OOPs (Object Oriented Programming System)

**Object** means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

* Object
* Class
* Inheritance
* Polymorphism
* Abstraction
* Encapsulation

## Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

## Class

**Collection of objects** is called class. It is a logical entity.

#### Inheritance

**When one object acquires all the properties and behaviours of parent object** i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.



#### Polymorphism

When **one task is performed by different ways** i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

#### Abstraction

**Hiding internal details and showing functionality** is known as abstraction. For example: phone call, we don't know the internal processing.

In java, we use abstract class and interface to achieve abstraction.



#### Encapsulation

**Binding (or wrapping) code and data together into a single unit is known as encapsulation**. For example: capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

## Advantage of OOPs over Procedure-oriented programming language

|  |
| --- |
| 1)OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows. |
| 2)OOPs provides data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere. |
| 3)OOPs provides ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language. |

|  |  |
| --- | --- |
| Global Data | Object Data |

## What is difference between object-oriented programming language and object-based programming language?

Object based programming language follows all the features of OOPs except Inheritance. JavaScript and VBScript are examples of object based programming languages.

# Java Naming conventions

Java **naming convention** is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method etc.

But, it is not forced to follow. So, it is known as convention not rule.

All the classes, interfaces, packages, methods and fields of java programming language are given according to java naming convention.

## Advantage of naming conventions in java

By using standard Java naming conventions, you make your code easier to read for yourself and for other programmers. Readability of Java program is very important. It indicates that **less time** is spent to figure out what the code does.

|  |  |
| --- | --- |
| **Name** | **Convention** |
| class name | should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc. |
| interface name | should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc. |
| method name | should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc. |
| variable name | should start with lowercase letter e.g. firstName, orderNumber etc. |
| package name | should be in lowercase letter e.g. java, lang, sql, util etc. |
| constants name | should be in uppercase letter. e.g. RED, YELLOW, MAX\_PRIORITY etc. |

## CamelCase in java naming conventions

Java follows camelcase syntax for naming the class, interface, method and variable.

If name is combined with two words, second word will start with uppercase letter always e.g. actionPerformed(), firstName, ActionEvent, ActionListener etc.

### 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**

### Syntax to declare a class:

1. class <class\_name>{
2. field;
3. method;
4. }



### Instance variable in Java

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

### Method in Java

In java, a method is like function i.e. used to expose behavior of an object.

#### Advantage of Method

* Code Reusability
* Code Optimization

### new keyword in Java

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

### Object and Class Example: main within class

In this example, we have created a Student class that have two data members id and name. We are creating the object of the Student class by new keyword and printing the objects value.

Here, we are creating main() method inside the class.

File: Student.java

1. class Student{
2. int id;//field or data member or instance variable
3. String name;
5. public static void main(String args[]){
6. Student s1=new Student();//creating an object of Student
7. System.out.println(s1.id);//accessing member through reference variable
8. System.out.println(s1.name);
9. }

}

### Object and Class Example: main outside class

In real time development, we create classes and use it from another class. It is a better approach than previous one. Let's see a simple example, where we are having main() method in another class.

We can have multiple classes in different java files or single java file. If you define multiple classes in a single java source file, it is a good idea to save the file name with the class name which has main() method.

File: TestStudent1.java

1. class Student{
2. int id;
3. String name;
4. }
5. class TestStudent1{
6. public static void main(String args[]){
7. Student s1=new Student();
8. System.out.println(s1.id);
9. System.out.println(s1.name);
10. }
11. }

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

File: TestStudent2.java

1. class Student{
2. int id;
3. String name;
4. }
5. class TestStudent2{
6. public static void main(String args[]){
7. Student s1=new Student();
8. s1.id=101;
9. s1.name="Sonoo";
10. System.out.println(s1.id+" "+s1.name);//printing members with a white space
11. }
12. }

We can also create multiple objects and store information in it through reference variable.

*File: TestStudent3.java*

1. **class** Student{
2. **int** id;
3. String name;
4. }
5. **class** TestStudent3{
6. **public** **static** **void** main(String args[]){
7. //Creating objects
8. Student s1=**new** Student();
9. Student s2=**new** Student();
10. //Initializing objects
11. s1.id=101;
12. s1.name="Sonoo";
13. s2.id=102;
14. s2.name="Amit";
15. //Printing data
16. System.out.println(s1.id+" "+s1.name);
17. System.out.println(s2.id+" "+s2.name);
18. }
19. }

### 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

1. class Student{
2. int rollno;
3. String name;
4. void insertRecord(int r, String n){
5. rollno=r;
6. name=n;
7. }
8. void displayInformation(){System.out.println(rollno+" "+name);}
9. }
10. class TestStudent4{
11. public static void main(String args[]){
12. Student s1=new Student();
13. Student s2=new Student();
14. s1.insertRecord(111,"Karan");
15. s2.insertRecord(222,"Aryan");
16. s1.displayInformation();
17. s2.displayInformation();
18. }
19. }

### 3) Object and Class Example: Initialization through 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

1. class Employee{
2. int id;
3. String name;
4. float salary;
5. void insert(int i, String n, float s) {
6. id=i;
7. name=n;
8. salary=s;
9. }
10. void display(){System.out.println(id+" "+name+" "+salary);}
11. }
12. public class TestEmployee {
13. public static void main(String[] args) {
14. Employee e1=new Employee();
15. Employee e2=new Employee();
16. Employee e3=new Employee();
17. e1.insert(101,"ajeet",45000);
18. e2.insert(102,"irfan",25000);
19. e3.insert(103,"nakul",55000);
20. e1.display();
21. e2.display();
22. e3.display();
23. }
24. }

### Object and Class Example: Rectangle

There is given another example that maintains the records of Rectangle class.

File: TestRectangle1.java

1. class Rectangle{
2. int length;
3. int width;
4. void insert(int l, int w){
5. length=l;
6. width=w;
7. }
8. void calculateArea(){System.out.println(length\*width);}
9. }
10. class TestRectangle1{
11. public static void main(String args[]){
12. Rectangle r1=new Rectangle();
13. Rectangle r2=new Rectangle();
14. r1.insert(11,5);
15. r2.insert(3,15);
16. r1.calculateArea();
17. r2.calculateArea();
18. }
19. }

## What are the different ways to create an object in Java?

There are many ways to create an object in java. They are:

* By new keyword
* By newInstance() method
* By clone() method
* By deserialization
* By factory method etc.

We will learn these ways to create object later.

## Anonymous object

Anonymous simply means nameless. An object which has no reference is known as anonymous object. It can be used at the time of object creation only.

If you have to use an object only once, anonymous object is a good approach. For example:

1. new Calculation();//anonymous object



Calling method through reference:

1. Calculation c=new Calculation();
2. c.fact(5);



Calling method through anonymous object

1. new Calculation().fact(5);



Let's see the full example of anonymous object in java.

1. class Calculation{
2. void fact(int  n){
3. int fact=1;
4. for(int i=1;i<=n;i++){
5. fact=fact\*i;
6. }
7. System.out.println("factorial is "+fact);
8. }
9. public static void main(String args[]){
10. new Calculation().fact(5);//calling method with anonymous object
11. }
12. }



Output:

Factorial is 120

### Creating multiple objects by one type only

We can create multiple objects by one type only as we do in case of primitives.

Initialization of primitive variables:

1. int a=10, b=20;



Initialization of refernce variables:

1. Rectangle r1=new Rectangle(), r2=new Rectangle();//creating two objects



Let's see the example:

1. class Rectangle{
2. int length;
3. int width;
4. void insert(int l,int w){
5. length=l;
6. width=w;
7. }
8. void calculateArea(){System.out.println(length\*width);}
9. }
10. class TestRectangle2{
11. public static void main(String args[]){
12. Rectangle r1=new Rectangle(),r2=new Rectangle();//creating two objects
13. r1.insert(11,5);
14. r2.insert(3,15);
15. r1.calculateArea();
16. r2.calculateArea();
17. }
18. }

### Real World Example: Account

File: TestAccount.java

1. class Account{
2. int acc\_no;
3. String name;
4. float amount;
5. void insert(int a,String n,float amt){
6. acc\_no=a;
7. name=n;
8. amount=amt;
9. }
10. void deposit(float amt){
11. amount=amount+amt;
12. System.out.println(amt+" deposited");
13. }
14. void withdraw(float amt){
15. if(amount<amt){
16. System.out.println("Insufficient Balance");
17. }else{
18. amount=amount-amt;
19. System.out.println(amt+" withdrawn");
20. }
21. }
22. void checkBalance(){System.out.println("Balance is: "+amount);}
23. void display(){System.out.println(acc\_no+" "+name+" "+amount);}
24. }
26. class TestAccount{
27. public static void main(String[] args){
28. Account a1=new Account();
29. a1.insert(832345,"Ankit",1000);
30. a1.display();
31. a1.checkBalance();
32. a1.deposit(40000);
33. a1.checkBalance();
34. a1.withdraw(15000);
35. a1.checkBalance();
36. }}