Java Fundamentals, Data Types, Operators, and Control Statements

1.1 History and Philosophy of Java

Java was developed by **James Gosling** at Sun Microsystems in 1995. It is designed to be **portable, secure, and high-performance**. Java follows the **WORA (Write Once, Run Anywhere)** principle, making it platform-independent. Java is widely used for web applications, enterprise software, mobile applications, and more.

1.2 Object-Oriented Programming (OOP)

Java follows the OOP principles:

- **Class:** Class is a blueprint for creating an objects.
- **Object:** An Object is an instance of a class.
- **Encapsulation**: Wrapping data and methods into a single unit (class).
- **Inheritance**: Allowing a class to inherit methods and fields from another class.
- **Polymorphism**: Ability to use the same function name for different purposes (overloading & overriding).
- **Abstraction**: Hiding implementation details from the user.

1.3 Java Development Kit (JDK)

JDK contains:

- **JRE (Java Runtime Environment)**: Required to run Java applications.
- **JVM (Java Virtual Machine)**: Converts bytecode into machine code.
- **Java Compiler (javac)**: Compiles Java source code into bytecode.

1.4 A First Simple Java Program

```
// A simple Java program to print "Hello, World!"
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Output:

```
Hello, World!
```

1.5 Packages in Java

In Java, a **package** is a mechanism for organizing Java classes into namespaces, providing a way to group related classes and interfaces together. Packages help avoid name conflicts and make the code easier to maintain. Packages help in organizing Java classes into namespaces.

There are two types of packages in Java:

1. **Built-in Packages**: These are predefined packages that come with the Java standard library. Some examples include:
    - java.util: Contains utility classes like ArrayList, HashMap, etc.
    - java.io: Contains classes for input and output, like File, BufferedReader, etc.
    - java.lang: Automatically imported in every Java program and contains essential classes like String, System, Math, etc.
    - java.net: Contains classes for networking, such as URL and Socket.
2. **User-defined Packages**: These are packages created by users to organize their own code. To create a package, you use the package keyword at the beginning of your Java file:

```
package com.example.myapp;
public class MyClass {
   // class code here
}
```

To access classes from other packages, you use the import keyword:

```
import com.example.myapp.MyClass;

public class Main {
   public static void main(String[] args) {
      MyClass obj = new MyClass();
   }
}
```

Simple Example of package

```
package mypackage;
// Declare a package
class MyClass {
   public static void main(String[] args) {
      System.out.println("Package Example");
   }
}
```

Output:

Package Example

1.6 Java’s Data Types

1.6.1 Integers

Stores whole numbers.

```
class IntegerExample {
    public static void main(String[] args) {
        int a = 10;
        System.out.println("Integer value: " + a);
    }
}
```

Output:

```
Integer value: 10
```

1.6.2 Characters

Stores a single character.

```
class CharExample {
    public static void main(String[] args) {
        char letter = 'A';
        System.out.println("Character: " + letter);
    }
}
```

Output:

```
Character: A
```

1.6.3 Floating Point Types

Stores decimal numbers.

```
class FloatExample {
    public static void main(String[] args) {
        double pi = 3.1415;
        System.out.println("Floating point value: " + pi);
    }
}
```

Output:

```
Floating point value: 3.1415
```

1.6.4 Strings

Used to store sequences of characters.

```
class StringExample {
   public static void main(String[] args) {
      String name = "Java";
      System.out.println("String: " + name);
   }
}
```

Output:

```
String: Java
```

1.6.5 Arrays

Used to store multiple values of the same type.

```
class ArrayExample {
   public static void main(String[] args) {
      int[] numbers = {1, 2, 3};
      System.out.println("First element of array: " + numbers[0]);
   }
}
```

Output:

```
First element of array: 1
```

1.6.6 Boolean Types

Stores true or false values.

```
class BooleanExample {
   public static void main(String[] args) {
      boolean isJavaFun = true;
      System.out.println("Boolean value: " + isJavaFun);
   }
}
```

Output:

```
Boolean value: true
```

1.7. Literals

Literals are constant values that are directly written into the code. These values represent data of different types like numbers, characters, and strings.

1.7.1. Hex, Octal and Binary Literals

Hexadecimal (Hex) Literals: Base 16 number system (digits 0-9 and letters A-F).

Syntax: 0x or 0X

Example: 0x1F (Hexadecimal 1F is decimal 31)

Octal Literals: Base 8 number system (digits 0-7).

Syntax: 0 (without x)

Example: 075 (Octal 75 is decimal 61)

Binary Literals: Base 2 number system (digits 0 and 1).

Syntax: 0b or 0B

Example: 0b101 (Binary 101 is decimal 5)

Program Example:

```
public class Main {
    public static void main(String[] args) {
        int hexValue = 0x1F;  // Hexadecimal literal
        System.out.println("Hexadecimal: " + hexValue);  // Output: 31

        int octalValue = 075; // Octal literal
        System.out.println("Octal: " + octalValue);  // Output: 61

        int binaryValue = 0b101; // Binary literal
        System.out.println("Binary: " + binaryValue); // Output: 5
    }
```

```
}
```

1.7.2. Character Escape Sequences

Escape sequences are special characters used in strings to represent characters that are difficult to type or display.

Examples include:

- Newline (\n): Moves to a new line.

- Tab (\t): Adds a tab space.

- Backslash (\\): Prints a backslash.

- Single Quote (\'): Prints a single quote.

- Double Quote (\"): Prints a double quote.

Code Example:

```
public class EscapeExample {
    public static void main(String[] args) {
        System.out.println("Hello\nWorld!");  // New line
        System.out.println("Tab\tTest");      // Tab space
        System.out.println("Backslash: \\"); // Backslash
        System.out.println("Quote: \"Hello\"");  // Double Quote
    }
}
```

1.7.3. String Literals

A String literal is a sequence of characters enclosed in double quotes.

Basic String: "Hello"

Multiline String (Java 13+): Enclosed in triple quotes.

Code Example:

```
public class StringLiteralExample {
    public static void main(String[] args) {
        String message = "Hello, World!";  // Basic String
        System.out.println(message);  // Output: Hello, World!

        String multilineMessage = """
                This is a multiline
                string literal.
                It preserves the format.
                """;
        System.out.println(multilineMessage);
    }
}
```

1.8. Variables and Constants

Variables are used to store values that can be changed during the program's execution.

Syntax: type variableName = value;

Example:

```
int age = 30;  // Variable of type int

String name = "Rajesh";  // Variable of type String
```

Constants are values that cannot be changed once they are initialized. Use the "final" keyword to declare constants.

Syntax: final type constantName = value;

Example:

```
final double PI = 3.14159;  // Constant value
```

Program Example:

```
public class Main {
    public static void main(String[] args) {
        int hexValue = 0x1F; // Hexadecimal
        System.out.println("Hexadecimal: " + hexValue); // Output: 31
        int octalValue = 075; // Octal
        System.out.println("Octal: " + octalValue); // Output: 61

        int binaryValue = 0b1011; // Binary
        System.out.println("Binary: " + binaryValue); // Output: 11
        String escapeString = "Hello\nWorld!\tThis is a backslash: \\ and a quote: \"";
        System.out.println("Escape Sequences Output:\n" + escapeString);
        String stringLiteral = "This is a simple string literal.";
        System.out.println("String Literal: " + stringLiteral);

        String multilineString = """
                This is a multiline
                string literal.
                It preserves the format.
                """;
        System.out.println("Multiline String Literal:\n" + multilineString);

        final double PI = 3.14159;  // Constant value
        System.out.println("Value of PI: " + PI);  // Output: 3.14159

        int age = 25;  // Variable
        String name = "Rajesh";  // Variable
        System.out.println(name + " is " + age + " years old.");  // Output: Rajesh is 25 years old.
    }
}
```

1.11 Control Statements

In Java, control statements are used to control the flow of execution in a program. The main control statements in Java are divided into three categories:

1. Selection Statements (Conditional Statements)

These statements allow you to choose a path of execution based on a condition.

- **if statement**: Executes a block of code if the specified condition is true.

```
if (condition) {
 // code to be executed if condition is true
}
```

- **if-else statement**: Executes one block of code if the condition is true, and another block if it is false.

```
if (condition) {
    // code to be executed if condition is true
} else {
    // code to be executed if condition is false
}
```

- **if-else if-else statement**: Tests multiple conditions sequentially.

```
if (condition1) {
    // code to be executed if condition1 is true
} else if (condition2) {
    // code to be executed if condition2 is true
} else {
    // code to be executed if none of the above conditions are true
}
```

- **switch statement**: Tests an expression against multiple values. It is often used when there are several possible conditions

```
switch (expression) {
    case value1:
        // code to be executed if expression == value1
        break;
    case value2:
        // code to be executed if expression == value2
        break;
    default:
        // code to be executed if expression doesn't match any case
}
```

2. Looping Statements

These statements allow you to repeat a block of code multiple times.

- **for loop**: Repeats a block of code a specific number of times.

  ```
  for (initialization; condition; update) {
     // code to be executed
  }
  ```

  Example:

  ```
  for (int i = 0; i < 5; i++) {
     System.out.println(i);
  }
  ```

- **while loop**: Repeats a block of code as long as the condition is true.

  ```
  while (condition) {
     // code to be executed
  }
  ```

  Example:

  ```
  int i = 0;
  while (i < 5) {
     System.out.println(i);
     i++;
  }
  ```

- **do-while loop**: Similar to the while loop, but it guarantees that the code block will execute at least once, as the condition is checked after the execution.

  ```
  do {
     // code to be executed
  } while (condition);
  ```

  Example:

  ```
  int i = 0;
  do {
     System.out.println(i);
     i++;
  } while (i < 5);
  ```

3. Jump Statements

These statements control the flow of execution by breaking, continuing, or jumping to other sections in the code.

1.11.4 Continue Statement

```
class ContinueExample {
   public static void main(String[] args) {
      for(int i = 0; i < 5; i++) {
         if(i == 2) continue;
         System.out.println("Iteration: " + i);
      }
   }
}
```

Output:

```
Iteration: 0
Iteration: 1
Iteration: 3
Iteration: 4
```

1.11.5 Break Statement

```
class BreakExample {
   public static void main(String[] args) {
      for(int i = 0; i < 5; i++) {
         if(i == 2) break;
         System.out.println("Iteration: " + i);
      }
   }
}
```

Output:

```
Iteration: 0
Iteration: 1
```