** PROJECT REPORT**

**JAVA PROGRAMMING FUNDAMENTALS**

**(EBDS22ET2)**

**2024-2025(EVEN SEMESTER**)

**DEPARTMENT OF B.Tech DS & AI (E&T)**

**COURSE : B-TECH CSE-DS (AI)**

**YEAR/SEM/SEC : I/II/BB1**

**PROJECT TITLE : JAVA BASICS**



**BONAFIDE CERTIFICATE  
  
 JAVA PROGRAMMING FUNDAMENTALS**

**. DEPARTMENT OF B.Tech DS & AI (E&T)**

Certified that this project report “JAVA BASICS” is the confirmed work of  
***Y.TARUN KUMAR REDDY, S. LAKSHMI KANTH, S. SRI PATHIRAO*** -I-year-sem-2.B-Tech CSE-DS (AI)  
in JAVA PROGRAMMING FUNDAMENTALS (EBDS22ET2), carried out under the supervision of the faculty.

Signature of Lab-in-Charge Signature of Head of Dept

Submitted for the Practical Examination held on \_\_\_\_\_\_\_\_\_\_\_\_

Internal Examiner External Examiner

# 

**BASICS** **OF** **JAVA** **PROGRAMMING**

1. **ABSTRACT**

This document provides a foundational understanding of Java programming, a versatile, object-oriented, and platform-independent language widely used for enterprise-level applications, mobile development, web development, and more. It delves into the core concepts of Java, starting from its inception and key characteristics, leading into practical application through a simple programming example. The aim is to equip beginners with the necessary knowledge to write, compile, and execute basic Java programs, fostering a strong base for further exploration in Java development. The document covers the essential steps from defining an algorithm to observing the program’s output, thereby offering a comprehensive introduction to the fundamental aspects of Java programming.

The primary focus Is on clarity and simplicity, ensuring that complex ideas are broken down into digestible components. We explore Java’s “Write Once, Run Anywhere” (WORA) philosophy, its robust nature, security features, and high performance, all of which contribute to its enduring popularity. Through a step-by-step approach, readers will gain insight into how Java code is structured, compiled into bytecode, and executed by the Java Virtual Machine (JVM). This holistic overview is designed to serve as an invaluable resource for anyone embarking on their Java programming journey, laying a solid groundwork for more advanced topics and real-world application development.

1. **INTRODUCTION** **TO** **JAVA**

Java is a high-level, class-based, object-oriented programming language designed to have as few implementation dependencies as possible. It was originally developed by James Gosling at Sun Microsystems (now acquired by Oracle Corporation) and released in 1995 as a core component of Sun’s Java platform. The language was conceived with the principle of “Write Once, Run Anywhere” (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.

**KEY** **FEATURES** **OF** **JAVA**:

* Platform Independence: Achieved through the Java Virtual Machine (JVM). Java source code is compiled into bytecode, which is then executed by the JVM on any compatible system. This makes Java highly portable.
* Object-Oriented: Java is a pure object-oriented language, meaning everything in Java is treated as an object (with the exception of primitive data types). It supports core OOP principles like encapsulation, inheritance, polymorphism, and abstraction.
* Simple: Java was designed to be easy to learn for programmers already familiar with C++. It removed many complex and rarely used features of C++ like explicit pointers and operator overloading.
* Secure: Java’s security features include memory management without explicit pointers, bytecode verification, and a security manager, which helps prevent malicious code from accessing system resources.
* Robust: Java has strong memory management, automatic garbage collection, and exception handling, which contribute to its robustness by reducing the chances of programming errors and crashes.
* Multithreaded: Java supports multithreading, allowing a program to perform multiple tasks concurrently within a single program. This is crucial for developing interactive and responsive applications.
* High Performance: While bytecode is interpreted, the use of Just-In-Time (JIT) compilers within the JVM optimizes performance by compiling frequently executed bytecode into native machine code at runtime.
* Distributed: Java is designed for distributed environments. Its networking capabilities are an integral part of the language, making it suitable for developing applications that communicate across networks.

APPLICATIONS OF JAVA:

Java’s versatility has led to its adoption in a wide array of domains:

* Android Apps: Java is the official language for Android app development.
* Web Applications: Extensive use of Java frameworks like Spring, Struts, and Hibernate for robust and scalable web applications.
* Enterprise Applications: Java Enterprise Edition (Java EE) provides a platform for building large-scale, distributed, and multi-tier network applications.
* Big Data: Technologies like Hadoop and Apache Spark are written in Java and Scala (which runs on JVM).
* Scientific Applications: Used for scientific computing, including natural language processing.
* Financial Services: Widely used in the banking and financial sector for server-side applications.
* Embedded Systems: Java ME (Micro Edition) is used for developing applications for small devices.

1. AIM

The primary aim of this document is to provide a clear, concise, and comprehensive introduction to the fundamental concepts of Java programming for beginners. By the end of this document, readers should be able to:

1. Understand the core principles and characteristics that make Java a powerful and popular programming language.
2. Grasp the concept of Java’s platform independence, including the roles of the Java compiler and the Java Virtual Machine (JVM).
3. Identify and comprehend the basic structure of a simple Java program.
4. Learn the steps involved in writing, compiling, and executing a basic Java application.
5. Interpret the output of a simple Java program and understand its implications.
6. Develop a foundational understanding of programming logic through an elementary algorithm and its corresponding Java implementation.

This document serves as a stepping stone for aspiring Java developers, offering practical insights and a structured approach to learning. It aims to demystify the initial hurdles of programming by presenting a hands-on example, thereby building confidence and encouraging further exploration into more advanced Java topics and application development. The goal is to provide a solid theoretical and practical base upon which more complex programming constructs and paradigms can be built.

1. ALGORITHM

To illustrate the basic process of programming in Java, we will consider a very simple problem: displaying a classic “Hello, World!” message on the console. This example is fundamental in almost all programming languages and serves as an excellent starting point to understand the execution flow.

PROBLEM STATEMENT:

Create a program that outputs the string “Hello, World!” to the standard output device (typically the console).

STEPS FOR THE ALGORITHM:

1. Start: Begin the program execution.
2. Define Program Structure: In Java, all code resides within classes.

Therefore, we need to define a class. Let’s name it HelloWorld .

1. Define Main Method: The Java Virtual Machine (JVM) looks for a special method called main to start the execution of any Java application. This method must have a specific signature: public static void main(String[] args) .
2. Print Output: Inside the main method, use the standard output stream to print the desired message. Java provides System.out.println() for this purpose. The string “Hello, World!” will be passed as an argument to this method.
3. End: Terminate the program execution after the message has been displayed.

This algorithm demonstrates the sequential execution of instructions common in procedural programming, encapsulated within an object-oriented structure (a class and a method) that is characteristic of Java. It highlights the basic input/output operations and the necessary boilerplate code for a runnable Java application.

CODE

class Main {

public static void main(String[] args) {

System.out.println("Hello world!");

}

}

**CODE** **EXPLANATION**:

* Public class HelloWorld { … } :

◦ public : This is an access modifier, meaning the class HelloWorld is accessible from any other class.

◦ class : This keyword is used to declare a class. In Java, all code resides inside classes.

◦ HelloWorld : This is the name of our class. By convention, Java class names start with an uppercase letter. The file containing this code must be named HelloWorld.java .

* Public static void main(String[] args) { … } :

◦ public : An access modifier, making the main method accessible from outside the class, allowing the JVM to invoke it.

◦ static : This keyword means that the main method belongs to the class itself, not to any specific object of the class. This allows the JVM to call main without creating an object of the HelloWorld class.

◦ void : This signifies that the main method does not return any value.

◦ main : This is the special method name that the JVM looks for as the entry point of the program.

◦ (String[] args) : This defines the parameters that the main method accepts. Args is an array of String objects, which can be used to receive command-line arguments passed to the program.

• System.out.println(“Hello, World!”); :

◦ System : A final class from the java.lang package (which is implicitly imported) that provides access to system resources.

◦ out : A static member of the System class, which is an instance of PrintStream . It represents the standard output stream (usually the console).

◦ println() : A method of the PrintStream class (the type of

System.out ) that prints the given argument to the console and then moves the cursor to the next line.

◦ “Hello, World!” : This is the string literal that will be printed to the console.

**OUTPUT**

To see the output of the Java code, you would first need to save the code in a file named HelloWorld.java . Then, compile it using the Java compiler ( javac ) and run it using the Java Virtual Machine ( java ).

COMPILATION COMMAND:

Javac HelloWorld.java

This command compiles the HelloWorld.java file into bytecode, creating a HelloWorld.class file in the same directory.

**EXECUTION** **COMMAND**:

This command instructs the JVM to load and execute the HelloWorld.class file.

**EXPECTED** **OUTPUT**:

When you run the java HelloWorld command, the program will execute the System.out.println(“Hello, World!”); statement, and the following text will be displayed on your console:

***Java code example demonstrating local, instance, and static variables, along with an explanation:***

CODE:

public class MinVariableDemo {

    static String universe = "Expanding"; // Static variable (1)

    String planet; // Instance variable (2)

    public MinVariableDemo(String p) { // Constructor

        this.planet = p; // Initialize instance variable

    }

    public void showInfo() { // Method

        String message = "Hello from "; // Local variable (3)

        System.out.println(message + this.planet + " in the " + MinVariableDemo.universe + " universe.");

    }

    public static void main(String[] args) { // Main method

        MinVariableDemo earth = new MinVariableDemo("Earth"); // Create instance

        earth.showInfo(); // Call instance method

        MinVariableDemo.universe = "Vast"; // Modify static variable

        earth.showInfo(); // Show updated static variable

    }

}.

***Output***:-

Hello from Earth in the Expanding universe.

Hello from Earth in the Vast universe.

**Explanation:**

**Local Variable (localVariable):**

* + **Scope:** Declared inside the demonstrateVariables() method. Its scope is limited to that method.
  + **Lifetime:** Created when demonstrateVariables() is called and destroyed when the method finishes execution.
  + **Memory:** Stored on the method's stack frame.
  + **Access:** Only accessible within the method where it's declared.

**Instance Variable (instanceVariable):**

* + **Scope:** Declared directly inside the VariableTypes class, but outside any method.
  + **Lifetime:** Created when an object of VariableTypes is instantiated (e.g., new VariableTypes()) and destroyed when the object is garbage collected.
  + **Memory:** Stored in the heap memory as part of the object.
  + **Access:** Each object of the class gets its own copy of the instance variables. Accessed using an object reference (e.g., obj.instanceVariable).

**Static Variable (staticVariable):**

* + **Scope:** Declared directly inside the VariableTypes class and marked with the static keyword.
  + **Lifetime:** Created when the class is loaded into memory (usually when the program starts or the class is first referenced) and exists until the program ends.
  + **Memory:** Stored in a special area of memory called the "method area" or "class area," separate from the heap and stack.
  + **Access:** There's only one copy of a static variable for the entire class, shared by all objects of that class. Accessed using the class name (e.g., VariableTypes.staticVariable) or an object reference (though class name is preferred for clarity).

RESULT

The successful execution of the “Hello, World!” program demonstrates the fundamental lifecycle of a Java application: from source code to compilation into bytecode, and finally to execution by the Java Virtual Machine. The output “Hello, World!” confirms that the program was correctly structured, compiled without errors, and executed as intended, successfully printing the specified message to the console.

This simple outcome, while seemingly trivial, encapsulates several crucial concepts introduced earlier:

* Proof of Platform Independence: Once compiled into .class bytecode, this same HelloWorld.class file can be moved to any operating system (Windows, macOS, Linux, etc.) that has a compatible JVM installed and will produce the identical output without any modification or recompilation. This validates Java’s “Write Once, Run Anywhere” principle.
* Understanding Program Entry Point: The successful execution confirms the critical role of the main method as the primary entry point for Java applications, where the JVM begins its execution flow.
* Basic I/O Operation: The result illustrates the most basic form of output operation in Java using System.out.println() , a fundamental component for communicating information from a program to the user.
* Foundation for Object-Oriented Programming: Even this simple program is structured within a class, highlighting Java’s object-oriented nature where all code and data are encapsulated within objects and classes. This sets the stage for understanding more complex objectoriented designs.
* Confirmation of Development Environment Setup: Achieving this output also implicitly confirms that the Java Development Kit (JDK), including the Java compiler ( javac ) and the Java Virtual Machine ( java ), is correctly installed and configured on the system, which is a prerequisite for any Java development.

In conclusion, the “Hello, World!” program serves as more than just a first step; it is a complete, albeit minimal, demonstration of a functional Java application. Its successful execution provides tangible proof of understanding the foundational elements of Java programming, paving the way for learning more complex data structures, control flows, and object-oriented paradigms required for building sophisticated applications. This initial success is crucial for building confidence and serves as a springboard for deeper exploration into the vast capabilities of the Java language.