WelCome

5th Semester Computer Science & Technology

Subject Name: Application Development using JAVA

Subject Code – 28551

T	P	C
2	3	3

Presented by
Bulbul Ahamed
Chief Instructor & Head of the Dept.
Computer Department
Sherpur Polytechnic Institute.

Chapter-1

Classes, Objects in Java

At the end of this chapter/session students will be able to know Classes, Objects, in Java

.

Declaration (syntax) of class and object in Java

Objects and Classes in Java

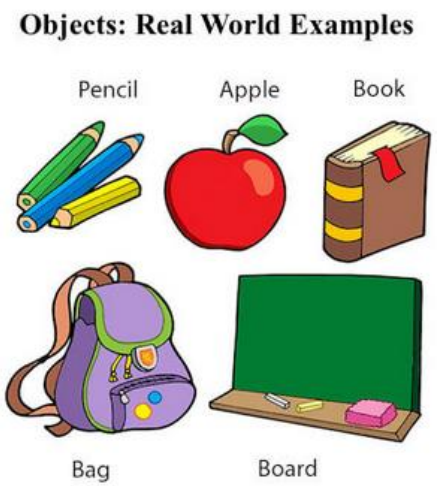
In this page, we will learn about Java objects and classes. In object-oriented programming technique, we design a program using objects and classes.

An object in Java is the physical as well as a logical entity, whereas, a class in Java is a logical entity only.

- Object in Java
- Class in Java
- Instance Variable in Java
- Method in Java
- Example of Object and class that main student
- Anonymous Object

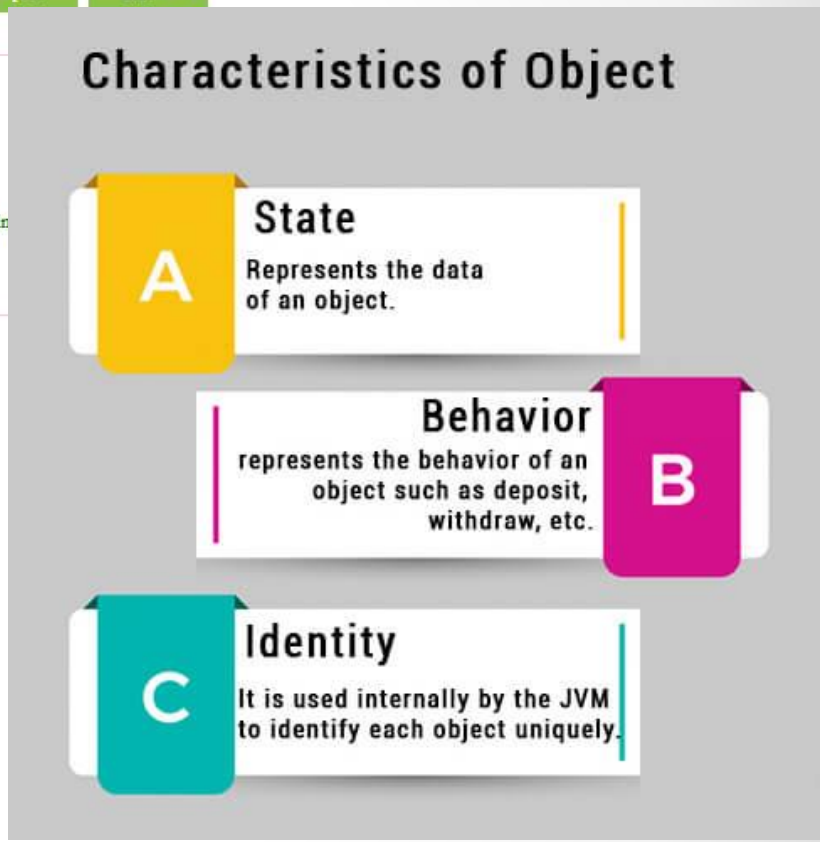
What is an object in Java

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.



An object has three characteristics:

- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.



Mohammad Hazrat Ali
Chief Instructor (Tech)
Computer.
Mymensingh Polytechnic Institute

For Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Object Definitions:

- An object is *a real-world entity*.
- An object is *a runtime entity*.
- The object is *an entity which has state and behavior*.
- The object is *an instance of a class*.

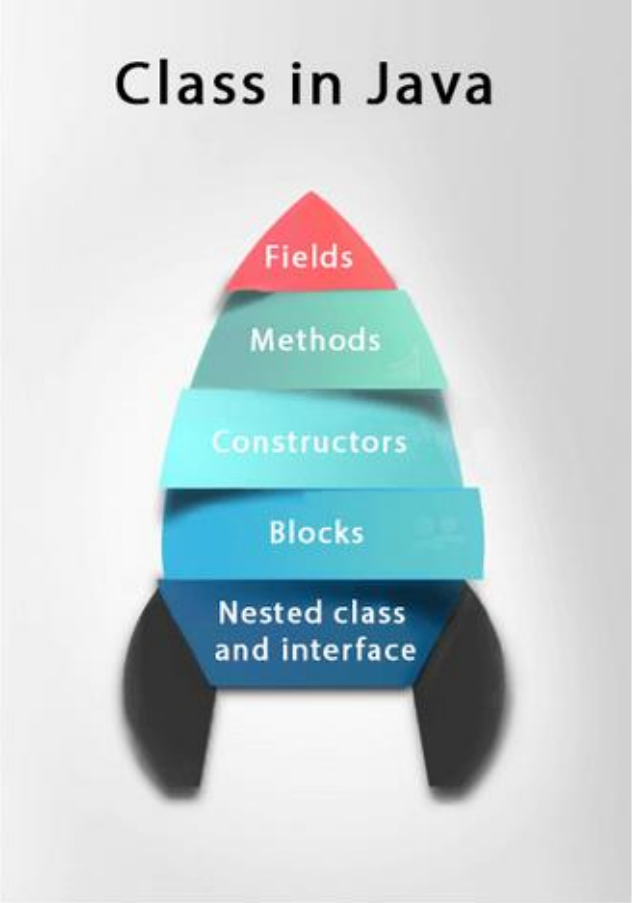
What is a class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

- **Fields**
- **Methods**
- **Constructors**
- **Blocks**
- **Nested class and interface**

Java



Syntax to declare a class:

```
class <class_name>{
    field;
    method;
}
```

Instance variable in Java

A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

Method in Java

In Java, a method is like a function which is used to expose the behavior of an object.

Advantage of Method

- Code Reusability
- Code Optimization

---

new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

Create an Object

In Java, an object is created from a class. We have already created the class named `MyClass`, so now we can use this to create objects.

To create an object of `MyClass`, specify the class name, followed by the object name, and use the keyword `new`:

Example

Create an object called "`myObj`" and print the value of x:

```
public class MyClass {
  int x = 5;

  public static void main(String[] args) {
    MyClass myObj = new MyClass();
    System.out.println(myObj.x);
  }
}
```

Assigning object reference variables in java.

3 Ways to initialize object

There are 3 ways to initialize object in Java.

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

1) Object and Class Example: Initialization through reference

Initializing an object means storing data into the object. Let's see a simple example where we are going to initialize the object through a reference variable.

File: TestStudent2.java

```
class Student{
 int id;
 String name;
}
class TestStudent2{
 public static void main(String args[]){
  Student s1=new Student();
  s1.id=101;
  s1.name="Sonoo";
  System.out.println(s1.id+" "+s1.name);//printing members with a white space
 }
}
```

Assigning object reference variables in java.

2) Object and Class Example: Initialization through method

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method. Here, we are displaying the state (data) of the objects by invoking the displayInformation() method.

File: TestStudent4.java

```
class Student{
 int rollno;
 String name;
 void insertRecord(int r, String n){
  rollno=r;
  name=n;
 }
 void displayInformation(){System.out.println(rollno+" "+name);}
}
class TestStudent4{
 public static void main(String args[]){
  Student s1=new Student();
  Student s2=new Student();
  s1.insertRecord(111,"Karan");
  s2.insertRecord(222,"Aryan");
  s1.displayInformation();
  s2.displayInformation();
 }
}
```

Assigning object reference variables in java.

3) Object and Class Example: Initialization through a constructor

We will learn about constructors in Java later.

Object and Class Example: Employee

Let's see an example where we are maintaining records of employees.

File: TestEmployee.java

```
class Employee{
   int id;
   String name;
   float salary;
   void insert(int i, String n, float s) {
      id=i;
      name=n;
      salary=s;
   }
   void display(){System.out.println(id+" "+name+" "+salary);}
}
public class TestEmployee {
public static void main(String[] args) {
   Employee e1=new Employee();
   Employee e2=new Employee();
   Employee e3=new Employee();
   e1.insert(101,"ajeet",45000);
   e2.insert(102,"irfan",25000);
   e3.insert(103,"nakul",55000);
   e1.display();
   e2.display();
   e3.display();
}
}
```

Chapter-2

Methods and Constructors in Java

At the end of this chapter/session students will be able to know Methods and Constructors in Java

.

Java Methods

‹ Previous Next

A **method** is a block of code which only runs when it is called.

You can pass data, known as parameters, into a method.

Methods are used to perform certain actions, and they are also known as **functions**.

Why use methods? To reuse code: define the code once, and use it many times.

Create a Method

A method must be declared within a class. It is defined with the name of the method, followed by parentheses **()**. Java provides some pre-defined methods, such as `System.out.println()`, but you can also create your own methods to perform certain actions:

Creating Method

Considering the following example to explain the syntax of a method −

Syntax

```
public static int methodName(int a, int b) {
   // body
}
```

Here,

- **public static** − modifier
- **int** − return type
- **methodName** − name of the method
- **a, b** − formal parameters
- **int a, int b** − list of parameters

Method definition consists of a method header and a method body. The same is shown in the following syntax −

Syntax

```
modifier returnType nameOfMethod (Parameter List) {
   // method body
}
```

The syntax shown above includes −

- **modifier** − It defines the access type of the method and it is optional to use.
- **returnType** − Method may return a value.
- **nameOfMethod** − This is the method name. The method signature consists of the method name and the parameter list.
- **Parameter List** − The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.
- **method body** − The method body defines what the method does with the statements.

Example

Here is the source code of the above defined method called **min()**. This method takes two parameters num1 and num2 and returns the maximum between the two −

```
/** the snippet returns the minimum between two numbers */

public static int minFunction(int n1, int n2) {
   int min;
   if (n1 > n2)
      min = n2;
   else
      min = n1;

   return min;
}
```

Call a Method

To call a method in Java, write the method's name followed by two parentheses **()** and a semicolon**;**

In the following example, `myMethod()` is used to print a text (the action), when it is called:

Example

Inside `main` , call the `myMethod()` method:

```
public class MyClass {
  static void myMethod() {
    System.out.println("I just got executed!");
  }

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

// Outputs "I just got executed!"
```

Procedure of adding Method to class.

Method Parameters

Information can be passed to methods as parameter. Parameters act as variables inside the method.

Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

The following example has a method that takes a `String` called **fname** as parameter. When the method is called, we pass along a first name, which is used inside the method to print the full name:

Example

```
public class MyClass {
  static void myMethod(String fname) {
    System.out.println(fname + " Refsnes");
  }

  public static void main(String[] args) {
    myMethod("Liam");
    myMethod("Jenny");
    myMethod("Anja");
  }
}
// Liam Refsnes
// Jenny Refsnes
// Anja Refsnes
```

Procedure of adding Method to class.

Return Values

The `void` keyword, used in the examples above, indicates that the method should not return a value. If you want the method to return a value, you can use a primitive data type (such as `int`, `char`, etc.) instead of `void`, and use the `return` keyword inside the method:

Example

```
public class MyClass {
  static int myMethod(int x) {
    return 5 + x;
  }

  public static void main(String[] args) {
    System.out.println(myMethod(3));
  }
}
// Outputs 8 (5 + 3)
```

What are the advantages of using methods?

- The main advantage is code reusability. You can write a method once, and use it multiple times. You do not have to rewrite the entire code each time. Think of it as, "write once, reuse multiple times."

- Methods make code more readable and easier to debug. For example, `getSalaryInformation()` method is so readable, that we can know what this method will be doing than actually reading the lines of code that make this method.

Advantages of methods are as follows :

- The methods help to implement our task easier .
- It saves a lot of time in case of several operations and several tasks .
- The methods help in data security .
- The methods also enable polymorphism .
- Methods can be overloaded or over ridden .
- Methods allow recursion and it acts as a loop .

If a class has multiple methods having same name but different in 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.

Advantage of method overloading

Method overloading *increases the readability of the program*.

Different ways to overload the method

There are two ways to overload the method in java

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

1) Method Overloading: changing no. of arguments

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

In this example, we are creating static methods so that we don't need to create instance for calling methods.

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
```

2) Method Overloading: changing data type of arguments

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

```
class Adder{
static int add(int a, int b){return a+b;}
static double add(double a, double b){return a+b;}
}
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}}
```

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating Java constructor

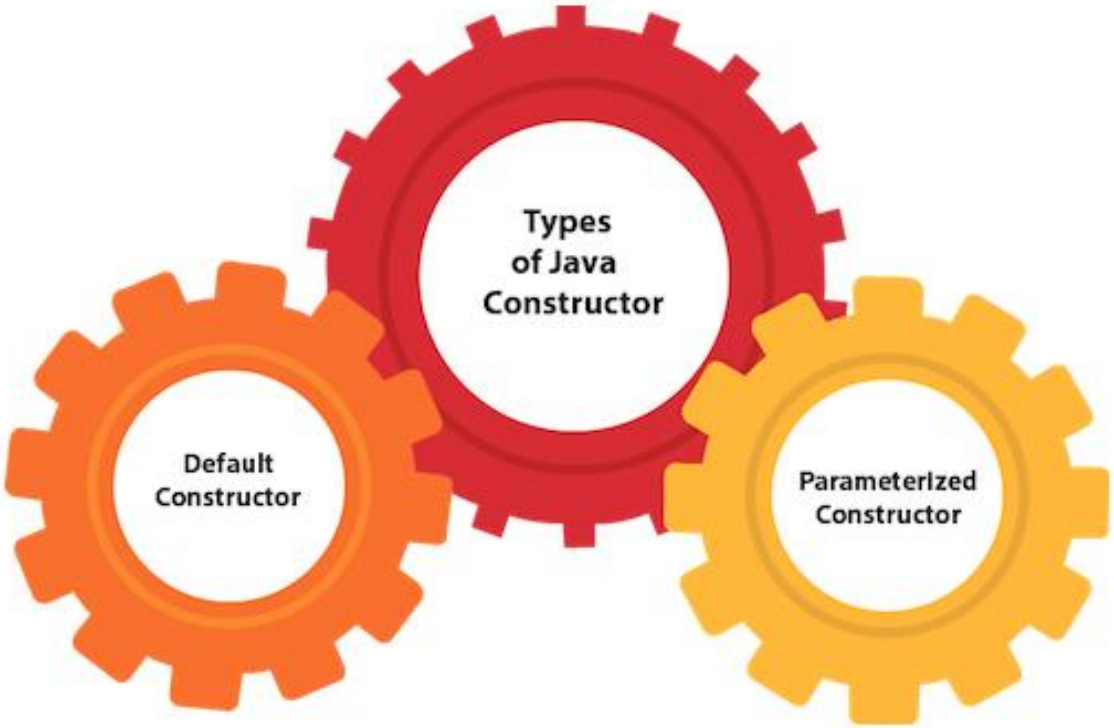
There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor



Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

```
<class_name>(){}
```

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
//Java Program to create and call a default constructor
class Bike1{
//creating a default constructor
Bike1(){System.out.println("Bike is created");}
//main method
public static void main(String args[]){
//calling a default constructor
Bike1 b=new Bike1();
}
}
```

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
//Java Program to demonstrate the use of the parameterized constructor.
class Student4{
    int id;
    String name;
    //creating a parameterized constructor
    Student4(int i,String n){
    id = i;
    name = n;
    }
    //method to display the values
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){
    //creating objects and passing values
    Student4 s1 = new Student4(111,"Karan");
    Student4 s2 = new Student4(222,"Aryan");
    //calling method to display the values of object
    s1.display();
    s2.display();
   }
}
```

Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

```java
//Java program to overload constructors
class Student5{
    int id;
    String name;
    int age;
    //creating two arg constructor
    Student5(int i,String n){
    id = i;
    name = n;
    }
    //creating three arg constructor
    Student5(int i,String n,int a){
    id = i;
    name = n;
    age=a;
    }
    void display(){System.out.println(id+" "+name+" "+age);}

    public static void main(String args[]){
    Student5 s1 = new Student5(111,"Karan");
    Student5 s2 = new Student5(222,"Aryan",25);
    s1.display();
    s2.display();
   }
}
```

Instance variable hiding in java

Whenever you inherit a superclass a copy of superclass's members is created at the subclass and you using its object you can access the superclass members.

If the superclass and the subclass have instance variable of same name, if you access it using the subclass object, the instance variables of the subclass hides the instance variables of the superclass irrespective of the types. This mechanism is known as field hiding or, instance variable hiding.

But, since it makes code complicated field hiding is not recommended.

Instance variable hiding in java

Example

In the following example we have two classes Super and Sub one extending the other. They both have two fields with same names (name and age).

When we print values of these fields using the object of Sub. The values of the subclass are printed.

```
class Super {
   String name = "Krishna";
   int age = 25;
}
class Sub extends Super {
   String name = "Vishnu";
   int age = 22;
   public void display(){
      Sub obj = new Sub();
      System.out.println("Name: "+obj.name);
      System.out.println("age: "+obj.age);
   }
}
public class FieldHiding{
   public static void main(String args[]){
      new Sub().display();
   }
}
```

Output

```
Name: Vishnu
age: 22
```

Garbage collection in Java

In java, garbage means unreferenced objects.

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.

Advantage of Garbage Collection

- It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
- It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

How can an object be unreferenced?

There are many ways:

- By nulling the reference
- By assigning a reference to another
- By anonymous object etc.

Garbage collection in Java

1) By nulling a reference:

```
Employee e=new Employee();
e=null;
```

2) By assigning a reference to another:

```
Employee e1=new Employee();
Employee e2=new Employee();
e1=e2;//now the first object referred by e1 is available for garbage collection
```

3) By anonymous object:

```
new Employee();
```

Chapter-3

OOP in Java

At the end of this chapter/session students will be able to know OOP in Java

.

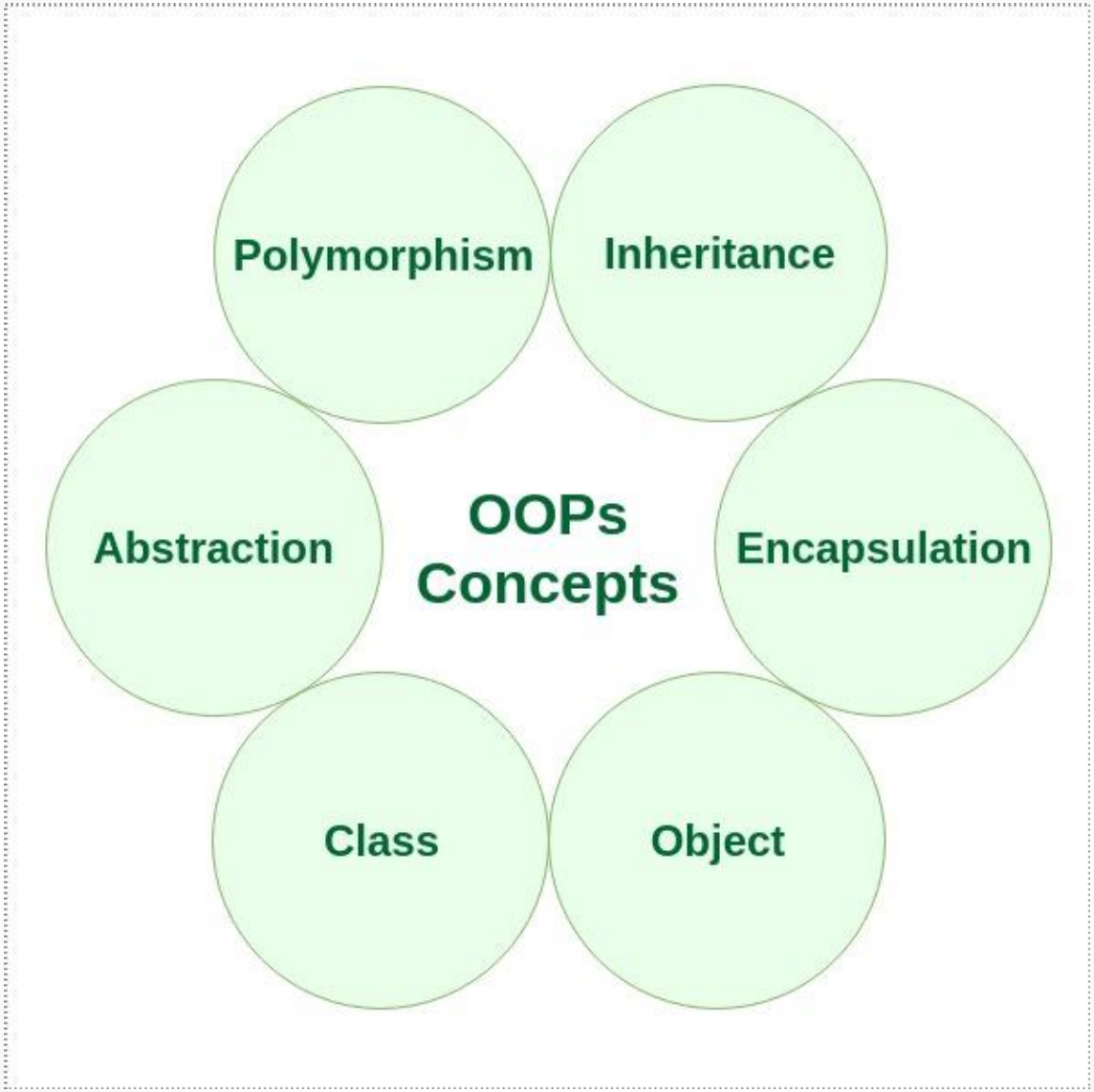
Object Oriented Programming (OOPs) Concept in Java

Object-oriented programming: As the name suggests, Object-Oriented Programming or OOPs refers to languages that uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

OOPs Concepts:

- Polymorphism
- Inheritence
- Encapsulation
- Abstraction
- Class
- Object
- Method
- Message Passing

State the terms used continued.....



State the terms used continued.....

. **Polymorphism:** Polymorphism refers to the ability of OOPs programming languages to differentiate between entities with the same name efficiently. This is done by Java with the help of the signature and declaration of these entities.

Inheritence: Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.

Important terminology:

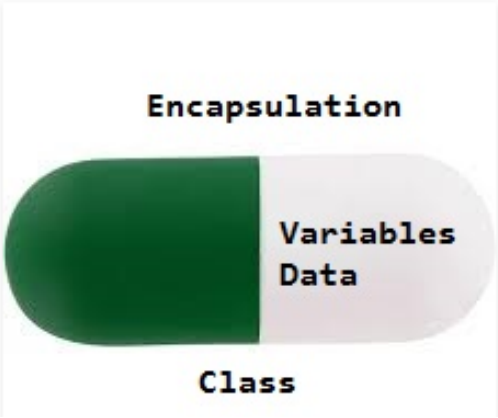
- **Super Class:** The class whose features are inherited is known as superclass(or a base class or a parent class).
- **Sub Class:** The class that inherits the other class is known as subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
- **Reusability:** Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

The keyword used for inheritance is **extends**.

State the terms used continued.....

Encapsulation: Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.

- Technically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of own class in which they are declared.
- As in encapsulation, the data in a class is hidden from other classes, so it is also known as **data-hiding**.
- Encapsulation can be achieved by Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.



State the terms used continued.....

. **Abstraction:** Data Abstraction is the property by virtue of which only the essential details are displayed to the user.The trivial or the non-essentials units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.
Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details. The properties and behaviours of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.

In java, abstraction is achieved by interfaces and abstract classes. We can achieve 100% abstraction using interfaces.

State the terms used continued.....

. **Class:** A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

1. **Modifiers**: A class can be public or has default access (Refer this for details).
2. **Class name:** The name should begin with a initial letter (capitalized by convention).
3. **Superclass(if any):** The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
4. **Interfaces(if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
5. **Body:** The class body surrounded by braces, { }.

. **Object:** It is a basic unit of Object Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of:

1. **State** : It is represented by attributes of an object. It also reflects the properties of an object.
2. **Behavior** : It is represented by methods of an object. It also reflects the response of an object with other objects.
3. **Identity** : It gives a unique name to an object and enables one object to interact with other objects.

Example of an object: dog

State the terms used continued.....

Dynamic Binding: In dynamic binding, the code to be executed in response to function call is decided at runtime. C++ has virtual functions to support this.

Message Passing: Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function and the information to be sent.

Some OOP languages

Languages with object-oriented features [edit]

- ABAP
- Ada 95
- AmigaE
- Apex
- BETA
- Boo
- C++
- C#
- Ceylon
- Chapel
- Clarion
- CLU
- COBOL
- Cobra
- ColdFusion
- Common Lisp
- COOL
- CorbaScript
- Curl
- D
- Dart
- DataFlex
- Omnis Studio
- OpenEdge Advanced Business Language
- Oz, Mozart Programming System
- Perl since v5
- PHP since v4, greatly enhanced in v5
- Power Builder
- Prototype-based languages
  - Actor-Based Concurrent Languages: ABCL/1, ABCL/R, ABCL/R2, ABCL/c+
  - Agora
  - Cecil
  - ECMAScript
    - ActionScript
    - JavaScript
    - JScript
  - Etoys (in Squeak)
  - Io
  - Lua
  - Lisaac
  - MOO
  - NewtonScript
  - Obliq

Some OOP languages

- Dylan
- E
- Eiffel
  - Sather
- Elixir
- Fortran 2003
- FPr
- FreeBASIC
- F-Script
- F#
- Gambas
- Genie
- Go
- Graphtalk
- IDLscript
- J
- J#
- JADE
- Java
  - Groovy
  - Join Java
  - X10
- Julia
  - REBOL
  - Self
- Python
- REALbasic
- Ruby
- Rust
- S
  - R
- Scala
- Scriptol
- Seed7
- SenseTalk
- Simula
- Smalltalk
  - Self
  - Bistro
  - Squeak
    - Pharo
    - Newspeak
- Squirrel
- Swift
- TADS
- Tcl

Some OOP languages

- Kotlin
- Lasso
- Lava
- Lexico
- Lingo
- LISP
- Logtalk
- MATLAB
- Modula-3
- Nemerle
- NetRexx
- Nim
- Noop
- Oberon (Oberon-1)
  - Oberon-2
- Object Pascal
  - Delphi
  - Free Pascal
  - Turbo Pascal
- Object REXX
- Objective-C
- OCaml
  - Xotcl (similar to CLOS)
  - incr Tcl (itcl; similar to C++)
- Transcript
- TypeScript
- Ubercode
- Vala
- Visual Basic
  - Visual Basic .NET (VB.NET)
  - VBScript
  - Visual Basic for Applications (VBA)
- Visual FoxPro
- Visual Prolog
- XBase++ (extends xBase standard language)
- Xojo
- ZZT-oop

Benefits of OOP

Some of the advantages of object-oriented programming include:

1. Improved software-development productivity: Object-oriented programming is modular, as it provides separation of duties in object-based program development. It is also extensible, as objects can be extended to include new attributes and behaviors. Objects can also be reused within an across applications. Because of these three factors – modularity, extensibility, and reusability – object-oriented programming provides improved software-development productivity over traditional procedure-based programming techniques.

2. Improved software maintainability: For the reasons mentioned above, object-oriented software is also easier to maintain. Since the design is modular, part of the system can be updated in case of issues without a need to make large-scale changes.

3. Faster development: Reuse enables faster development. Object-oriented programming languages come with rich libraries of objects, and code developed during projects is also reusable in future projects.

4. Lower cost of development: The reuse of software also lowers the cost of development. Typically, more effort is put into the object-oriented analysis and design, which lowers the overall cost of development.

5. Higher-quality software: Faster development of software and lower cost of development allows more time and resources to be used in the verification of the software. Although quality is dependent upon the experience of the teams, object-oriented programming tends to result in higher-quality software.

Benefits of OOP

Benefits of OOP:

- It is easy to model a real system as real objects are represented by programming objects in OOP. The objects are processed by their member data and functions. It is easy to analyze the user requirements.
- With the help of inheritance, we can reuse the existing class to derive a new class such that the redundant code is eliminated and the use of existing class is extended. This saves time and cost of program.
- In OOP, data can be made private to a class such that only member functions of the class can access the data. This principle of data hiding helps the programmer to build a secure program that can not be invaded by code in other part of the program.
- With the help of polymorphism, the same function or same operator can be used for different purposes. This helps to manage software complexity easily.
- Large problems can be reduced to smaller and more manageable problems. It is easy to partition the work in a project based on objects.
- It is possible to have multiple instances of an object to co-exist without any interference i.e. each object has its own separate member data and function.

Applications of OOP

Applications of Object Oriented Programming

Main application areas of OOP are:

- User interface design such as windows, menu.
- Real Time Systems
- Simulation and Modeling
- Object oriented databases
- AI and Expert System
- Neural Networks and parallel programming
- Decision support and office automation systems etc.

Chapter-4

Classes, Objects, Methodsand Constructors in Java

At the end of this chapter/session students will be able to know Classes, Objects, Methods, and Constructors in Java

.

Declaration (syntax) of class and object in Java

Objects and Classes in Java

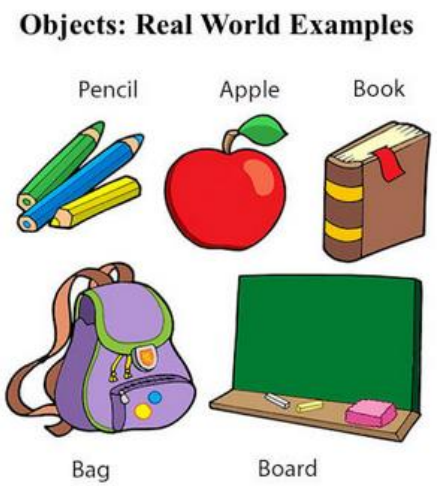
In this page, we will learn about Java objects and classes. In object-oriented programming technique, we design a program using objects and classes.

An object in Java is the physical as well as a logical entity, whereas, a class in Java is a logical entity only.

- Object in Java
- Class in Java
- Instance Variable in Java
- Method in Java
- Example of Object and class that main student
- Anonymous Object

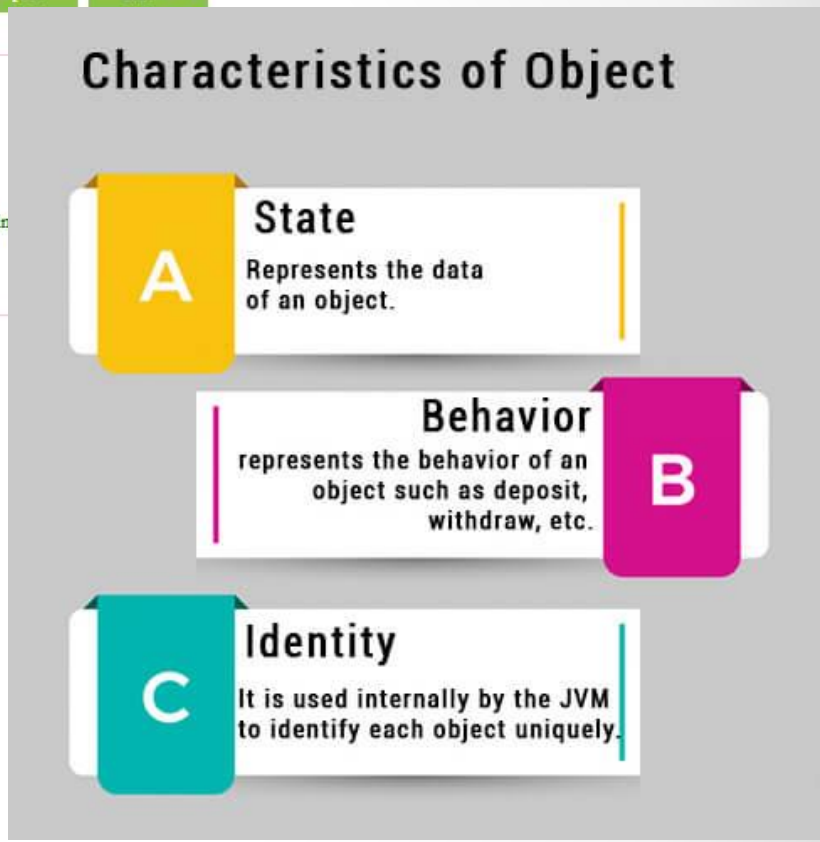
What is an object in Java

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.



An object has three characteristics:

- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.



Mohammad Hazrat Ali
Chief Instructor (Tech)
Computer.
Mymensingh Polytechnic Institute

For Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Object Definitions:

- An object is *a real-world entity*.
- An object is *a runtime entity*.
- The object is *an entity which has state and behavior*.
- The object is *an instance of a class*.

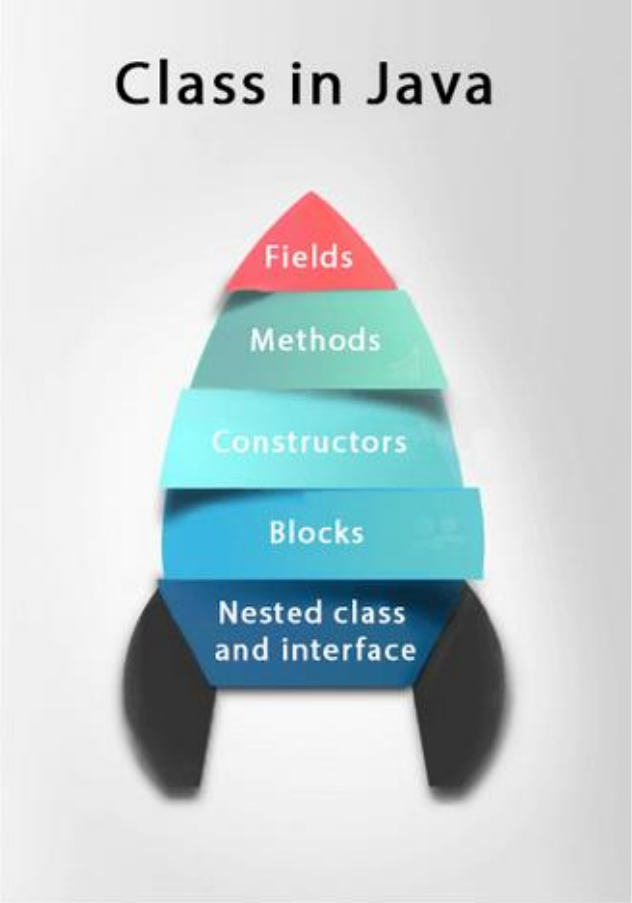
What is a class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

- **Fields**
- **Methods**
- **Constructors**
- **Blocks**
- **Nested class and interface**

Java



Syntax to declare a class:

```
class <class_name>{
    field;
    method;
}
```

Instance variable in Java

A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

Method in Java

In Java, a method is like a function which is used to expose the behavior of an object.

Advantage of Method

- Code Reusability
- Code Optimization

---

new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

Create an Object

In Java, an object is created from a class. We have already created the class named `MyClass`, so now we can use this to create objects.

To create an object of `MyClass`, specify the class name, followed by the object name, and use the keyword `new`:

Example

Create an object called "`myObj`" and print the value of x:

```
public class MyClass {
  int x = 5;

  public static void main(String[] args) {
    MyClass myObj = new MyClass();
    System.out.println(myObj.x);
  }
}
```

Assigning object reference variables in java.

3 Ways to initialize object

There are 3 ways to initialize object in Java.

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

1) Object and Class Example: Initialization through reference

Initializing an object means storing data into the object. Let's see a simple example where we are going to initialize the object through a reference variable.

File: TestStudent2.java

```
class Student{
 int id;
 String name;
}
class TestStudent2{
 public static void main(String args[]){
  Student s1=new Student();
  s1.id=101;
  s1.name="Sonoo";
  System.out.println(s1.id+" "+s1.name);//printing members with a white space
 }
}
```

Assigning object reference variables in java.

2) Object and Class Example: Initialization through method

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method. Here, we are displaying the state (data) of the objects by invoking the displayInformation() method.

File: TestStudent4.java

```
class Student{
 int rollno;
 String name;
 void insertRecord(int r, String n){
  rollno=r;
  name=n;
 }
 void displayInformation(){System.out.println(rollno+" "+name);}
}
class TestStudent4{
 public static void main(String args[]){
  Student s1=new Student();
  Student s2=new Student();
  s1.insertRecord(111,"Karan");
  s2.insertRecord(222,"Aryan");
  s1.displayInformation();
  s2.displayInformation();
 }
}
```

Assigning object reference variables in java.

3) Object and Class Example: Initialization through a constructor

We will learn about constructors in Java later.

Object and Class Example: Employee

Let's see an example where we are maintaining records of employees.

File: TestEmployee.java

```
class Employee{
   int id;
   String name;
   float salary;
   void insert(int i, String n, float s) {
      id=i;
      name=n;
      salary=s;
   }
   void display(){System.out.println(id+" "+name+" "+salary);}
}
public class TestEmployee {
public static void main(String[] args) {
   Employee e1=new Employee();
   Employee e2=new Employee();
   Employee e3=new Employee();
   e1.insert(101,"ajeet",45000);
   e2.insert(102,"irfan",25000);
   e3.insert(103,"nakul",55000);
   e1.display();
   e2.display();
   e3.display();
}
}
```

Java Methods

‹ Previous Next

A **method** is a block of code which only runs when it is called.

You can pass data, known as parameters, into a method.

Methods are used to perform certain actions, and they are also known as **functions**.

Why use methods? To reuse code: define the code once, and use it many times.

Create a Method

A method must be declared within a class. It is defined with the name of the method, followed by parentheses **()**. Java provides some pre-defined methods, such as `System.out.println()`, but you can also create your own methods to perform certain actions:

Creating Method

Considering the following example to explain the syntax of a method −

Syntax

```
public static int methodName(int a, int b) {
   // body
}
```

Here,

- **public static** − modifier
- **int** − return type
- **methodName** − name of the method
- **a, b** − formal parameters
- **int a, int b** − list of parameters

Method definition consists of a method header and a method body. The same is shown in the following syntax −

Syntax

```
modifier returnType nameOfMethod (Parameter List) {
   // method body
}
```

The syntax shown above includes −

- **modifier** − It defines the access type of the method and it is optional to use.
- **returnType** − Method may return a value.
- **nameOfMethod** − This is the method name. The method signature consists of the method name and the parameter list.
- **Parameter List** − The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.
- **method body** − The method body defines what the method does with the statements.

Example

Here is the source code of the above defined method called **min()**. This method takes two parameters num1 and num2 and returns the maximum between the two −

```
/** the snippet returns the minimum between two numbers */

public static int minFunction(int n1, int n2) {
   int min;
   if (n1 > n2)
      min = n2;
   else
      min = n1;

   return min;
}
```

Call a Method

To call a method in Java, write the method's name followed by two parentheses **()** and a semicolon**;**

In the following example, `myMethod()` is used to print a text (the action), when it is called:

Example

Inside `main` , call the `myMethod()` method:

```
public class MyClass {
  static void myMethod() {
    System.out.println("I just got executed!");
  }

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

// Outputs "I just got executed!"
```

Procedure of adding Method to class.

Method Parameters

Information can be passed to methods as parameter. Parameters act as variables inside the method.

Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

The following example has a method that takes a `String` called **fname** as parameter. When the method is called, we pass along a first name, which is used inside the method to print the full name:

Example

```
public class MyClass {
  static void myMethod(String fname) {
    System.out.println(fname + " Refsnes");
  }

  public static void main(String[] args) {
    myMethod("Liam");
    myMethod("Jenny");
    myMethod("Anja");
  }
}
// Liam Refsnes
// Jenny Refsnes
// Anja Refsnes
```

Procedure of adding Method to class.

Return Values

The `void` keyword, used in the examples above, indicates that the method should not return a value. If you want the method to return a value, you can use a primitive data type (such as `int`, `char`, etc.) instead of `void`, and use the `return` keyword inside the method:

Example

```
public class MyClass {
  static int myMethod(int x) {
    return 5 + x;
  }

  public static void main(String[] args) {
    System.out.println(myMethod(3));
  }
}
// Outputs 8 (5 + 3)
```

What are the advantages of using methods?

- The main advantage is code reusability. You can write a method once, and use it multiple times. You do not have to rewrite the entire code each time. Think of it as, "write once, reuse multiple times."

- Methods make code more readable and easier to debug. For example, `getSalaryInformation()` method is so readable, that we can know what this method will be doing than actually reading the lines of code that make this method.

Advantages of methods are as follows :

- The methods help to implement our task easier .
- It saves a lot of time in case of several operations and several tasks .
- The methods help in data security .
- The methods also enable polymorphism .
- Methods can be overloaded or over ridden .
- Methods allow recursion and it acts as a loop .

If a class has multiple methods having same name but different in 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.

Advantage of method overloading

Method overloading *increases the readability of the program*.

Different ways to overload the method

There are two ways to overload the method in java

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

1) Method Overloading: changing no. of arguments

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

In this example, we are creating static methods so that we don't need to create instance for calling methods.

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
```

2) Method Overloading: changing data type of arguments

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

```
class Adder{
static int add(int a, int b){return a+b;}
static double add(double a, double b){return a+b;}
}
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}}
```

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating Java constructor

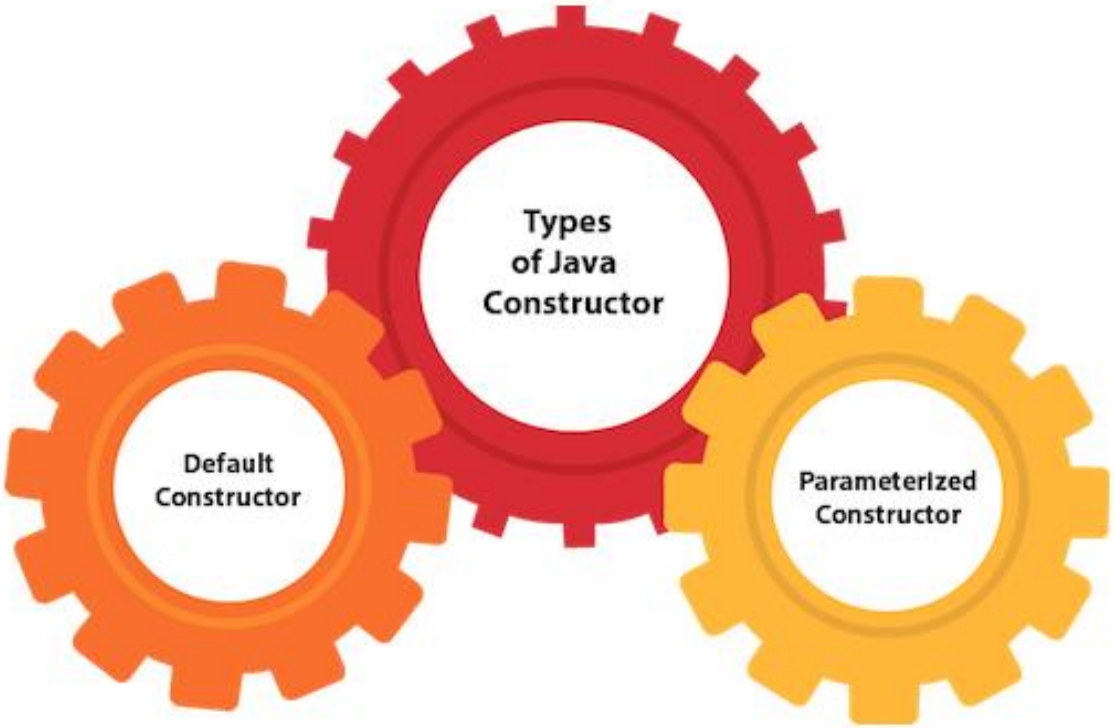
There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor



Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

```
<class_name>(){}
```

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
//Java Program to create and call a default constructor
class Bike1{
//creating a default constructor
Bike1(){System.out.println("Bike is created");}
//main method
public static void main(String args[]){
//calling a default constructor
Bike1 b=new Bike1();
}
}
```

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
//Java Program to demonstrate the use of the parameterized constructor.
class Student4{
    int id;
    String name;
    //creating a parameterized constructor
    Student4(int i,String n){
    id = i;
    name = n;
    }
    //method to display the values
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){
    //creating objects and passing values
    Student4 s1 = new Student4(111,"Karan");
    Student4 s2 = new Student4(222,"Aryan");
    //calling method to display the values of object
    s1.display();
    s2.display();
   }
}
```

Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

```java
//Java program to overload constructors
class Student5{
    int id;
    String name;
    int age;
    //creating two arg constructor
    Student5(int i,String n){
    id = i;
    name = n;
    }
    //creating three arg constructor
    Student5(int i,String n,int a){
    id = i;
    name = n;
    age=a;
    }
    void display(){System.out.println(id+" "+name+" "+age);}

    public static void main(String args[]){
    Student5 s1 = new Student5(111,"Karan");
    Student5 s2 = new Student5(222,"Aryan",25);
    s1.display();
    s2.display();
   }
}
```

Instance variable hiding in java

Whenever you inherit a superclass a copy of superclass's members is created at the subclass and you using its object you can access the superclass members.

If the superclass and the subclass have instance variable of same name, if you access it using the subclass object, the instance variables of the subclass hides the instance variables of the superclass irrespective of the types. This mechanism is known as field hiding or, instance variable hiding.

But, since it makes code complicated field hiding is not recommended.

Instance variable hiding in java

Example

In the following example we have two classes Super and Sub one extending the other. They both have two fields with same names (name and age).

When we print values of these fields using the object of Sub. The values of the subclass are printed.

```
class Super {
   String name = "Krishna";
   int age = 25;
}
class Sub extends Super {
   String name = "Vishnu";
   int age = 22;
   public void display(){
      Sub obj = new Sub();
      System.out.println("Name: "+obj.name);
      System.out.println("age: "+obj.age);
   }
}
public class FieldHiding{
   public static void main(String args[]){
      new Sub().display();
   }
}
```

Output

```
Name: Vishnu
age: 22
```

Garbage collection in Java

In java, garbage means unreferenced objects.

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.

Advantage of Garbage Collection

- It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
- It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

How can an object be unreferenced?

There are many ways:

- By nulling the reference
- By assigning a reference to another
- By anonymous object etc.

Garbage collection in Java

1) By nulling a reference:

```
Employee e=new Employee();
e=null;
```

2) By assigning a reference to another:

```
Employee e1=new Employee();
Employee e2=new Employee();
e1=e2;//now the first object referred by e1 is available for garbage collection
```

3) By anonymous object:

```
new Employee();
```

PRACTICAL: 01

Install a Java Development Kit /Net beans software

At the end of this chapter/session students will be able to Install a Java Development Kit /Net beans software .

Kit

First Java Program | Hello World Example

In this page, we will learn how to write the simple program of java. We can write a simple hello java program easily after installing the JDK.

To create a simple java program, you need to create a class that contains the main method. Let's understand the requirement first.

- Software Requireme
- Creating Hello Java
- Resolving javac is n

The requirement for Java Hello World Example

For executing any java program, you need to

- Install the JDK if you don't have installed it, download the JDK and install it.
- Set path of the jdk/bin directory. http://www.javatpoint.com/how-to-set-path-in-java
- Create the java program
- Compile and run the java program

Kit

Steps

1. Click the “Download” button beneath “**JDK**.” This will open a new page containing several download options.
2. Scroll to the latest version of **Java** SE **Development Kit**. ...
3. Click “Accept License Agreement.” ...
4. Click the download link next to your operating system.
5. **Install** the **JDK** on your computer.

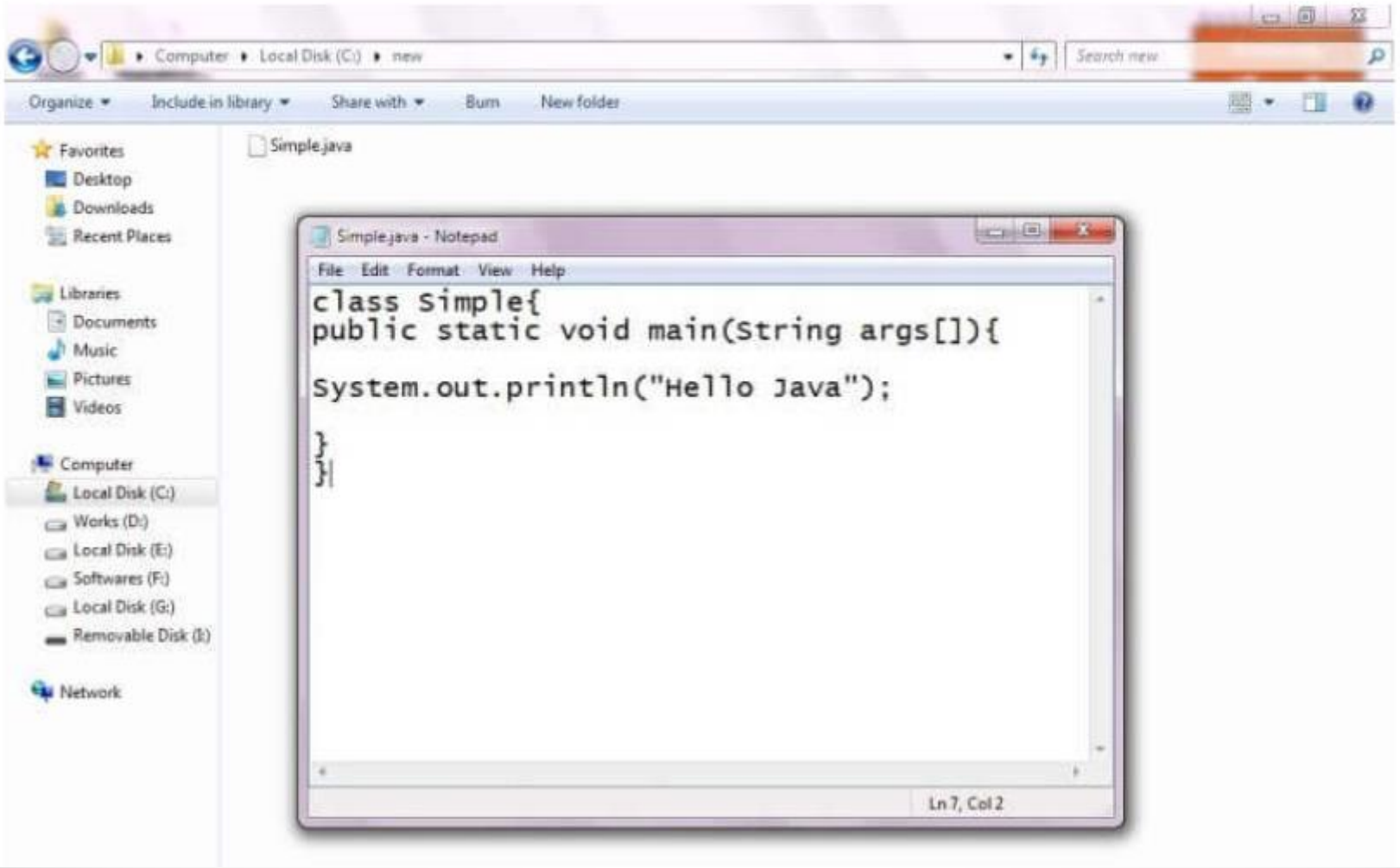
PRACTICAL: 02

Write and execute java program for displaying text messages

At the end of this chapter/session students will be able to Write and execute java program for displaying text messages.

messages

To write the simple program, you need to open notepad by **start menu -> All Programs -> Accessories -> notepad** and write a simple program as displayed below:



program for displaying text messages

Creating Hello World Example

Let's create the hello java program:

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

Test it Now

save this file as Simple.java

To compile:	javac Simple.java
To execute:	java Simple

```
Output:Hello Java
```

PRACTICAL: 03

Write and execute java programs using arrays and control flow statements

At the end of this chapter/session students will be able to Write and execute java programs using arrays and control flow statements .

For loop example to iterate an array:

Here we are iterating and displaying array elements using the for loop.

```
class ForLoopExample3 {
    public static void main(String args[]){
         int arr[]={2,11,45,9};
         //i starts with 0 as array index starts with 0 too
         for(int i=0; i<arr.length; i++){
              System.out.println(arr[i]);
         }
    }
}
```

Output:

```
2
11
45
9
```

PRACTICAL: 04

Write and execute java programs using class, object, method and constructor

At the end of this chapter/session students will be able to Write and execute java programs using class, object, method and constructor .

and constructor

```
//Let us see another example of default constructor
//which displays the default values
class Student3{
int id;
String name;
//method to display the value of id and name
void display(){System.out.println(id+" "+name);}

public static void main(String args[]){
//creating objects
Student3 s1=new Student3();
Student3 s2=new Student3();
//displaying values of the object
s1.display();
s2.display();
}
}
```

Test it Now

Output:

```
0 null
0 null
```

PRACTICAL: 05

Compile and run your program using Ant, Maven, Gradle packaging tool in Java.

At the end of this chapter/session students will be able to
Compile and run your program using Ant, Maven, Gradle packaging tool in Java .

Compile and run your program using Ant, Maven, Gradle packaging tool in Java

Ant with Ivy

Ivy dependencies need to be specified in the ivy.xml file. Our example is fairly simple and requires only JUnit and Hamcrest dependencies.

[ivy.xml]

```
<ivy-module version="2.0">
    <info organisation="org.apache" module="java-build-tools"/>
    <dependencies>
        <dependency org="junit" name="junit" rev="4.11"/>
        <dependency org="org.hamcrest" name="hamcrest-all" rev="1.3"/>
    </dependencies>
</ivy-module>
```

Now we'll create our Ant build script. Its task will be only to compile a JAR file. The end result is the following build.xml.

Compile and run your program using Ant, Maven, Gradle packaging tool in Java

```
<project xmlns:ivy="antlib:org.apache.ivy.ant" name="java-build-tools" default="ja

    <property name="src.dir" value="src"/>
    <property name="build.dir" value="build"/>
    <property name="classes.dir" value="${build.dir}/classes"/>
    <property name="jar.dir" value="${build.dir}/jar"/>
    <property name="lib.dir" value="lib" />
    <path id="lib.path.id">
        <fileset dir="${lib.dir}" />
    </path>

    <target name="resolve">
        <ivy:retrieve />
    </target>

    <target name="clean">
        <delete dir="${build.dir}"/>
    </target>
```

Compile and run your program using Ant, Maven, Gradle packaging tool in Java

```
    <target name="compile" depends="resolve">
        <mkdir dir="${classes.dir}"/>
        <javac srcdir="${src.dir}" destdir="${classes.dir}" classpathref="lib.path
    </target>

    <target name="jar" depends="compile">
        <mkdir dir="${jar.dir}"/>
        <jar destfile="${jar.dir}/${ant.project.name}.jar" basedir="${classes.dir}
    </target>

</project>
```

First we specify several properties. From there on it is one task after another. We use Ivy to resolve dependencies, clean, compile and, finally, create the JAR file. That is quite a lot of configuration for a task that almost every Java project needs to perform.

To run the Ant task that creates the JAR file, execute following.

```
ant jar
```

Maven

[pom.xml]

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <groupId>com.technologyconversations</groupId>
    <artifactId>java-build-tools</artifactId>
    <packaging>jar</packaging>
    <version>1.0</version>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.11</version>
        </dependency>
```

```
        <dependency>
            <groupId>org.hamcrest</groupId>
            <artifactId>hamcrest-all</artifactId>
            <version>1.3</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.3.2</version>
            </plugin>
        </plugins>
    </build>

</project>
```

To run the Maven goal that creates the JAR file, execute following.

```
mvn package
```

Gradle

[build.gradle]

```
apply plugin: 'java'
apply plugin: 'checkstyle'
apply plugin: 'findbugs'
apply plugin: 'pmd'

version = '1.0'

repositories {
    mavenCentral()
}

dependencies {
    testCompile group: 'junit', name: 'junit', version: '4.11'
    testCompile group: 'org.hamcrest', name: 'hamcrest-all', version: '1.3'
}
```

Not only that the Gradle code is much shorter and, to those familiar with Gradle, easier to understand than Maven, but it actually introduces many useful tasks not covered with the Maven code we just wrote. To get the list of all tasks that Gradle can run with the current configuration, please execute the following.

```
gradle tasks --all
```

PRACTICAL: 06

Write and execute java programs using inheritance and polymorphism

At the end of this chapter/session students will be able to
Write and execute java programs using inheritance and polymorphism.

```java
class Animal {
  public void animalSound() {
    System.out.println("The animal makes a sound");
  }
}

class Pig extends Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
}

class Dog extends Animal {
  public void animalSound() {
    System.out.println("The dog says: bow wow");
  }
}
```

PRACTICAL: 07

Write and execute java programs using package

At the end of this chapter/session students will be able to
Write and execute java programs using package.

```java
//save as Simple.java
package mypack;
public class Simple{
 public static void main(String args[]){
    System.out.println("Welcome to package");
   }
}
```

To Compile:

e:\sources> javac -d c:\classes Simple.java

To Run:

To run this program from e:\source directory, you need to set classpath of the directory where the class file resides.

e:\sources> set classpath=c:\classes;.;

e:\sources> java mypack.Simple

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

Package p=Package.getPackage("java.lang");

System.out.println("package name: "+p.getName());

System.out.println("Specification Title: "+p.getSpecificationTitle());
System.out.println("Specification Vendor: "+p.getSpecificationVendor());
System.out.println("Specification Version: "+p.getSpecificationVersion());

System.out.println("Implementaion Title: "+p.getImplementationTitle());
System.out.println("Implementation Vendor: "+p.getImplementationVendor());
System.out.println("Implementation Version: "+p.getImplementationVersion());
System.out.println("Is sealed: "+p.isSealed());


 }
}
```

PRACTICAL: 08

Write and execute java programs using interface

At the end of this chapter/session students will be able to
Write and execute java programs using interface.

```
interface printable{
void print();
}
class A6 implements printable{
public void print(){System.out.println("Hello");}

public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }
}
```

Test it Now

Output:

```
Hello
```

PRACTICAL: 09

Write and execute java programs using multithreaded programming method

At the end of this chapter/session students will be able to
Write and execute java programs using multithreaded programming method .

```java
public class TestThreadTwice1 extends Thread{
 public void run(){
   System.out.println("running...");
 }
 public static void main(String args[]){
  TestThreadTwice1 t1=new TestThreadTwice1();
  t1.start();
  t1.start();
 }
}
```

Test it Now

```
running
Exception in thread "main" java.lang.IllegalThreadStateException
```

PRACTICAL: 10

Write and execute java programs using I/O operation

At the end of this chapter/session students will be able to
Write and execute java programs using I/O operation .

```java
import java.io.FileInputStream;
public class DataStreamExample {
    public static void main(String args[]){
        try{
          FileInputStream fin=new FileInputStream("D:\\testout.txt");
          int i=fin.read();
          System.out.print((char)i);

          fin.close();
        }catch(Exception e){System.out.println(e);}
       }
      }
```

```java
import java.io.FileOutputStream;
public class FileOutputStreamExample {
    public static void main(String args[]){
          try{
            FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
            fout.write(65);
            fout.close();
            System.out.println("success...");
           }catch(Exception e){System.out.println(e);}
     }
}
```

Thanks