**Java Language Fundamentals**

**1. Classes and Objects:**

* **Class**: A class is a blueprint for creating objects. It defines the properties and behaviors that the objects of the class will have.

class Car {

String make;

String model;

void start() {

System.out.println("Car is starting");

}

}

* **Object**: An object is an instance of a class. It holds actual values for the properties defined in the class.

Car myCar = new Car();

myCar.make = "Toyota";

myCar.model = "Corolla";

myCar.start();

**2. Class Syntax:**

* A class begins with the class keyword and contains member variables (fields) and methods.

public class MyClass {

int number;

String name;

void display() {

System.out.println("Number: " + number + ", Name: " + name);

}

}

**3. Primitive Data Types:**

Java provides eight primitive data types:

* byte (1 byte)
* short (2 bytes)
* int (4 bytes)
* long (8 bytes)
* float (4 bytes)
* double (8 bytes)
* char (2 bytes)
* boolean (1 bit)

Example:

int number = 10;

double price = 19.99;

char letter = 'A';

boolean isValid = true;

**4. Implicit and Explicit Conversion:**

* **Implicit Conversion**: Automatic conversion done by the compiler, usually when a smaller data type is converted to a larger one (widening).

int a = 100;

long b = a; // Implicit conversion from int to long

* **Explicit Conversion**: The programmer must manually convert a larger data type to a smaller one (narrowing).   
  double d = 9.78;  
  int i = (int) d; // Explicit conversion from double to int

float f=(float)d;

**5. Access Specifiers: To access the variable,method or any class or interface we use access specifier.**

Java has four access specifiers:

* public: Accessible from any class.
* protected: Accessible within the same package and subclasses.
* default (no specifier): Accessible only within the same package.
* private: Accessible only within the same class.

Example:

public class Example {

private int x;

protected int y;

}

**6. Constructor:**

A constructor is a special method that is called when an object is instantiated. It has the same name as the class and no return type.

class Car {

String make;

// Constructor

Car(String make) {

this.make = make;

}

}

**7. Creating an Object:**

An object is created using the new keyword, followed by the constructor of the class.

Car myCar = new Car("Toyota");

**8. Parameterized Constructor:**

A constructor that takes parameters is used to initialize object properties during creation.

class Car {

String make;

int year;

// Parameterized constructor

Car(String make, int year) {

this.make = make;

this.year = year;

}

}

**9. Memory Allocation:**

* **Stack Memory**: Stores local variables and function call frames.
* **Heap Memory**: Used for dynamic memory allocation, where objects are created.

**10. Garbage Collection:**

Java has an automatic garbage collection mechanism to reclaim memory by removing objects that are no longer referenced.

* The System.gc() method can be called, though the JVM typically handles garbage collection automatically.

**11. this Keyword:**

The this keyword refers to the current instance of the class. It is used to differentiate instance variables from local variables with the same name.

class Car {

String make;

Car(String make) {

this.make = make; // 'this' refers to the current instance variable

}

}

**12. static Keyword – Variable & Method:**

* **Static Variables**: Shared among all instances of the class.
* **Static Methods**: Can be called without creating an object of the class.

class Math {

static int add(int a, int b) {

return a + b;

}

}

**13. Main Method:**

The entry point for a Java application. It must be defined as:

public static void main(String[] args) {

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

}

**14. Instance Init Block & Static Init Block:**

* **Instance Init Block**: Code that runs when an object is created.

class Car {

{

System.out.println("Instance Block executed");

}

}

* **Static Init Block**: Code that runs when the class is loaded.

class Car {

static {

System.out.println("Static Block executed");

}

}

**15. Method Overloading:**

Java allows methods to have the same name but with different parameters (either in type, number, or both).

class Math {

int add(int a, int b) {

return a + b;

}

double add(double a, double b) {

return a + b;

}

}

**16. Array – 1D, 2D:**

* **1D Array**: A simple array.

int[] numbers = {1, 2, 3, 4, 5};

* **2D Array**: An array of arrays.

int[][] matrix = {{1, 2}, {3, 4}, {5, 6}};

**17. Enhanced For Loop (forEach Loop):**

This is a shorthand loop for iterating over collections or arrays.

int[] numbers = {1, 2, 3, 4, 5};

for (int num : numbers) {

System.out.println(num);

}

**18. Variable Arguments (Varargs):**

Java allows methods to accept a variable number of arguments using ....

class Math {

static int sum(int... numbers) {

int total = 0;

for (int num : numbers) {

total += num;

}

return total;

}

}

**19. Package:**

A package is a namespace for organizing classes and interfaces.

package com.example;

public class MyClass {

public static void main(String[] args) {

System.out.println("Inside MyClass");

}

}

**20. Importing from Packages:**

Java allows importing classes or entire packages to avoid using fully qualified names.

import java.util.Scanner;

class Test {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

}

}

**21. Static Import:**

You can import static members (fields and methods) of a class directly.

import static java.lang.Math.\*;

class Test {

public static void main(String[] args) {

double result = sqrt(25); // No need to use Math.sqrt()

System.out.println(result);

}

}