### Why was Object-Oriented Programming Needed?

1. **Manage Complexity:** As software applications grew larger and more complex, procedural programming (where code is written in a linear top-down approach) became harder to manage and maintain. OOP helps manage this complexity by organizing code into reusable and modular components.
2. **Code Reusability:** Through inheritance, OOP allows the creation of new classes that reuse, extend, and modify the behavior defined in other classes. This promotes code reuse and reduces redundancy.
3. **Improved Maintainability:** Encapsulation and abstraction make it easier to manage and update code. Changes to a class's implementation can be made with minimal impact on the code that uses the class, as long as the interface remains consistent.
4. **Real-World Modelling:** OOP is more aligned with real-world concepts, making it intuitive for developers to model complex systems by thinking in terms of objects and their interactions.
5. **Scalability:** OOP facilitates the development of applications that are scalable and can grow over time. The modular nature of OOP allows for easier addition of new features without disrupting existing functionality.

**Introduction and Editions of Javafooter line**

**Introduction**

Java is an object-oriented, class-based, secured, platform-independent, and general-purpose programming language. Java was originally developed by James Gosling at Sun Microsystems and released in 1995 as a core component of Sun Microsystem’s Java platform. Java programming language is based on the write once, run anywhere (WORA) principle, meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java Virtual Machine (JVM) regardless of the underlying operating system.

**Different Editions of Java**

There are four editions of the Java programming language.

* Java Platform, Standard Edition (Java SE)
* Java Platform, Enterprise Edition (Java EE)
* Java Platform, Micro Edition (Java ME)
* JavaFX

1. Java SE: Java SE, also known as Core Java, is the most basic and standard version of java. It consists of a wide variety of general-purpose APIs (like java.lang, java.util) as well as many special-purpose APIs. Java SE is used to create Desktop applications. It defines everything from the basic types and objects of java programming language to high-level classes that are used for networking, security, database access, graphical user interface development (GUI), and XML parsing.
2. Java EE: The Java EE platform is built on top of the Java SE platform. The Enterprise Edition version of java has a much larger usage of Java, like the development of web services, networking, server-side scripting, and other various web-based applications. Java EE uses HTML, CSS, JavaScript, etc., so as to create web pages and web services. It is also one of the most widely used web development standards.
3. Java ME: The Java ME platform is widely used for developing embedded systems, mobiles, and small devices. Java ME uses many libraries and APIs of Java SE, as well as many of its own. The basic aim of this edition is to work on mobiles, wireless devices, set top boxes, etc. Most of the apps developed for the phones were built on Java ME only.
4. JavaFX: JavaFX is another edition of java technology, which is now merged with Java SE 8. It is mainly used to create rich GUI (Graphical User Interface) in java apps. It is supported by both desktop environments as well as web browsers.

**Features and Uses of Java**

The features of the Java programming language are given below.

1. **Simple:**Java programming language is easy to learn and its syntax is simple and easy to understand. According to Sun Microsystem, the java programming language is simple because the java syntax is similar to C++. So, it is easier for the programmer to learn java after C++.  Java has removed many rarely-used features like explicit pointers, operator overloading, etc. In java, there is no need to remove unused objects because there is an automatic garbage collection in java.
2. **Object-Oriented:**Java is an object-oriented programming language. Everything in java is an object. We can use the functionality of our program by using objects. Object-oriented programming language simplifies software development and maintenance by providing some rules. The basic concepts of OOPs are classes and objects, packages, polymorphism, inheritance, abstraction, encapsulation, and etc.
3. **Platform independent:**Java is a platform-independent programming language. We can write java code once and run it anywhere. Java code can be run on multiple platforms, for example, Windows, Linux, Mac, etc. Java code is compiled by the compiler and converted into byte code. This byte code is platform-independent code because it can be run on multiple platforms.
4. **Robust:**Java programming language is robust because it uses strong memory management. There is an automatic garbage collection in java to destroy the objects which are unused. Exception handling is another feature that makes java programming robust.
5. **Portable:**Java programming language is portable because it enables you to carry byte code to any platform.
6. **Multithreaded:**Java also supports multithreaded programming. We can create multiple threads in java. The main advantage of multithreading is that it doesn’t occupy the memory for each thread. It shares a common memory area. Threads are important for web applications, multimedia applications, etc.
7. **Distributed:** Java programming language is distributed because it facilitates the user to create distributed applications. We can create distributed applications in java by using RMI and EJB.
8. **Secure:**When a java program is compiled, it generates a byte code that is in the non-readable form. The java byte code cannot be read by humans. That makes the java language secure.

**Uses of Java**

Java is the most popular, widely used object-oriented programming language. By using java, we can develop a variety of applications such as web applications, mobile applications, desktop applications, network applications, and many more. Java is used for developing different applications. Some of them are listed below:

1. **Banking:**In banking, the Java programming language is used to develop transaction management.
2. **Mobile App Development:**Java can be used to develop mobile applications. Most of the android applications are built using java. So, if you are familiar with java, it will become much easier to develop mobile applications.
3. **Desktop Applications:** We can also create GUI applications in java. Java provides AWT, Swings, and JavaFX to develop desktop applications.
4. **Big Data:** In Big Data, the Hadoop MapReduce framework is written using java.
5. **Web applications:**We can also create web applications using java. The most popular frameworks like Spring, Spring Boot, Hibernate used for developing web applications are based on java.

JDK, JRE, and JVM

JDK, JRE, and JVM are the most important parts of the Java programming language. Without these, you can not develop and run java programs on your machine.

JDK: JDK stands for Java Development Kit. JDK provides an environment to develop and execute the java program. JDK is a kit that includes two things - Development Tools to provide an environment to develop your java programs and JRE to execute your Java programs.

JRE: JRE stands for Java Runtime Environment. JRE provides an environment to only run (not develop) the java programs onto your machine. JRE is only used by the end-users of the system. JRE consists of libraries and other files that JVM uses at runtime.

JVM: JVM stands for Java Virtual Machine, which is a very important part of both JDK and JRE because it is inbuilt in both. Whatever java program you run using JDK and JRE goes into the JVM and JVM is responsible for executing the java program line by line.

**main() Method**

Before explaining the java main() method, let’s first create a simple program to print Hello World. After that, we will explain why the main() method in java is public static void main(String args[]).

public class HelloWorld {

public static void main(String args[]) {

System.out.println("Hello World");

}

}

**public**: the public is an access modifier that can be used to specify who can access this main() method. It simply defines the visibility of the method. The JVM calls the main() method outside the class. Therefore it is necessary to make the java main() method as public.

**static**: static is a keyword in java. We can make static variables and static methods in java with the help of the static keyword. The main advantage of a static method is that we can call this method without creating an instance of the class. JVM calls the main() method without creating an instance of the class, therefore it is necessary to make the java main() method static.

**void**: void is a return type of method. The java main() method doesn’t return any value. Therefore, it is necessary to have a void return type.

**main**: main is the name of the method. It is a method where program execution starts.

**String args[]:** String in java is a class that is used to work on Strings and args is a reference variable that refers to an array of type String. If you want to pass the argument through the command line then it is necessary to make the argument of the main() method as String args[].

**Compile and run Java programs:**

* Below are the steps to compile and run your first java program
* Open a text editor like notepad or notepad++.
* Write a java program in your text editor and save it with (.java extension), remember that the file name and the class name may be the same. For example, if you are creating a class HelloWorld, then you need to save it with HelloWorld.java.
* Open your command prompt.
* Set the directory where your java program is saved.
* Once you enter the directory where your java program is saved, now it’s time to compile your java program, to compile the java program use the command javac HelloWorld.java (In this case my java file is HelloWorld.java).
* Once your Java program compiles fine, now it’s time to run your java program. You can run your java program by using java HelloWorld (java file\_Name) command.

**Comments in Java**

Comments are statements that are not executed by the compiler. Comments make the program more human-readable by including the details of the code involved. The proper use of comments makes maintenance and debugging of the code easier.

Types of Comments

There are three types of comments in java

* Single line comments
* Multi-line comments
* Documentation comments

* **Single line comment**: The single line comment is used to comment only one line. A beginner-level programmer uses mostly single-line comments for describing the code functionality.

Syntax:

// write your comment here

Example:

// Java program to show single line comment

public class HelloWorld {

public static void main(String args[]) {

// Print "Hello World" on console

System.out.println("Hello World");

}

}

* **Multi-line comments**: The multi-line comments are used to comment multiple lines of code.

Syntax:

/\*

This

is

multiline

comment

\*/

Example:

public class HelloWorld {

public static void main(String args[]) {

/\*

Here, we have

declared a

variable and we are

printing its value

\*/

int a = 10;

System.out.println(a);

}

}

* **Documentation Comments**: This type of comment is used generally when we write the code for projects. It helps to generate a documentation page for reference, which can be used for getting information about methods present, their parameters, etc.

Syntax:

/\*\*Comment start

\*

\*tags are used in order to specify a parameter

\*or method or heading.

\*HTML tags can also be used

\*such as <h1>

\*

\*comment ends\*/

Example:

// Java program to illustrate documentation comments

/\*\*

\* Find product of four numbers! The FindPro program implements an application

\* that simply calculates product of four integers and Prints the output on the

\* screen.

\* \* @author Prashant Srivastava

\* @version 1.0

\* @since 2021-02-26

\*/

public class FindPro {

/\*\*

\* This method is used to find the product of four integers.

\* \* @param num1 This is the first parameter to FindPro method

\* @param num2 This is the second parameter to FindPro method

\* @param num3 This is the third parameter to FindPro method

\* @param num4 This is the fourth parameter to FindPro method

\* @return int This returns average of numA, numB and numC.

\*/

public int FindPro(int num1, int num2, int num3, int num4) {

return (num1 \* num2 \* num3 \* num4);

}

/\*\*

\* This is the main method which makes use of the FindPro method.

\* \* @param args Unused.

\* @return Nothing.

\*/

public static void main(String args[]) {

FindPro obj = new FindPro();

int pro = obj.FindPro(10, 20, 30, 40);

System.out.println("Product of 10, 20, 30 and 40 is : " + pro);

}

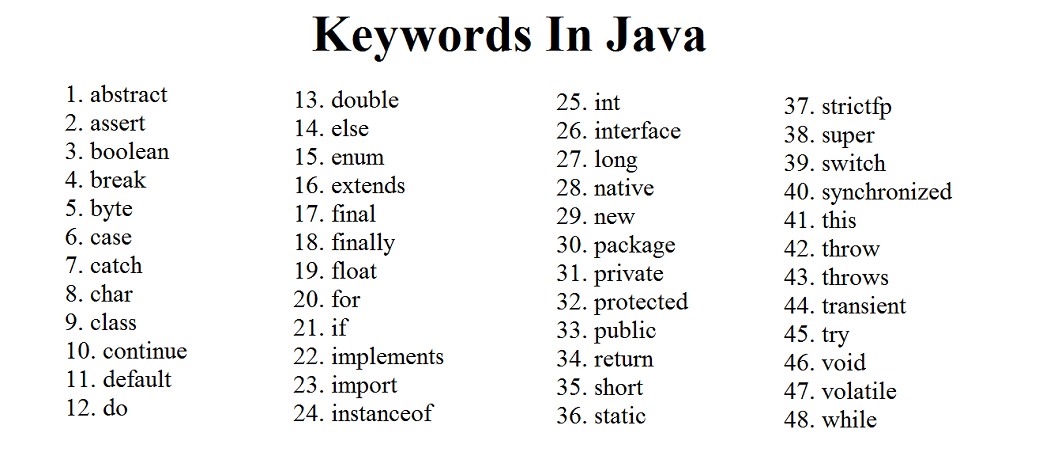
}

Output:

Product of 10, 20, 30 and 40 is : 240000

# **Java Keywords:**

Keywords in Java are also known as reserved words. These are the predefined words therefore they can’t be used as a variable name. If we will use keywords as a variable name, the result will be a compile-time error. The list of all the Java Keywords is given below.



**Data types in Java**

The data type defines the type of value that can be stored in a variable. For example, if a variable has an int data type, it can only store an integer value. In java, there are two categories of data types.

Primitive Data Type: A primitive data type is predefined by the language and is named by a keyword or reserved keyword. There are eight types of primitive data types in java such as boolean, char, int, short, byte, long, float, and double.

Boolean: Boolean data type specifies only one bit of information and it is used to store only two possible values either true or false. Default value is false.

byte: byte data type is 8 bit signed two’s complement integer. Its value lies between -128 to 127. It has a minimum value of -128 and a maximum value of 127 (inclusive). The byte data type is most commonly used to save memory in large arrays.

short: short data type is a 16-bit signed two’s complement integer. It can hold any number between -32768 to 32767 (inclusive). Like byte data type, it is commonly used to save memory in large arrays.

int: int data type is 32-bit signed two’s complement integer. It can hold the number between -2,147,483,648 to 2,147,483,648. The default value of the int data type is 0.

long: long data type is 64-bit two’s complement integer. It can hold the number between -2^63 to 2^63-1. The default value of long data type is 0.

float: float data type is used to store floating-point numbers. The float data type is a single-precision 32-bit IEEE 754 floating-point. It can hold 6 to 7 decimal digits. It is recommended to use float instead of double if you need to save memory in large arrays of floating-point numbers. The default value of float is 0.0f.

double: double data type is generally used to store decimal values. The double data type is a double-precision 64-bit IEEE 754 floating-point. For decimal values, this data type is generally the default choice. The default value of double is 0.0d.

char: The char data type is used to store characters. The char data type is a single 16-bit Unicode character.

2. Non-Primitive Data Type: Non-Primitive data type refers to the objects. ArrayList and String are some of the examples of Non-Primitive data type. We will discuss the Non-Primitive data type later.

Example

// Primitive Data Types

int price = 5000; // Integer Value

float rateOfInterest = 5.99f; // Floating point number

char ch = 'a'; // Character

// Non-Primitive Data Types

String str = "Coding Ninjas"; // String

## ****Scope of Variables in Java****

The variable scope is the part of the program where the variable is accessible. The scope of the variable can be determined at compile time. There are mainly two types of variable scope.

**1) Local Variables Scope:**A variable that is defined inside a block, method body, or constructor is called a local variable. These variables can’t be accessed outside the method.

**Example:**

public class VariableScope {

void method() {

*// local variable (Method Level Scope)*

*// This method can’t be accessed outside*

*// method body.*

int x;

}

}

**2) Member/Class Level Variable Scope:**A variable that is declared inside the class but outside the method body, block, or constructor is known as member/class level variable. These variables can be directly accessed anywhere in the class.

**Example:**

class VariableScope {

*// variable defined inside the class*

int x;

}

public class VariableScopeDemo {

public static void main(String args[]) {

*// Creating VariableScope class object*

VariableScope obj = new VariableScope ();

*// Assigning values in the variable*

obj.x = 10;

*// Printing the value*

System.out.println(obj.x);

}

}

**Output:**

10

**Types of Variable**

A variable is a name given to memory location. There are three types of variables in java.

* Local Variable.
* Instance Variable.
* Static variable

**1.** **Local Variables:**A variable that is defined inside a block, method body, or constructor is called a local variable. These variables are created when the methods are called and they get destroyed when the methods are executed and return to the caller.

The initialization of the local variable is mandatory. If you don’t initialize the variable before use, the compiler will give a compile-time error.

**Example:**

public class Addition {

*// Function to add two numbers*

  public void add() {

*// Local variables*

     int a = 10;

     int b = 20;

     int c = a + b;

*// Printing the sum*

     System.out.println(c);

  }

*// Driver Code*

  public static void main(String args[]) {

*// Creating an object of Addition class*

     Addition obj = new Addition();

*// Function Call*

     obj.add();

  }

}

**Output:**

30

**2. Instance Variables:**A variable that is declared inside the class but outside the method body, block, or constructor is known as an instance variable. It is a non-static variable. These variables are created when an instance (object) of the class is created and are destroyed when the object is destroyed. Initialization of the instance variable is not mandatory. **Even If you don’t initialize the instance variable, it has a default value in it**. Instance variables can be accessed only by creating the object of the class.

**Example:**

class Student {

*// These are instance variables*

*// these are declared inside the*

*// class but outside the method body*

String name;

int rollno;

}

public class StudentRecords {

public static void main(String args[]) {

*// Creating Student class object*

Student obj = new Student();

*// Assigning values in the variables*

obj.name = "Ram";

obj.rollno = 10;

*// Printing name and rollno*

System.out.println(obj.name);

System.out.println(obj.rollno);

}

}

**Output:**

Ram

10

**3.** **Static Variables:**A variable that is declared as static is known as a static variable. It is also known as a class variable. These variables are created at the beginning of the program execution and destroyed automatically when the program execution ends. We can create only a single copy of a static variable. To access the static variables, we don’t need to create the object of the class. We can simply access the static variable as

class\_Name.variable\_Name;

**Example:**

class Student {

*// static variables*

public static int rollno;

public static String name = "Ram";

}

public class StudentDemo {

public static void main(String args[])

{

*// accessing static variable without creating object*

Student.rollno = 10;

System.out.println(Student.name + " 's rollno is :" + Student.rollno);

}

}

**Output:**

Ram’s rollno is 10