Date : 13th feb 2024

## **What is Java?**

* Developed By James Gosling at Sun Microsystems in 1995
* It is a Programming language and a Platform.
* Platform: any hardware or software env. Where program runs.
* Since Java has JRE (Java runtime env) and API . so it is a platform

##### **Types of Java Applications :**

1. **Standalone application :**

* Also known as Desktop app or window based app
* Traditional software that we need to install on our system
* Ex. : media player, antivirus

1. **Web application**

* Application that runs on server side and created dynamic page
* Ex. Servlets , JSP , Spring

1. **Enterprise Application**

* Application that is distributed in nature such as bank applications

1. **Mobile Application**

* Application created for mobile device
* Android and JAVA ME are used to create mobile applications

##### **Java Platforms / Editions :**

1. **Java SE (Java standard edition) :**

* Include java programming APIs such as java.lang, java.io, java.net , java.util etc
* Includes core topics like OOPs, String . Regex , Exception

1. **Java EE (Enterprise edition) :**

* Used to develop web and enterprise applications
* Build on top of Java SE platform
* Includes topics like Servlets , JSP, Web services etc.

1. **Java ME (Micro edition) :**

* Micro platform that is dedicated to mobile applications

1. **Java FX :**

* Used to develop rich internet applications
* Uses lightweight user interface API

## Features of Java :

1. **Simple :**

* Easy to learn
* Simple syntax . based on C++
* Clean and easy to understand
* Removed many complicated features like - explicit pointers and operator overloading
* Have automatic garbage collector

1. **Object-Oriented :**

* Object-oriented programming language i.e everything is an object
* Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior.
* Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.
* Basic concepts of OOPs are:
  + Object
  + Class
  + Inheritance
  + Polymorphism
  + Abstraction
  + Encapsulation

1. **Portable :**

* Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

1. **Platform independent :**

* Java is write once , run anywhere so it is not platform dependent
* C, C++ are complied into platform specific machines
* Java is software-based platform which runs on top of other hardware based platforms .
* Java has two components :
  + Runtime env.
  + API
* Java code is complied by compiler and converted into bytecode . this bytecode is platform-independent code

1. **Secured :**

* Java is best known for security
* Java can be used to developed virus free system
* Java is secured by bcz :
  + No explicit pointer
  + Java Programs run inside a virtual machine sandbox
  + **Classloader:** Classloader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
  + **Bytecode Verifier:** It checks the code fragments for illegal code that can violate access rights to objects.
  + **Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.

1. **Robust**

* Robust means strong
* It is robust bcz :
  + Uses string memory management
  + Lack of pointers
  + Automatic garbage collection which runs on JVM
  + Exception handling
  + Type checking mechanism

1. **Architecture neutral :**

* There is no implementation-dependent feature .
* Ex: size of primitive type is fixed
* In c int occupies 2 bytes incase of 32 bit architecture and 4 bytes in case of 64 bit architecture but Java occupies 4 bytes for both 32 and 64 bit

1. **Interpreted**
2. **High Performance :**

* Java is faster as java bytecode is close to native code
* Still little slower than compiler language like C++
* Since java is an interpreted language so it is slower than the compiler language

1. **Multithreaded :**

* Thread is separate program executing concurrently
* We can write Java programs that deal with many tasks at once by defining multiple threads.
* The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area.
* Threads are important for multi-media, Web applications, etc.

1. **Distributed :**

* Distributed because it facilitates user to create distributed applications
* RMI and EJB is used to crete distributed applications

1. **Dynamic :**

* Java supports dynamic loading of classes i.e clases are loaded on demand
* Java supports dynamic compilation and automatic memory management (garbage collection).

## **C++ VS Java**

|  |  |  |
| --- | --- | --- |
| **Comparison Index** | **C++** | **Java** |
| Platform independent | Platform dependent | Platform independent |
| Used for | For system programming | Application programming, window based , web-based, enterprise , mobile applications |
| Goto statement | Support Goto statement | Does not support Goto |
| Multiple inheritance | Supports Multiple inheritance | Does not support Multiple Inheritance |
| Operator overloading | Supports Operator overloading | Doesnot supports operator overloading |
| Pointers | Supports Pointers | Supports pointers internally . but cant write pointers in program explicitly Java restrict pointer support |
| Compiler & interpreter | Uses compiler only | Uses both compiler and interpreter |
| Flow | C++ is compiled and run using the compiler which converts source cod into machine code so it is platform dependent | Java source code is converted into bytecode at compilation time . The interpreter executes this bytecode at runtime and produces output. It is interpreted that is why it is platform-independent |
| Call by value and call by reference | Supports both | Supports only call by value |
| Structure and Union | Supports both | Supports NOne |
| Thread Support | Relies on Third-party libraries for thread support | Have build in thread support |
| Multi-comment / Documentation comment | Does not support | Support documentation comment (/\* \*/) |
| Virtual keyword | Supports virtual keyword so that we can decide whether or not to override a function | Has no virtual keyword . We can override all non static method by default , in other word , non-static methods are virtual by default |

14TH Feb 2024

## **First Java Program | Hello World Example**

To create a simple Java program, you need to create a class that contains the main method.

##### **The requirement for Java Hello World Example :**

##### For executing any Java program, the following software or application must be properly installed.

##### Install the JDK if you don't have installed it, download the JDK and install it.

##### Set path of the jdk/bin directory.

##### Create the Java program

##### Compile and run the Java program

Java Program :

**Class Test{**

**Public static void main (String argos[])**

**{**

**system.out.println(“Hello world”)**

**}**

**}**

##### 

|  |  |
| --- | --- |
| **To compile:** | javac Simple.java |
| **To execute:** | java Simple |

**Parameters used in Java program :**

* **Class :** Keyword used to declare class
* **Public :** Access Modifier that represents visibility . it means it is visible to all
* **Static :** It is a keyword . If we declare any method as static then that method is called static method . Advantage if using static methos us that there is no need to create objects to invoke static method. Main() method is executed by JVM so it doesnot require creating an object to invoke the main method which saves memory
* **Void :** return type . thi doesnt return a value
* **Main :** represents starting point of the program
* **String args[] :** command line argument .\
* **System.out.println :** used to print statement in new line . System is class , out is object of Printstream class and println()is method os Printstream class .

**Valid Ways to write the parameters in program :**

**1) By changing the sequence of the modifiers, method prototype is not changed in Java.**

static public void main(String args[])

**2) The subscript notation in the Java array can be used after type, before the variable or after the variable.**

public static void main(String[] args)

public static void main(String []args)

public static void main(String args[])

**3) You can provide var-args support to the main() method by passing 3 ellipses (dots)**

public static void main(String... args)

**NOTE :**

Variable Arguments (Varargs) in Java is a method that takes a variable number of arguments. Variable Arguments in Java simplifies the creation of methods that need to take a variable number of arguments.

Syntax of Varargs :

public static void fun(int ... a)

{

// method body

}

The … syntax tells the compiler that varargs have been used, and these arguments should be stored in the array referred to by a.

**4) Having a semicolon at the end of class is optional in Java.**

class A{

static public void main(String... args){

System.out.println("hello java4");

}

};

**5) Valid Java main() method signature**

public static void main(String[] args)

public static void main(String []args)

public static void main(String args[])

public static void main(String... args)

static public void main(String[] args)

public static final void main(String[] args)

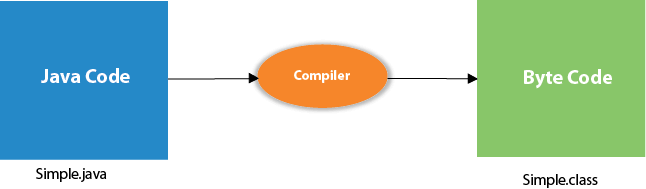
final public static void main(String[] args)

final strictfp public static void main(String[] args)

## **Internal Details of Hello Java Program**

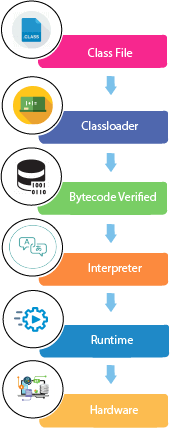
## What happens at compile time?

At compile time, the Java file is compiled by Java Compiler (It does not interact with OS) and converts the Java code into bytecode.



## What happens at runtime?

At runtime, the following steps are performed:



**Classloader:** It is the subsystem of JVM that is used to load class files.

**Bytecode Verifier:** Checks the code fragments for illegal code that can violate access rights to objects.

**Interpreter:** Read bytecode stream then execute the instructions.

### Q) Can you save a Java source file by another name than the class name?

Yes, if the class is not public. It is explained in the figure given below:

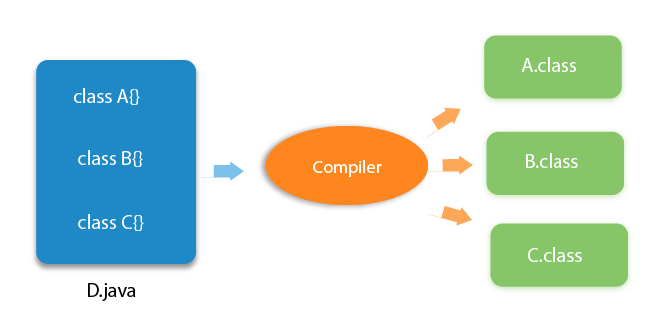


|  |  |
| --- | --- |
| **To compile:** | javac Hard.java |
| **To execute:** | java Simple |

Observe that, we have compiled the code with file name but running the program with class name. Therefore, we can save a Java program other than class name.

### Q) Can you have multiple classes in a java source file?

Yes, like the figure given below illustrates:



# How to set path in Java

The path is required to be set for using tools such as javac, java, etc.

If you are saving the Java source file inside the JDK/bin directory, the path is not required to be set because all the tools will be available in the current directory.

However, if you have your Java file outside the JDK/bin folder, it is necessary to set the path of JDK.

There are two ways to set the path in Java:

1. Temporary
2. Permanent

## 1) How to set the Temporary Path of JDK in Windows

To set the temporary path of JDK, you need to follow the following steps:

* Open the command prompt
* Copy the path of the JDK/bin directory
* Write in command prompt: set path=copied\_path

### For Example:

set path=C:\Program Files\Java\jdk1.6.0\_23\bin

## 2) How to set Permanent Path of JDK in Windows

For setting the permanent path of JDK, you need to follow these steps:

* Go to MyComputer properties -> advanced tab -> environment variables -> new tab of user variable -> write path in variable name -> write path of bin folder in variable value -> ok -> ok -> ok

## **JDK, JRE, and JVM**

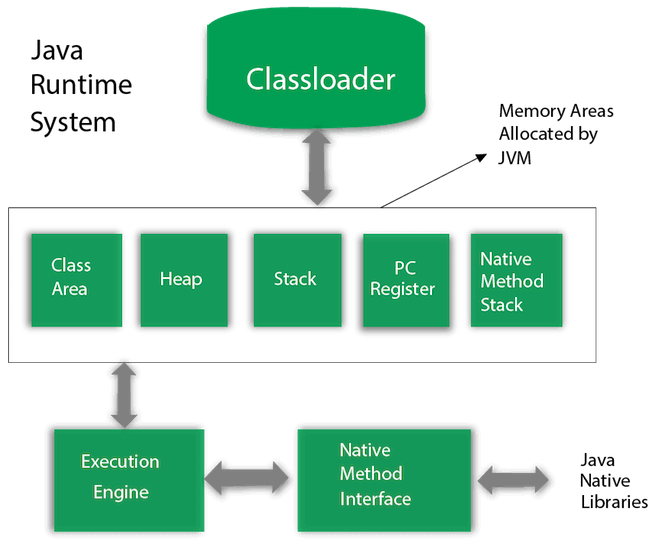
1. **JVM : Java Virtual Machine**

* Abstract machine
* Called virtual machine because it doesn't physically exist
* Provides runtime env. Where java bytecode is executed
* Runs those program also which are written in diff. Language and complied to Java bytecode
* JVM performs the following main task :
  + Loads the code
  + Verifies the code
  + Executes the code
  + Provides the runtime env.
* JVM is platform dependent
* It is:
  + A specification where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Oracle and other companies.
  + An implementation Its implementation is known as JRE (Java Runtime Environment).
  + Runtime Instance Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.
* JVM provides the definition for the
  + Memory area
  + Class file format
  + Register set
  + Garbage collected Heap
  + Fatal error reporting

## 

## **JVM Architecture**

Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc.



**A) Classloader**

Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.

Bootstrap ClassLoader: This is the first classloader which is the super class of Extension classloader. It loads the rt.jar file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes etc.

Extension ClassLoader: This is the child classloader of Bootstrap and parent classloader of System classloader. It loades the jar files located inside $JAVA\_HOME/jre/lib/ext directory.

System/Application ClassLoader: This is the child classloader of Extension classloader. It loads the classfiles from classpath. By default, classpath is set to current directory. You can change the classpath using "-cp" or "-classpath" switch. It is also known as Application classloader.

**B) Class(Method) Area**

Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

**C) Heap**

It is the runtime data area in which objects are allocated.

**D) Stack**

Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

**E) Program Counter Register**

PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.

**F) Native Method Stack**

It contains all the native methods used in the application.

**G) Execution Engine**

It contains:

A virtual processor

Interpreter: Read bytecode stream then execute the instructions.

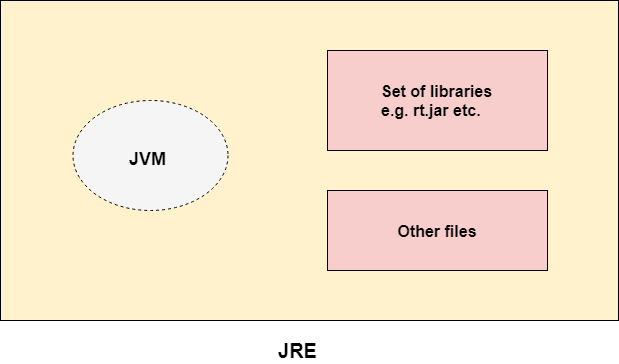
Just-In-Time(JIT) compiler: It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here, the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

**H) Java Native Interface**

Java Native Interface (JNI) is a framework which provides an interface to communicate with another application written in another language like C, C++, Assembly etc. Java uses JNI framework to send output to the Console or interact with OS libraries.

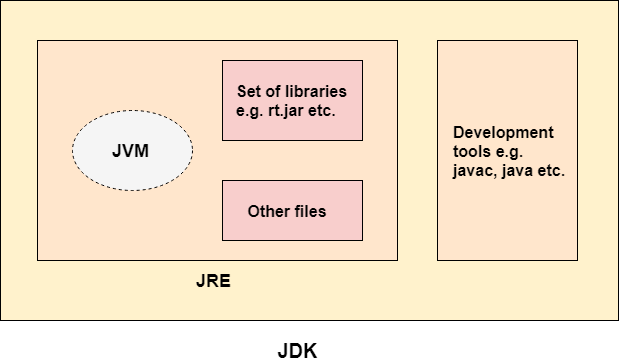
1. **JRE : Java Runtime Environment**

* Also written as Java RTE
* JRE is a set of software tools which are used for developing java applications
* It is the implementation of JVM , it physically exist
* Contains set of libraries + other files that JVM uses at runtime
* The implementation of JVM is also actively released by other companies besides Sun Micro Systems.



1. **JDK : Java Development Kit**

* Software development env. Which is used to develop java applications and applets .
* Physically exist
* Contains JRE + development tools
* JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:
  + Standard Edition Java Platform
  + Enterprise Edition Java Platform
  + Micro Edition Java Platform
* JDK contains private JVM and few other resources such as interpreter/loader(java) , compiler (javac) , archiver(jar) , document generator (javadoc) etc. to complete the development of java applications

****

## **Java Variable :**

* Variable is container that holds value while java program is executed
* Variable is assigned with Data type
* It is a name of memory location
* It is name of reserved area allocated in memory
* Ex : int a = 50 // here a is a variable

**Types of Variable :**

1. Local variables
2. Instance Variable
3. Static Variable
4. **Local variable :**

* Varibel declared inside the body of the method
* Can Use this variable within the method
* Other methods didnt know this variable exist int he method
* Cannot be defined as static

1. **Instance Variable:**

* Varibel declared inside the class but outside the body of the method
* Called instance variable bcz its value is instance- specific and is not shared among instances
* An object that is created using a class is said to be an instance of that class.
* Cannot declared Static

1. **Static variables :**

* Variable desclared as satic
* Cannot be local
* Can create single copy of static variable and share it among all the instance of class
* Momory allocations for static variables happens only once – when the class is loaded in memory

**EX :**

**public class A {**

**static int m=100;//static variable**

**void method()**

**{**

**int n=90;//local variable**

**}**

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

**{**

**int data=50;//instance variable**

**}**

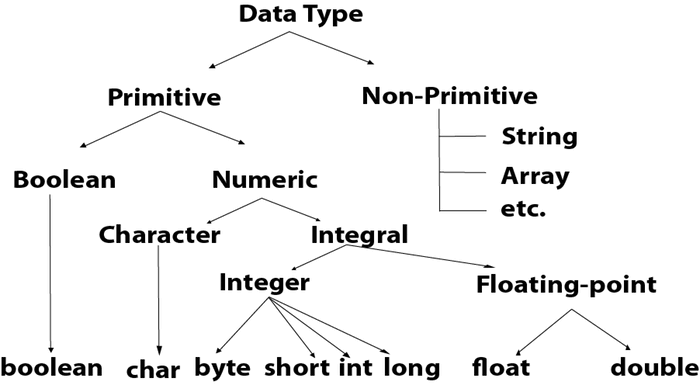
**}//end of class**

## **Data Types in Java :**

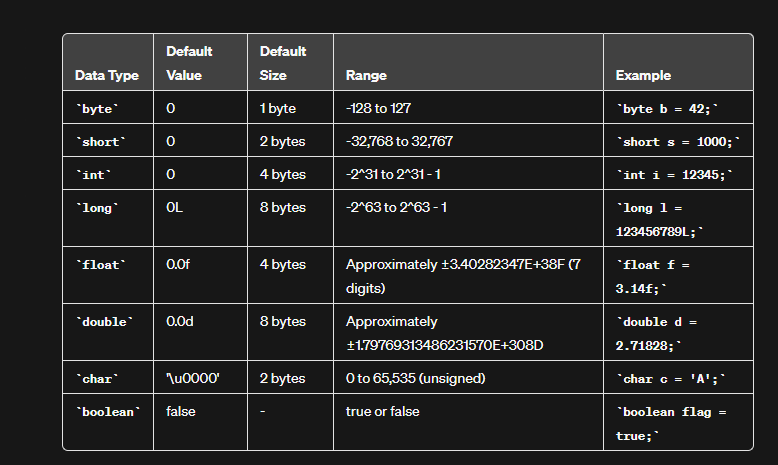
Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include [Classes](https://www.javatpoint.com/object-and-class-in-java), [Interfaces](https://www.javatpoint.com/interface-in-java), and [Arrays](https://www.javatpoint.com/array-in-java).

**NOTE : Java is a statically-typed programming language. It means, all variables must be declared before its use. That is why we need to declare variable's type and name.**



1. **Primitive data types:**



Signed and unsigned are terms used to describe how numbers are represented in binary, indicating whether a number includes a sign (positive or negative) or is strictly positive.

**Signed:**In a signed number representation, a certain bit is used to represent the sign of the number (positive or negative).

For example, in a signed 8-bit representation, the range is typically from -128 to 127. The leftmost bit (most significant bit) is reserved for the sign (0 for positive, 1 for negative).

Signed 8-bit:

* 01111111 (127)
* 00000000 (0)
* 11111111 (-1)
* 10000000 (-128)

**Unsigned:**In an unsigned number representation, all bits are used to represent the magnitude of the number, and there is no dedicated sign bit.

For example, in an unsigned 8-bit representation, the range is from 0 to 255.

Unsigned 8-bit:

* 11111111 (255)
* 00000000 (0)

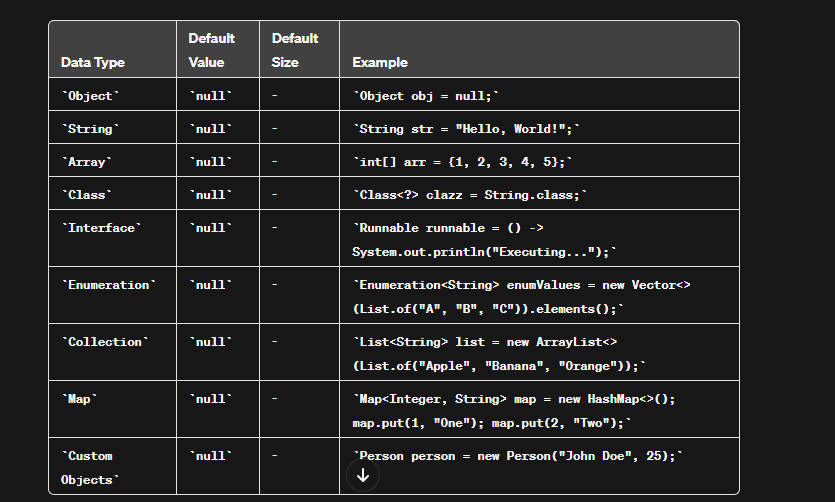
In Java, all primitive data types (except char) are signed. This means that the leftmost bit is used to represent the sign. For example, in a signed byte, the range is from -128 to 127. The char type, although 16 bits, is unsigned and represents Unicode characters ranging from 0 to 65,535.

### Why char uses 2 byte in java and what is \u0000 ?

It is because java uses Unicode system not ASCII code system. The \u0000 is the lowest range of Unicode system. To get detail explanation about Unicode visit next page.

1. **Non-primitive data types:** The non-primitive data types include [Classes](https://www.javatpoint.com/object-and-class-in-java), [Interfaces](https://www.javatpoint.com/interface-in-java), and [Arrays](https://www.javatpoint.com/array-in-java).

* Java non-primitive data types, also known as reference types or objects,
* they have a default value of null



* In the above examples, the default value for all non-primitive types is null. The size of non-primitive types is not fixed and depends on the specific implementation and the data stored in them.

# Unicode System

|  |
| --- |
| Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages. |

### Why java uses Unicode System?

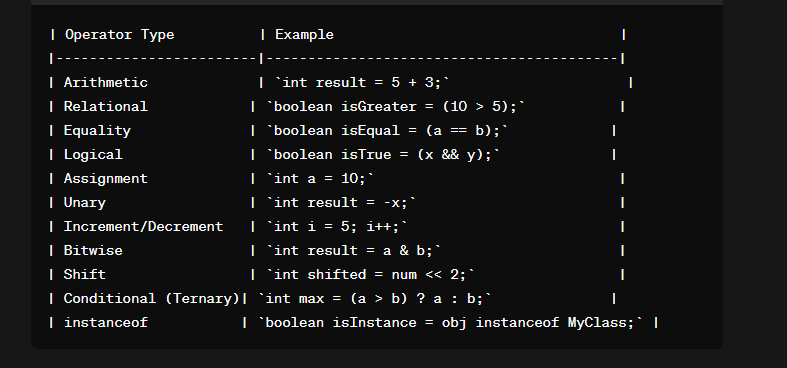
|  |
| --- |
| Before Unicode, there were many language standards: |
| * **ASCII** (American Standard Code for Information Interchange) for the United States. * **ISO 8859-1** for Western European Language. * **KOI-8** for Russian. * **GB18030 and BIG-5** for chinese, and so on. |

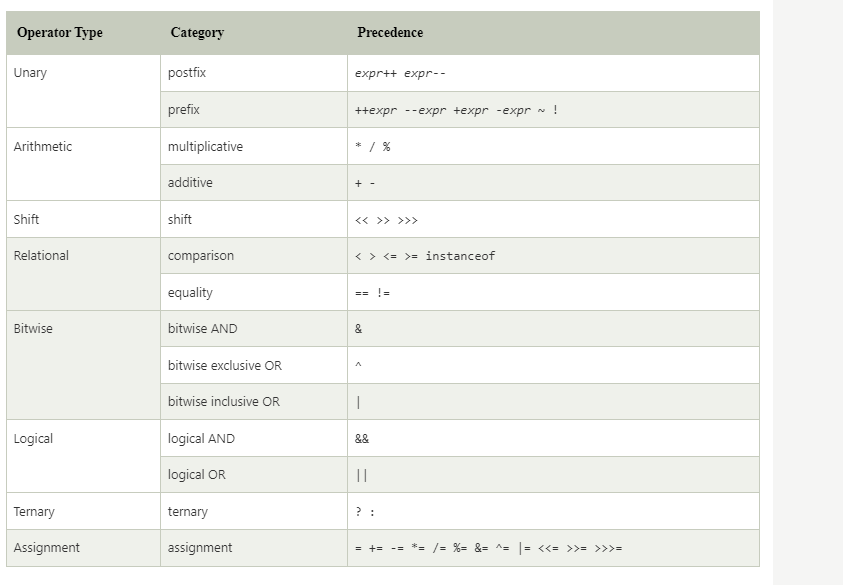
## Problem

|  |
| --- |
| **This caused two problems:**   1. A particular code value corresponds to different letters in the various language standards. 2. The encodings for languages with large character sets have variable length.Some common characters are encoded as single bytes, other require two or more byte. |

## Solution

|  |
| --- |
| To solve these problems, a new language standard was developed i.e. Unicode System. |
| In unicode, character holds 2 byte, so java also uses 2 byte for characters. |
| **lowest value:**\u0000 |
| **highest value:**\uFFFF Operators in Java **Operator** in [Java](https://www.javatpoint.com/java-tutorial) is a symbol that is used to perform operations. |





##### **Java Unary Operator**

The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:

* incrementing/decrementing a value by one
* negating an expression
* inverting the value of a boolean

Increment :

a = a+1 = a++ / ++a

Post and pre-increment:

Pre : ++a

* assign/change the value
* use the value

Ex : int a = 10 , b = 0 ;

B = ++a

sysout(a) // 11

sysout(b) // 11

Post : a++

* Use the value
* Change the value

Ex : int a = 10 , b = 0 ;

B = a++ // a = 11

sysout(a) // 11

sysout(b) // 10

Decrement :

A = a-1 = a - - /– - a

**Post and pre decrement :**

Pre : - - a

Int a = 10 , b = 0

B = - - a // a = 9 == b

Sysout (a) // 9

sysout(b) // 9

**Post : a - -**

* **Assing / use**
* **Change the value**

Int a = 10 , b = 0

B = a - - // b = 10 , a = 9

Sysout (a) // a = 9

sysout(b) // 10

### Java Unary Operator Example: ++ and --

**public** **class** OperatorExample{

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

**int** x=10;

System.out.println(x++);//10 (11)

System.out.println(++x);//12

System.out.println(x--);//12 (11)

System.out.println(--x);//10

}}

**Output:**

10

12

12

10

### Java Unary Operator Example 2: ++ and --

1. **public** **class** OperatorExample{
2. **public** **static** **void** main(String args[]){
3. **int** a=10;
4. **int** b=10;
5. System.out.println(a++ + ++a);//10+12=22
6. System.out.println(b++ + b++);//10+11=21
8. }}

**Output:**

22

21

### Java Unary Operator Example: ~ and !

1. **public** **class** OperatorExample{
2. **public** **static** **void** main(String args[]){
3. **int** a=10;
4. **int** b=-10;
5. **boolean** c=**true**;
6. **boolean** d=**false**;
7. System.out.println(~a);//-11 (minus of total positive value which starts from 0)
8. System.out.println(~b);//9 (positive of total minus, positive starts from 0)
9. System.out.println(!c);//false (opposite of boolean value)
10. System.out.println(!d);//true
11. }}

# Java Keywords

Java keywords are also known as reserved words. Keywords are particular words that act as a key to a code. These are predefined words by Java so they cannot be used as a variable or object name or class name.

## List of Java Keywords

A list of Java keywords or reserved words are given below:

1. [**abstract**](https://www.javatpoint.com/abstract-keyword-in-java)**:** Java abstract keyword is used to declare an abstract class. An abstract class can provide the implementation of the interface. It can have abstract and non-abstract methods.
2. [**boolean:**](https://www.javatpoint.com/boolean-keyword-in-java) Java boolean keyword is used to declare a variable as a boolean type. It can hold True and False values only.
3. [**break**](https://www.javatpoint.com/java-break)**:** Java break keyword is used to break the loop or switch statement. It breaks the current flow of the program at specified conditions.
4. [**byte**](https://www.javatpoint.com/byte-keyword-in-java)**:** Java byte keyword is used to declare a variable that can hold 8-bit data values.
5. [**case**](https://www.javatpoint.com/case-keyword-in-java)**:** Java case keyword is used with the switch statements to mark blocks of text.
6. [**catch**](https://www.javatpoint.com/try-catch-block)**:** Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.
7. [**char**](https://www.javatpoint.com/char-keyword-in-java)**:** Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters
8. [**class**](https://www.javatpoint.com/class-keyword-in-java)**:** Java class keyword is used to declare a class.
9. [**continue**](https://www.javatpoint.com/java-continue)**:** Java continue keyword is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.
10. [**default**](https://www.javatpoint.com/default-keyword-in-java)**:** Java default keyword is used to specify the default block of code in a switch statement.
11. [**do**](https://www.javatpoint.com/java-do-while-loop)**:** Java do keyword is used in the control statement to declare a loop. It can iterate a part of the program several times.
12. [**double**](https://www.javatpoint.com/double-keyword-in-java)**:** Java double keyword is used to declare a variable that can hold 64-bit floating-point number.
13. [**else**](https://www.javatpoint.com/java-if-else)**:** Java else keyword is used to indicate the alternative branches in an if statement.
14. [**enum**](https://www.javatpoint.com/enum-in-java)**:** Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.
15. [**extends**](https://www.javatpoint.com/inheritance-in-java)**:** Java extends keyword is used to indicate that a class is derived from another class or interface.
16. [**final**](https://www.javatpoint.com/final-keyword)**:** Java final keyword is used to indicate that a variable holds a constant value. It is used with a variable. It is used to restrict the user from updating the value of the variable.
17. [**finally**](https://www.javatpoint.com/finally-block-in-exception-handling)**:** Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether an exception is handled or not.
18. [**float**](https://www.javatpoint.com/float-keyword-in-java)**:** Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.
19. [**for**](https://www.javatpoint.com/java-for-loop)**:** Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some condition becomes true. If the number of iteration is fixed, it is recommended to use for loop.
20. [**if**](https://www.javatpoint.com/java-if-else)**:** Java if keyword tests the condition. It executes the if block if the condition is true.
21. [**implements**](https://www.javatpoint.com/interface-in-java)**:** Java implements keyword is used to implement an interface.
22. [**import**](https://www.javatpoint.com/package)**:** Java import keyword makes classes and interfaces available and accessible to the current source code.
23. [**instanceof**](https://www.javatpoint.com/downcasting-with-instanceof-operator)**:** Java instanceof keyword is used to test whether the object is an instance of the specified class or implements an interface.
24. [**int**](https://www.javatpoint.com/int-keyword-in-java)**:** Java int keyword is used to declare a variable that can hold a 32-bit signed integer.
25. [**interface**](https://www.javatpoint.com/interface-in-java)**:** Java interface keyword is used to declare an interface. It can have only abstract methods.
26. [**long**](https://www.javatpoint.com/long-keyword-in-java)**:** Java long keyword is used to declare a variable that can hold a 64-bit integer.
27. **native:** Java native keyword is used to specify that a method is implemented in native code using JNI (Java Native Interface).
28. [**new**](https://www.javatpoint.com/new-keyword-in-java)**:** Java new keyword is used to create new objects.
29. [**null**](https://www.javatpoint.com/null-keyword-in-java)**:** Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.
30. [**package**](https://www.javatpoint.com/package)**:** Java package keyword is used to declare a Java package that includes the classes.
31. [**private**](https://www.javatpoint.com/private-keyword-in-java)**:** Java private keyword is an access modifier. It is used to indicate that a method or variable may be accessed only in the class in which it is declared.
32. [**protected**](https://www.javatpoint.com/protected-keyword-in-java)**:** Java protected keyword is an access modifier. It can be accessible within the package and outside the package but through inheritance only. It can't be applied with the class.
33. [**public**](https://www.javatpoint.com/public-keyword-in-java)**:** Java public keyword is an access modifier. It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.
34. [**return**](https://www.javatpoint.com/return-keyword-in-java)**:** Java return keyword is used to return from a method when its execution is complete.
35. [**short**](https://www.javatpoint.com/short-keyword-in-java)**:** Java short keyword is used to declare a variable that can hold a 16-bit integer.
36. [**static**](https://www.javatpoint.com/static-keyword-in-java)**:** Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is mainly used for memory management.
37. [**strictfp**](https://www.javatpoint.com/strictfp-keyword)**:** Java strictfp is used to restrict the floating-point calculations to ensure portability.
38. [**super**](https://www.javatpoint.com/super-keyword)**:** Java super keyword is a reference variable that is used to refer to parent class objects. It can be used to invoke the immediate parent class method.
39. [**switch**](https://www.javatpoint.com/java-switch)**:** The Java switch keyword contains a switch statement that executes code based on test value. The switch statement tests the equality of a variable against multiple values.
40. [**synchronized**](https://www.javatpoint.com/synchronization-in-java)**:** Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.
41. [**this**](https://www.javatpoint.com/this-keyword)**:** Java this keyword can be used to refer the current object in a method or constructor.
42. [**throw**](https://www.javatpoint.com/throw-keyword)**:** The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exceptions. It is followed by an instance.
43. [**throws**](https://www.javatpoint.com/throws-keyword-and-difference-between-throw-and-throws)**:** The Java throws keyword is used to declare an exception. Checked exceptions can be propagated with throws.
44. [**transient**](https://www.javatpoint.com/transient-keyword)**:** Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.
45. [**try**](https://www.javatpoint.com/try-catch-block)**:** Java try keyword is used to start a block of code that will be tested for exceptions. The try block must be followed by either catch or finally block.
46. **void:** Java void keyword is used to specify that a method does not have a return value.
47. [**volatile**](https://www.javatpoint.com/volatile-keyword-in-java)**:** Java volatile keyword is used to indicate that a variable may change asynchronously.
48. [**while**](https://www.javatpoint.com/java-while-loop)**:** Java while keyword is used to start a while loop. This loop iterates a part of the program several times. If the number of iteration is not fixed, it is recommended to use the while loop.

20th feb :

# 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.

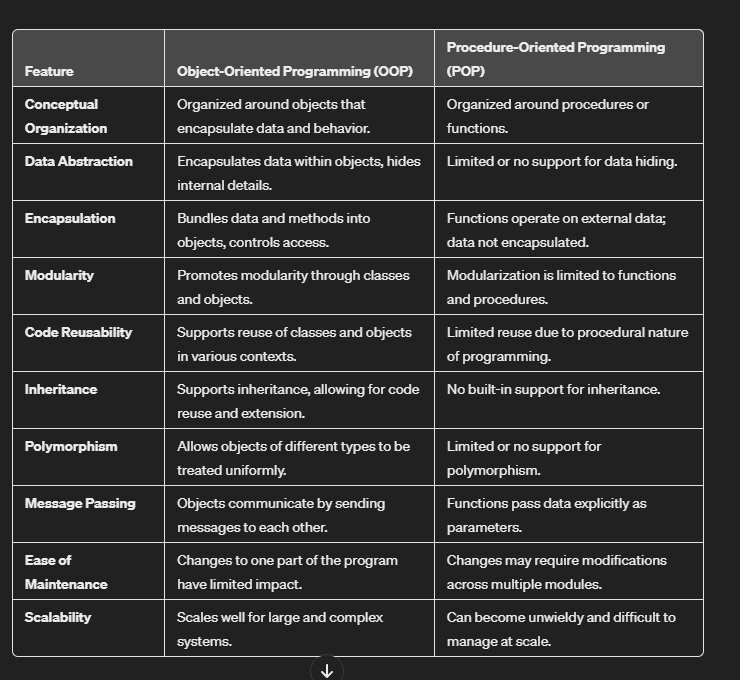
### **Aggregation :**

* 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

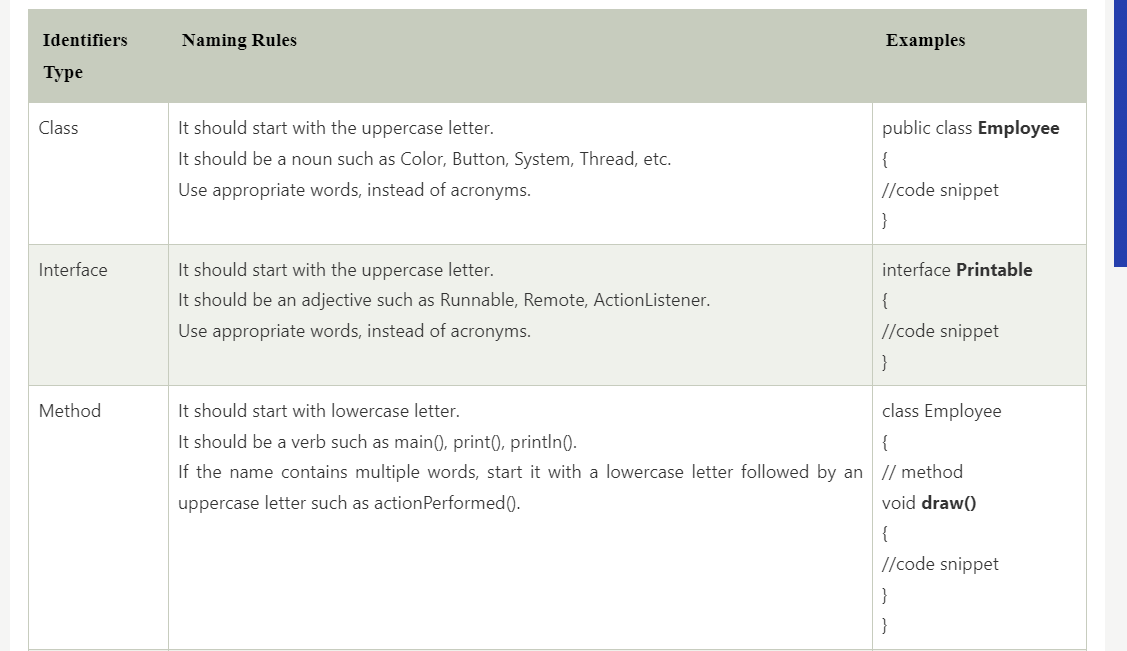


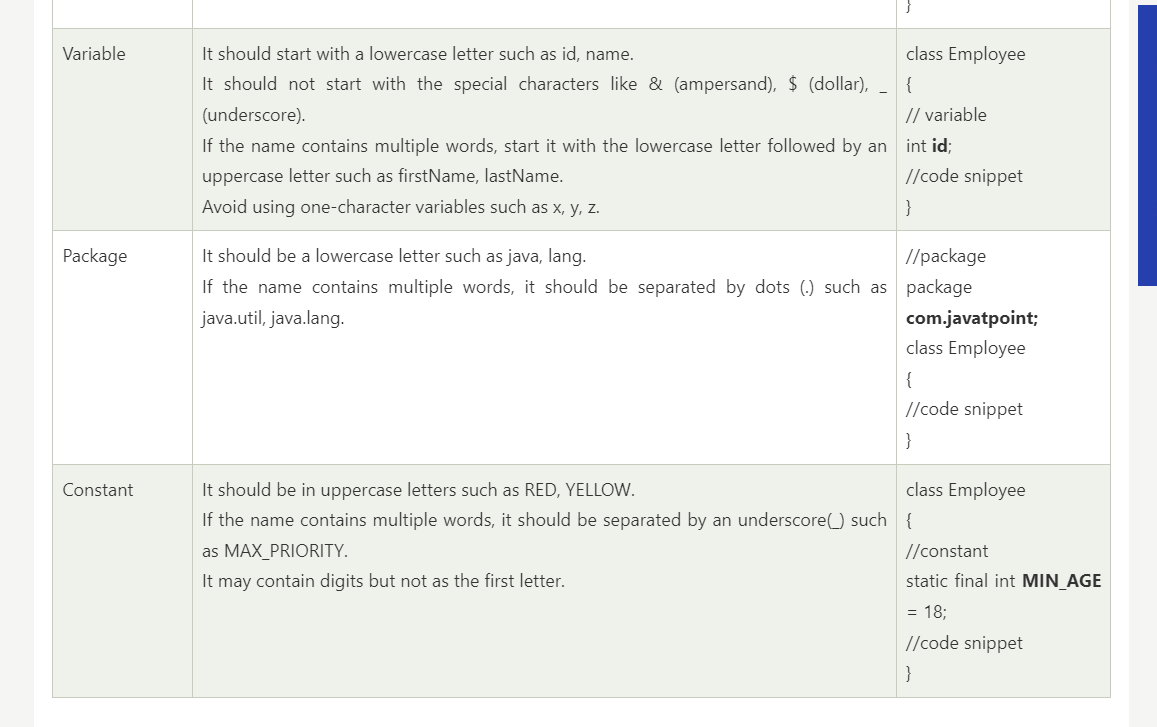
## 

# 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 :**

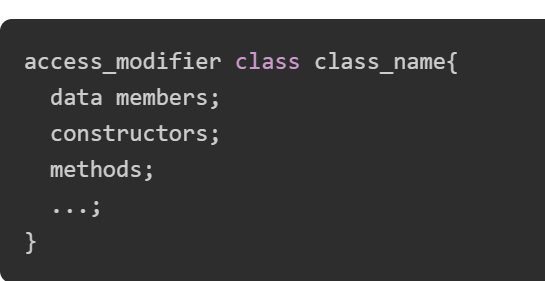
* Blueprint from which the individual object are created
* A class is a group of objects which have common properties
* It is a logical entity. It can't be physical.
* Occupies NO SPACE in memory
* A class in Java can contain:
* Fields
* Methods
* Constructors
* Blocks
* Nested class and interface

* 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:

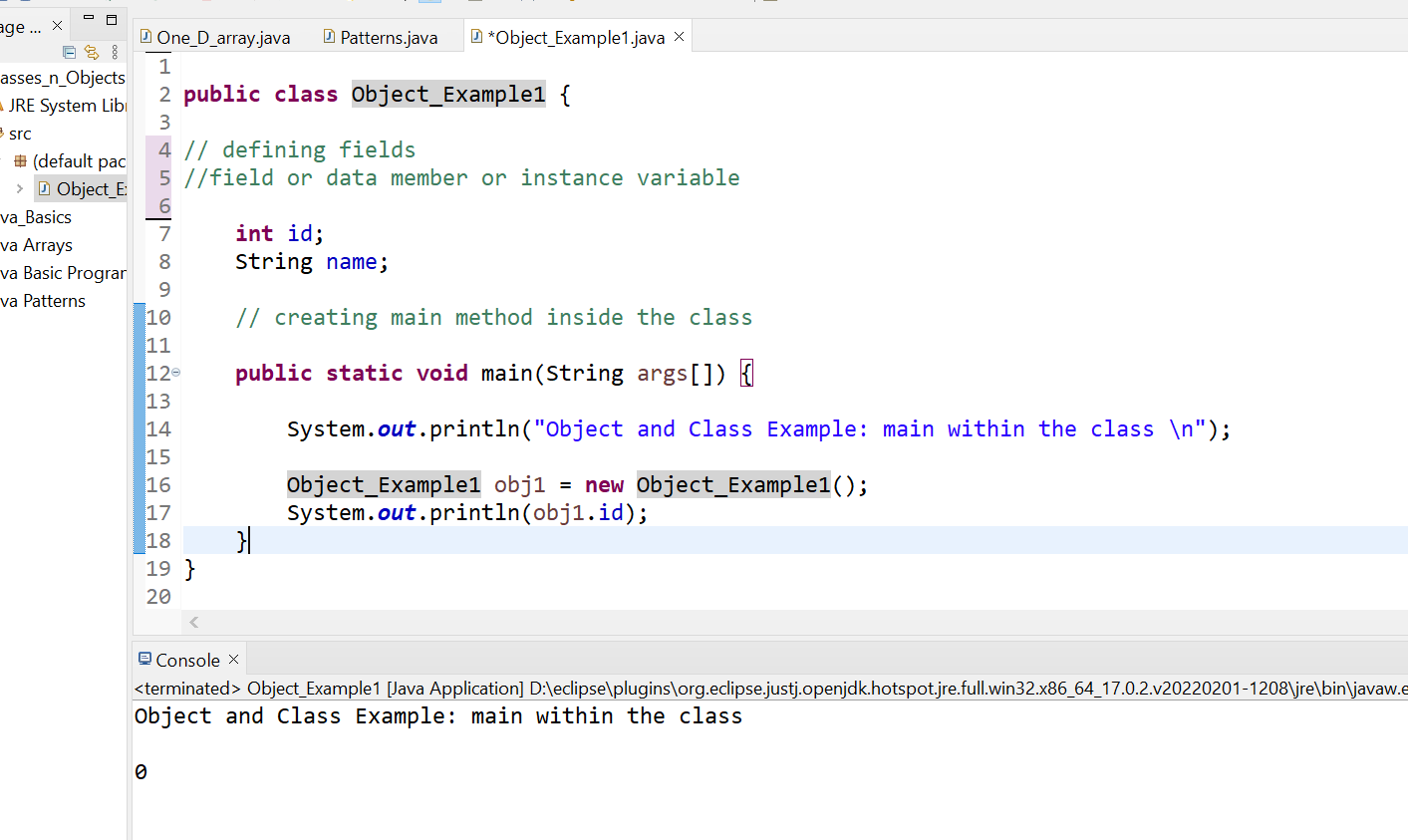
Golden Retriever

2

Golden

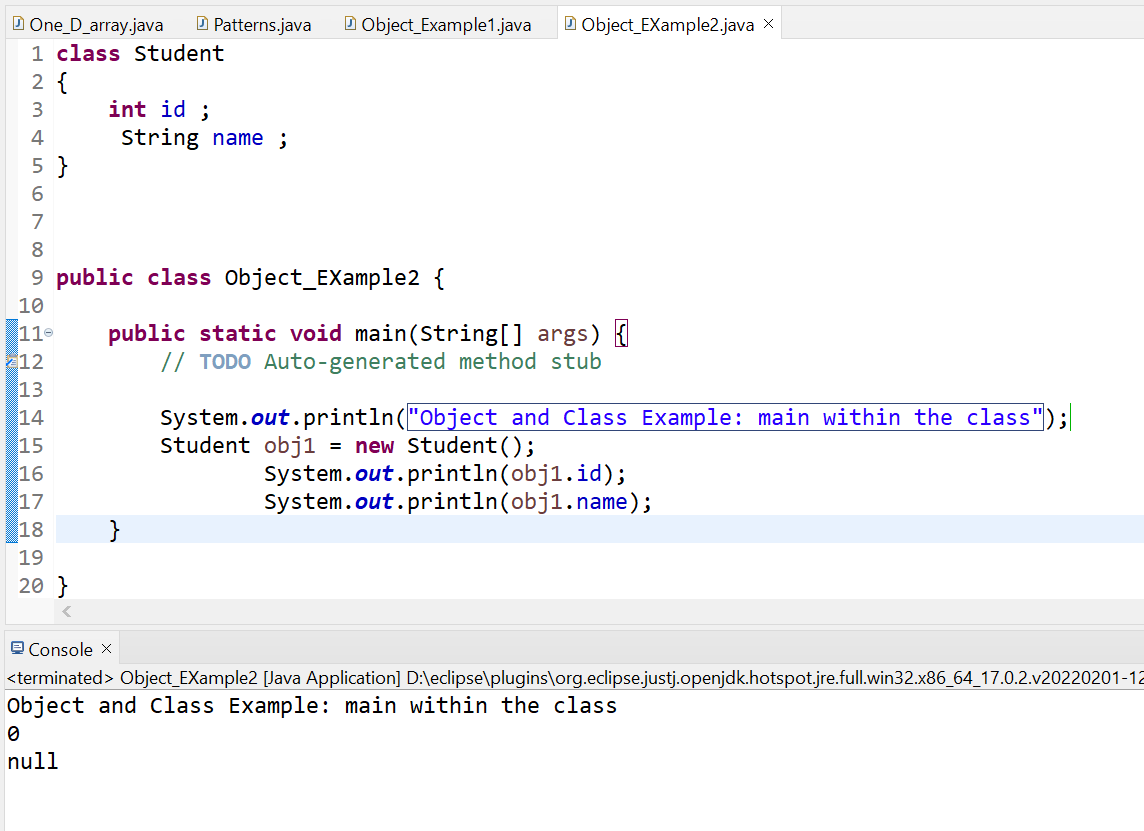
### Object and Class Example: main within the class

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



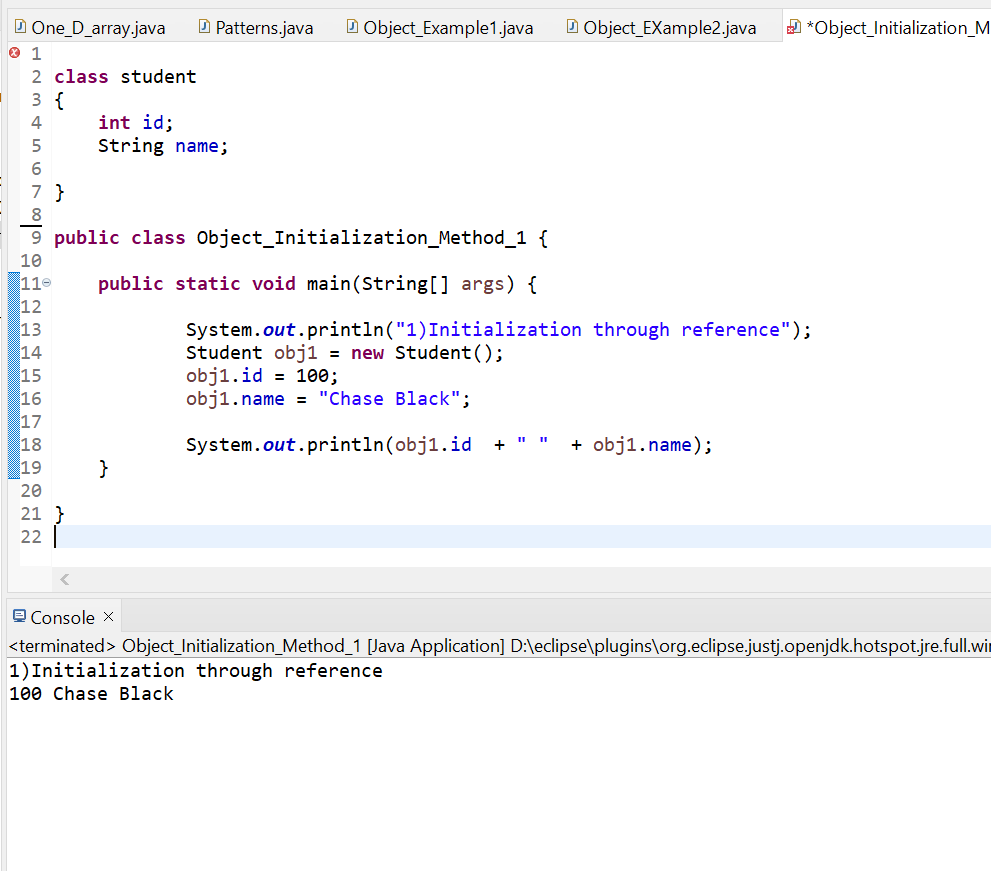
##### **3 Ways to initialize object**

There are 3 ways to initialize object in Java.

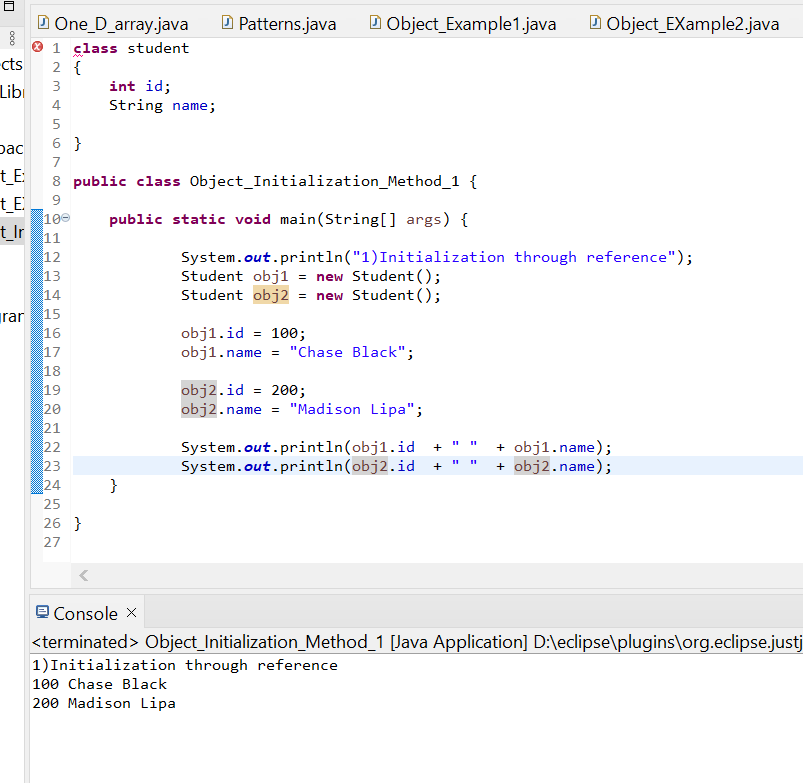
1. By reference variable
2. By method
3. By constructor

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

Initializing an object means storing data into the object.

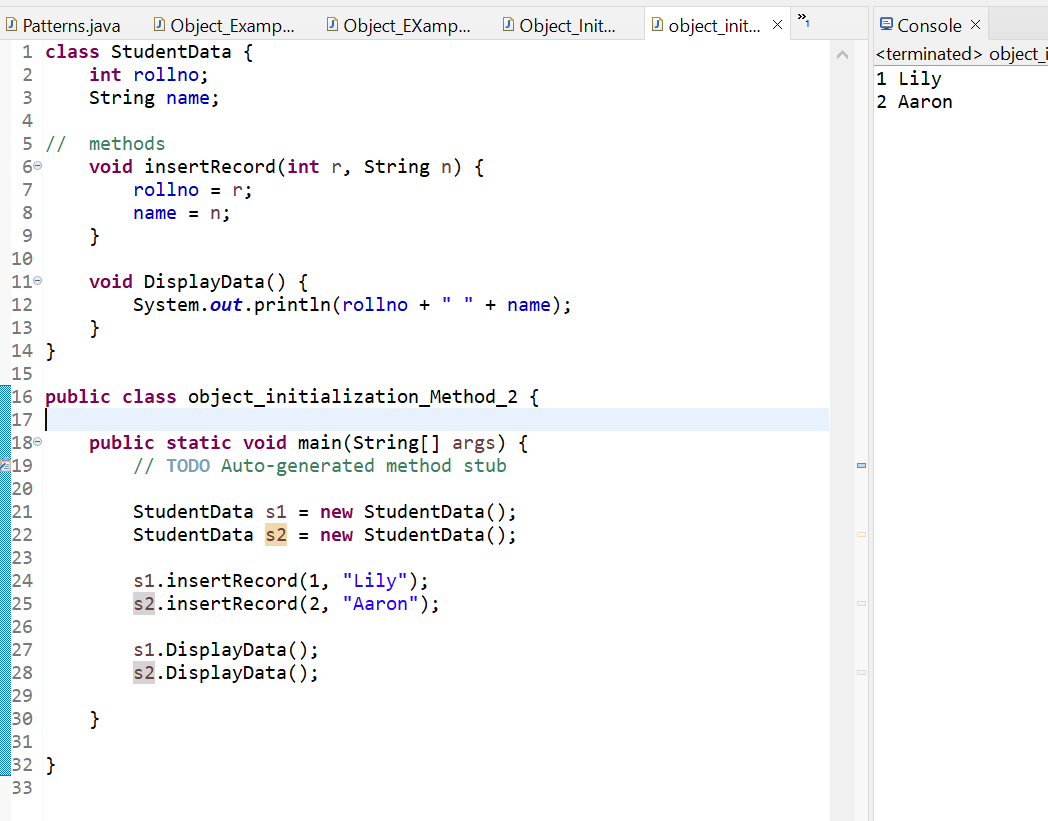


We can also create multiple objects and store information in it through reference variable.



### 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:

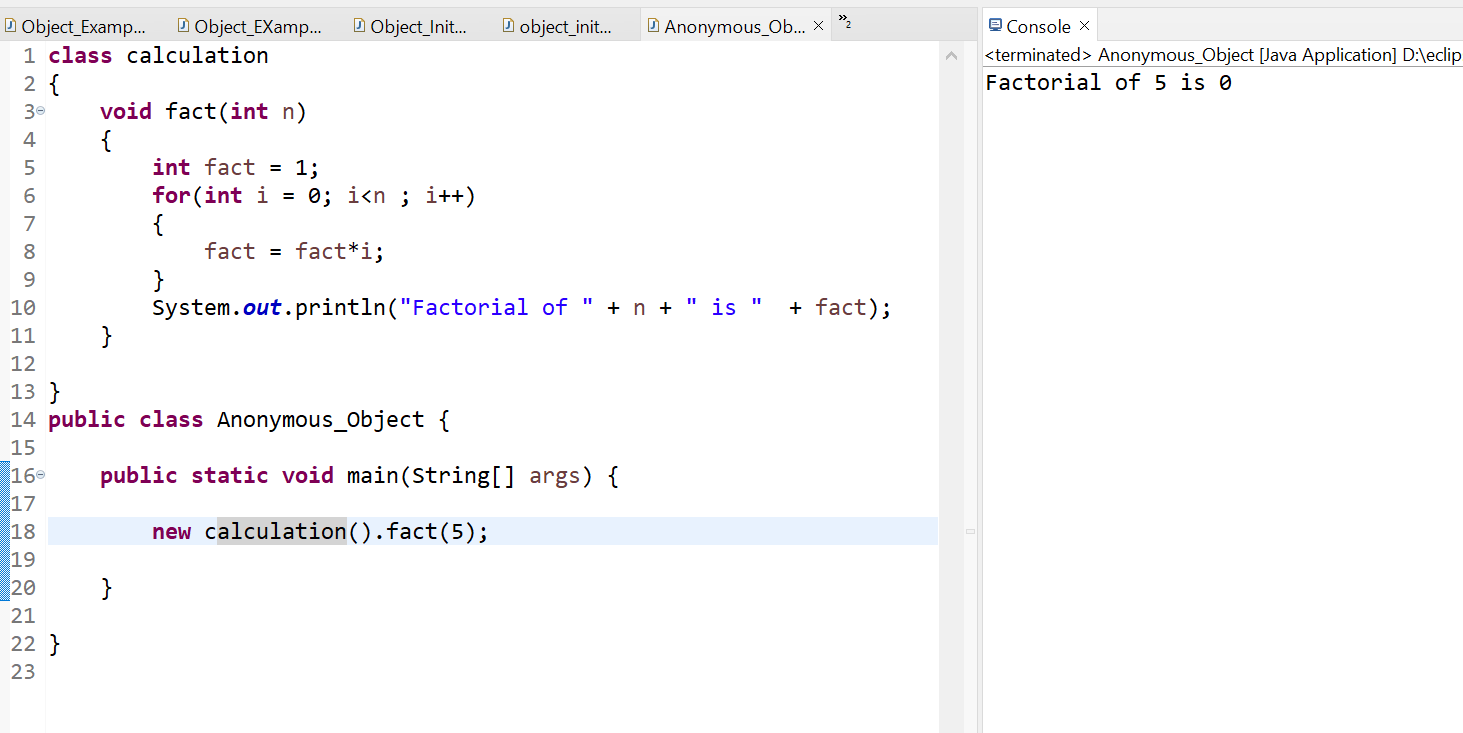
1. **new** Calculation();//anonymous object

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

**21st Feb 2024 :**

# **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.

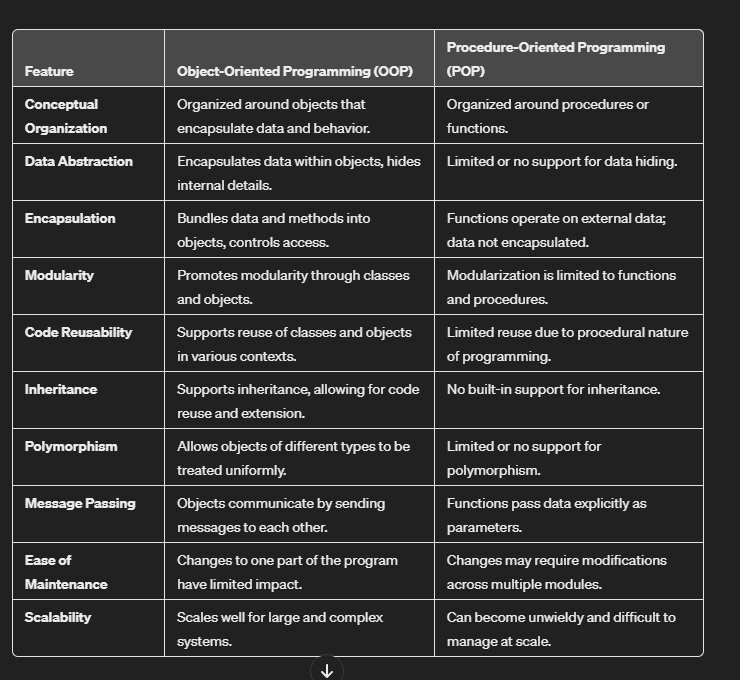
### Aggregation  :

* 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.

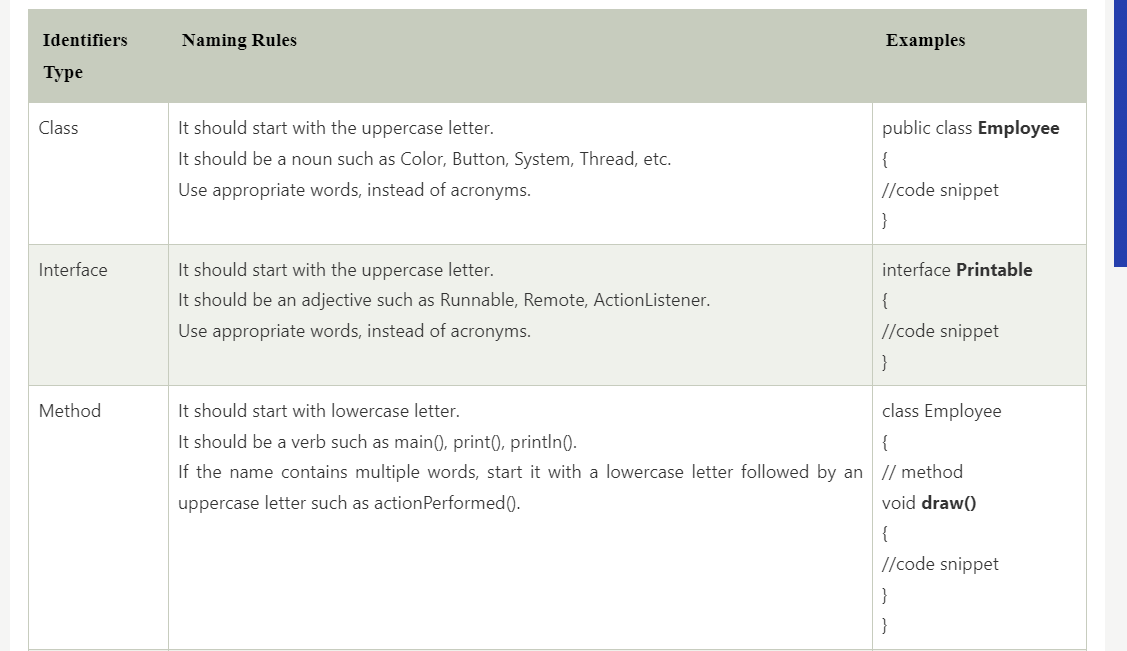
## **Advantage of OOPs over Procedure-oriented programming language**

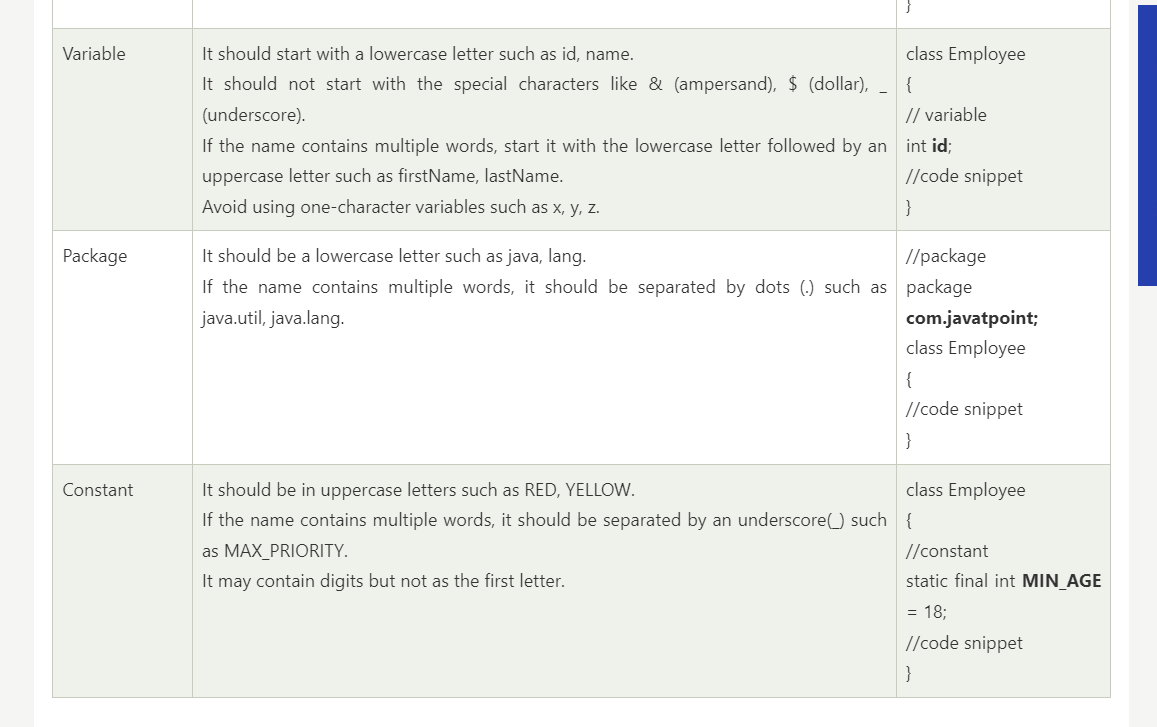


# **Java Naming Convention**

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  :

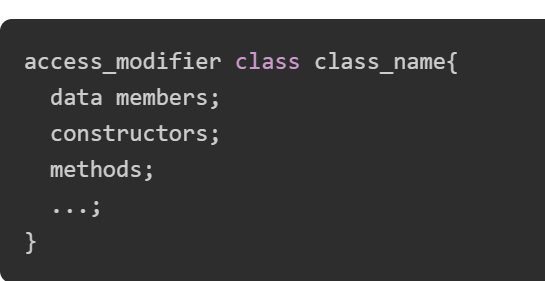
* Blueprint from  which the individual object are  created
* A class is a group of objects which have common properties
* It is a logical entity. It can't be physical.
* Occupies NO SPACE in memory
* A class in Java can contain:
* Fields
* Methods
* Constructors
* Blocks
* Nested class and interface

* 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:

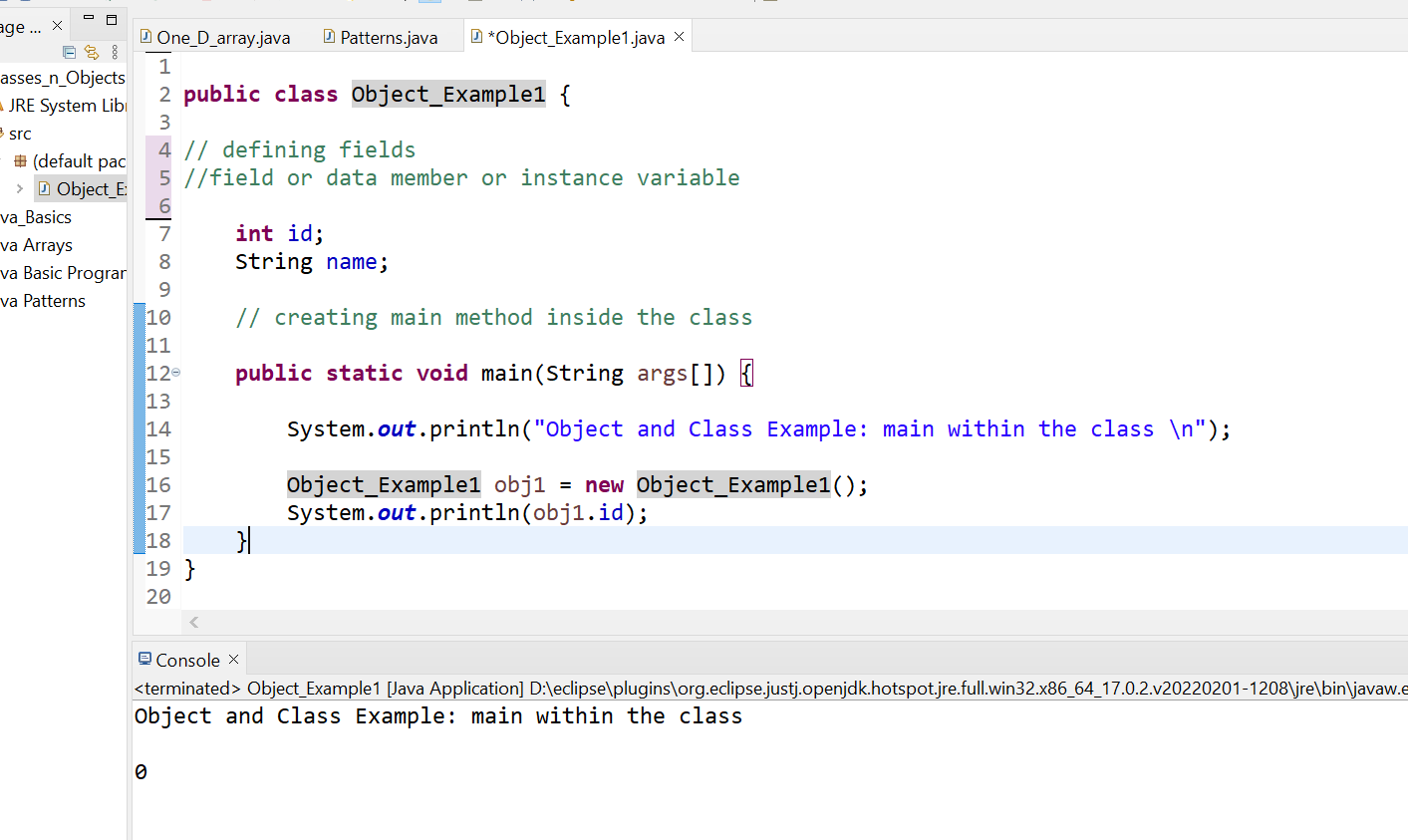
Golden Retriever

2

Golden

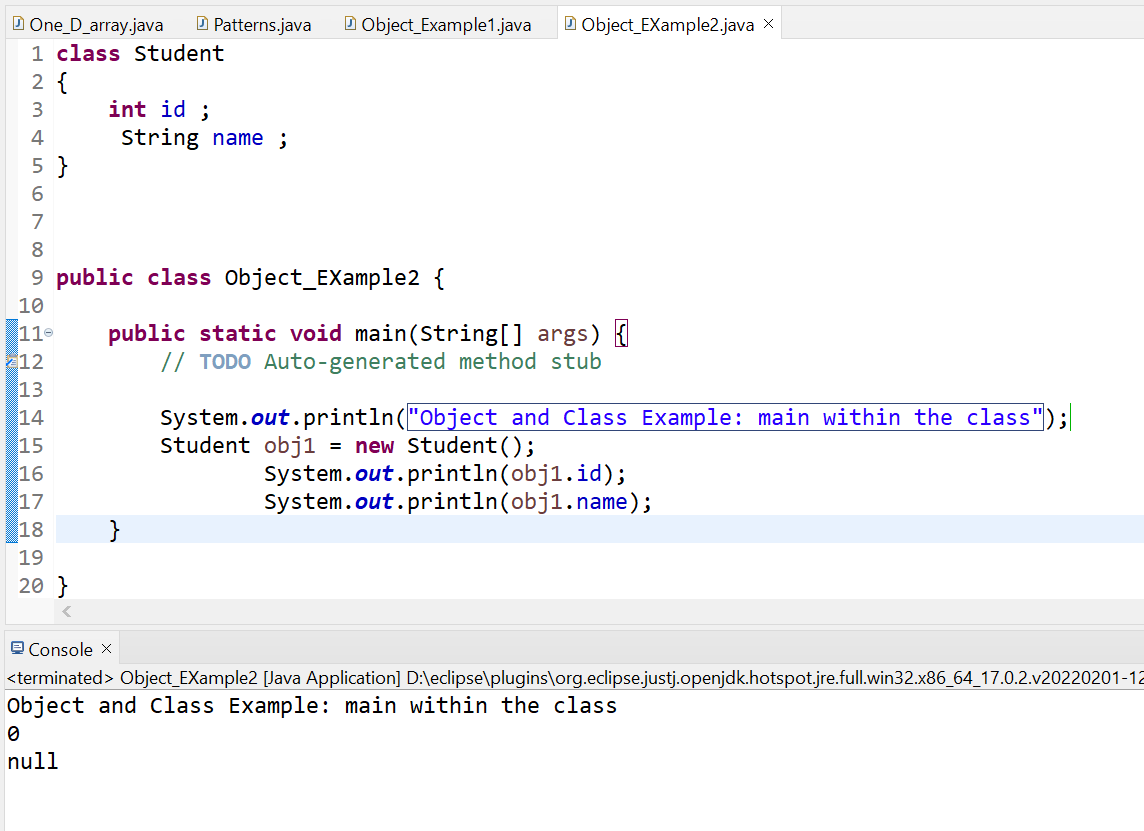
### **Object and Class Example: main within the class**

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



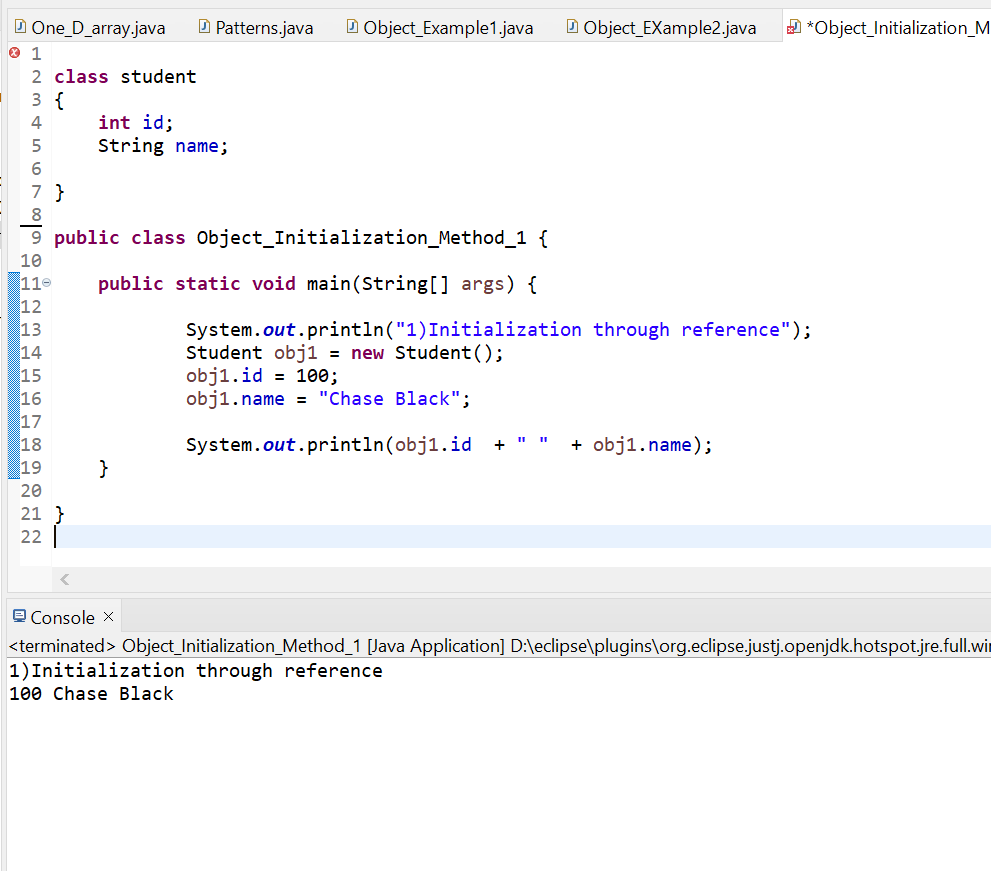
##### 3 Ways to initialize object

There are 3 ways to initialize object in Java.

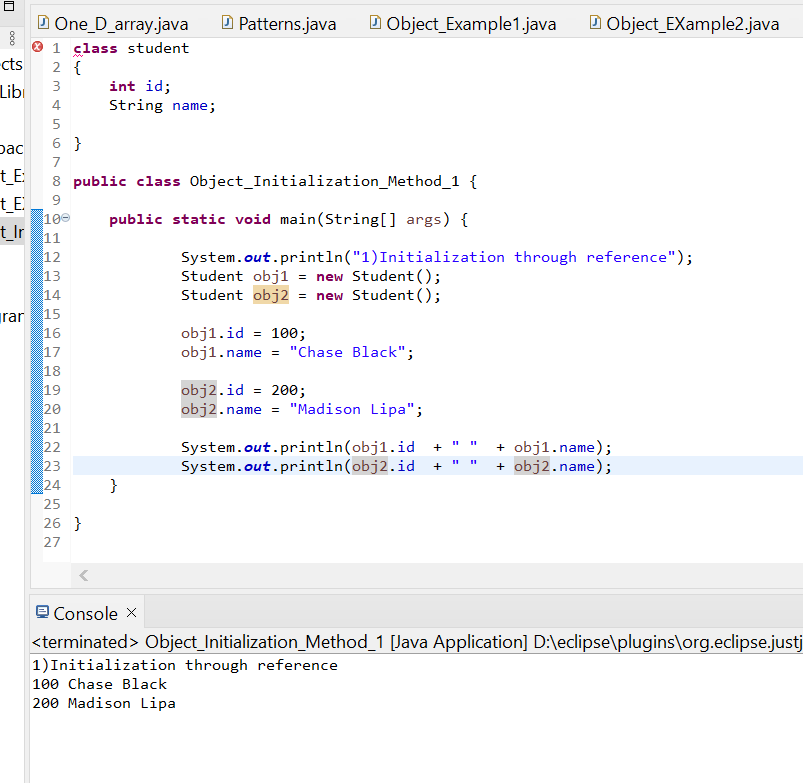
1. By reference variable
2. By method
3. By constructor

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

Initializing an object means storing data into the object.

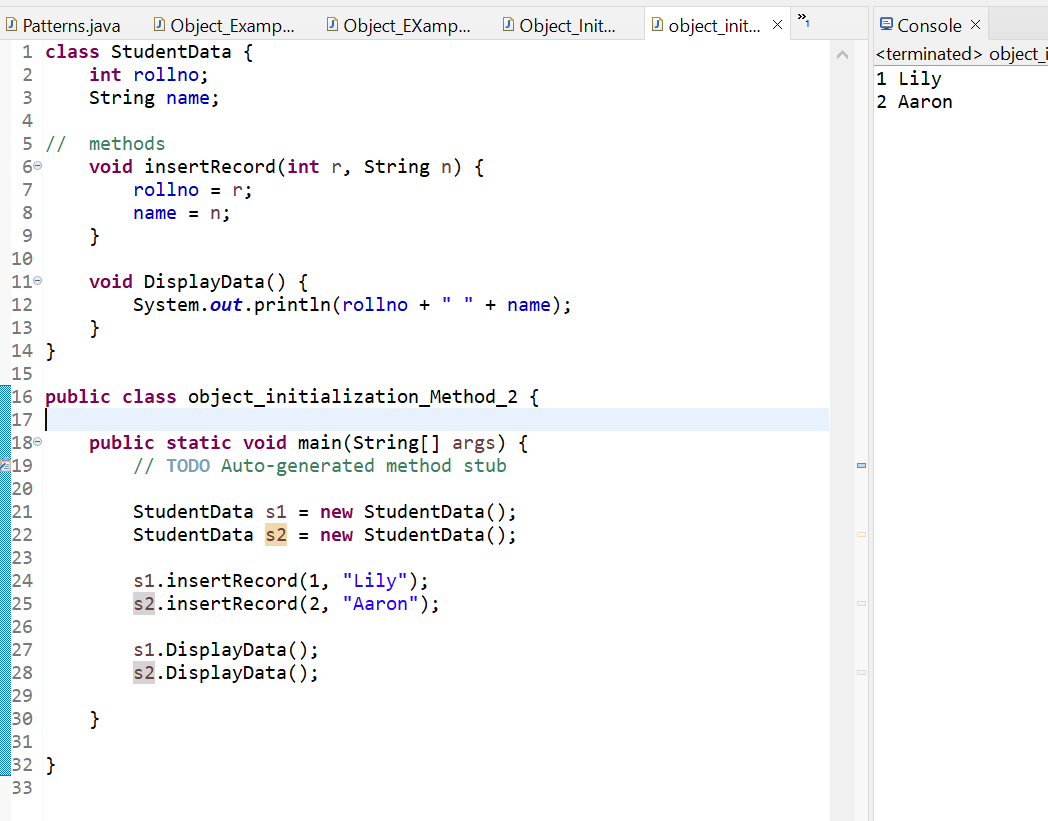


We can also create multiple objects and store information in it through reference variable.



### **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:

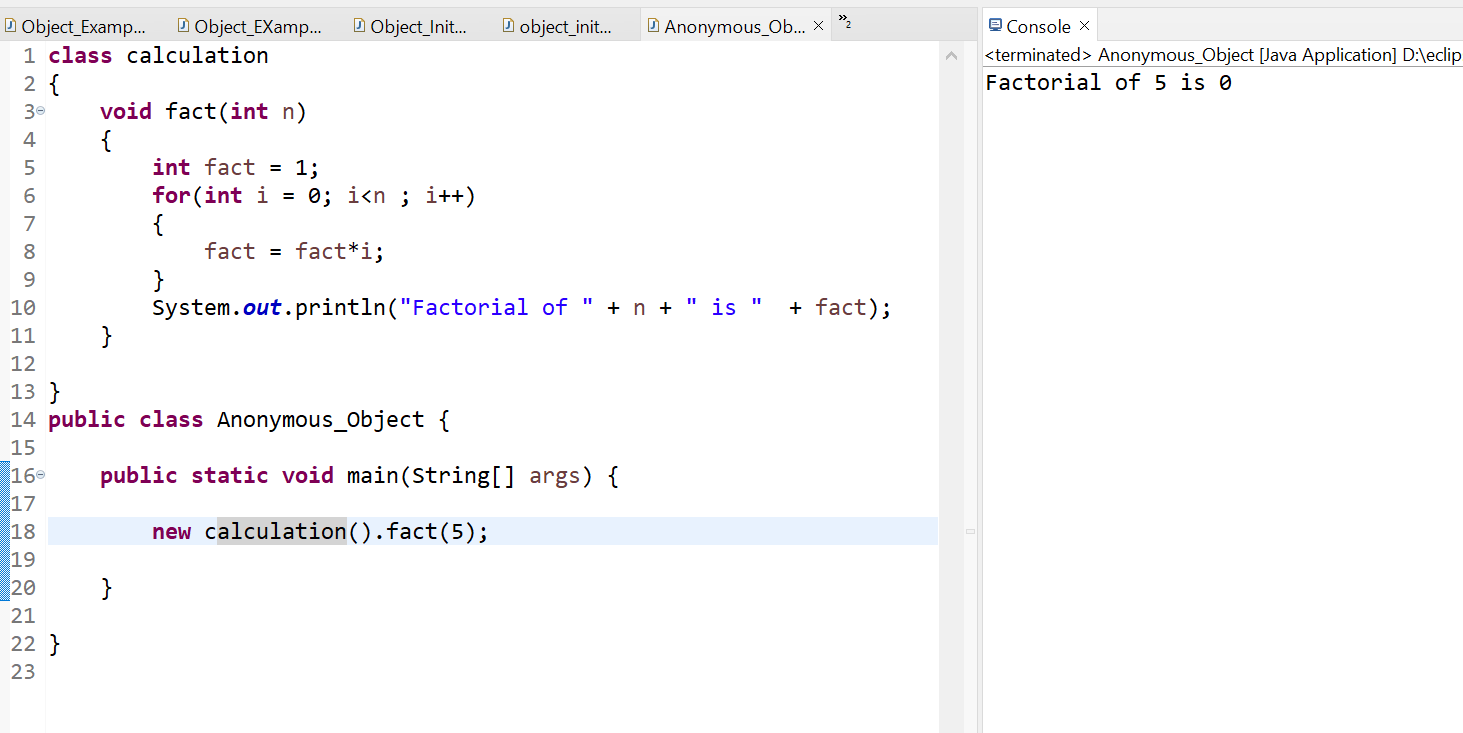
1. **new** Calculation();//anonymous object

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

# **Method in Java**

* In general, a **method** is a way to perform some task.
* Similarly, the **method in Java** is a collection of instructions that performs a specific task.
* It provides the reusability of code.

## **What is a method in Java?**

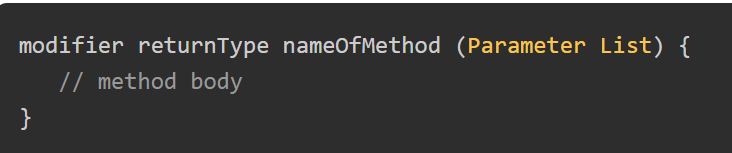
* A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation.
* It is used to achieve the **reusability** of code. We write a method once and use it many times.
* We do not require to write code again and again. It also provides the **easy modification** and **readability** of code, just by adding or removing a chunk of code. The method is executed only when we call or invoke it.
* The most important method in Java is the **main()** method.

## **Creating a Java Method**

To create a Java method, there should be an [access modifier](https://www.tutorialspoint.com/java/java_access_modifiers.htm) followed by the return type, method's name, and parameters list.

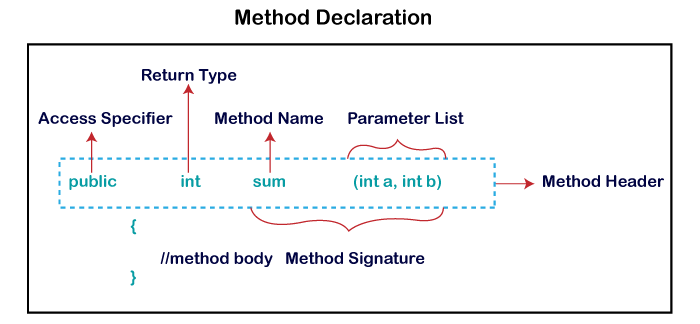
### **Syntax to Create a Java Method**

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



### **Method Declaration**

The method declaration provides information about method attributes, such as visibility, return-type, name, and arguments. It has six components that are known as **method header**, as we have shown in the following figure.



**Method Signature:** Every method has a method signature. It is a part of the method declaration. It includes the **method name** and **parameter list**.

**Access Specifier:** Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides **four** types of access specifier:

* **Public:** The method is accessible by all classes when we use public specifier in our application.
* **Private:** When we use a private access specifier, the method is accessible only in the classes in which it is defined.
* **Protected:** When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.
* **Default:** When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

**Return Type:** Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.

**Method Name:** It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method. Suppose, if we are creating a method for subtraction of two numbers, the method name must be **subtraction().** A method is invoked by its name.

**Parameter List:** It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, left the parentheses blank.

**Method Body:** It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.

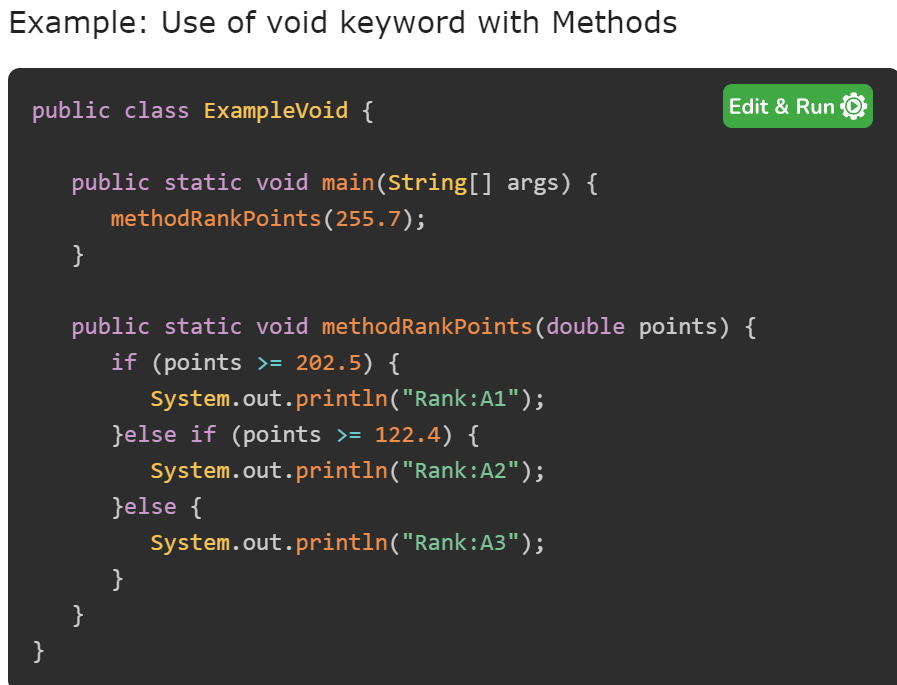
**Note : It is also possible that a method has the same name as another method name in the same class, it is known as method overloading.**

## **Calling a Java Method**

* For using a method, it should be called. There are two ways in which a method is called i.e., method returns a value or returning nothing (no return value).
* The process of method calling is simple. When a program invokes a method, the program control gets transferred to the called method. This called method then returns control to the caller in two conditions, when −
* the return statement is executed.
* it reaches the method ending closing brace.
* The methods returning void is considered as call to a statement. Lets consider an example −
* System.out.println("This is tutorialspoint.com!");

* The method returning value can be understood by the following example −
* int result = sum(6, 9);

## **The void Keyword with Java Methods**

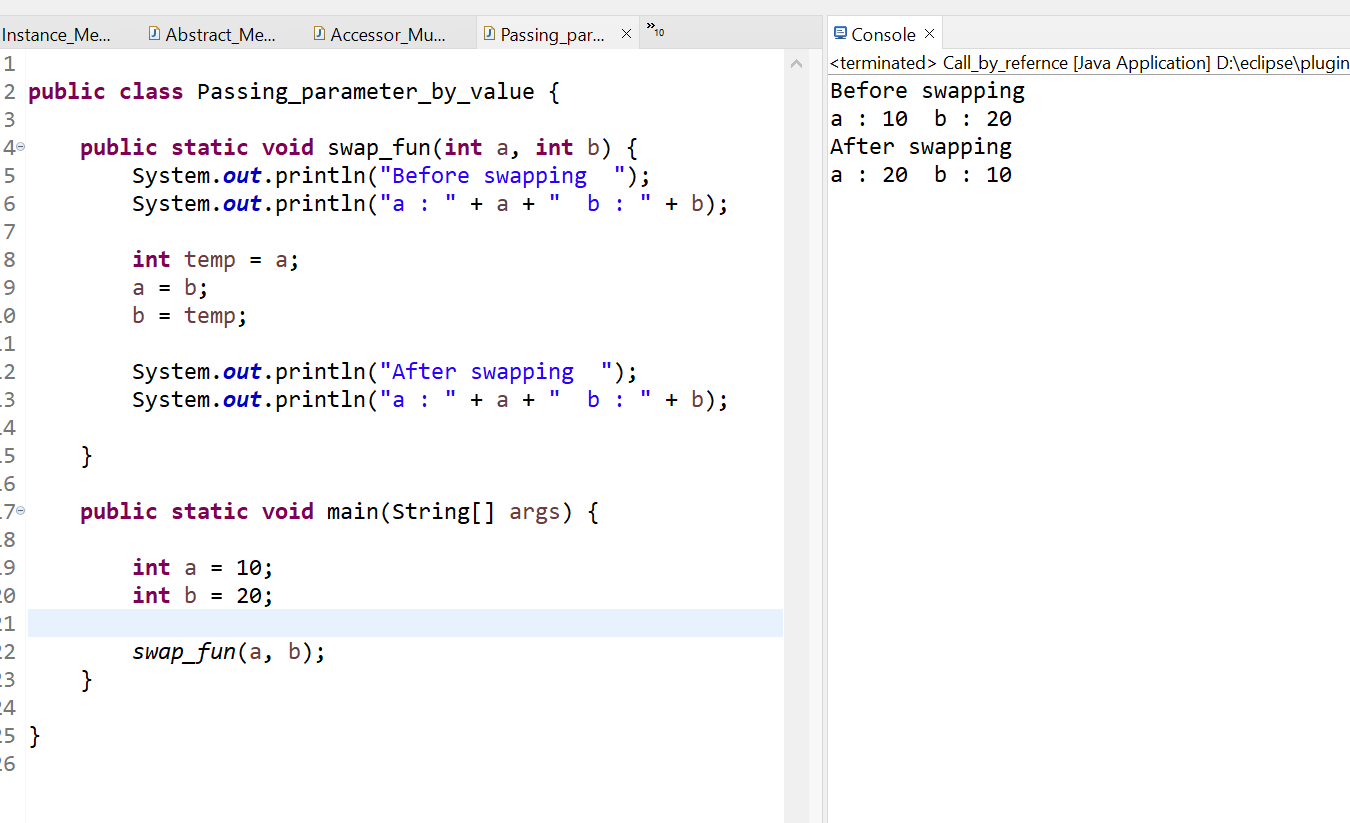
* The void keyword allows us to create methods which do not return a value.
* Here, in the following example we're considering a void method methodRankPoints.
* This method is a void method, which does not return any value.
* Call to a void method must be a statement i.e. methodRankPoints(255.7);.
* 

#### **Output**

**Rank:A1**

## **Passing Parameters by Value in Java Methods**

* While working under calling process, arguments is to be passed. These should be in the same order as their respective parameters in the method specification. Parameters can be passed by value or by reference.
* Passing Parameters by Value means calling a method with a parameter. Through this, the argument value is passed to the parameter.



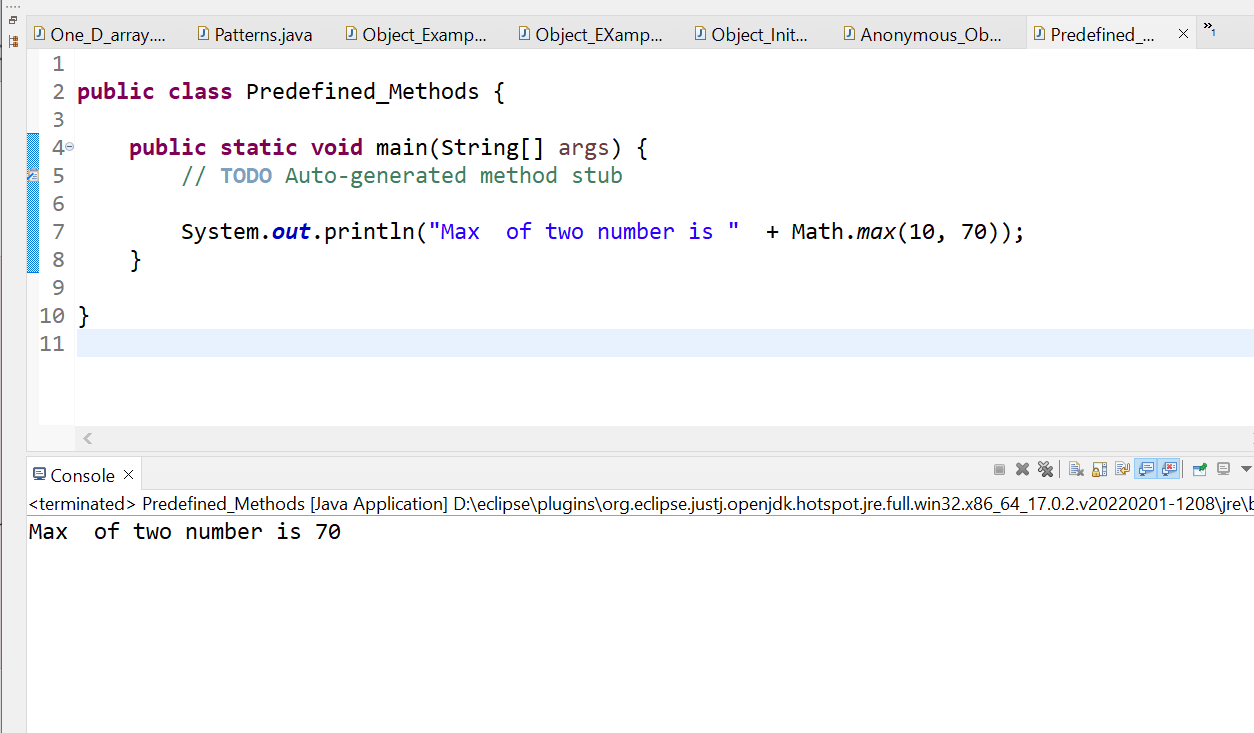
**Types of Method**

There are two types of methods in Java:

* Predefined Method
* User-defined Method

### **Predefined Method**

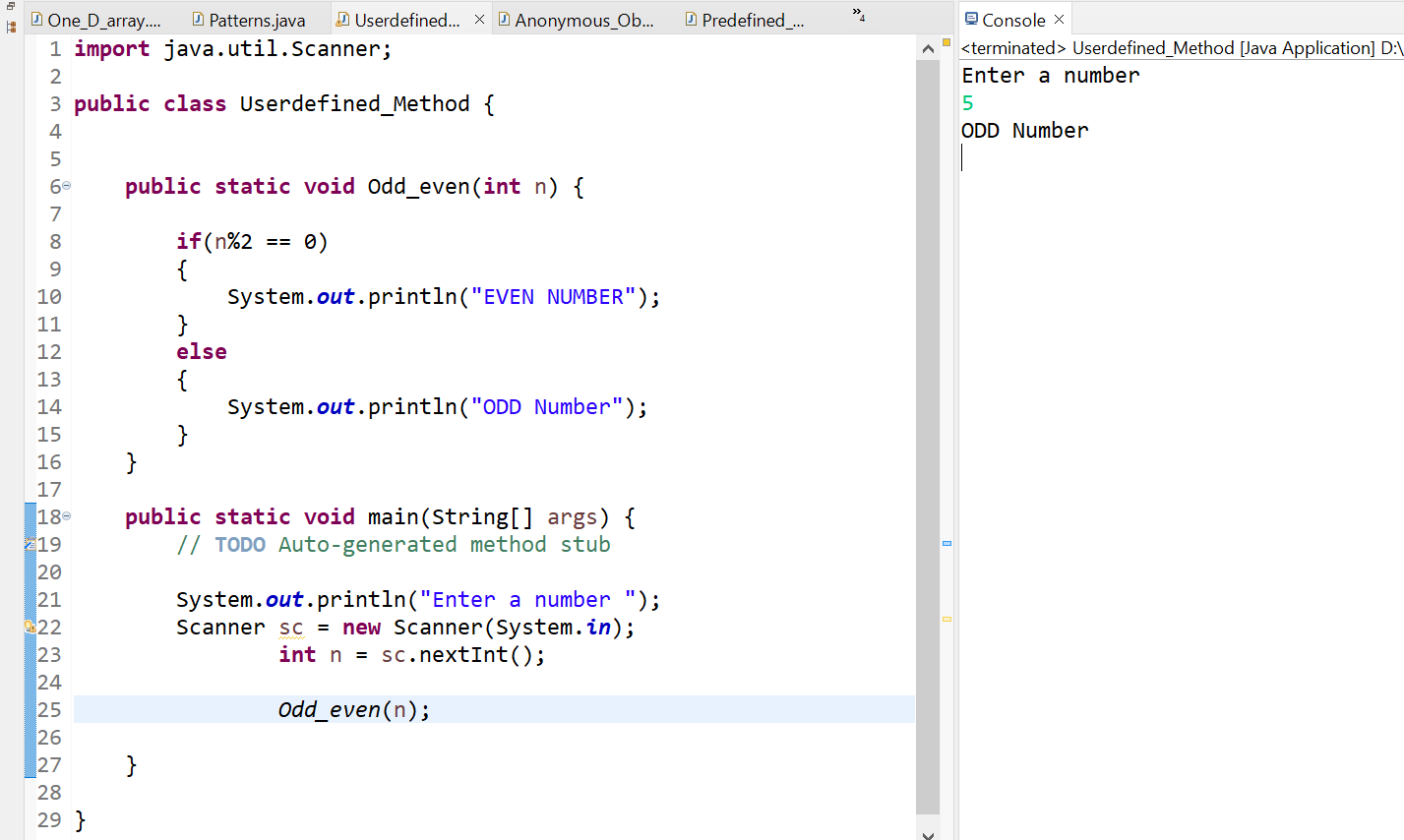
* In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods.
* It is also known as the **standard library method** or **built-in method**.
* We can directly use these methods just by calling them in the program at any point.
* Some pre-defined methods are **length(), equals(), compareTo(), sqrt(),** etc. When we call any of the predefined methods in our program, a series of codes related to the corresponding method runs in the background that is already stored in the library.
* Each and every predefined method is defined inside a class. Such as **print()** method is defined in the **java.io.PrintStream** class. It prints the statement that we write inside the method.
* For example, **print("Java")**, it prints Java on the console.



In the above example, we have used three predefined methods **main(), print(),** and **max()**. We have used these methods directly without declaration because they are predefined. The print() method is a method of **PrintStream** class that prints the result on the console. The max() method is a method of the **Math** class that returns the greater of two numbers.

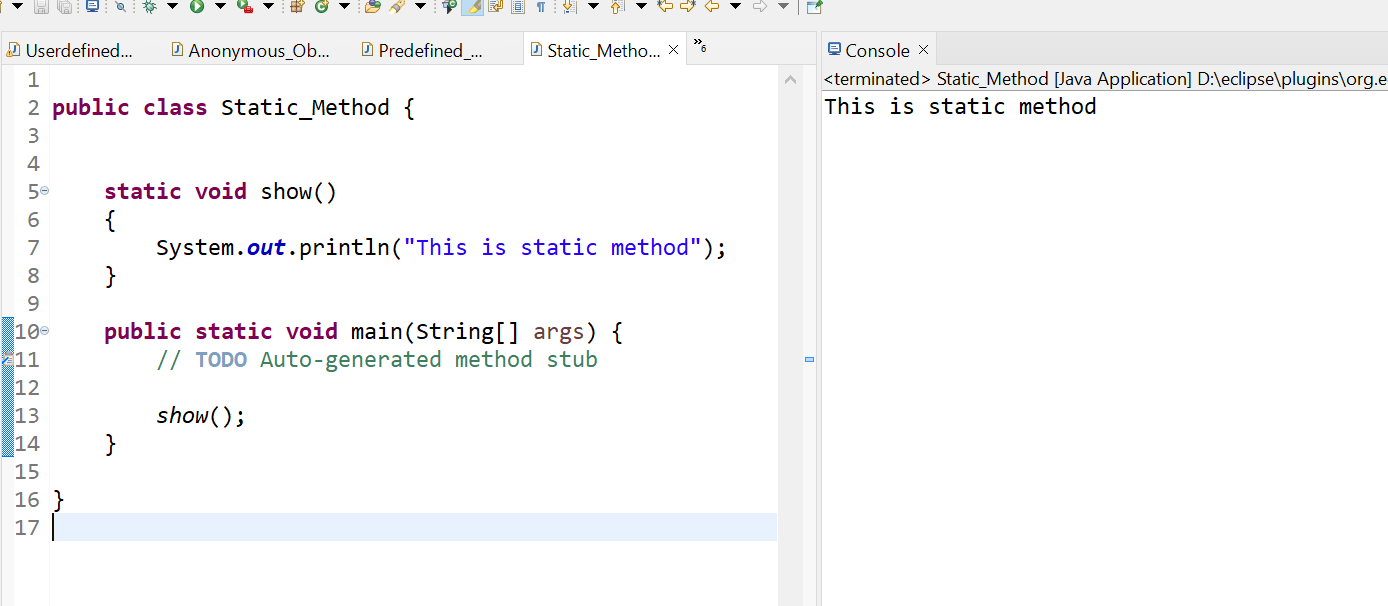
2. User-defined Method

* The method written by the user or programmer is known as **a user-defined** method. These methods are modified according to the requirement.



### **Static Method :**

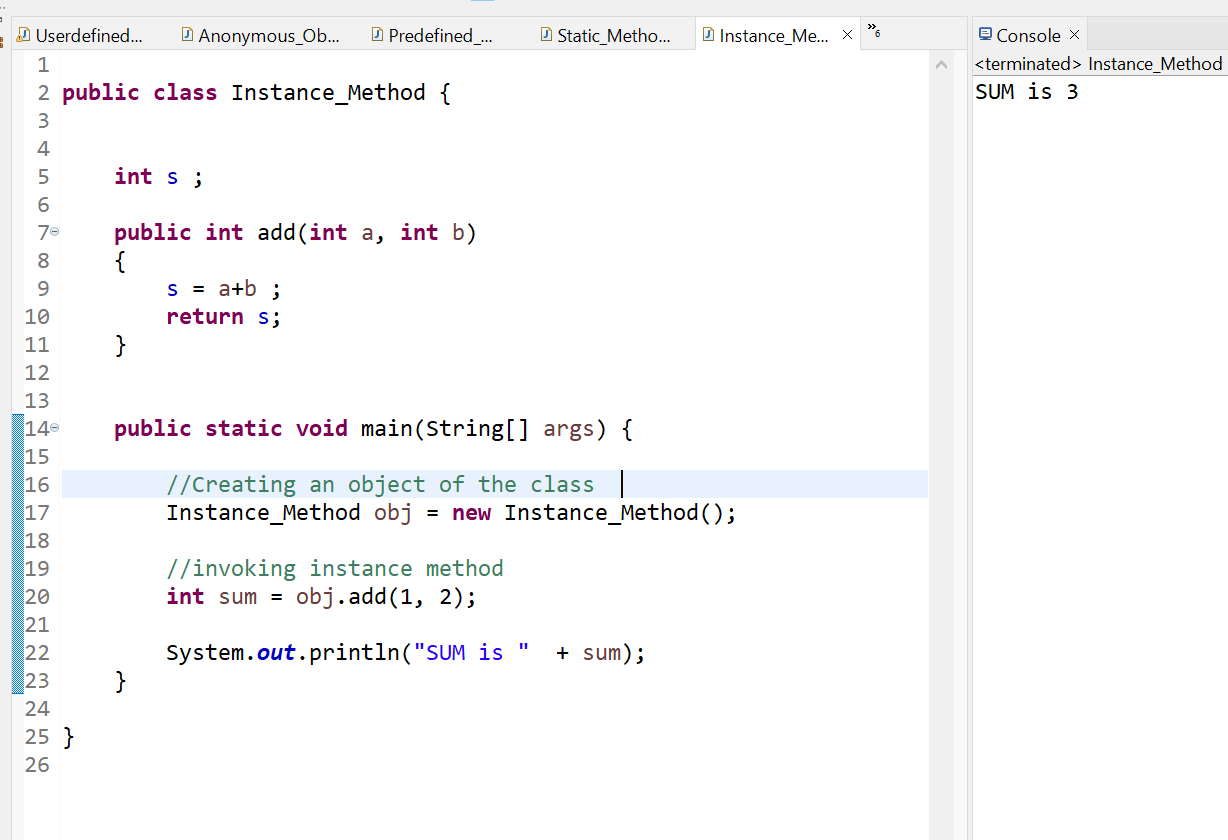
* A method that has static keyword is known as static method.
* Method that belongs to a class rather than an instance of a class is known as a static method.
* We can also create a static method by using the keyword **static** before the method name.
* The main advantage of a static method is that we can call it without creating an object.
* It can access static data members and also change the value of it. It is used to create an instance method. It is invoked by using the class name.
* The best example of a static method is the **main()** method.



### **Instance Method**

* The method of the class is known as an **instance method**.
* It is a **non-static** method defined in the class.
* Before calling or invoking the instance method, it is necessary to create an object of its class.

Example :



There are two types of instance method:

* **Accessor Method**
* **Mutator Method**

1. **Accessor Method  :**

* The method(s) that reads the instance variable(s) is known as the accessor method.
* We can easily identify it because the method is prefixed with the word get.
* It is also known as getters. It returns the value of the private field. It is used to get the value of the private field.
* **Example**

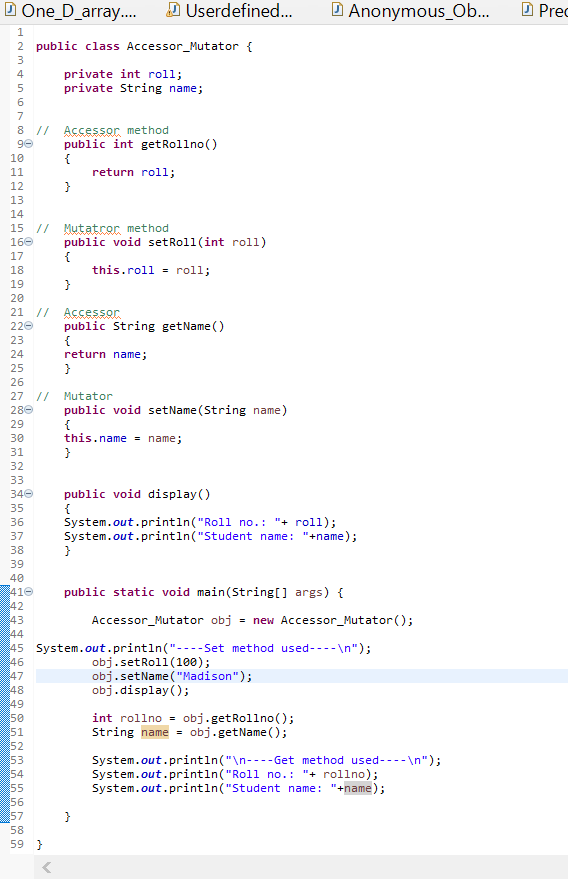
1. **public int getId()**
2. **{**
3. **return Id;**
4. **}**

1. **Mutator Method  :**

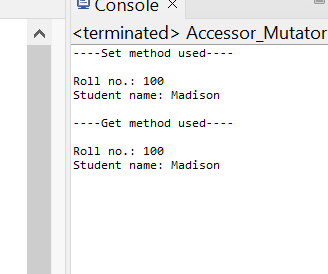
* The method(s) read the instance variable(s) and also modify the values.
* We can easily identify it because the method is prefixed with the word set.
* It is also known as setters or modifiers.
* It does not return anything.
* It accepts a parameter of the same data type that depends on the field. It is used to set the value of the private field.
* **Example**

1. **public** **void** setRoll(**int** roll)
2. {
3. **this**.roll = roll;
4. }

**Example of Accessor and Mutator :**



**Output :**

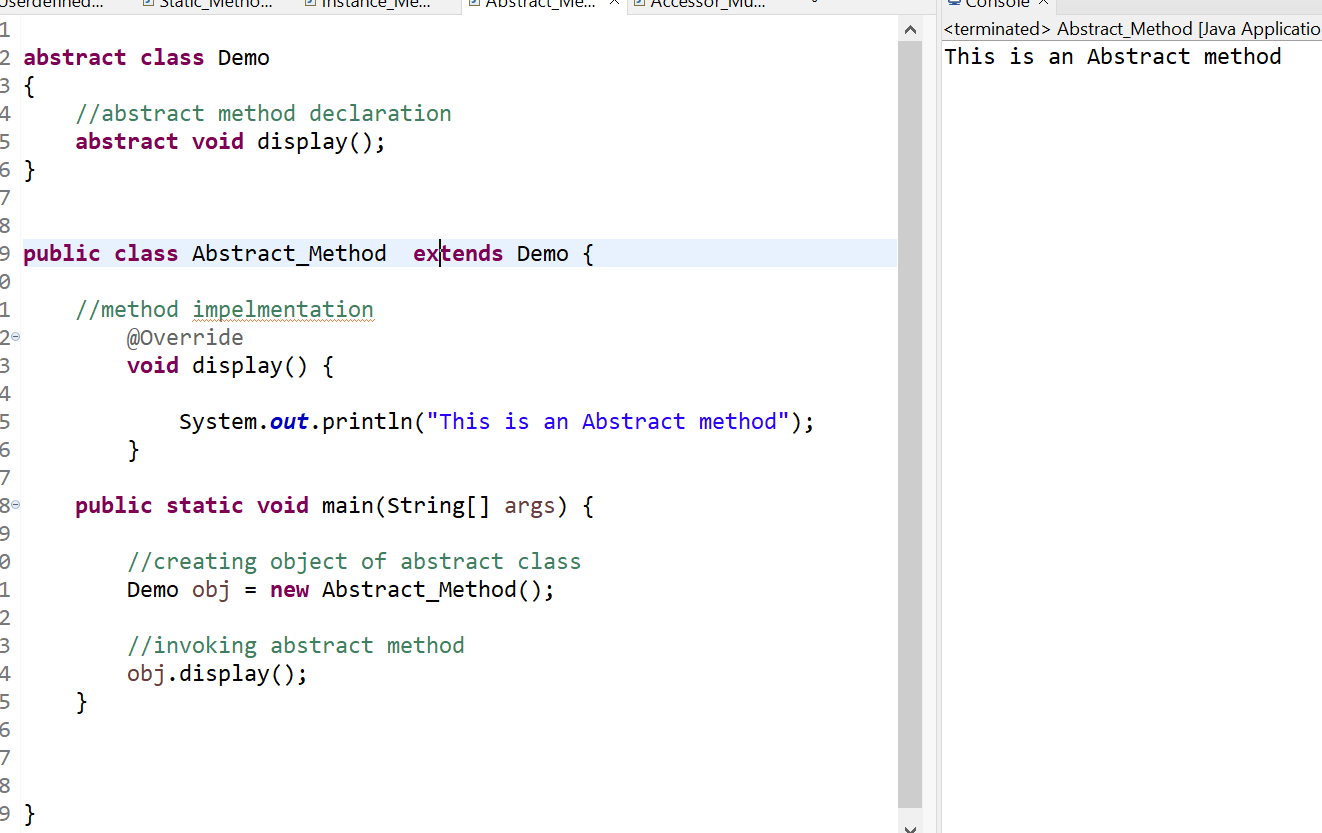
****

### **Abstract Method**

* The method that does not has method body is known as abstract method.
* In other words, without an implementation is known as abstract method.
* It always declares in the **abstract class**.
* It means the class itself must be abstract if it has abstract method.
* To create an abstract method, we use the keyword **abstract**.
* **Syntax**

**abstract** **void** method\_name();

* Example :

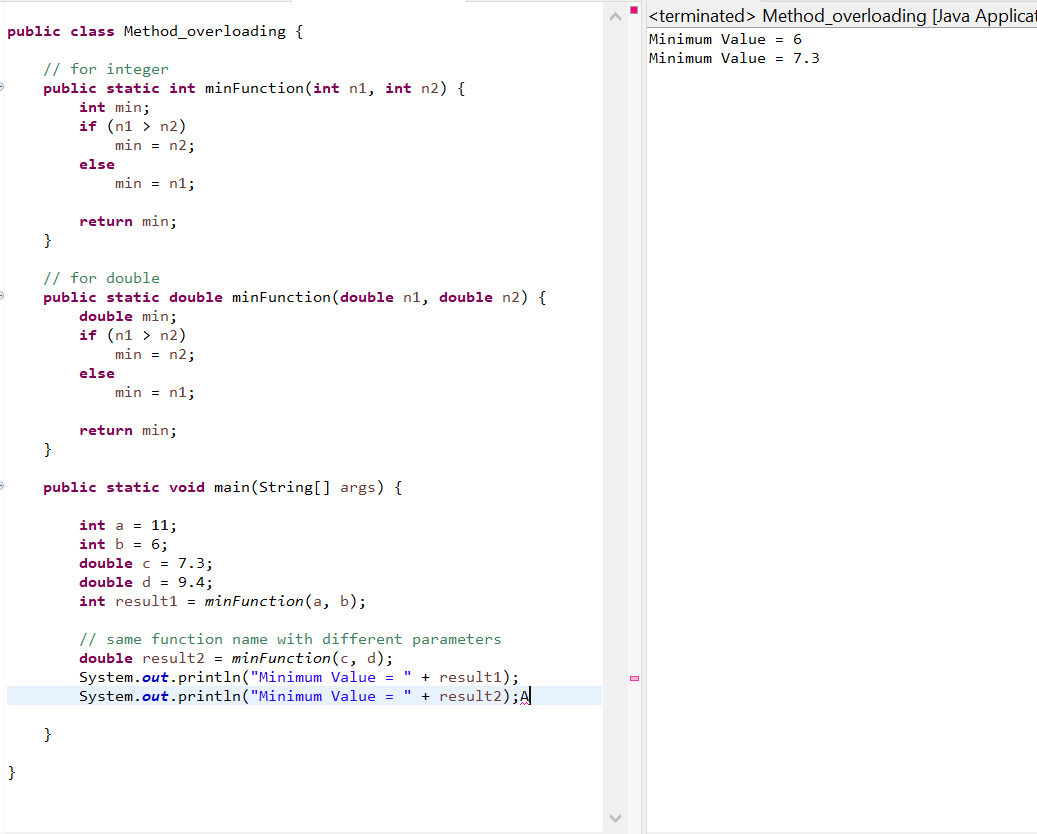


### **Factory method**

* It is a method that returns an object to the class to which it belongs. All static methods are factory methods.
* For example, **NumberFormat obj = NumberFormat.getNumberInstance();**

## **Java Methods Overloading**

* When a class has two or more methods by the same name but different parameters, it is known as method overloading.
* It is different from overriding.
* In overriding, a method has the same method name, type, number of parameters, etc.

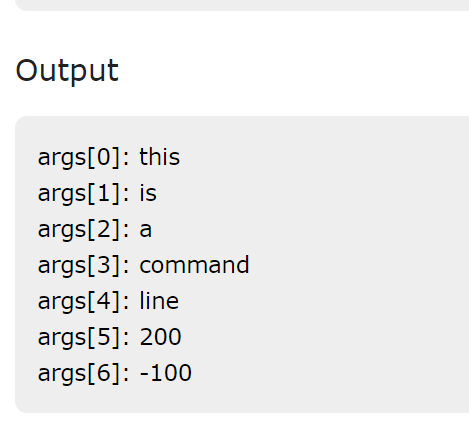


* Overloading methods makes program readable. Here, two methods are given by the same name but with different parameters. The minimum number from integer and double types is the result.

## **Using Command-Line Arguments**

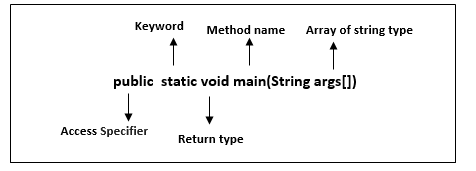
* Sometimes you will want to pass some information into a program when you run it. This is accomplished by passing command-line arguments to main( ).
* A command-line argument is the information that directly follows the program's name on the command line when it is executed. To access the command-line arguments inside a Java program is quite easy. They are stored as strings in the String array passed to main( ).





# **Java main() method**

The main() is the starting point for JVM to start execution of a Java program. Without the main() method, JVM will not execute the program. The syntax of the main() method is:



**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.

**String args[]:** 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.

1. main(String args[])

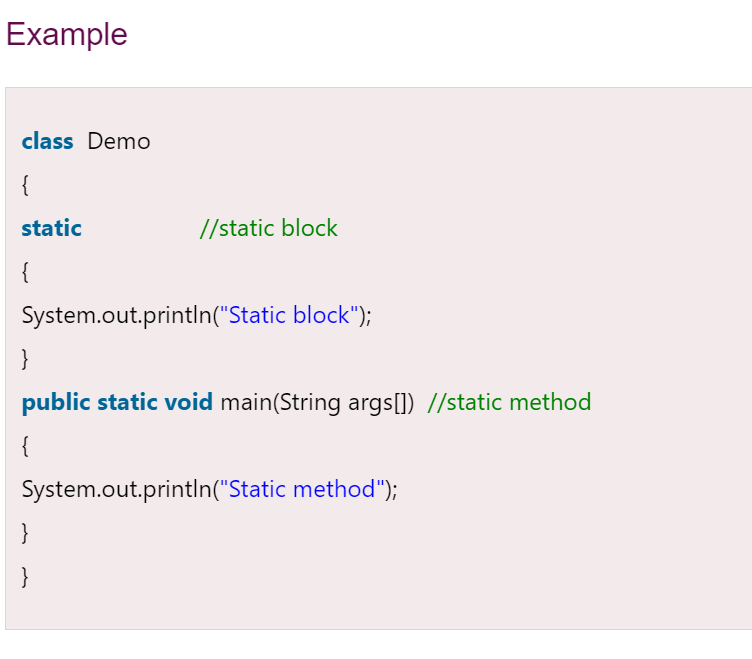
Here, agrs[] is the array name, and it is of String type. It means that it can store a group of string. Remember, this array can also store a group of numbers but in the form of string only. Values passed to the main() method is called arguments. These arguments are stored into args[] array, so the name args[] is generally used for it.

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

* 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.



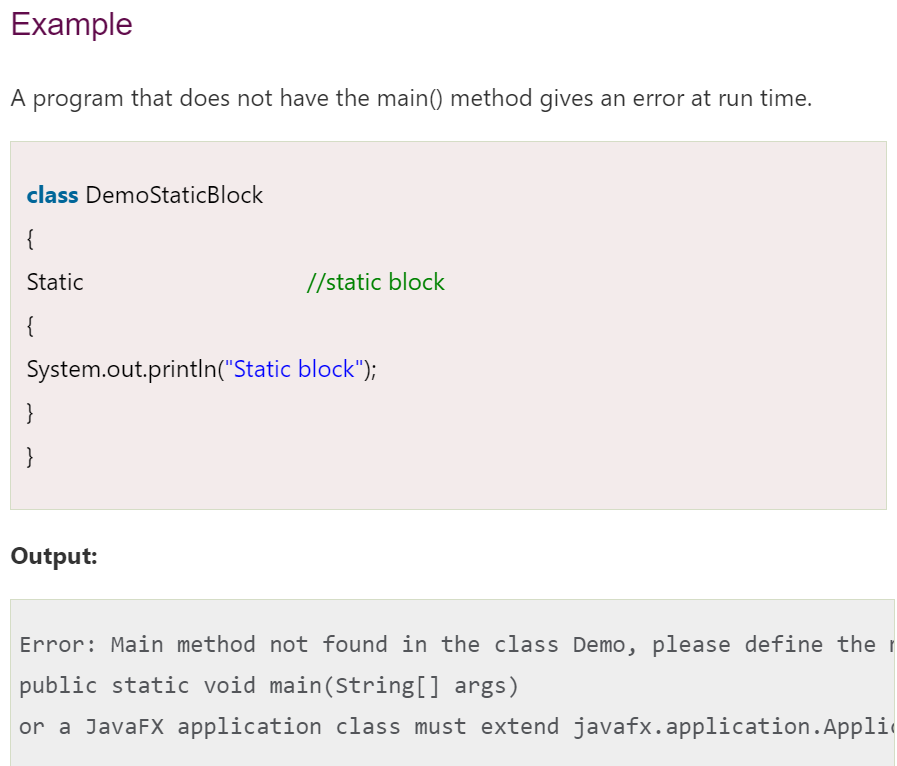
Output :

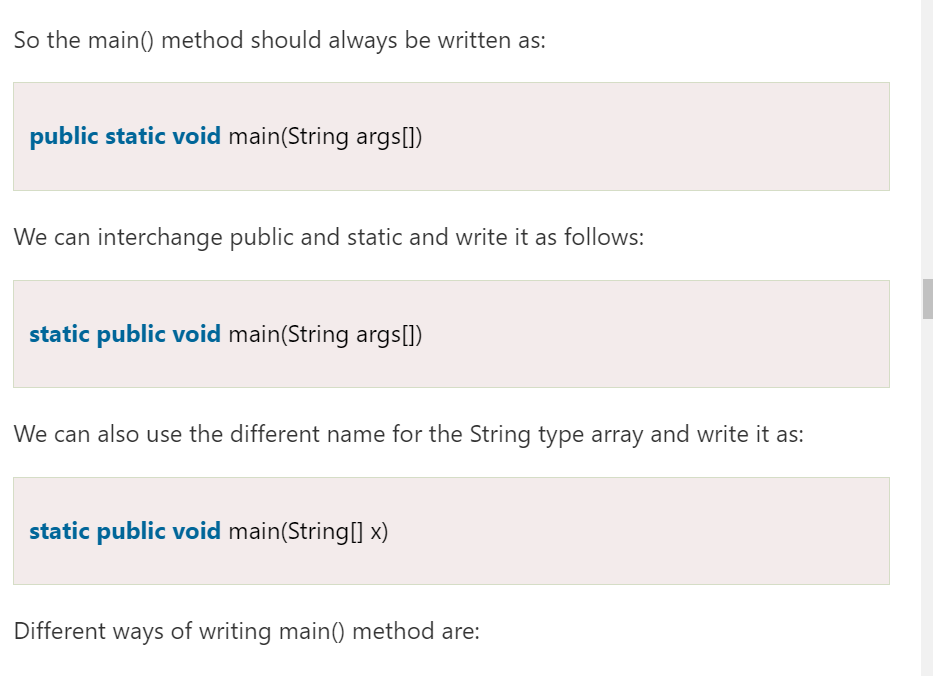
Static block

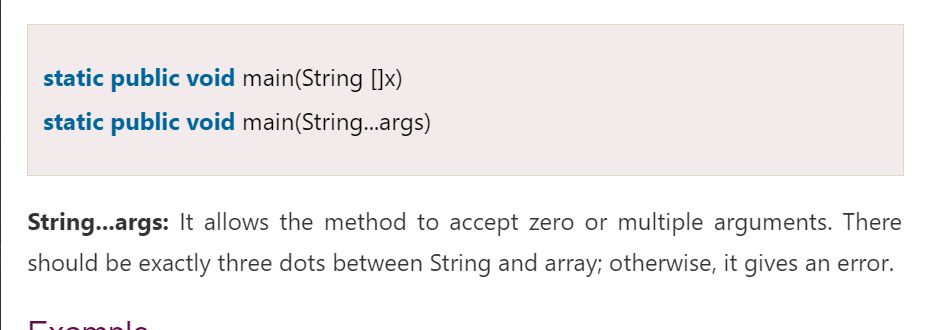
Static method

We observe that JVM first executes the static block, if it is present in the program. After that it searches for the main() method. If the main() method is not found, it gives error.

### **Example**

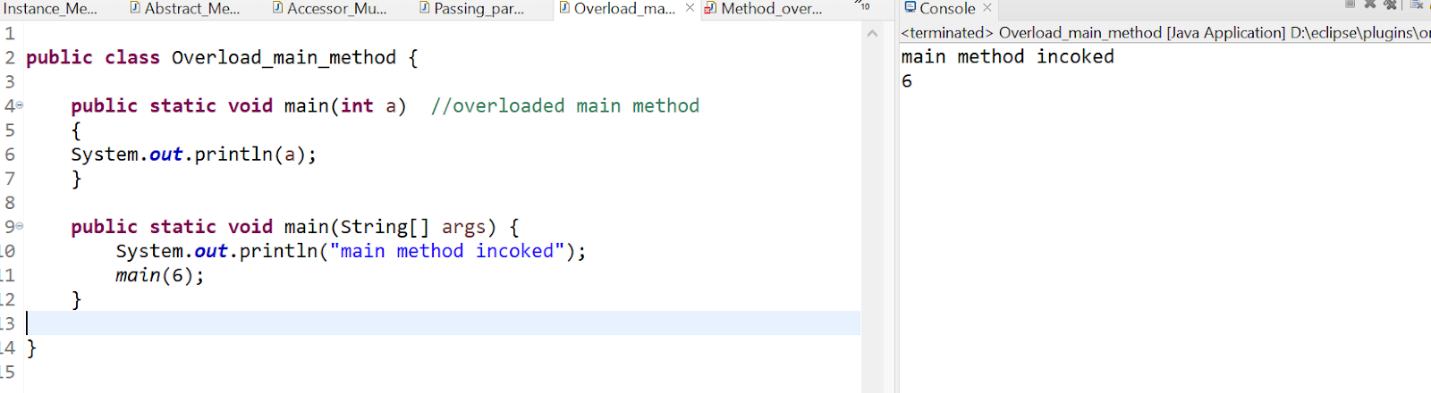






## **Overloading of main() method**

We can also overload the main() method. We can define any number of main() method in the class, but the method signature must be different.



# **Constructors in Java**

* In [Java](https://www.javatpoint.com/java-tutorial), a constructor is a block of codes similar to the method. It is called when an instance of the [class](https://www.javatpoint.com/object-and-class-in-java) 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.

**Note:**

1. 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.
2. We can use access modifiers while declaring a constructor. It controls the object creation. In other words, we can have private, protected, public or default constructor in Java.

### **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**

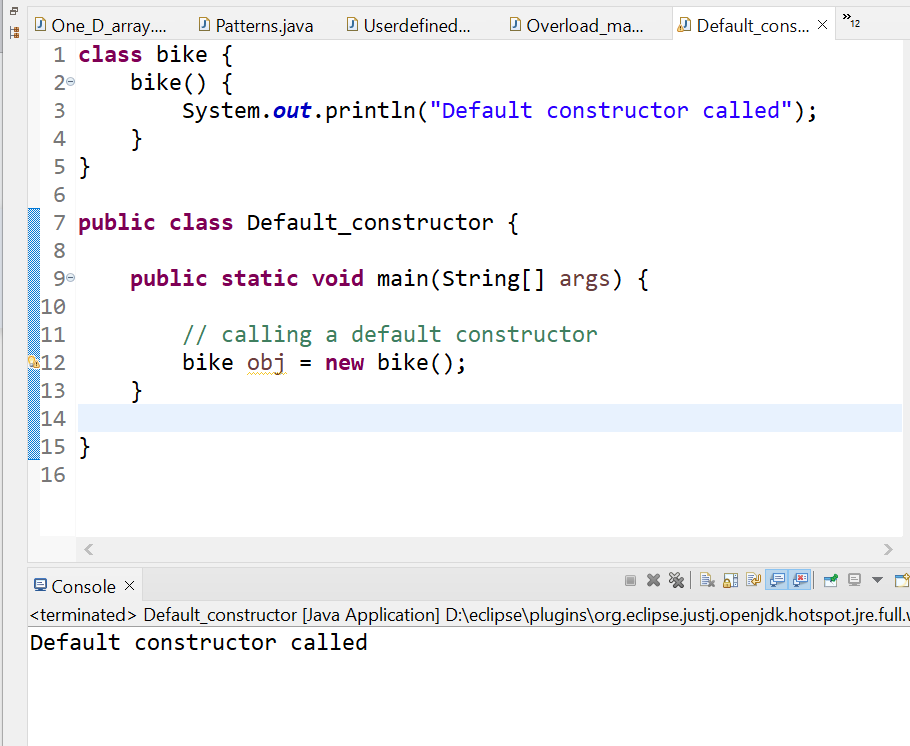
There are two types of constructors in Java:

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

Java Default Constructor

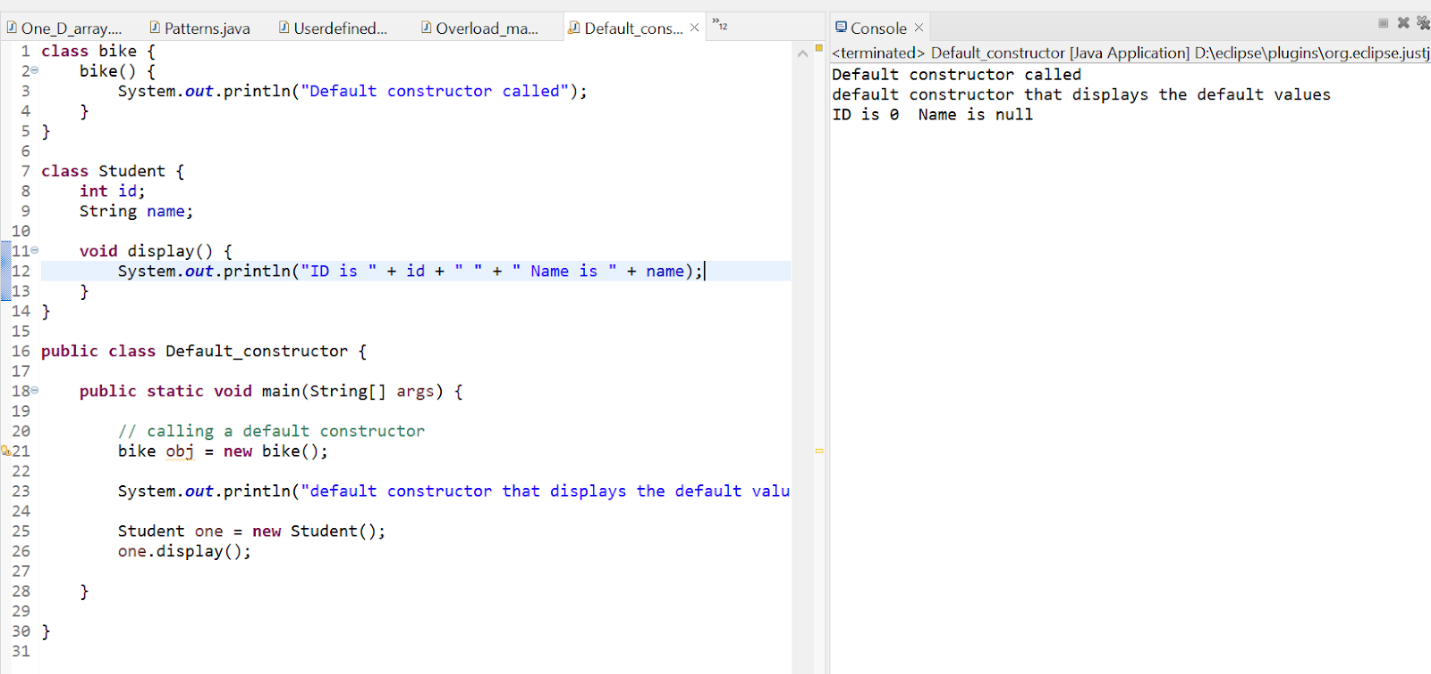
* A constructor is called "Default Constructor" when it doesn't have any parameter.
* Syntax :
  + <class\_name>(){}

* Example :



 Q) What is the purpose of a default constructor?

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.



**:**In the above class,you are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

### **Java Parameterized Constructor**

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

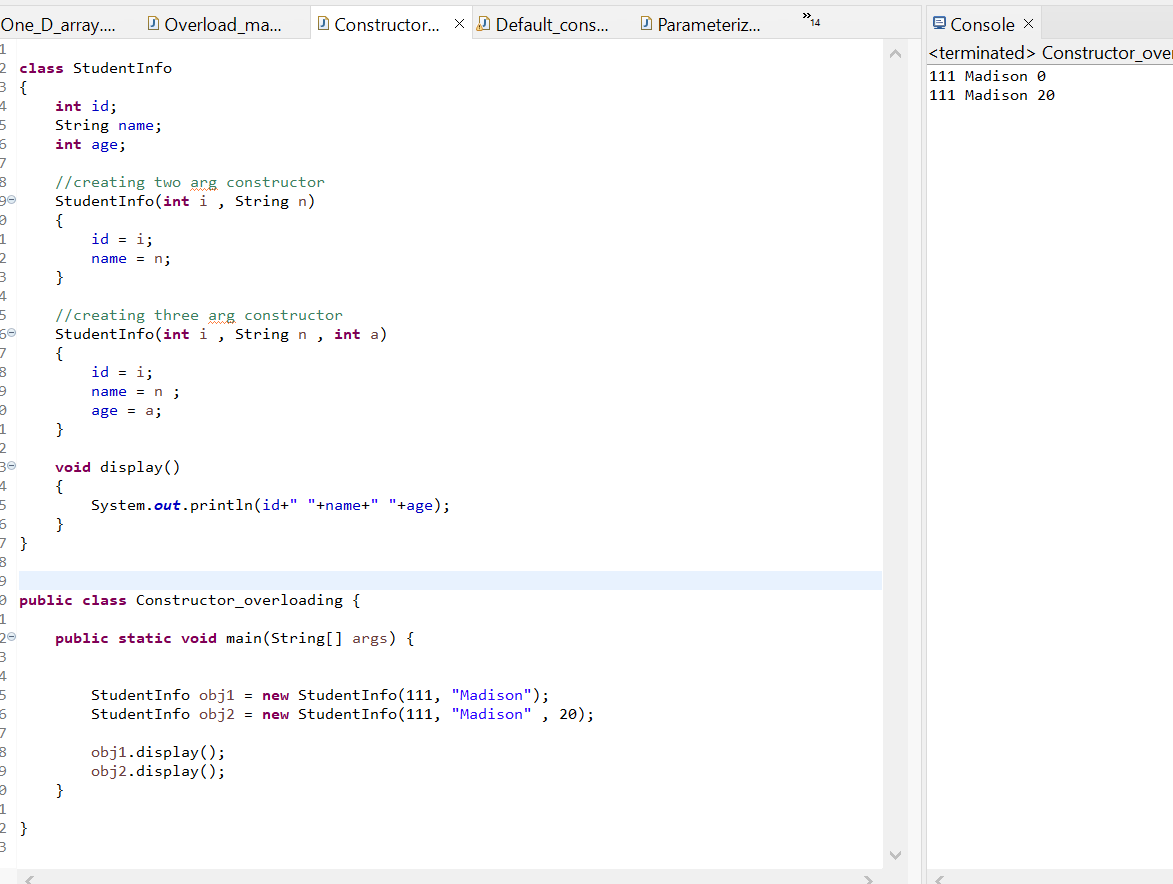
### **Why use the parameterized constructor?**

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

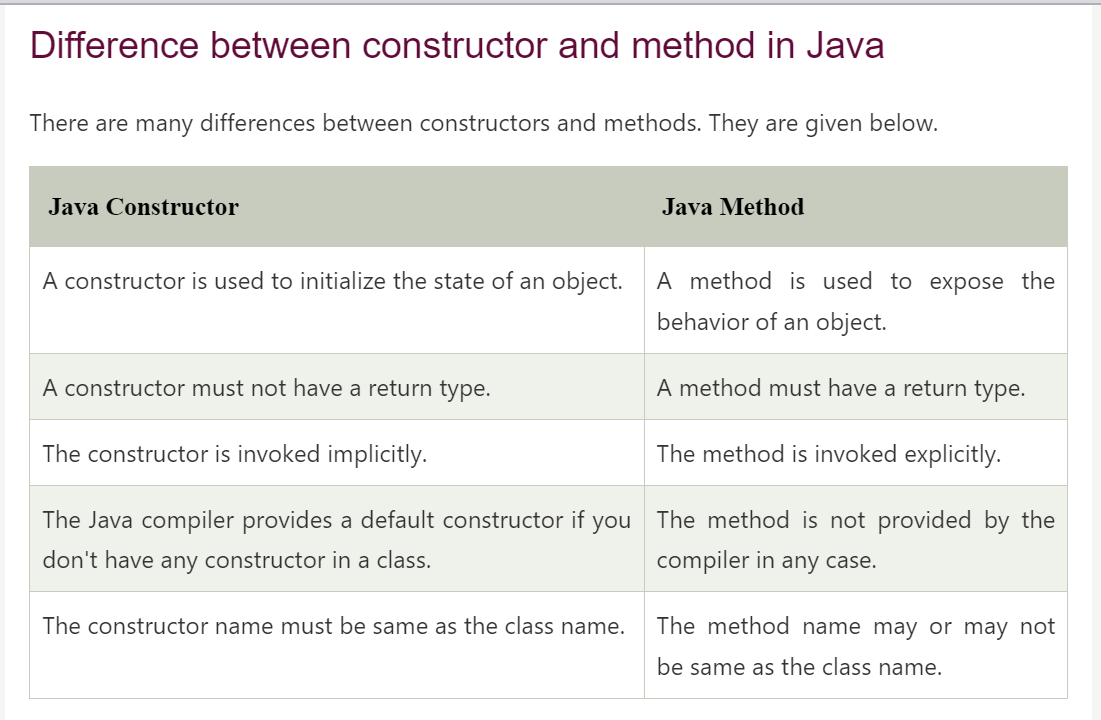


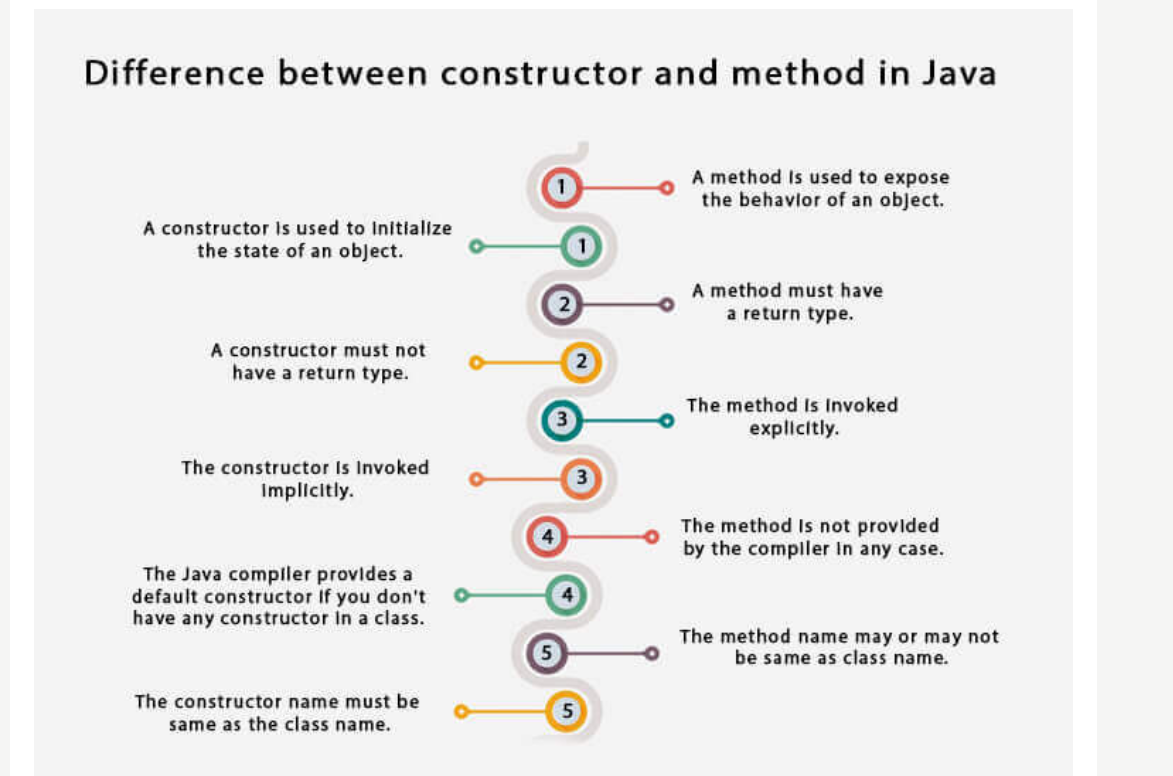
## **Constructor Overloading in Java**

* 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](https://www.javatpoint.com/method-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.

****

## **Difference between constructor and method in Java**

****

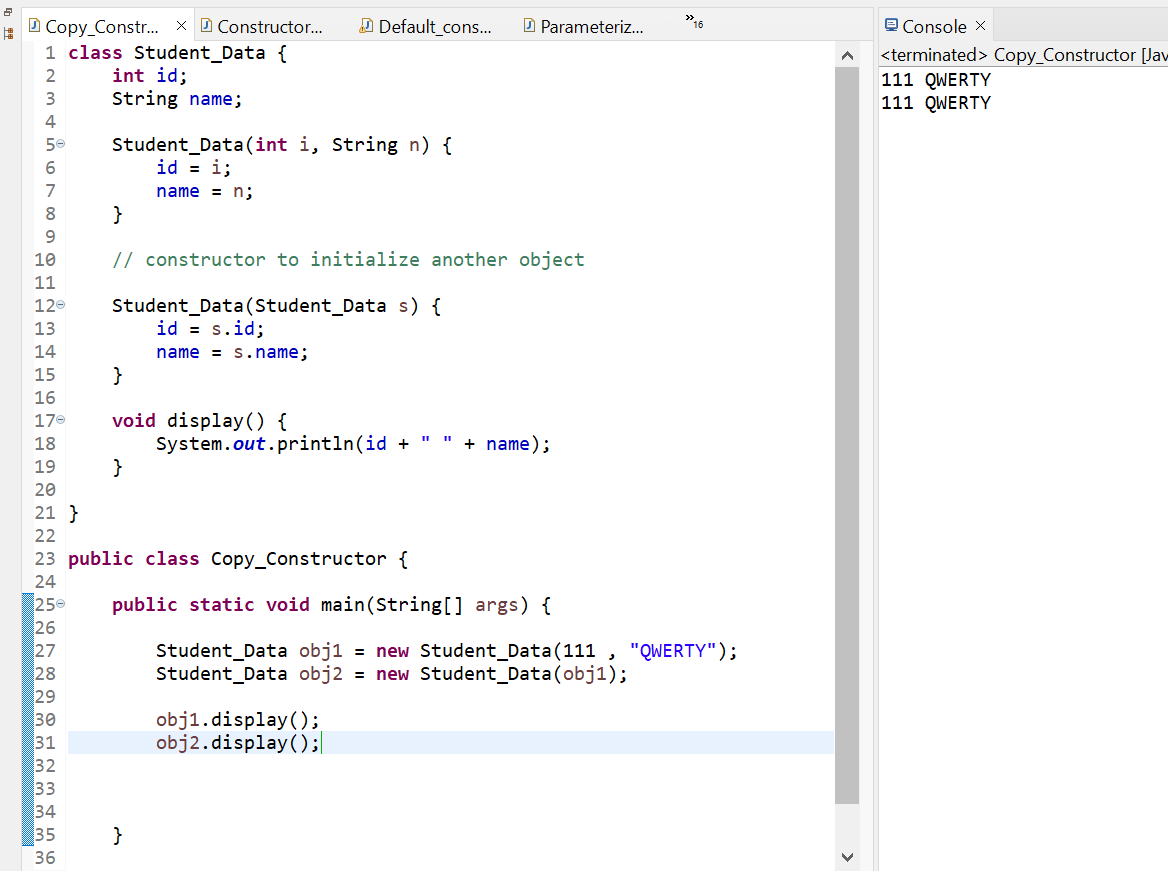
****

## **Java Copy Constructor**

* There is no copy constructor in Java. However, we can copy the values from one object to another like copy constructor in C++.
* There is no copy constructor in Java. However, we can copy the values from one object to another like copy constructor in C++.
  + By constructor
  + By assigning the values of one object into another
  + By clone() method of Object class

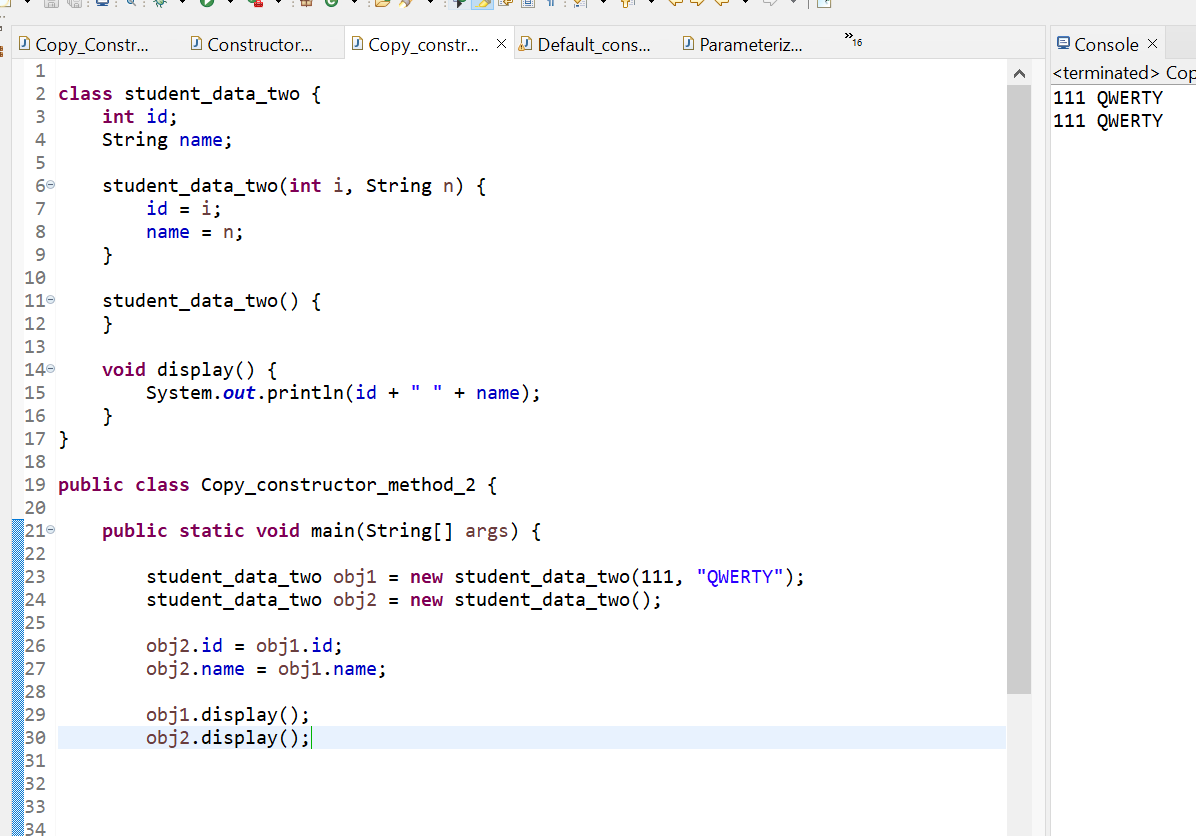
1. By Constructor

we are going to copy the values of one object into another using Java constructor.



1. **assigning the objects values to another object.(Copying values without constructor)**

We can copy the values of one object into another by assigning the objects values to another object. In this case, there is no need to create the constructor.



### **Q) Does constructor return any value?**

Yes, it is the current class instance (You cannot use return type yet it returns a value).

### **Can constructor perform other tasks instead of initialization?**

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

### **Is there Constructor class in Java?**

Yes.

### **What is the purpose of Constructor class?**

Java provides a Constructor class which can be used to get the internal information of a constructor in the class. It is found in the java.lang.reflect package.

# **Java Destructor**

* In Java, when we create an object of the class it occupies some space in the memory (heap).
* If we do not delete these objects, it remains in the memory and occupies unnecessary space that is not upright from the aspect of programming. To resolve this problem, we use the **destructor**.
* The **destructor** is the opposite of the constructor.
* The constructor is used to initialize objects while the destructor is used to delete or destroy the object that releases the resource occupied by the object.
* Remember that **there is no concept of destructor in Java**. In place of the destructor, Java provides the garbage collector that works the same as the destructor.
* The [garbage collector](https://www.javatpoint.com/Garbage-Collection) is a program (thread) that runs on the [JVM](https://www.javatpoint.com/jvm-java-virtual-machine). It automatically deletes the unused objects (objects that are no longer used) and free-up the memory.
* The programmer has no need to manage memory, manually. It can be error-prone, vulnerable, and may lead to a memory leak.
* It is a special method that automatically gets called when an object is no longer used.
* When an object completes its life-cycle the garbage collector deletes that object and deallocates or releases the memory occupied by the object.
* It is also known as **finalizers** that are non-deterministic. In [Java](https://www.javatpoint.com/java-tutorial), the allocation and deallocation of objects handled by the **garbage collector**. The invocation of finalizers is not guaranteed because it invokes implicitly.

## **Advantages of Destructor**

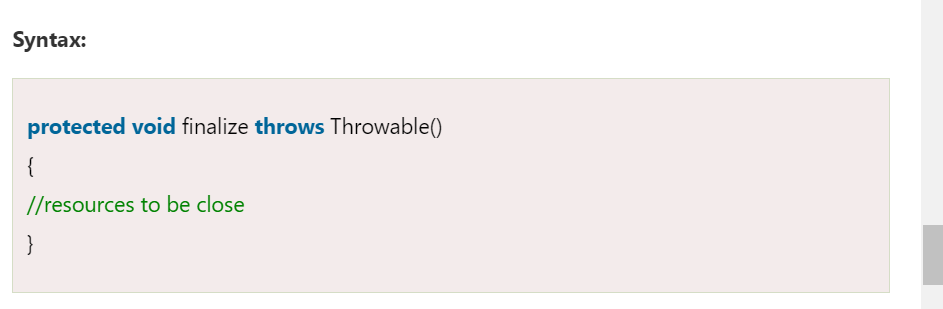
* It releases the resources occupied by the object.
* No explicit call is required, it is automatically invoked at the end of the program execution.
* It does not accept any parameter and cannot be overloaded.

## **How does destructor work?**

* When the object is created it occupies the space in the heap.
* These objects are used by the threads. If the objects are no longer is used by the thread it becomes eligible for the garbage collection.
* The memory occupied by that object is now available for new objects that are being created.
* It is noted that when the garbage collector destroys the object, the JRE calls the finalize() method to close the connections such as database and network connection.
* From the above, we can conclude that using the destructor and garbage collector is the level of developer's interference to memory management.
* It is the main difference between the two. The destructor notifies exactly when the object will be destroyed. While in Java the garbage collector does the same work automatically.
* These two approaches to memory management have positive and negative effects. But the main issue is that sometimes the developer needs immediate access to memory management.

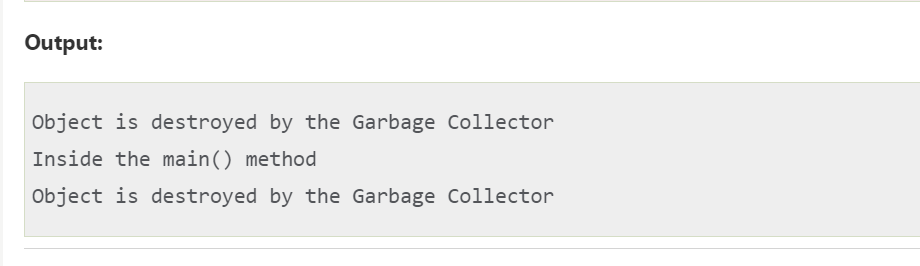
## **Java finalize() Method**

It is difficult for the programmer to forcefully execute the garbage collector to destroy the object. But Java provides an alternative way to do the same. The Java Object class provides the **finalize()** method that works the same as the destructor. The syntax of the finalize() method is as follows:



* It is not a destructor but it provides extra security. It ensures the use of external resources like closing the file, etc. before shutting down the program. We can call it by using the method itself or invoking the method **System.runFinalizersOnExit(true)**.
* It is a protected method of the Object class that is defined in the java.lang package.
* It can be called only once.
* We need to call the finalize() method explicitly if we want to override the method.
* The gc() is a method of JVM executed by the Garbage Collector. It invokes when the heap memory is full and requires more memory for new arriving objects.
* Except for the unchecked exceptions, the JVM ignores all the exceptions that occur by the finalize() method.





# Java - Access Modifiers

* The **access modifiers** in Java are used to specify the [scope of the variables](https://www.tutorialspoint.com/java/java_variable_scope.htm), [data members](https://www.tutorialspoint.com/java/java_class_attributes.htm), [methods](https://www.tutorialspoint.com/java/java_class_methods.htm), [classes](https://www.tutorialspoint.com/java/java_object_classes.htm), or [constructors](https://www.tutorialspoint.com/java/java_constructors.htm). These help to restrict and secure the access (or, level of access) of the data.
* There are four level of access:
  + Visible to the [package](https://www.tutorialspoint.com/java/java_packages.htm), the default. No modifiers are needed.
  + Visible to the class only (private).
  + Visible to the world (public).
  + Visible to the package and all subclasses (protected).

## **Types of Java Access Modifiers**

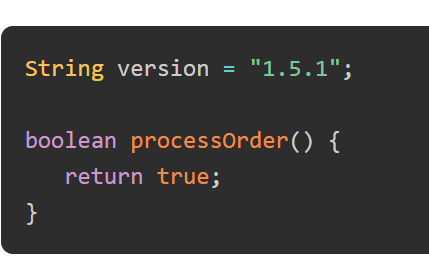
* There are four different types of access modifiers in Java, we have listed them as follows:
  + Default (No keyword required)
  + Private
  + Protected
  + Public

### **1. Default Access Modifier**

* Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.
* A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public.

#### **Example: Default Access Modifier**

* Variables and methods can be declared without any modifiers, as in the following examples −



### **2. Private Access Modifier**

* Methods, variables, and constructors that are declared private can only be accessed within the declared class itself.
* Private access modifier is the most restrictive access level. Class and interfaces cannot be private.
* Variables that are declared private can be accessed outside the class, if public getter methods are present in the class.
* Using the private modifier is the main way that an object encapsulates itself and hides data from the outside world.

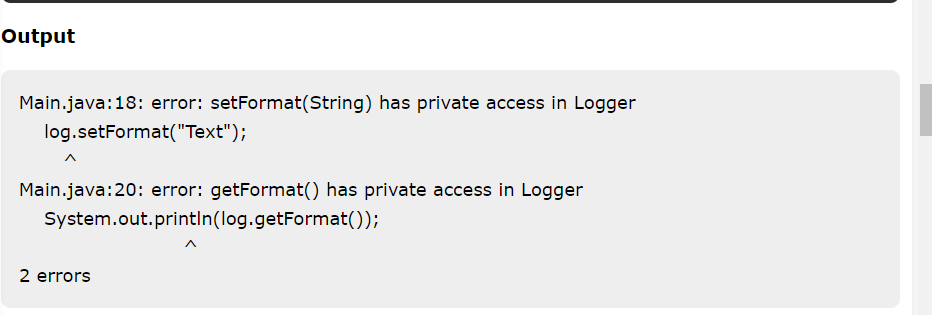


* Here, the format variable of the Logger class is private, so there's no way for other classes to retrieve or set its value directly.
* So, to make this variable available to the outside world, we defined two public methods: getFormat(), which returns the value of format, and setFormat(String), which sets its value.

#### **Example 2: Private Access Modifier**

* In this example, the data members and class methods of the **Logger** class are private. We are trying to access those class methods in another class **Main**.





### **3. Protected Access Modifier**

* Variables, methods, and constructors, which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.
* The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.
* Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.



Here, if we define openSpeaker() method as private, then it would not be accessible from any other class other than AudioPlayer. If we define it as public, then it would become accessible to all the outside world. But our intention is to expose this method to its subclass only, that's why we have used protected modifier.



**Output**

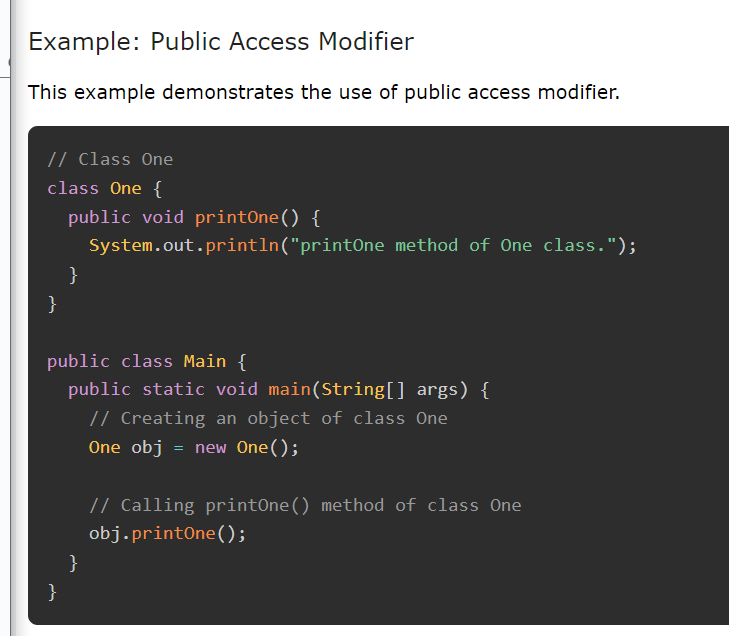
printOne method of One class.

### **4. Public Access Modifier**

* A class, method, constructor, interface, etc. declared public can be accessed from any other class. Therefore, fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.
* However, if the public class we are trying to access is in a different package, then the public class still needs to be imported. Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.



The main() method of an application has to be public. Otherwise, it could not be called by a Java interpreter (such as java) to run the class.



**Output**

This example demonstrates the use of public access modifier.

## **Java Access Modifiers and Inheritance**

The following rules for inherited methods are enforced −

* Methods declared public in a superclass also must be public in all subclasses.
* Methods declared protected in a superclass must either be protected or public in subclasses; they cannot be private.
* Methods declared private are not inherited at all, so there is no rule for them.

The following table shows the summary of the accessibility in the same/different classes (or, packages) based on the access modifiers.



### **Example**

In this example, we've created a class with a private variable age and a variable with default scope as name. Using setter/getter method, we're updating age and getting value and name is updated directly.



**Output**

Age: 2, name: Tommy

##### 