***Introduction to Java and Fundamentals of Java Programming

• Introduction to Java: History, Features, and Architecture (JVM, JDK, JRE), The Java Buzzwords
• Java's Magic: The Bytecode, Object-Oriented Programming Properties
• Java Program Structure and Syntax, The Java Keywords
• Identifiers in Java, The Java Class Libraries
• A First Simple Program, Handling Syntax Errors

***Introducing Data Types and Operators:

• Why Data Types Are Important, Java's Primitive Types
• Literals (Hexadecimal, Octal, and Binary Literals, Character Escape Sequences, String Literals)
• Variables, Types of variables in JAVA, Constants, The Scope and Lifetime of Variables
• Operators (Arithmetic Operators, Relational and Logical Operators, Short Circuit Logical Operators, The Assignment Operator, Shorthand Assignments, Ternary)
• Type Conversion and Type Casting • Operator Precedence, Expressions
• A Second Simple Program, Create Blocks of Code, Semicolons and Positioning, Indentation Practices

***Program Control Flow Statements:

• Conditional/Branching Statements, Looping: (Entry control loop, Exit Control loop), Jump Statements: break, continue, return
• Input/Output: Using Scanner

1. Introduction to Java

History

Developed by James Gosling at Sun Microsystems in the early 1990s.

Initially named "Oak," then renamed "Java" in 1995.

Designed for interactive television, but evolved for the internet.

Acquired by Oracle Corporation in 2010.

Features

Simple, Object-Oriented, Platform-Independent. Secure, Robust,

Multithreaded.

High Performance, Distributed, Dynamic. Interpreted (partially) and Compiled.

Architecture

JVM (Java Virtual Machine): Abstract machine that enables a computer to run Java programs.

JDK (Java Development Kit): Software development environment for writing Java applications.

JRE (Java Runtime Environment): Runtime environment to execute Java applications.

The Java Buzzwords

Simple

Secure

Portable

Object-Oriented

Robust

Multithreaded

Architecture

Neutral

Interpreted

High Performance

Distributed

Dynamic

Java's journey from a project for interactive television to a dominant language for enterprise and web applications highlights its adaptability and robust design. Its platform independence, achieved through the JVM, makes it a powerful tool for developers targeting diverse environments.

2. Java's Magic: Bytecode and OOP

Object-Oriented Programming Properties

- **Encapsulation:** Bundling data and methods that operate on the data within a single unit (class).

  Achieved using access modifiers.
- **Inheritance:** A mechanism where one class acquires the properties and behaviors of another class.

  Promotes code reusability.
- **Polymorphism:** The ability of an object to take on many forms. Achieved through method overloading and method overriding.
- **Abstraction:** Hiding the complex implementation details and showing only the essential features of the object. Achieved using abstract classes and interfaces.

The Bytecode

- **Intermediate Representation:** Java source code (.java files) is compiled into an intermediate format called bytecode (.class files).
- **Platform Independence:** This bytecode is not machine-specific, making Java "write once, run anywhere" (WORA).
- **JVM Interpretation:** The JVM interprets this bytecode into machine-specific instructions at runtime.
- **Security:** The JVM acts as a sandbox, enforcing security policies by verifying bytecode before execution.

Java's bytecode is the cornerstone of its platform independence, allowing applications to run seamlessly across various devices. Its strong adherence to Object-Oriented Programming (OOP) principles promotes modular, reusable, and maintainable code, which is crucial for large-scale software development.

3. Java Program Basics

Structure and Syntax

```
class MyClass {
  public static void main(String[] args) {
    System.out.println("Hello, Java!");
  }
}
```

- **Classes:** All Java code resides within classes.
- **Main Method:** The entry point for execution (public static void main(String[] args)).
- **Statements:** End with a semicolon (;).
- **Blocks:** Code enclosed in curly braces ({}).

The Java Keywords

Reserved words that have a predefined meaning and purpose in Java. Examples: public, class, static, void, int, if, else, for, while, new.

Identifiers in Java

Names used for classes, methods, variables, etc. Must start with a letter, underscore (_), or dollar sign ($). Cannot be keywords. Case-sensitive.

The Java Class Libraries

A collection of pre-written classes and interfaces that provide common functionalities. Organized into packages (e.g., java.lang, java.util, java.io).

A First Simple Program

```
// MyFirstProgram.java class MyFirstProgram {
  public static void main(String[] args) {
    System.out.println("My first Java program!");
  }
}
```

Handling Syntax Errors

Compiler will report syntax errors (e.g., missing semicolons, undeclared variables). Errors must be fixed before the program can compile and run.

Understanding the basic structure, syntax, and reserved keywords is the first step in writing functional Java programs. The robust Java Class Libraries provide a wealth of pre-built functionalities, significantly accelerating development.

4. Data Types and Operators

Why Data Types Are Important

Define the type of data a variable can hold, determining the memory allocated and the operations that can be performed.

Java9s Primitive Types

Integers: byte, short, int, long.

Floating-Point: float, double.

Character: char (stores single Unicode characters).

Boolean: boolean (stores true or false).

Literals

Integer: 10, 0xFF (hex), 0b101 (binary).

Floating-Point: 3.14, 1.23f.

Character: 'A', '\n' (escape sequence).

String: "Hello World".

Variables, Types, Constants

Variables: Local, Instance, Static.

Constants: Declared using the final keyword (e.g., final int MAX_VALUE = 100;).

Scope and Lifetime of Variables

Scope: The region of a program where a variable is accessible.

Lifetime: The period during which a variable exists in memory

Java's strong typing ensures data integrity and helps prevent common programming errors. Understanding how to declare and use variables, along with different literal representations, is fundamental for effective programming.

4. Data Types and Operators (Continued)

Operators

Arithmetic: +, –, *, /, %.

Relational: ==, !=, >, <, >=, <=.

Logical: && (AND), || (OR), !(NOT).

Short Circuit: Optimized logical operators (&&, ||).

Assignment: =.

Shorthand Assignment: +=, -=, *=, etc.

Ternary: condition ? expr1 : expr2.

Type Conversion and Type Casting

Implicit (Widening): Automatic conversion from a smaller to a larger data type (e.g., int to long).

Explicit (Narrowing/Casting): Manual conversion from a larger to a smaller data type, potentially leading to data loss (e.g., (int) 3.14).

Operator Precedence, Expressions

Precedence: Determines the order in which operators are evaluated (e.g., * and / before + and –).

Expressions: Combinations of variables, literals, operators, and method calls that evaluate to a single value.

A Second Simple Program

```
class OperatorsExample {
  public static void main(String[]
    args) { int a = 10, b = 5;
    int sum = a + b; // Arithmetic
    boolean isEqual = (a == b); //
    Relational
    System.out.println("Sum: " + sum);
    System.out.println("Is Equal: " +
      isEqual);
  }
}
```

A solid grasp of Java's operators and their precedence is essential for writing correct and efficient logic. Understanding type conversion and casting allows for flexible data manipulation while avoiding runtime errors.

5. Program Control Flow Statements (Continued)

Jump Statements

break

Immediately terminates the innermost loop or switch statement and transfers control to the statement immediately following the terminated statement.

```
for (int i = 0; i < 10; i++) { if (i == 5) {
    break; // Loop terminates when i is 5
  }
  System.out.println(i);
}
```

continue

Skips the rest of the current iteration of a loop and proceeds to the next iteration.

```
for (int i = 0; i < 5; i++) { if (i == 2) {
   continue; // Skips printing 2
  }
  System.out.println(i);
}
```

return

Transfers control back to the caller of the method. Can optionally return a value.

```
public int add(int a, int b) { return a + b; // Returns
the sum
}
```

Mastering control flow is paramount for writing complex and responsive programs. These statements allow developers to guide the program's execution based on conditions and to manage iterative tasks efficiently.

6. Input/Output Operations with Scanner

Input/Output (I/O) operations are fundamental to programming, enabling programs to interact with the outside world. This includes receiving data from users (input) and displaying results or information (output). In Java, the Scanner class is a versatile tool for reading various types of input.

The Scanner Class

The Scanner class in Java's java.util package is primarily used for parsing primitive types and strings using regular expressions. It can read input from various sources, including the console (System.in), files, and strings.

To use the Scanner class, you must import it:

```
import java.util.Scanner;
```

Then, you create an instance of Scanner, typically linked to System.in for console input:

```
Scanner scanner =new Scanner(System.in);
```

Reading Different Types of Input

next()

Reads the next complete token (word) from the input. A token is typically delimited by whitespace.

```
String name = scanner.next(); // Reads "John" from "John Doe"
```

nextLine()

Reads the entire line until the next line separator is found. This is useful for reading sentences or phrases.

```
String sentence = scanner.nextLine(); // Reads "Hello World!"
```

nextInt()

Reads the next token as an int. If the next token cannot be parsed as an integer, it will throw an InputMismatchException.

```
int age =scanner.nextInt(); // Reads 25
```

nextDouble()

Reads the next token as a double. Similar to nextInt(), it throws an exception for invalid input.

```
double price =scanner.nextDouble(); // Reads 19.99
```

Other next methods

The Scanner class also provides methods like nextBoolean(), nextByte(), nextFloat(), nextLong(), and nextShort() for reading other primitive data types.

Common Pitfalls and Best Practices

nextLine() **after** nextInt()/nextDouble()**:** When you read an integer or double using nextInt() or nextDouble(), the newline character (produced when you press Enter) remains in the input buffer. If you then call nextLine(), it will immediately consume this leftover newline character, appearing to "skip" your input. To fix this, always call an extra scanner.nextLine() after nextInt() or nextDouble() to consume the remaining newline.

Closing the Scanner: Always close the Scanner object using scanner.close() when you are finished with it. This releases system resources.

Input Validation: Use hasNextInt(), hasNextDouble(), etc., to check if the next token is of the expected type before attempting to read it, to prevent InputMismatchException.

Complete Example Program

```
import java.util.InputMismatchException; import java.util.Scanner;

public class InputExample {
    public static void main(String[] args) {       Scanner scanner =new Scanner(System.in);
System.out.println("Welcome to the Interactive Program!");

// Reading an integer int age =0;
boolean validInput =false; while (!validInput) {
    System.out.print("Please enter your age: ");
    if (scanner.hasNextInt()) { age =scanner.nextInt(); validInput =true;
    } else {
        System.out.println("Invalid input. Please enter a number for age."); scanner.next(); // Consume the invalid input
    }
}
scanner.nextLine(); // Consume the leftover newline

// Reading a double double height =0.0; validInput =false; while (!validInput) {
    System.out.print("Please enter your height in meters (e.g., 1.75): "); if (scanner.hasNextDouble()) {
        height =scanner.nextDouble();
        validInput =true;
    } else {
        System.out.println("Invalid input. Please enter a decimal number for height."); scanner.next(); // Consume the invalid input
    }
}
scanner.nextLine(); // Consume the leftover newline

// Reading a single word (token) System.out.print("What is your favorite color? "); String color =scanner.next();
scanner.nextLine(); // Consume the leftover newline

// Reading an entire line
System.out.print("Tell us something about yourself: "); String aboutMe =scanner.nextLine();

System.out.println("\n-— Your Information —"); System.out.println("Age: " +age +" years"); System.out.println("Height: " +height +" meters"); System.out.println("Favorite
    Color: " +color); System.out.println("About You: " +aboutMe);

    scanner.close(); // Close the scanner to release resources
  }
}
```