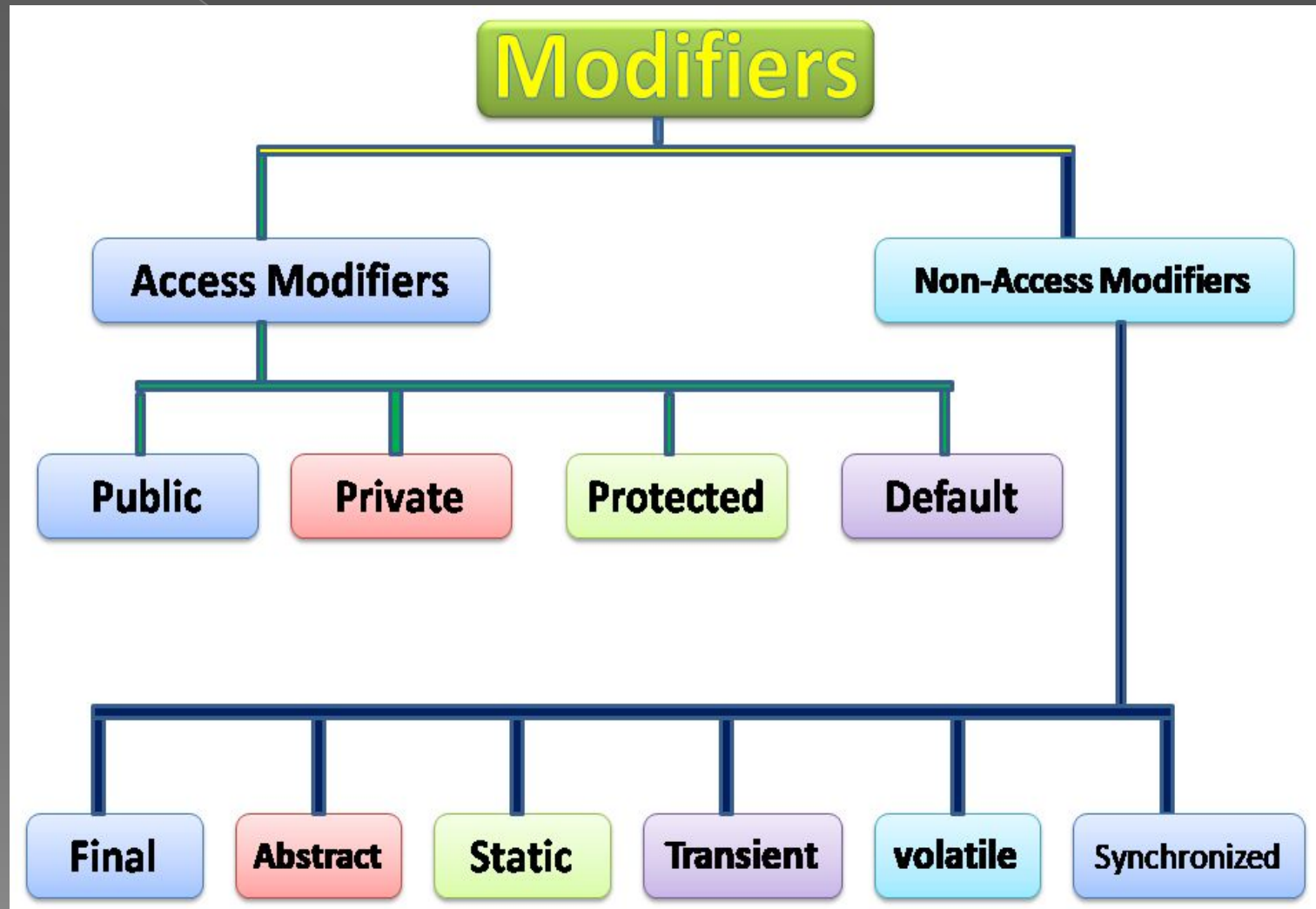
- Again 'o' will reassign to 'ob3' as the following statement execute.

```
System.out.println("ob1 == ob3: " + ob1.equalTo(ob3));
```



- Now as we can see, `equalTo` method is called on 'ob1', and 'o' is referring to 'ob3'. Since values of 'a' and 'b' are not same for both the references, so `if(condition)` is false, so else block will execute and false will be return.

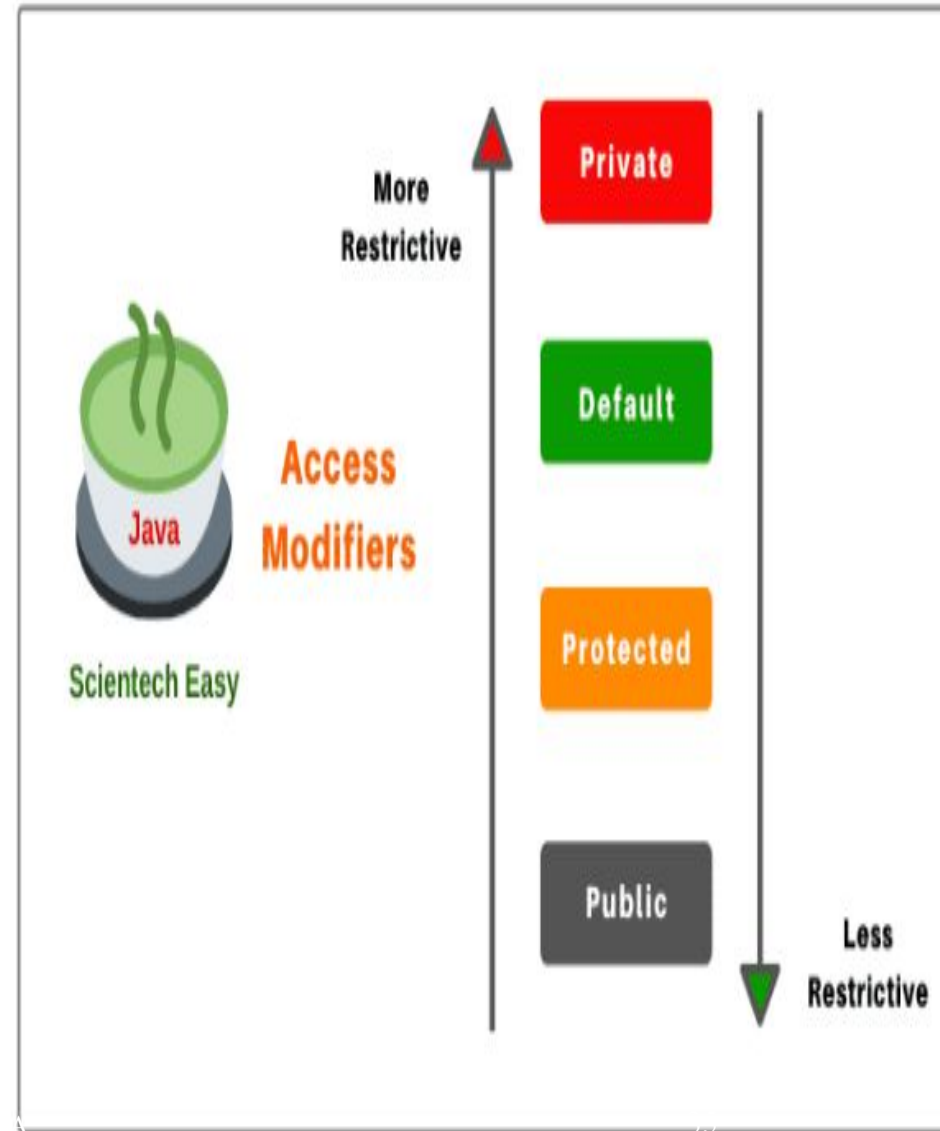
# Access control



# Access Specifiers

- access specifiers are those modifiers that are used to restrict the visibility / accessibility of classes, fields, methods, or constructors.
- The access specifiers are also known as visibility modifiers

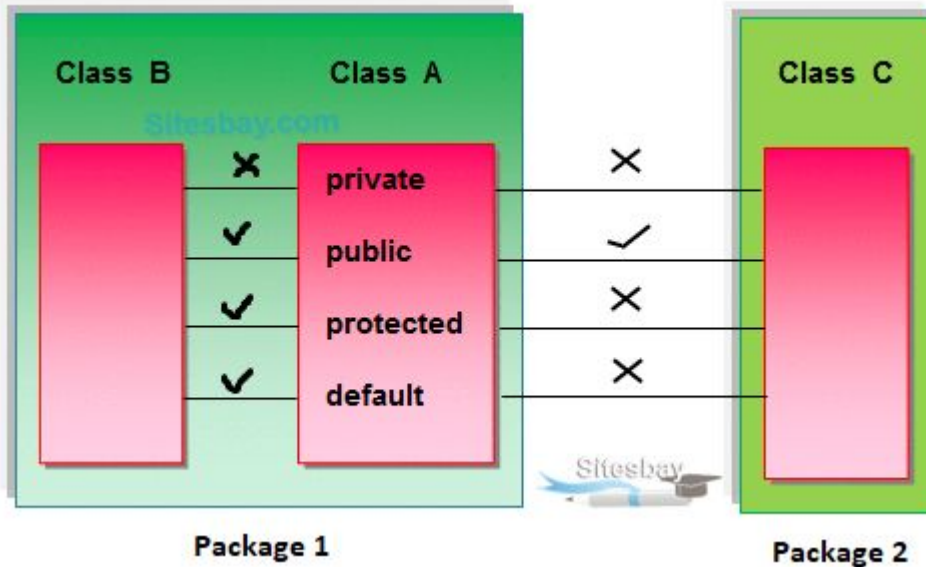
Java supports four types of access modifiers.



# Access modifiers

- **Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
- **Default:** The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
- **Protected:** The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
- **Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

# Lets understand Java Access Modifiers with a simple table

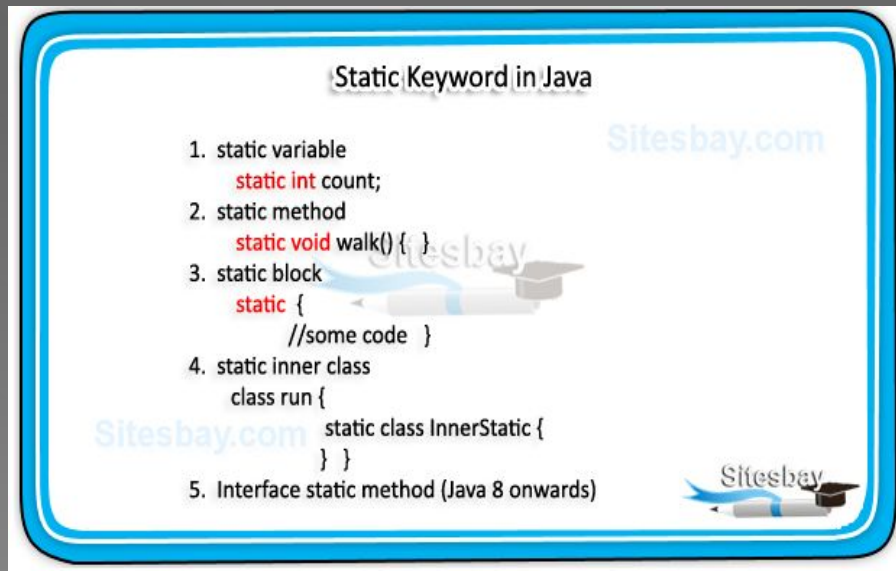


Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

# Non Access Modifier - Static keyword in java

- static is a non-access modifier in Java which is applicable for the following:

To create a static member (block, variable, method, nested class), precede its declaration with the keyword static. When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object.



# Static variable( also known as class variables)

If any variable we declared as static is known as static variable.

- A Static variable is used to fulfill the common requirement. For Example the Company name of employees, the college name of students, etc. The Name of the college is common for all students.
- The static variable allocates memory only once in the class area at the time of class loading.

## Advantage of static variable

- Using a static variable we make our program memory efficient (i.e it saves memory).



# Instance variable versus Class variable

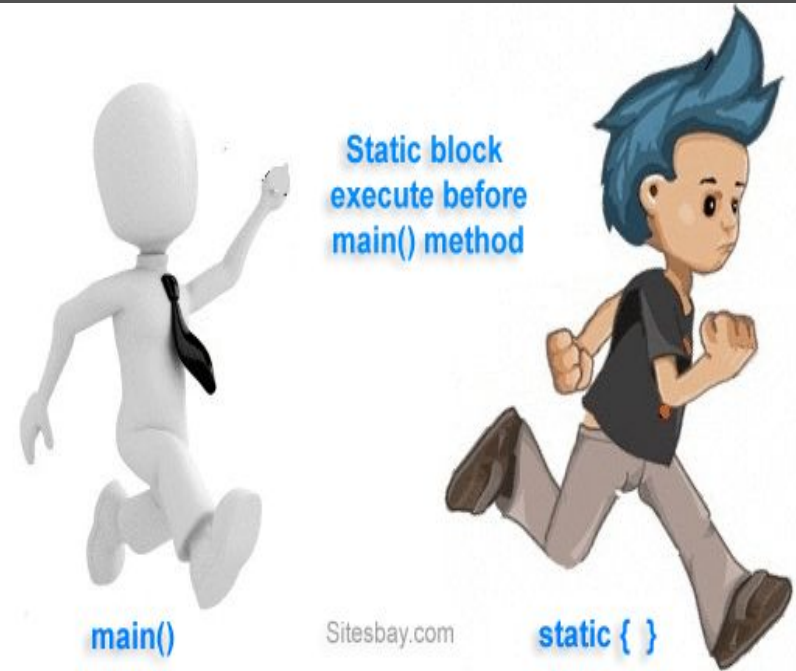
- ⦿ Java does not allow global variables.
- ⦿ Every variable in Java must be declared inside a class.
- ⦿ The keyword static is used to make a variable just like global variable.
- ⦿ A variable declared with static keyword is called class variable.
- ⦿ It acts like a global variable, that is only one copy of the variable associated with the class.

NOTE: one copy of the variable regardless of the number of instances of the class.



# Static Block in Java

- **Static block** is a set of statements, which will be executed by the JVM before execution of main method ( Class Loading).
- In a class we can take any number of static block but all these blocks will be execute from top to bottom



## Syntax

```
static
{
    .....
    //Set of Statements
    .....
}
```

# Static Block in Java - Example

- Static block example
- [..\Programs\Module2\StaticDemo1.java](#)
- Static block without main function
- [..\Programs\Module2\StaticDemo2.java](#)
- Multiple Static blocks
- [..\Programs\Module2\StaticDemo3.java](#)

## Why a static block executes before the main method ?

A class has to be loaded in main memory before we start using it. Static block is executed during class loading. This is the reason why a static block executes before the main method.

# Static methods and non static methods

	Non-Static method	Static method
1	<p>These method never be preceded by static keyword Example:</p> <pre>void fun1() {     .....     ..... }</pre>	<p>These method always preceded by static keyword Example:</p> <pre>static void fun2() {     .....     ..... }</pre>
2	Memory is allocated multiple time whenever method is calling.	Memory is allocated only once at the time of class loading.
3	It is specific to an object so that these are also known as instance method.	These are common to every object so that it is also known as member method or class method.
4	<p>These methods always access with object reference Syntax:</p> <p><b>Objref.methodname();</b></p>	<p>These property always access with class reference Syntax:</p> <p><b>className.methodname();</b></p>
5	If any method wants to be execute multiple time that can be declare as non static.	If any method wants to be execute only once in the program that can be declare as static .

# Static methods and non static methods - Example

## ◎ ..\Programs\Module2\StaticMethodDemo.java

Following table represent how the static and non-static properties are accessed in the different static or non-static method of same class or other class.

	within non-static method of same class	within static method of same class	within non-static method of other class	within static method of other class
Non-Static Method or Variable	Access directly	Access with object reference	Access with object reference	Access with object reference
Static Method or Variable	Access directly	Access directly	Access with object reference	Access with object reference

# Static inner classes

## Inner classes:

- If one class is existing within another class is known as inner class or nested class.

### Syntax

```
class Outerclass_name
{
.....
.....

class Innerclass_name1
{
.....
.....
}
class Innerclass_name1
{
.....
.....
}
}
```

# What is the main purpose of using inner class???

- To provide more security by making those inner class properties specific to only outer class but not for external classes.
- To make more than one property of classes private properties.
- If more than one property of class wants to make as private properties then all can be capped under private inner class.

## Syntax

```
class Outerclass_name
{
    private class Innerclass_name
    {
        .....
        ..... //private properties
    }
}
```

**Note:** No outer class made as private class otherwise this is not available for JVM at the time of execution.

# Rules to access properties of inner classes

- Inner class properties can be accessed in the outer class with the object reference but not directly.
- Outer class properties can be access directly within the inner class.
- [..\Programs\Module2\InnerClassDemo.java](#)



# Accessing inner class properties in the external class

1. If inner class is non static the object can be created with the following syntax

## Syntax

```
class Outer_class
{
    class Inner_class
    {
        ....
        ....
    }
    ....
    ....
}
class External_class
{
    Outer_class.Inner_Class objectreference=new Outer_Class.External_Class();
}
```

2. If inner class is static the object reference can be created with the following syntax

## Syntax

```
class Outer_class
{
    static class Inner_Class
    {
        ....
        ....
    }
}
class External_Class
{
    Outer_class.Inner_Class objectreference=new Outer_Class.External_Class();
}
}
```



# Accessing inner class properties in the external class

- ◎ [..\Programs\Module2\InnerClassDemo1.java](#)

# Introducing Final keyword







- The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context.

Final can be:

- variable
- method
- class



## Final Keyword in Java

- |   |                                     |   |                |
|---|-------------------------------------|---|----------------|
|  | Restrict Changing Value of Variable |  | Final Variable |
|  | Restrict Method Overriding          |  | Final Method   |
|  | Restrict Inheritance                |  | Final Class    |

## Java final variable

- If you make any variable as final, you cannot change the value of final variable(It will be constant).
- [..\Programs\Module2\FinalVariable.java](#)

## Java final Method

- If you make any method as final, you cannot override it
- [..\Programs\Module2\FinalMethod.java](#)

# Java final class

- ⦿ If you make any class as final, you cannot extend it.
- ⦿ ..\Programs\Module2\FinalClass.java

Q) Is final method inherited?

Ans) Yes, final method is inherited but you cannot override it.

Q) What is blank or uninitialized final variable?

Ans) A final variable that is not initialized at the time of declaration is known as blank final variable.

Q) Can we initialize blank final variable?

Ans) Yes, but only in constructor.

# Constructors for automatic initialization of objects

1. It can be tedious to initialize all of the variables in a class each time an object is instantiated.
2. Java allows objects to initialize themselves when they are created/declared.
3. This automatic initialization is performed through the concept of constructor.

# Properties of Constructors

1. A constructor **initializes an object** immediately upon creation.
2. Constructor in **Java** is a **method**.
3. This method has the **same name** as the class in which it resides.
4. Once defined, the constructor is **automatically called** immediately after object is created.
5. Constructor is a method which has **no return type**.
6. In fact, the implicit return type of a class constructor is the class type itself.
7. Constructor initialize the internal state of an object.

# Constructor – an Example

```
class Circle
{
    double x,y;
    double r;
    double circumference()
    {
        return 2*3.14159*r;
    }

    double area()
    {
        return(22/7)*r*r;
    }
    Circle(double a, double b, double c)
    {
        x = a;
        y = b;
        r = c;
    }
}
```

# Constructor – an Example

```
class CircleDemo1
{
    public static void main(String args[])
    {
        Circle c1 = new Circle(3.0,4.0,5.0);
        Circle c2 = new Circle(-5.0,-4.0,5.0);
        System.out.println("Circumference Circle 1" + c1.circumference());
        System.out.println("Area Circle 1" + c1.area());
        System.out.println("Circumference Circle 2" + c2.circumference());
        System.out.println("Area Circle 2" + c2.area());
    }
}
```



# Multiple constructors...

- ◉ Sometimes , it is necessary to initialize an object in a number of ways.
- ◉ Java allows this using the concept of Constructor Overloading.
- ◉ In other words, Java allows to declare one or more constructor method with different lists of parameters and different method definition.
- ◉ [..\Programs\Module2\CircleDemoMultiCons.java](#)

# this keyword concept

- ⦿ **this** is used to reduce name-space collisions.
- ⦿ Sometimes a method will need to refer to the object that invoked it. To allow this Java defines **this** keyword.
- ⦿ **this** can be used inside any method to refer to the current object.
- ⦿ **this** is always a reference to the object on which the method is invoked.
- ⦿ If we want to resolve same name which belong to the same class with other variable we should use **this** keyword.

# this – an example

```
class Circle
{
    double x,y;
    double r;
    double circumference()
    {
        return 2*3.14159*r;
    }

    double area()
    {
        return(22/7)*r*r;
    }
    Circle(double x, double y, double r)
    {
        this.x = x;
        this.y = y;
        this.r = r;
    }
}
```

# this – an example

```
class CircleDemo1
{
    public static void main(String args[])
    {
        Circle c1 = new Circle(3.0,4.0,5.0);
        Circle c2 = new Circle(-5.0,-4.0,5.0);
        System.out.println("Circumference Circle 1" + c1.circumference());
        System.out.println("Area Circle 1" + c1.area());
        System.out.println("Circumference Circle 2" + c2.circumference());
        System.out.println("Area Circle 2" + c2.area());
    }
}
```

# this keyword concept

- ..\Programs\Module2\thisDemo.java
- If we want to call any constructor with in any other constructor then we can use **this** keyword.
- ..\Programs\Module2\thisDemo1.java

# Garbage collection

- In java, garbage means unreferenced objects.  
**How can an object be unreferenced?**



# Garbage collection

- Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.
- To do so, we were using free() function in C language and delete in C++. But, in java it is performed automatically. So, java provides better memory management.
- The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

**Syntax: public static void gc() { }**

Note: Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.

# Finalize() Method

- ⦿ A constructor helps to initialize an object just after it has been created.
- ⦿ In contrast, the finalize method is invoked just before the object is destroyed:

1) implemented inside a class as:

```
protected void finalize() { ... }
```

2) implemented when the usual way of removing objects from memory is insufficient, and some special actions has to be carried out.



# Finalize() method

- The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

**Syntax: protected void finalize() { }**

Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

# gc() and finalize() method - example

```
public class TestGarbage1{  
  
    public void finalize(){System.out.println("object is garbage collected");}  
  
    public static void main(String args[]){  
        TestGarbage1 s1=new TestGarbage1();  
        TestGarbage1 s2=new TestGarbage1();  
        s1=null;  
        s2=null;  
        System.gc();  
    }  
}
```

Compile by: javac TestGarbage1.java

Run by: java TestGarbage1

object is garbage collected  
object is garbage collected

# Java run time INPUT- Command line argument

- The java command-line argument is an argument i.e. passed at the time of running the java program.
- The arguments passed from the console can be received in the java program and it can be used as an input.
- So, it provides a convenient way to check the behavior of the program for the different values.

# Java run time INPUT- Command line argument

- In the below example we are receiving arguments and printing them.

```
class CommandLineExample{  
    public static void main(String args[]){  
        System.out.println("Your first argument is: "+args[0]);  
    }  
}
```

```
compile by > javac CommandLineExample.java  
run by > java CommandLineExample sonoo
```

Output: Your first argument is: sonoo

```
class A{  
    public static void main(String args[]){  
  
        for(int i=0;i<args.length;i++)  
            System.out.println(args[i]);  
  
    }  
}
```

```
compile by > javac A.java  
run by > java A sonoo jaiswal 1 3 abc
```

Output: sonoo  
 jaiswal  
 1  
 3  
 abc

# Numeric Input to the program

```
import java.lang.*;

class Calculator{
    double i;
    double x = Math.sqrt(i);
}

class Example{
    public static void main(String args[]){
        Calculator a = new Calculator();
        a.i = Integer.parseInt(args[0]);
        System.out.println("Square root of "+a.i+" is "+a.x);
    }
}
```

# Recursion in Java

- Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method.



## Syntax:

```
returntype methodname() {  
    //code to be executed  
    methodname(); //calling same method  
}
```

# Java Recursion Example 1: Infinite times

```
public class RecursionExample1 {  
static void p(){  
    System.out.println("hello");  
    p();  
}
```

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

# Java Recursion Example 2: Finite times

```
public class RecursionExample2 {  
    static int count=0;  
    static void p() {  
        count++;  
        if(count<=5){  
            System.out.println("hello "+count);  
            p();  
        }  
    }  
    public static void main(String[] args) {  
        p();  
    }  
}
```



# Java Recursion Example 3:

## Factorial Number

```
public class RecursionExample3 {  
    static int factorial(int n){  
        if (n == 1)  
            return 1;  
        else  
            return(n * factorial(n-1));  
    }  
  
    public static void main(String[] args) {  
        System.out.println("Factorial of 5 is: "+factorial(5));  
    }  
}
```

# Java Recursion Example 4: Fibonacci Series

```
public class RecursionExample4 {  
    static int n1=0,n2=1,n3=0;  
    static void printFibo(int count){  
        if(count>0){  
            n3 = n1 + n2;  
            n1 = n2;  
            n2 = n3;  
            System.out.print(" "+n3);  
            printFibo(count-1);  
        }  
    }  
  
    public static void main(String[] args) {  
        int count=15;  
        System.out.print(n1+" "+n2);//printing 0 and 1  
        printFibo(count-2);//n-2 because 2 numbers are already printed  
    }  
}
```

# THANK YOU