**Unit-1 Notes**

**Object-Oriented Programming (OOP)**

* A programming paradigm that organizes software design around data (objects), rather than functions and logic.
* Focuses on creating objects that contain data (attributes) and methods (behaviors).
* Key benefits: code reusability, modularity, flexibility, and maintainability.

**OOP Principles**

* **Encapsulation:**
  + Bundling data (attributes) and methods (behaviors) within a single unit called a class.
  + Protecting data from external access (data hiding) by using access modifiers (public, private, protected).
  + Improves code organization, security, and maintainability.
* **Inheritance:**
  + Creating new classes (subclasses or derived classes) by inheriting properties and methods from existing classes (superclasses or base classes).
  + Promotes code reusability and hierarchical relationships between classes.
  + Types: single inheritance (one parent), multilevel inheritance (multiple levels), hierarchical inheritance (multiple subclasses from one parent), hybrid inheritance (combination of multiple and multilevel).
* **Polymorphism:**
  + Ability of objects to take on multiple forms.
  + Enables different objects to be treated as if they were of the same type.
  + Types:
    - Static polymorphism (compile-time polymorphism): Method overloading (multiple methods with the same name but different parameters).
    - Dynamic polymorphism (runtime polymorphism): Method overriding (redefining methods in subclasses).

**Java as an OOP & Internet-Enabled Language**

* Java is a pure object-oriented language, meaning everything in Java is an object except primitive data types.
* Provides robust support for OOP concepts through classes, objects, inheritance, polymorphism, and encapsulation.
* Designed for network and internet applications with features like sockets, URL handling, and security.
* Platform independence due to bytecode compilation.

**Importance of Java**

* Platform independence: Write once, run anywhere (WORA) principle.
* Robustness: Strong memory management and exception handling.
* Security: Built-in security features for developing secure applications.
* High performance: Just-In-Time (JIT) compilation for improved execution speed.
* Large and active community: Extensive support, libraries, and frameworks.
* Rich API: Provides a vast collection of pre-built classes and interfaces.

**Java Usage in Industry**

* Enterprise applications: Large-scale business software.
* Web applications: E-commerce, content management systems, etc.
* Android app development: Mobile applications for the Android platform.
* Embedded systems: Devices with limited resources (e.g., IoT).
* Big data technologies: Processing and analyzing large datasets.

**The Bytecode, Compiling, and Running of Simple Java Program**

* Java source code (.java) is compiled into bytecode (.class) by the Java compiler (javac).
* Bytecode is platform-independent and can be executed on any system with a Java Virtual Machine (JVM).
* JVM interprets the bytecode and translates it into machine code for the specific platform.

**JVM, JDK, JRE**

* **JVM (Java Virtual Machine):** The runtime environment that executes Java bytecode.
* **JDK (Java Development Kit):** Contains tools for developing Java applications (compiler, debugger, etc.).
* **JRE (Java Runtime Environment):** Includes the JVM and necessary libraries for running Java applications.

**Simple Java Program Structure**

Java

public class HelloWorld {

public static void main(String[] args) {

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

}

}

Use code

* public class HelloWorld: Defines a public class named HelloWorld.
* public static void main(String[] args): The entry point of the program.
* System.out.println("Hello, World!");: Prints "Hello, World!" to the console.