**1. Basics**

* **Hello World**
  + **Definition**: A simple Java program to display "Hello, World!" on the console.
  + **Example**:

public class HelloWorld {

public static void main(String[] args) {

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

}

}

* **Data Types**
  + **Primitive Types**: The basic data types provided by Java.
    - **int**: Integer (e.g., int age = 25;)
    - **long**: Long integer (e.g., long distance = 100000L;)
    - **short**: Short integer (e.g., short smallNumber = 1000;)
    - **byte**: Byte integer (e.g., byte b = 127;)
    - **float**: Floating-point number (e.g., float price = 19.99f;)
    - **double**: Double-precision floating-point number (e.g., double pi = 3.14159;)
    - **char**: Single character (e.g., char letter = 'A';)
    - **boolean**: True/false value (e.g., boolean isJavaFun = true;)
  + **Wrapper Classes**: Objects corresponding to the primitive types.
    - **Integer**, **Long**, **Short**, **Byte**, **Float**, **Double**, **Character**, **Boolean**
* **Variables**
  + **Definition**: Containers for storing data values.
  + **Example**:

int number = 10; // Integer variable

double price = 9.99; // Double variable

String name = "Java"; // String variable

**2. Control Flow**

* **If-Else**
  + **Definition**: Conditional statements that execute code blocks based on boolean expressions.
  + **Example**:

int x = 10;

if (x > 0) {

System.out.println("Positive");

} else if (x < 0) {

System.out.println("Negative");

} else {

System.out.println("Zero");

}

* **Switch**
  + **Definition**: A multi-way branch statement that allows execution based on the value of an expression.
  + **Example**:

int day = 2;

switch (day) {

case 1:

System.out.println("Monday");

break;

case 2:

System.out.println("Tuesday");

break;

default:

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

}

* **Loops**
  + **For Loop**: Repeats a block of code a specific number of times.

for (int i = 0; i < 10; i++) {

System.out.println(i);

}

* + **While Loop**: Repeats a block of code as long as a condition is true.

int i = 0;

while (i < 10) {

System.out.println(i);

i++;

}

* + **Do-While Loop**: Similar to while but executes at least once before checking the condition.

int j = 0;

do {

System.out.println(j);

j++;

} while (j < 10);

**3. Methods**

* **Method Definition**
  + **Definition**: A block of code that performs a specific task and can return a value.
  + **Example**:

java

public int add(int a, int b) {

return a + b;

}

* **Method Overloading**
  + **Definition**: Defining multiple methods with the same name but different parameters.
  + **Example**:

java

public void display(int a) {

System.out.println("Integer: " + a);

}

public void display(String a) {

System.out.println("String: " + a);

}

**4. Object-Oriented Programming**

* **Classes and Objects**
  + **Definition**: Classes define the blueprint for objects. Objects are instances of classes.
  + **Example**:

public class Person {

String name;

int age;

public void introduce() {

System.out.println("My name is " + name + " and I am " + age + " years old.");

}

}

* **Inheritance**
  + **Definition**: A mechanism where one class inherits fields and methods from another class.
  + **Example**:

public class Animal {

public void eat() {

System.out.println("This animal eats food.");

}

}

public class Dog extends Animal {

public void bark() {

System.out.println("The dog barks.");

}

}

* **Polymorphism**
  + **Definition**: The ability to process objects differently based on their data type or class.
  + **Example**:

Animal myAnimal = new Dog(); // Dog is an Animal

myAnimal.eat(); // Calls the method from Animal class

* **Abstraction and Encapsulation**
  + **Abstraction**: Hiding complex implementation details and showing only the necessary features.
    - **Example**:

java

public abstract class Shape {

public abstract void draw(); // Abstract method

}

* + **Encapsulation**: Protecting the data by bundling the data and methods that operate on the data.
    - **Example**:

java

public class Employee {

private String name; // Private field

public String getName() {

return name; // Getter method

}

public void setName(String name) {

this.name = name; // Setter method

}

}

**5. Exception Handling**

* **Try-Catch**
  + **Definition**: Blocks used to handle exceptions or errors that occur during program execution.
  + **Example**:

java

try {

int result = 10 / 0;

} catch (ArithmeticException e) {

System.out.println("Error: " + e.getMessage());

}

* **Finally**
  + **Definition**: A block that is always executed after try and catch blocks, regardless of whether an exception occurred.
  + **Example**:

java

try {

// Code that might throw an exception

} finally {

// Code that will always execute

}

**6. Collections Framework**

* **List**
  + **Definition**: An ordered collection that allows duplicate elements.
  + **Example**:

java

List<String> list = new ArrayList<>();

list.add("Java");

list.add("Python");

* **Set**
  + **Definition**: A collection that does not allow duplicate elements.
  + **Example**:

java

Set<String> set = new HashSet<>();

set.add("Java");

set.add("Java"); // Duplicate will not be added

* **Map**
  + **Definition**: A collection of key-value pairs where each key is unique.
  + **Example**:

java

Map<String, Integer> map = new HashMap<>();

map.put("Java", 1);

map.put("Python", 2);

**7. Multithreading**

* **Creating a Thread**
  + **Definition**: Creating a thread by extending the Thread class and overriding the run method.
  + **Example**:

java

public class MyThread extends Thread {

public void run() {

System.out.println("Thread running");

}

}

public class Main {

public static void main(String[] args) {

MyThread thread = new MyThread();

thread.start();

}

}

* **Runnable Interface**
  + **Definition**: Creating a thread by implementing the Runnable interface and passing it to a Thread object.
  + **Example**:

java

public class MyRunnable implements Runnable {

public void run() {

System.out.println("Runnable running");

}

}

public class Main {

public static void main(String[] args) {

Thread thread = new Thread(new MyRunnable());

thread.start();

}

}

**8. File I/O**

* **File Writing**
  + **Definition**: Writing data to a file using FileWriter.
  + **Example**:

java

try (FileWriter writer = new FileWriter("file.txt")) {

writer.write("Hello, World!");

} catch (IOException e) {

e.printStackTrace();

}

* **File Reading**
  + **Definition**: Reading data from a file using BufferedReader.
  + **Example**:

java

try (BufferedReader reader = new BufferedReader(new FileReader("file.txt"))) {

String line;

while ((line = reader.readLine()) != null) {

System.out.println(line);

}

} catch (IOException e) {

e.printStackTrace();

}