**Java Intro**

What is Java?

- Definition:

- High-level, object-oriented programming language developed by Sun Microsystems (now Oracle).

- Emphasizes "Write Once, Run Anywhere" (WORA).

- Key Features:

- Platform independence through Java Virtual Machine (JVM).

- Object-oriented paradigm.

- Automatic memory management (garbage collection).

- Rich standard library (Java API).

- Multithreading support.

- Security features.

- Large and active developer community.

---

Writing Your First Java Program:

```java

public class HelloWorld {

public static void main(String[] args) {

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

}

}

```

- Explanation:

- `public class HelloWorld`: Declares a class named `HelloWorld`.

- `public static void main(String[] args)`: Declares the main method, the entry point of the program.

- `System.out.println("Hello, World!");`: Prints "Hello, World!" to the console.

---

Typical Structure of a Java Program:

1. Package Declaration (Optional):

```java

package com.example; // Optional: Organizes classes into packages

```

2. Import Statements (Optional):

```java

import java.util.\*; // Optional: Imports classes from the Java Standard Library

```

3. Class Declaration:

```java

public class MyClass {

// Class body

}

```

4. Main Method:

```java

public static void main(String[] args) {

// Main method body

}

```

5. Class Body:

```java

public class MyClass {

// Fields (variables)

int myVariable;

// Constructor

public MyClass() {

// Constructor body

}

// Methods

public void myMethod() {

// Method body

}

}

```

6. Comments (Optional):

```java

// Single-line comment

/\*

Multi-line comment

Can span multiple lines

\*/

```

**Fundamentals**

Variable

Definition:

- A *variable* is a named storage location that holds data.

- Its value can be changed during program execution.

Key Concepts:

- Declaration: Specify variable name and data type.

- Initialization: Assign an initial value.

- Data Types: Define the kind of data the variable can hold.

- Naming Conventions: Follow specific rules.

- Scope: Determines where a variable can be accessed.

---

Data Types in Java

Definition:

- Data types define the type of data a variable can hold.

- Java has primitive (int, char) and reference (objects) types.

Primitive Data Types:

- Numeric (byte, short, int, long, float, double).

- Character (char).

- Boolean (boolean).

Reference Data Types:

- Class Types (objects).

- Array Types (collections).

- Interface Types (method contracts).

- Enumerations (fixed set of constants).

---

Operators in Java

Definition:

- Operators in Java are symbols representing computations or operations.

- Types include arithmetic, relational, logical, bitwise, and assignment operators.

Types of Operators:

- Arithmetic: `+`, `-`, `\*`, `/`, `%`.

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

- Logical: `&&`, `||`, `!`.

- Bitwise: `&`, `|`, `^`, `~`, `<<`, `>>`.

- Assignment: `=`, `+=`, `-=`, `\*=`, `/=`, `%=`.

---

Java Classes and Objects

Classes:

- A class is a blueprint or template for objects.

- It defines attributes (fields) and methods.

Objects:

- An object is an instance of a class.

- Represents real-world entities, encapsulating data and behaviour.

Key Concepts:

- Fields: Object attributes.

- Constructor: Initializes object.

- Methods: Define object behaviour.

- Encapsulation: Bundling data and methods.

- Instantiation: Creating an object from a class.

---

**OOP**

Object-Oriented Programming (OOP)

Definition:

- A programming paradigm that revolves around the concept of "objects."

- Objects combine data (attributes) and behavior (methods).

- Provides a structured way to design and organize code.

Key Concepts:

1. Objects:

- Instances of classes representing real-world entities.

- Combine data and behavior.

2. Classes:

- Blueprints or templates for creating objects.

- Define attributes and methods.

3. Encapsulation:

- Bundling data and methods within a class.

- Controls access to internal details.

4. Inheritance:

- New class inherits properties and behaviors from an existing class.

- Promotes code reuse and establishes hierarchy.

5. Polymorphism:

- Objects of different types can be treated as objects of a common type.

- Enhances flexibility and extensibility.

---

Features of OOP – Core Features:

1. Encapsulation:

-Definition: Bundling data and methods within a class.

- Purpose: Controls access, enhances modularity, protects object integrity.

2. Inheritance:

- Definition: New class inherits properties and behaviors.

- Purpose: Promotes code reuse, establishes hierarchy, enables specialization.

3. Polymorphism:

- Definition: Objects of different types as objects of a common type.

- Purpose: Enhances flexibility, enables generic coding, supports method overriding.

4. Abstraction:

- Definition: Simplifying complex systems by modeling essential features.

- Purpose: Focuses on essentials, hides unnecessary details, provides clarity.

---

Features of OOP – Other Features:

1. Association:

- Definition: Represents relationships between classes.

- Purpose: Defines collaboration and interaction between objects.

2. Composition:

- Definition: Class contains an object of another class.

- Purpose: Creates complex objects by combining simpler ones.

3. Interfaces:

- Definition: Defines a contract for classes to implement.

- Purpose: Supports multiple inheritance, ensures method implementation.

4. Packages and Namespaces:

- Definition: Organizes classes, avoids naming conflicts.

- Purpose: Improves code organization, prevents naming collisions.

5. Object-Oriented Analysis and Design (OOAD):

- Definition: Systematic approach to software development.

- Purpose: Ensures requirements are met, provides a structured methodology.

---

OOP is widely used for creating modular, scalable, and maintainable software. Its core and additional features contribute to effective code organization and design.

**Decision Control**

Decision Control in Programming

Introduction:

- Decision control structures manage the flow of a program based on conditions.

- Enable execution of specific code blocks based on the evaluation of conditions.

If Statement:

```java

if (condition) {

// Code to execute if the condition is true

}

```

Example:

```java

int num = 10;

if (num > 0) {

System.out.println("The number is positive.");

}

```

If-Else Statement:

```java

if (condition) {

// Code to execute if the condition is true

} else {

// Code to execute if the condition is false

}

```

Example:

```java

int num = -5;

if (num > 0) {

System.out.println("The number is positive.");

} else {

System.out.println("The number is non-positive.");

}

```

Nested If Statement:

```java

if (condition1) {

// Code to execute if condition1 is true

if (condition2) {

// Code to execute if condition2 is true

}

}

```

Example:

```java

int x = 5, y = 10;

if (x > 0) {

if (y > 0) {

System.out.println("Both x and y are positive.");

}

}

```

Switch Statement:

```java

switch (expression) {

case value1:

// Code to execute if expression equals value1

break;

case value2:

// Code to execute if expression equals value2

break;

// ... additional cases ...

default:

// Code to execute if none of the cases match

}

```

Example:

```java

int dayOfWeek = 3;

switch (dayOfWeek) {

case 1:

System.out.println("Monday");

break;

case 2:

System.out.println("Tuesday");

break;

// ... other cases ...

default:

System.out.println("Invalid day");

}

```

Ternary Operator (`?`):

```java

variable = (condition) ? expressionIfTrue : expressionIfFalse;

```

Example:

```java

int x = 10;

String result = (x > 0) ? "Positive" : "Non-positive";

System.out.println(result);

```

Decision control structures provide flexibility in program execution based on conditions. `if`, `if-else`, `nested if`, `switch`, and the ternary operator are essential for decision-making in Java.

**Java Identifiers**

Rules for Java Identifiers:

1. Start Character:

- Must begin with a letter, currency character (\$), or underscore (\\_).

- Valid: `variable`, `\_count`, `$amount`.

- Invalid: `2days`, `@value`.

2. Subsequent Characters:

- Can include letters, digits, currency characters, or underscores.

- Valid: `totalAmount`, `max\_value\_1`, `class\_name2`.

- Invalid: `user@name`, `variable#count`.

3. No Reserved Keywords:

- Cannot use Java reserved words as identifiers.

- Examples: `int`, `class`, `if`, `while`.

4. Case-Sensitive:

- Java is case-sensitive (`variableName` and `variablename` are different).

5. No Spaces or Special Characters:

- Except for \$ and \\_.

- Invalid: `my variable`, `user-name`.

6. No Length Limitation:

- No strict limit, but keep identifiers meaningful and concise.

7. Cannot Begin with a Digit:

- Valid: `value1`, `variable2`.

- Invalid: `2days`, `123count`.

Conventions for Java Identifiers:

1. Camel Case for Variables and Methods:

- Start with lowercase and capitalize each subsequent word.

- Examples: `myVariable`, `calculateTotalAmount()`.

2. Pascal Case for Classes:

- Start with uppercase and capitalize each subsequent word.

- Examples: `CarModel`, `EmployeeDetails`.

3. UPPERCASE for Constants:

- Use uppercase letters with underscores.

- Examples: `MAX\_VALUE`, `PI\_VALUE`.

4. Descriptive and Meaningful Names:

- Choose names that clearly convey the purpose.

- Examples: `totalAmount`, `userName`, `averageValue`.

5. Avoid Single Letters for Variable Names:

- Except for loop counters, use descriptive names.

- Example: `i` for loop counters.

Examples:

```java

// Variables

int totalAmount;

String userName;

double averageValue;

// Classes

class CarModel {

// Class body

}

// Constants

final int MAX\_VALUE = 100;

final double PI\_VALUE = 3.14;

```

Following these rules and conventions ensures code readability, maintainability, and adherence to best practices in Java development.