## **Features of Java**

**Open Source**

Java since its birth, till today, is an open-source product and it has all the public access. programmers are free to post entire source code for anyone to download, copy, redistribute them, which is usually part of the GPL  (General Public License,) the license that usually comes with open-source software.

**High Performance**

Java is an interpreted language, so it will never be as fast as a compiled language like C or C++. But, Java enables high performance with the use of the just-in-time compiler.

**Multi-Threading**

Java multithreading feature makes it possible to write a program that can do many tasks simultaneously. The benefit of multithreading is that it utilizes the same memory and other resources to execute multiple threads at the same time, like While typing, grammatical errors are checked along.

**Secure**

When it comes to security, Java is always the first choice. With java secure features it enables us to develop virus-free, temper free system. Java program always runs in Java runtime environment with almost null interaction with system OS, hence it is more secure.

**Platform independent**

Unlike other programming languages such as C, C++ etc which are compiled into platform-specific machines. Java is guaranteed to be write-once, run-anywhere language. On compilation, Java program is compiled into bytecode. This bytecode is platform-independent and can be run on any machine, plus this bytecode format also provides security. Any machine with Java Runtime Environment can run Java Programs.

**Portability**

The cross-platform feature enables the Java code to be highly portable. Programmers can write code in windows and execute the same code in Linux Operating System.

**Object-Oriented**

In java, everything is an object which has some data and behaviour. Java can be easily extended as it is based on Object Model.

**Robust**

Java makes an effort to eliminate error-prone codes by emphasizing mainly on compile-time error checking and runtime checking. But the main areas which Java improved were Memory Management and mishandled Exceptions by introducing automatic Garbage Collector and Exception Handling.

**Java: Member Variables**

A member variable plays a major role in a class as it is used to store a data value. When we define a class, we can declare a member variable. These variables are members of a class.  
Member variables are further classified into three types:

* Local variable
* Instance variable
* Class/Static variable

Local variable: These are the variables which are declared within the method of a class. Let’s understand this with a programmatic example:

**public** **class** Car {

**public** **void** display(**int** m){  // Method

**int** model=m;  // Created a local variable model

           System.out.println("Model of the car is" +model);

     }

}

Instance variable: Instance variable is declared in a class but outside a method, constructor or any block.

**public** **class** Car {

**public** String color;     // Created an instance variable color

      Car(String c){

            color=c;

      }

**public** **void** display() {  // Method

            System.out.println("color of the car is"+color);

      }

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

           Car obj=**new** Car("black");

                  obj.display();

      }

}

Class variable: Class variables are also called as static variables. These variables have only one copy that is shared by all the different objects in a class.

**public** **class** Car {

**public** **static** **int** tyres;   // Created a class variable tyres

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

           tyres=4;

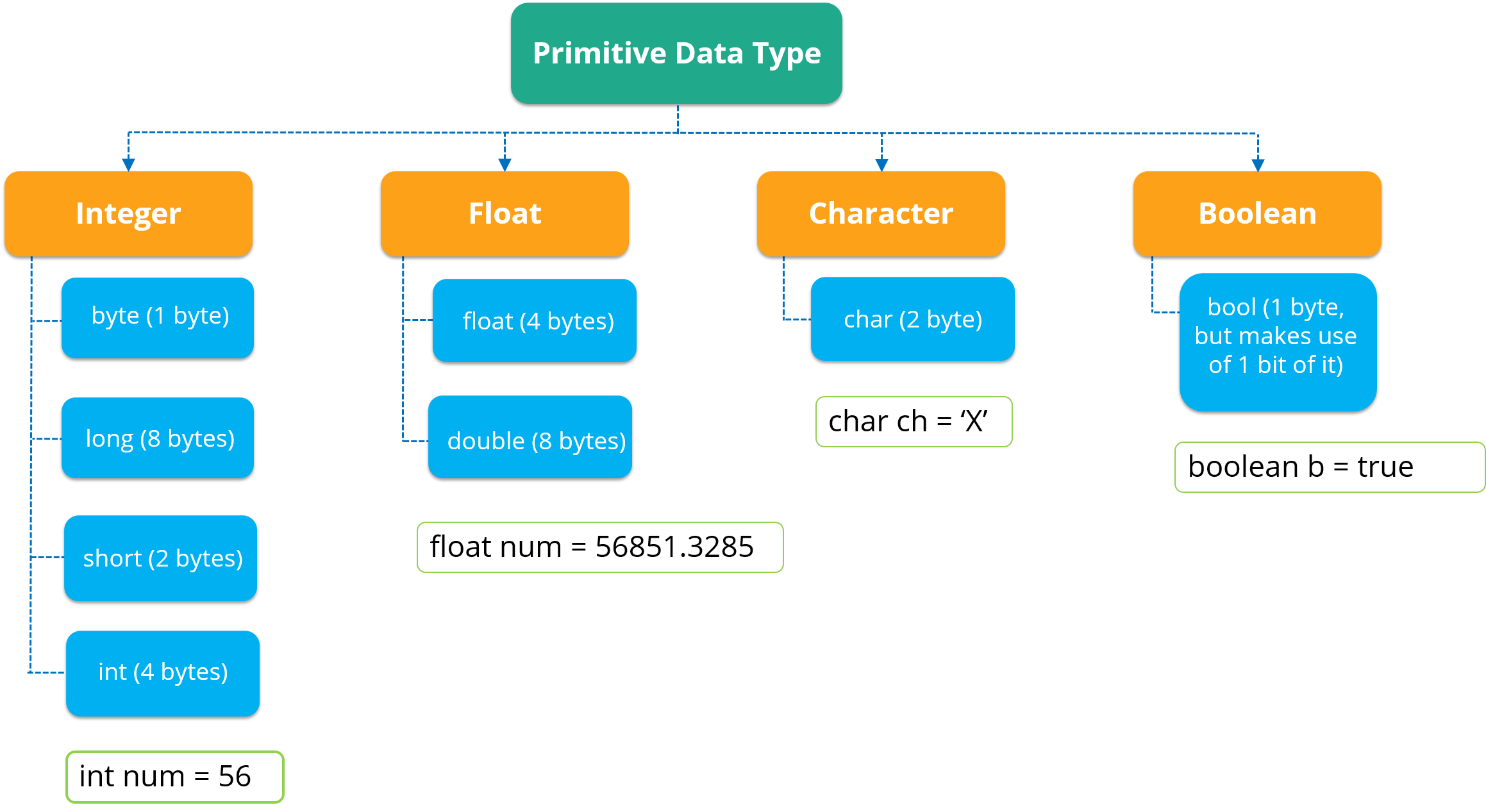
           System.out.println("Number of tyres are"+tyres);

      }

}

## **Java Tutorial: Datatypes**

A data type is used to represent different values which are stored in a variable. They are mainly classified into 4 different aspects – Integer, Float, Character and Boolean.



## **Java : Data Operators**

There are mainly 4 different types of operators, which are listed below:

* **Arithmetic Operator:** Perform arithmetic operations such as addition, subtraction, multiplication, division and modulus.
* **Unary Operator:** Unary operators are used to increments or decrements a particular value. For example: ++ stands for increment, – – stands for decrements.
* **Relational Operator:** It defines some kind of relation between two entities. For example: <, >, <=, >=, !=, ==.
* **Logical Operator:** Logical operators are typically used with boolean (logical) values.

## **Java : Classes and Objects**

A class in Java is a blueprint which includes all your data.  A class contain fields(variables) and methods to describe the behaviour of an object.

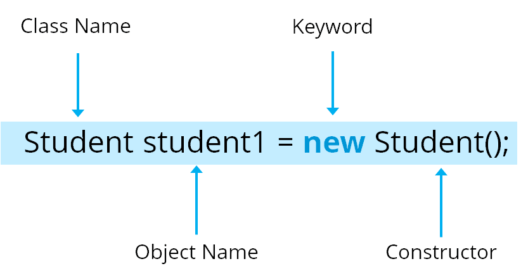
**class** Abc {

      member variables // class body

        methods

 }

An object is a major element in a class which has a state and behaviour. It is an instance of a class which can access your data.

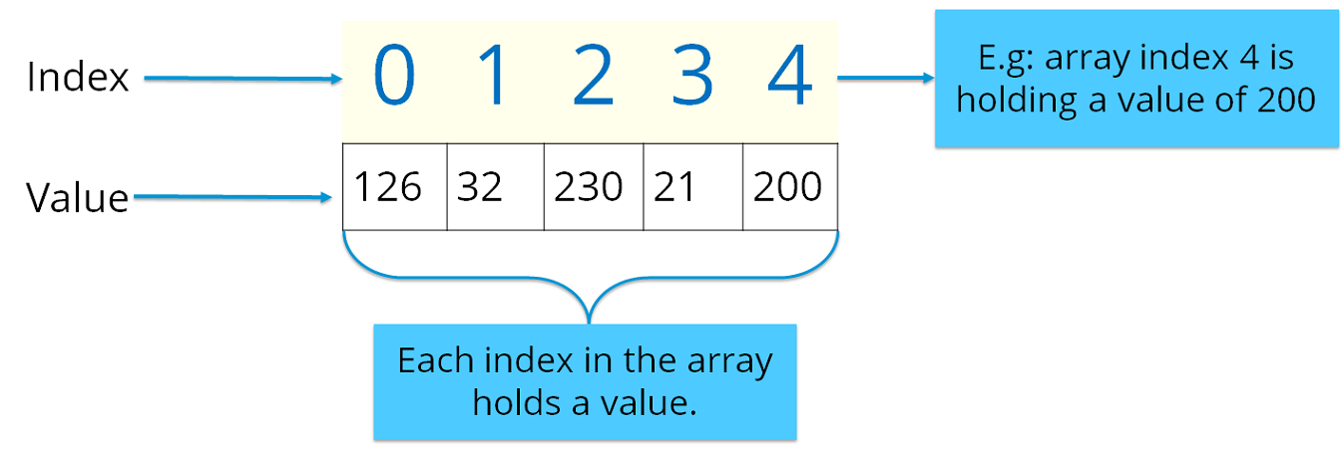


## **Java : Arrays**

Arrays in Java is similar to that of C++ or any other programming language. An array is a data structure which holds the sequential elements of the same type.

Let’s say you want to store 50 numbers. Instead of declaring individual variables, such as number0, number1, … and so on. You can declare one array variable – “numbers” and use number[0], the number[1] to represent individual variables. This will ease your task and minimizes the redundancy.

Each array has two components: index and value.



There are two types of arrays in Java:

* Single-dimension Array
* Multi-dimension Array

**Single-dimension Array:** In a single-dimension array, a list of variables of the same type can be accessed by a common name. You can initialize the array using the following syntax:

int a[] = new int[12];

**Multi**–**dimension** **Array:**In a multi-dimension array, your data is stored in a matrix form. Here, you can initialize the array using the following syntax:

int table[][]= new int[4][5];

**import** java.util.\*;

**public** **class** ArrayExample {

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

     {

**double** invoice[][]= **new** **double**[10][2];  // Initializing array

          Scanner obj= **new** Scanner(System.in);    // creating a scanner object to take input from user

**for**(i=0;i<10;i++){                       // nested for loops

**for**(j=0;j<2;j++);

               {

               System.out.println("Enter the value");

               invoice[i][j]=obj.nextDouble();         // store values to array

**for**(i=0;i<10;i++){

**for**(j=0;j<2;j++)

               {

                System.out.println(invoice[i][j]);

               }

         }

     }

}

**public** **class** Car{                    // Class creation

       String color;                        // Member variables

       String model;

**public** **void** SpeedCheck(**int** s)        // Method

         {

**int** speed=s;

**if**(speed>100)                        // Control statement

           {

           System.out.println(" You are driving very fast");

           }

**else**

           {

            System.out.println("You are driving at normal speed");

           }

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

         {

         Car obj= **new** Car();                  // Object creation

         obj.speed(60);

         }

## **File Input/Output Handling**

Java has a dedicated library to handle all its input and output functionalities. It is the java.io package/library that handles all the input/output streams in Java. Java has two types of streams, they are:

* Input Stream
* Output Stream

**Input Stream**

It is responsible for reading data from all the sources.

**Output Stream**

It is responsible for writing the data to the destination.

**import** java.io.\*;

**public** **class** fileStreamTest {

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

**try** {

**byte** bWrite[] = { 1, 2, 3, 4, 5 };

                      OutputStream os = **new** FileOutputStream("Edureka.txt");

**for** (**int** x = 0; x < bWrite.length; x++) {

                            os.write(bWrite[x]);

                      }

                      os.close();

                      InputStream is = **new** FileInputStream("Edureka.txt");

**int** size = is.available();

**for** (**int** i = 0; i < size; i++) {

                            System.out.print((**char**) is.read() + " ");

                      }

                      is.close();

                      }

**catch** (IOException e) {

                       System.out.print("Exception");

                 }

         }

}

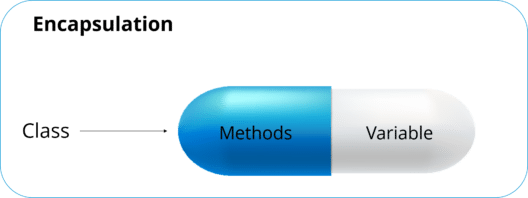
## **Java : OOPs Concept**

We have already discussed classes and objects in Java. Let’s discuss the 4 main concepts of object-oriented programming – Inheritance, Encapsulation, Polymorphism and Abstraction.

Let’s begin with the first concept i.e. Inheritance.

**Inheritance:** Most of you must be familiar with inheritance. Inheritance is a process where one class acquires the properties of another. But whose properties are inherited? Here we have two classes, a child class which inherits the properties of a base class.  
A Class which inherits the properties are known as **Child class.**It is also referred to as a derived class or a subclass. Next, the class whose properties are inherited are known as **Parent class**or a base class.

Encapsulation: Encapsulation in Java is a mechanism of wrapping up the data and code together as a single unit. Refer to the below image where all your methods, variables are bound together in a single class.



In encapsulation, the variables of a class will be hidden from other classes and can be accessed only through the methods of their current class.

**Polymorphism:**Polymorphism is the ability of a variable, function or object to take multiple forms. The most common use of polymorphism in OOPs occurs when a parent class is used to refer to a child class object. Polymorphism is also achieved through function overloading.

Abstraction: It is basically the quality of dealing with ideas rather than events. Abstraction is the methodology of hiding the implementation details from the user and only providing the functionality to the users. Let’s see this real-life example of a car where i’ll help you understand what exactly abstraction is.



## **Exception Handling**

An exception can be defined as an unexpected problem that arises during the execution of the program. The exception interrupts the sequential and normal flow of the program. Hence, it becomes mandatory to resolve these exceptions to avoid any problems.

## Hierarchy of Java Exception classes

The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:



### Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

## Difference between Checked and Unchecked Exceptions

### 1) Checked Exception

The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

### 2) Unchecked Exception

The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

### 3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

## Java Exception Keywords

There are 5 keywords which are used in handling exceptions in Java.

|  |  |
| --- | --- |
| Keyword | Description |
| try | The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone. |
| catch | The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later. |
| finally | The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not. |
| throw | The "throw" keyword is used to throw an exception. |
| throws | The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature. |