# Java OOPs Concepts

* Simula == first object oriented programming language
* Object - oriented Programming language : programming paradigm where everything is represented as an object .
* Main aim of the OOP is to implement real world entities such as object , classes , abstraction etc.
* OOP is a methodology or paradigm to design a program using classes and objects .
* It simplified software development and maintenance by provisioning some concepts
* Object means real world entity such as Chair , table , computer etc.
* OOPs concepts include :
  + [Object](https://www.javatpoint.com/object-and-class-in-java)
  + Class
  + [Inheritance](https://www.javatpoint.com/inheritance-in-java)
  + [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
  + [Abstraction](https://www.javatpoint.com/abstract-class-in-java)
  + [Encapsulation](https://www.javatpoint.com/encapsulation)

Apart from these concepts, there are some other terms which are used in Object-Oriented design:

* Coupling
* Cohesion
* Association
* Aggregation
* Composition

### **Object :**

* Any entity that has state and behavior is known as Object
* Example : pen , table , chair etc.
* It can be physical and logical
* Object can be defined as Instance of class
* Object contains an address and takes up some space in memory
* Example : Dog is an Object as it has ste like color , name , breed etc. as well as behaviour such as wagging a tail , barking , eating etc.

### **Class :**

* Collection of object
* Logical Entity
* Blueprint from which we can create individual object
* Doesnot consume space

### **Inheritance :**

* When one object acquires all the the property and behaviour of parent object , it is called inheritance
* Advantage : code reusablitiy
* Used to achieve runtime polymorphism

### **Polymorphism :**

* If one task performed in diff ways then it is called polymorphism
* Ex. convincing a customer to draw diff things like circle , triangle , square etc.
* In java we use Method Overloading and Method overriding to achieve Polymorphism

### **Abstraction :**

* Hiding Internal details and showing functionality is called Abstraction
* Ex . Phone call , we don't know th internal processing
* In java , we use Abstract classes and Interface to achieve Abstraction

### **Encapsulation :**

* Binding or wrapping code and data together into a single unit are known as Encapsulation
* Java class is an ex. Of Encapsulation
* Java bean is fully encapsulated class bcz all daat members are private here

### **Coupling :**

* Coupling refers to the knowledge or info or dependency of another class
* It arises when classes are aware of each other
* If a class has the details information of another class, there is strong coupling.
* In Java, we use private, protected, and public modifiers to display the visibility level of a class, method, and field.
* You can use interfaces for the weaker coupling because there is no concrete implementation.

### **Cohesion :**

* Cohesion refers to the level of a component which performs a single well-defined task.
* A single well-defined task is done by a highly cohesive method.
* The weakly cohesive method will split the task into separate parts.
* The java.io package is a highly cohesive package because it has I/O related classes and interface.
* However, the java.util package is a weakly cohesive package because it has unrelated classes and interfaces.

### **Association :**

* Association represents the relationship between the objects.
* Here, one object can be associated with one object or many objects.
* There can be four types of association between the objects:
* One to One
* One to Many
* Many to One, and
* Many to Many
* For example, One country can have one prime minister (one to one), and a prime minister can have many ministers (one to many). Also, many MP's can have one prime minister (many to one), and many ministers can have many departments (many to many).
* Association can be undirectional or bidirectional.

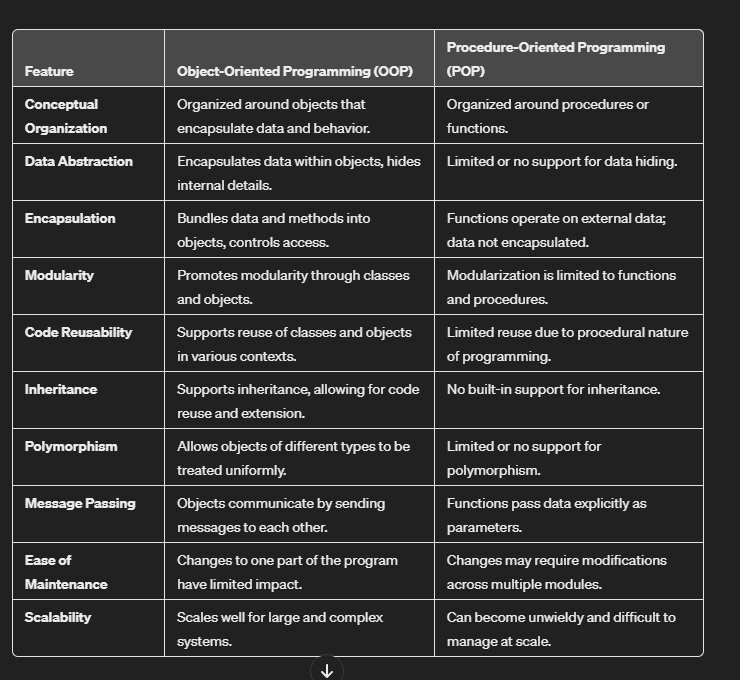
### **Aggregation :**

* Aggregation is a way to achieve Association.
* Aggregation represents the relationship where one object contains other objects as a part of its state.
* It represents the weak relationship between objects. It is also termed as a *has-a* relationship in Java.
* Like, inheritance represents the *is-a* relationship. It is another way to reuse objects.

### **Composition :**

* The composition is also a way to achieve Association.
* The composition represents the relationship where one object contains other objects as a part of its state.
* There is a strong relationship between the containing object and the dependent object.
* It is the state where containing objects do not have an independent existence.
* If you delete the parent object, all the child objects will be deleted automatically.

## Advantage of OOPs over Procedure-oriented programming language

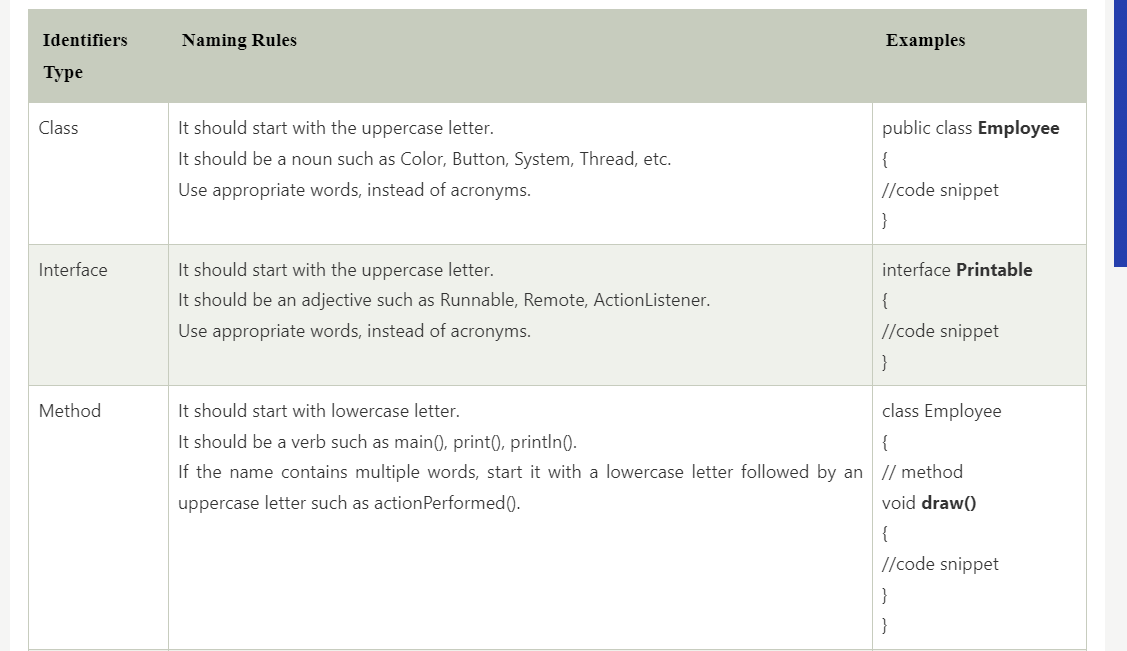


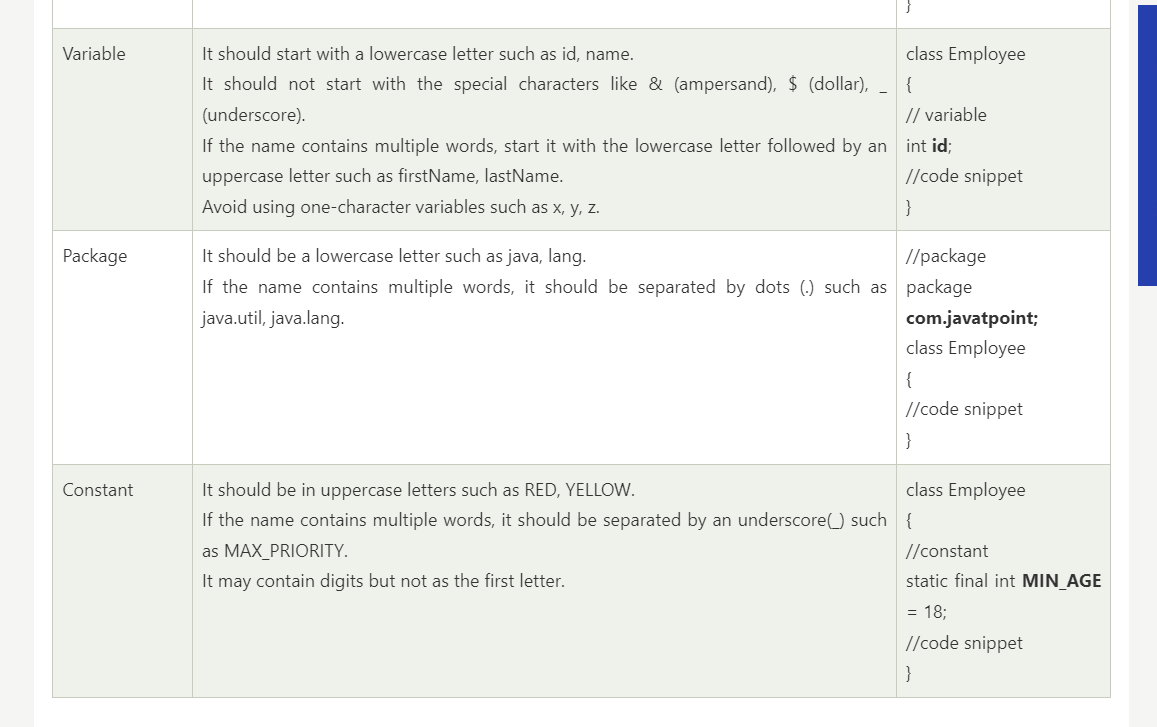
## 

# Java Naming Convention

Java follows camel-case syntax for naming the class, interface, method, and variable.

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





## Advantages of Java OOPs

The following are the advantages of using the OOPs in Java:

* The implementations of OOPs concepts are easier.
* The execution of the OOPs is faster than procedural-oriented programming.
* OOPs provide code reusability so that a programmer can reuse an existing code.
* OOPs help us to keep the important data hidden.

# **Java - Object and Classes**

### **Java Classes :**

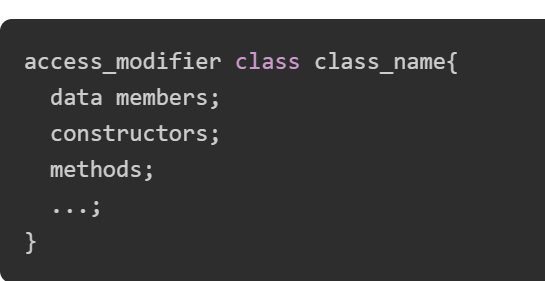
* Blueprint from which the individual object are created
* A class is a group of objects which have common properties
* It is a logical entity. It can't be physical.
* Occupies NO SPACE in memory
* A class in Java can contain:
* Fields
* Methods
* Constructors
* Blocks
* Nested class and interface

* A class can contain any of the following variable types.
* **Local variables** − Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
* **Instance variables** − Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
* **Class variables** − Class variables are variables declared within a class, outside any method, with the static keyword.

##### **Creating (Declaring) a Java Class**

##### To create (declare) a class, you need to use [access modifiers](https://www.tutorialspoint.com/java/java_access_modifiers.htm) followed by class keyword and class\_name.

* Syntax :



* Example :

// Creating a Java class

class Dog {

// Declaring and initializing the attributes

String breed;

int age;

String color;

// methods to set breed, age, and color of the dog

public void setBreed(String breed) {

this.breed = breed;

}

public void setAge(int age) {

this.age = age;

}

public void setColor(String color) {

this.color = color;

}

// method to print all three values

public void printDetails() {

System.out.println("Dog detials:");

System.out.println(this.breed);

System.out.println(this.age);

System.out.println(this.color);

}

}

### **Java Objects**

* An **object** is a variable of the type **class**, it is a basic component of an object-oriented programming system.
* A class has the methods and data members (attributes), these methods and data members are accessed through an **object**.
* Thus, an object is an instance of a class.
* All these objects have a state and a behavior.
* If we consider a dog, then its state is - name, breed, and color, and the behavior is - barking, wagging the tail, and running.
* If you compare the software object with a real-world object, they have very similar characteristics. Software objects also have a state and a behavior. A software object's state is stored in fields and behavior is shown via methods.
* So, in software development, methods operate on the internal state of an object, and the object-to-object communication is done via methods.

**Creating (Declaring) a Java Object :**

* In Java, the new keyword is used to create new objects.
* There are three steps when creating an object from a class −
  + Declaration − A variable declaration with a variable name with an object type.
  + Instantiation − The 'new' keyword is used to create the object.
  + Initialization − The 'new' keyword is followed by a call to a constructor. This call initializes the new object.
* Syntax :



Note: parameters are optional and can be used while you're using [constructors](https://www.tutorialspoint.com/java/java_constructors.htm) in the class.

* **Example to Create a Java Object**

**// Creating a Java class**

**class Dog {**

**// Declaring and initializing the attributes**

**String breed;**

**int age;**

**String color;**

**// methods to set breed, age, and color of the dog**

**public void setBreed(String breed) {**

**this.breed = breed;**

**}**

**public void setAge(int age) {**

**this.age = age;**

**}**

**public void setColor(String color) {**

**this.color = color;**

**}**

**// method to print all three values**

**public void printDetails() {**

**System.out.println("Dog detials:");**

**System.out.println(this.breed);**

**System.out.println(this.age);**

**System.out.println(this.color);**

**}**

**}**

**public class Main {**

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

**// Creating an object of the class Dog**

**Dog obj = new Dog();**

**// setting the attributes**

**obj.setBreed("Golden Retriever");**

**obj.setAge(2);**

**obj.setColor("Golden");**

**// Printing values**

**obj.printDetails();**

**}**

**}**

#### Output

Dog detials:

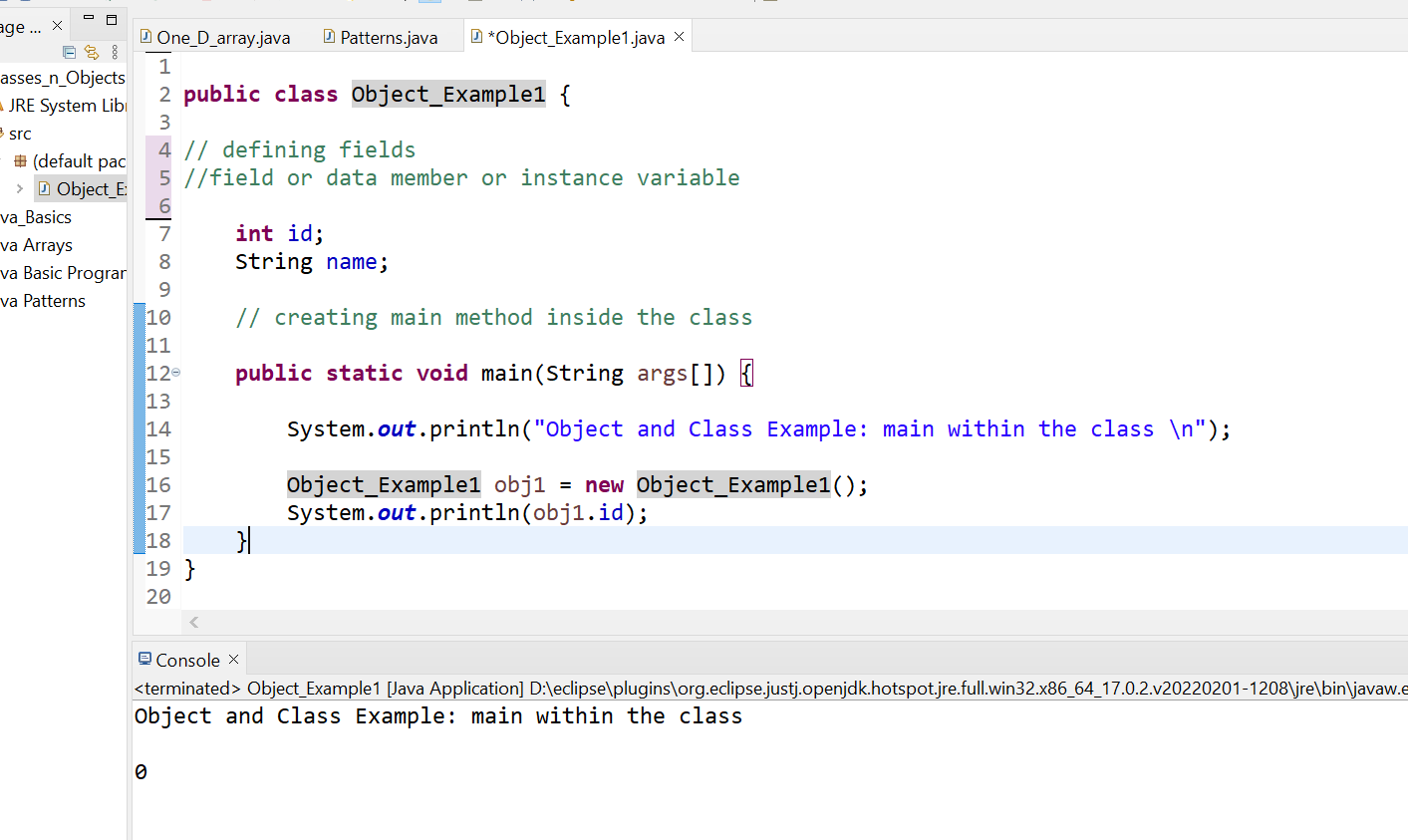
Golden Retriever

2

Golden

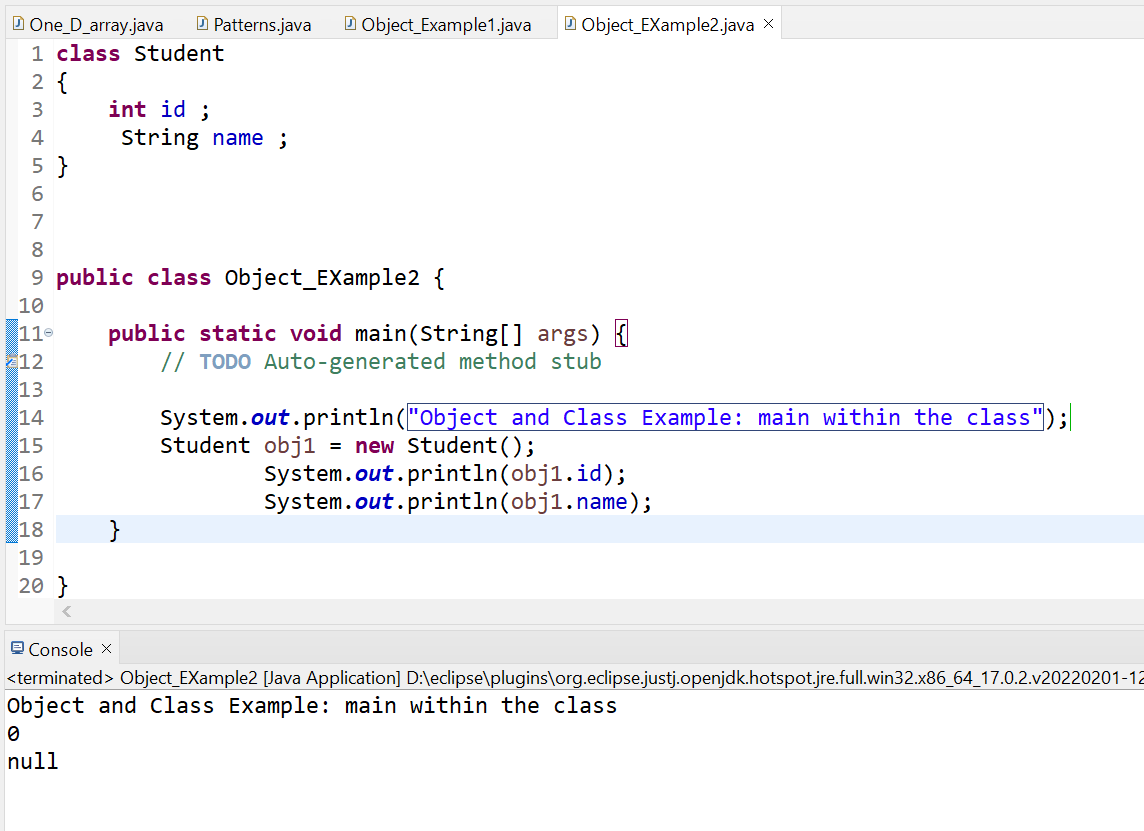
### Object and Class Example: main within the class

In this example, we have created a Object\_Example1 class which has two data members id and name. We are creating the object of the Object\_Example1 class by new keyword and printing the object's value



### Object and Class Example: main outside the 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



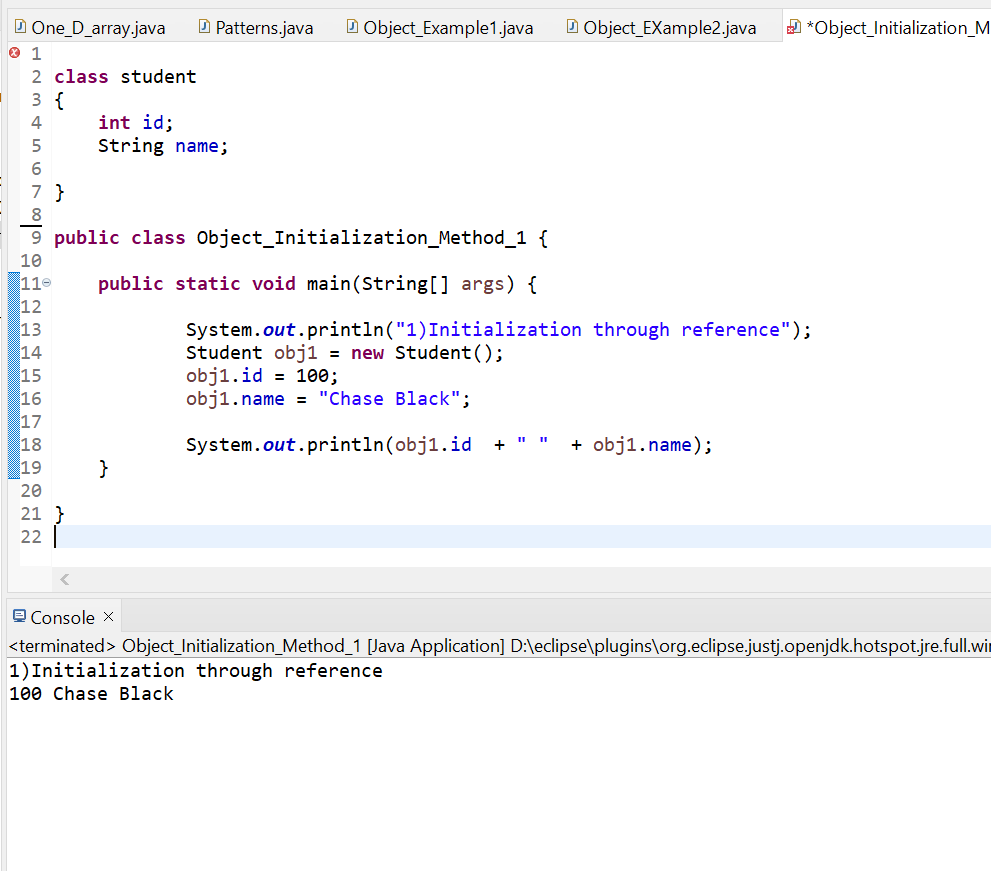
##### **3 Ways to initialize object**

There are 3 ways to initialize object in Java.

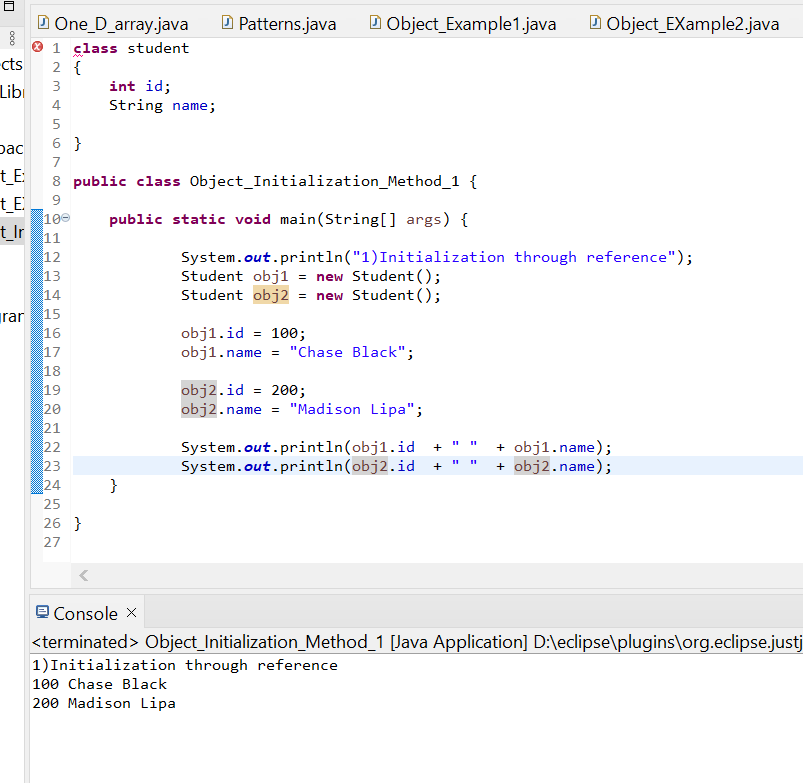
1. By reference variable
2. By method
3. By constructor

### 1) Object and Class Example: Initialization through reference

Initializing an object means storing data into the object.

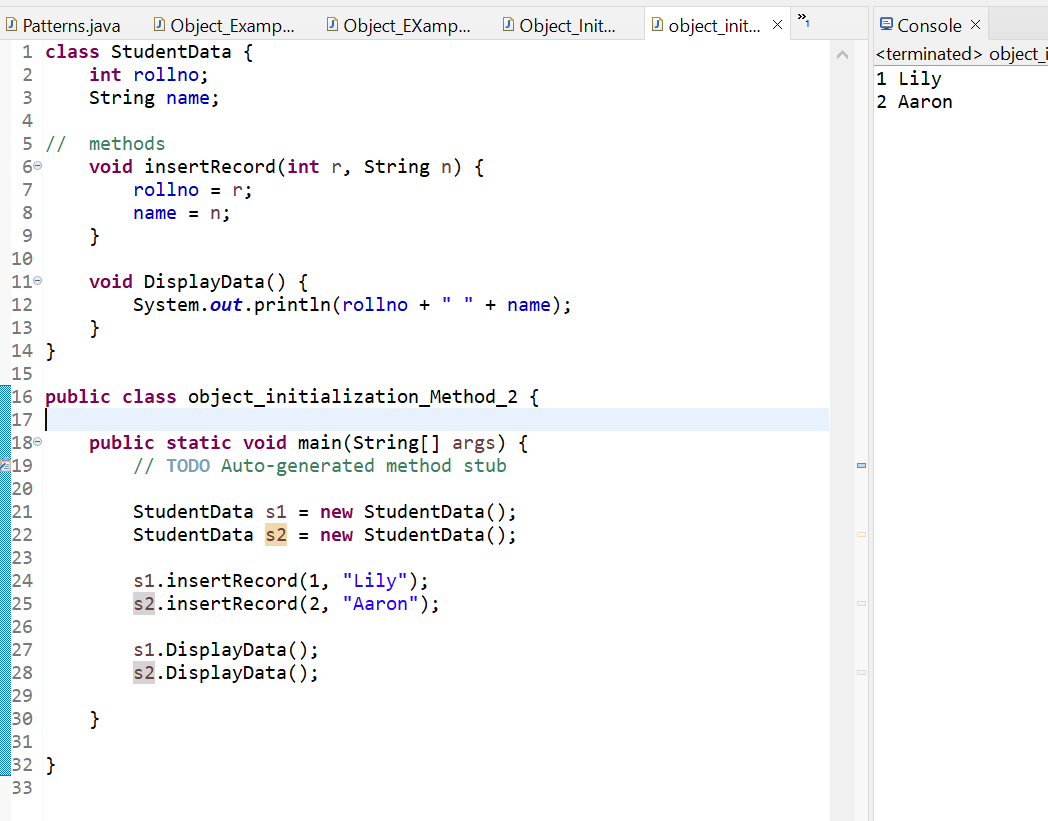


We can also create multiple objects and store information in it through reference variable.



### 2) Object and Class Example: Initialization through method

In this example, we are creating the two objects of StudentData 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.



As you can see in the above figure, object gets the memory in heap memory area. The reference variable refers to the object allocated in the heap memory area. Here, s1 and s2 both are reference variables that refer to the objects allocated in memory.



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

## **DONE LATER ON**

## Anonymous object

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

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

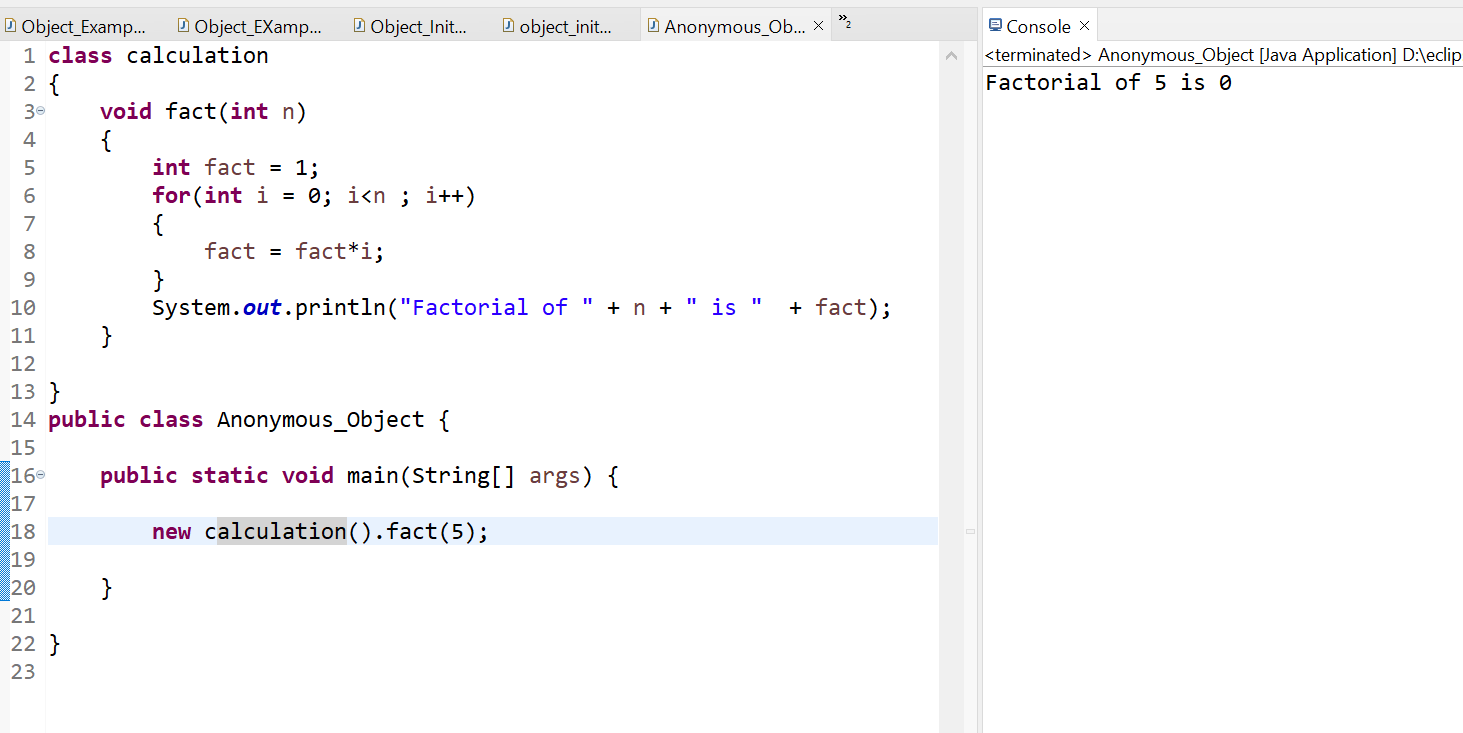
1. **new** Calculation();//anonymous object

Calling method through a reference:

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

Calling method through an anonymous object

1. **new** Calculation().fact(5);



### 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