Experiment No. 1

Study of Java basics and Implementation of programs to demonstrate control flow and arrays.

Instructions:

This manual consists of three parts: **A) Theory and Concepts, B) Problems for Implementation, and C) Write-up Questions.**

1. Students must understand the **theory and concepts** provided before implementing the problem statement(s) for **Experiment 1**.
2. They should **practice the given code snippets** within the theory section.
3. Later, they need to **implement the problems provided**.
4. **Write-up:** Students are required to **write answers** to the questions on journal pages, **maintain a file**, and get it checked regularly. The file should include index, write-up, and implementation code with results.
5. Referencing: Include proper sources or references for the content used.
6. Use of Generative AI: Clearly mention if you have used any AI tools (e.g., ChatGPT, Copilot, Bard) to generate text, explanations, or code. Cite the AI-generated content appropriately in the write-up.

Part A. Theory and Concepts:

Features of Java language –

1. **Simple** – Java is very easy to learn, and its syntax is simple, clean and easy to understand.
2. **Object-oriented** – Java is an object-oriented programming language. Everything in Java is an object.
3. **Robust** – It uses strong memory management.
    - There is a lack of pointers that avoids security problems.
    - Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
    - There are exception handling and the type checking mechanism in Java.
4. **Multithreading** – Java supports multithreaded programming which allows you to write programs that do many things simultaneously.

5. **Architecture-Neutral** – Java program execution does not depend on OS upgrades, processor upgrades and changes in core system resources.
6. **Interpreted and high-performance** – Java enables the creation of cross- platform programs by compiling into an intermediate representation called Java bytecode. This code can be interpreted on any system that provides a Java Virtual Machine. Java bytecode was carefully designed so that it would be easy to translate directly into native machine code for very high performance by using Just-in-time compiler.
7. **Distributed** – Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.
8. **Dynamic** – Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand. Java programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at run-time.

Data-types –

Primitive Data Types - Primitive types are the most basic data types built into Java.

Data Type	Size	Min Value	Max Value	Example
byte	1 byte	-128	127	byte b = 100;
short	2 bytes	-32,768	32,767	short s = 1000;
int	4 bytes	-2,147,483,648	2,147,483,647	int num = 50000;
long	8 bytes	-9,223,372,036,854,775,808	9,223,372,036,854,775,807	long bigNum = 100000L;
float	4 bytes	±1.4E-45	±3.4E+38	float pi = 3.14f;
double	8 bytes	±4.9E-324	±1.8E+308	double precise = 99.99;
char	2 bytes	\u0000 (0)	\uffff (65,535)	char grade = 'A';
boolean	1 bit	false	true	boolean isJavaFun = true;

Non-Primitive Data Types - These are more complex data types that store references to objects.

Data Type	Description	Example
String	A sequence of characters	String name = "Java";
Array	A collection of elements of the same type	Int[ ] numbers = {1, 2, 3};
Class	A blueprint for objects	class Student { }
Interface	A contract for implementing classes	interface Animal { }

Control Statements in Java

Java provides three types of control flow statements.

- **Decision Making statements**
  - if statements
  - switch statement
- **Loop statements**
  - do while loop
  - while loop
  - for loop
  - for-each loop
- **Jump statements**
  - break statement: It used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement. However, it breaks only the inner loop in the case of the nested loop.
  - continue statement: it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

Structure of the Java program–

a) Documentation section (suggested)
b) package statement (optional)
c) import statement (optional)
d) interface statement (optional)
e) class definition (optional)
f) main method class(essential)
{……}

```
class FirstProgram
{   public static void main(String args[]){
    System.out.println(“My First Java Program”);
}
}
```

- Name of the file must be the same as the name of the class.
- The file extensionis.*java.*
- Java is *case-sensitive*.
- To compile java program use – ***javac filename.java***
- To execute java program use – ***java filename***
- ***main()*** must be declared as ***public*** since it must be called by the code outside of its class when the program is started.
- ***static*** keyword allows main() to be called without having to instantiate a particular instance of the class.

Array– Array is a group of same type variables referred by common name. Array of any type can be created and may have one or two dimensions.

One dimensional array–

Declaration –

type var-name[ ];

Memory allocation for array–

var-name=new type[size];

Two dimensional array–

type var-name[ ][ ]=new type[rows][columns];

Part B. Problems for Implementation:

Aim: Implement programs to demonstrate control flow and arrays using Java.

1. Implement a Java program to find the **factorial** of a given number.
2. Implement a Java program to check whether a given number is **prime** or not. (Take the number as a command-line argument.)
3. Implement a Java program to **sort** a given list of 10 numbers in ascending order.
4. Implement a Java program to **merge two sorted arrays**.
5. Implement a Java program to perform 2×2 **matrix** multiplication, addition, and transpose (using a switch case).

Part C. Write-up Questions:

1. Describe OOP concepts with real-world examples.
2. Explain the features of the Java programming language.
3. Describe the JVM and JIT compiler.
4. Write the structure of a Java program and provide an example of a simple Java program.
5. Explain arrays with an example. Also, explain jagged arrays.

Conclusion:

Students should be able to understand the Java program structure and write simple Java programs.