Chapter 1: INTRODUCTION TO JAVA

Java is a powerful programming language used for developing mobile and desktop applications, big data processing, embedded systems etc.

1. TYPES OF JAVA APPLICATIONS

There are mainly 4 types of applications that can be created using java programming

1. Stand alone application: They are also called desktop applications. These are traditional software that we must install on each machine. Example pf standalone applications are media players, antivirus etc.
2. Web app: It is an application that runs on the server site and creates a dynamic page. Currently, technologies like servlet, GSP, Spring, etc are used to create web applications in java.
3. Enterprise app: An application that is distributed in nature such as banking applications are called enterprise application. It has benefits such as high-level security, load balancing and clustering. In java, EJB (Enterprise Java Bean) is used in creating enterprise applications.
4. Mobile App: An application created for mobile devices is called a mobile application. Currently, android studio and JAVAMI are used to create mobile applications.
5. JAVA PLATFORMS OR EDITIONS

A java platform is a specific setting that where java programming language can be used.

1. JAVA SE(Java Standard Edition) :

Java SE is made up of APIs (Application Programming Interface) like java.lang, java.net, java.io , etc. That provides java’s fundamental programming features. It specifies every aspect of the java programming language from its fundamental types and objects, to its high level classes for security, graphical user interface development, database manipulation, networking etc.

1. JAVA EE (Java Enterprise Edition) :

Java EE is an enterprise platform primarily used for developing web and enterprise application. It was developed to expand the java SE with the addition of a collection of standard or specifications that describes frequently used features by commercial applications. The most used features of java EE are Java Server Page(JSP), Java Server Faces (JSF), Java Persistence API (JPA) and Enterprise Java Bean (EJB).

1. JAVA MICRO EDITION (JAVA ME):

It was developed to facilitate mobile and embedded device applications. The java ME offers an API and a small footprint virtual machine that helps in running java ME applications on compact devices.

1. JAVA FX (JAVA Special Effects):

Java FX is a platform that provides a modern hardware accelerated graphics and the media engine for building rich online and desktop applications with a simple and powerful scripting language. It was created in java to take the place of Swing as the default GUI library.

1. JAVA VIRTUAL MACHINES

Java is both a compiled and an interpreted language. Java source codes is transformed into simple binary instructions called byte codes

Java code

Byte code

code

Linux

code

Windows

code

Mac os

code

Java code

Byte code

code

Executable

code

javac

java

.java

.class

1. JAVA BASIS
2. A variable is a name of an area in memeory. In other words it is the name of a memory location and the stored value can be changed during execution.

**Declaration or syntax**: *type variable\_name;*

**Example:** *int phone;*

**Types of variables:** There are 3 types of variables in java;

1. Local variable: A variable declared in the body of a method is called a local variable. You can only use this variable in this method and other methods in the class don’t even know that the variable exists

NB: A local variable cannot be defined with the static keywords

**1;static**

The static keyword in Java is used to declare class-level variables, methods, or blocks that belong to the class rather than instances of the class.

* **Static Variables**: These variables are shared by all instances of the class. They are initialized only once when the class is loaded into memory.
* **Static Methods**: These methods can be called on the class itself without needing an instance. They can only access static variables or call other static methods.
* **Static Blocks**: A static block is used to initialize static variables. It runs only once when the class is loaded.

**Example**:

class Example {

static int count = 0; // static variable

static void increment() { // static method

count++;

}

}

**2. final**

The final keyword is used to define constants or prevent modification of variables, methods, and classes.

* **Final Variables**: Once assigned, their values cannot be changed.
* **Final Methods**: These methods cannot be overridden by subclasses.
* **Final Classes**: These classes cannot be subclassed.

**Example**:

final int MAX\_SPEED = 120; // final variable

class Car {

final void startEngine() { // final method

System.out.println("Engine started.");

}

}

final class SportsCar { // final class

void accelerate() {

System.out.println("Accelerating!");

}

}

**3. this**

The this keyword is used to refer to the current instance of the class. It is primarily used to differentiate between instance variables and parameters when they have the same name, or to call another constructor of the same class.

* **Referring to instance variables**: this is used when the instance variable and method parameter have the same name.
* **Calling another constructor**: this() can be used to call another constructor within the same class.

**Example**:

class Car {

String model;

Car(String model) {

this.model = model; // refers to the instance variable

}

void printModel() {

System.out.println("Model: " + this.model); // this is optional, but shows clarity

}

}

**4. synchronized**

The synchronized keyword is used to control access to a block of code or method by multiple threads to ensure thread safety.

* **Synchronized Method**: When a method is declared as synchronized, only one thread can access the method at a time for a given object.
* **Synchronized Block**: It allows more granular control by synchronizing only a block of code instead of an entire method.

**Example**:

class Counter {

private int count = 0;

synchronized void increment() { // synchronized method

count++;

}

void decrement() {

synchronized(this) { // synchronized block

count--;

}

}

}

**5. volatile**

The volatile keyword is used to indicate that a variable may be changed asynchronously by multiple threads. It ensures that reads and writes to the variable are directly from and to the main memory (instead of being cached), which provides visibility guarantees.

* It tells the JVM to always read the value from the main memory rather than from local thread caches.

**Example**:

class SharedResource {

private volatile boolean flag = false;

void setFlag() {

flag = true; // Writing to the volatile variable

}

boolean checkFlag() {

return flag; // Always reads the latest value from main memory

}

}

**6. native**

The native keyword in Java is used to indicate that a method is implemented in another language, typically C or C++, and not in Java. It is often used for performance reasons or to interact with system-level resources.

* Native methods are declared using the native modifier, and their implementation is provided outside of Java.

**Example**:

class Example {

// Declaring a native method

public native void someNativeMethod();

static {

// Loading the native library

System.loadLibrary("nativeLib");

}

}