**Section 5: Java Basic Course**

**Key Concepts & Implementation Steps**

1. **Setting Up Java in Android Studio**
   * **Concept**: Create Java-only projects (no Android activity).
   * **Steps**:

java

New Project → "No Activity" → Name project (e.g., "Java Examples") →

Set package (e.g., `com.mastercoding.javaexamples`) → File > New Module > "Java Library"

* + **Tools**: Android Studio, Java/Kotlin Library module.
  + **Best Practice**: Use "No Activity" to avoid unnecessary Android components.

1. **Comments**
   * **Concept**: Non-executable notes for code documentation.
   * **Syntax**:

java

*// Single-line comment*

*/\* Multi-line*

*comment \*/*

* + **Best Practice**: Use comments to explain complex logic or disable code temporarily.

1. **The**main**Method**
   * **Concept**: Entry point for Java programs.
   * **Syntax**:

java

public static void main(String[] args) {

*// Code here*

}

* + **Key Terms**:
    - public: Accessible globally.
    - static: No class instance needed.
    - void: No return value.
    - String[] args: Command-line arguments.

1. **Output with**System.out
   * **Concept**: Print messages to the console.
   * **Syntax**:

java

System.out.print("Hello"); *// No newline*

System.out.println("World"); *// With newline*

1. **Variables**
   * **Concept**: Containers for storing data.
   * **Syntax**:

java

int age = 25; *// Declaration & initialization*

age = 30; *// Update value*

* + **Best Practice**: Use meaningful names (e.g., userAge instead of a).

1. **Primitive Data Types**

| **Type** | **Range/Values** | **Example** |
| --- | --- | --- |
| byte | -128 to 127 | byte x = 100; |
| short | -32,768 to 32,767 | short y = 7777; |
| int | -2³¹ to 2³¹-1 | int z = 88776655; |
| long | -2⁶³ to 2⁶³-1 (suffix L) | long w = 112233L; |
| float | Single-precision (suffix F) | float pi = 3.14F; |
| double | Double-precision (default) | double pi2 = 3.1415926535; |
| boolean | true or false | boolean isRaining = true; |
| char | Single character (Unicode) | char grade = 'A'; |

1. **Type Casting**
   * **Implicit (Safe)**: Automatic conversion (e.g., int → double).

java

int num = 10;

double numDouble = num; *// 10.0*

* + **Explicit (Manual)**: Potential data loss (e.g., double → int).

java

double pi = 3.14;

int numPi = (int) pi; *// 3 (truncates decimal)*

1. **Operators**
   * **Arithmetic**: +, -, \*, /, % (modulus).

java

int remainder = 10 % 3; *// 1*

* + **Comparison**: ==, !=, >, <, >=, <= (return boolean).

java

boolean isEqual = (x == 10);

* + **Logical**: && (AND), || (OR), ! (NOT).

java

boolean goOutside = isSunny && !needUmbrella;

1. **Strings**
   * **Concept**: Objects for text (non-primitive).
   * **Syntax**:

java

String greeting = "Hello";

* + **Concatenation**: Use +.

java

String message = greeting + " World!";

* + **Methods**:

java

int len = message.length(); *// Returns string length*

1. **Conditional Statements**
   * if-else:

java

if (age >= 18) {

System.out.println("Adult");

} else {

System.out.println("Minor");

}

* + switch:

java

switch (day) {

case 1: System.out.println("Sunday"); break;

default: System.out.println("Unknown");

}

1. **Loops**
   * while:

java

int i = 1;

while (i <= 5) {

System.out.println(i);

i++;

}

* + for:

java

for (int j = 0; j < 5; j++) {

System.out.println(j);

}

* + break**/**continue:

java

if (i == 4) break; *// Exit loop*

if (i == 2) continue; *// Skip iteration*

1. **Arrays**
   * **1D Array**:

java

int[] numbers = new int[3];

numbers[0] = 10; *// Index starts at 0*

* + **2D Array**:

java

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

System.out.println(matrix[1][0]); *// 3*

* + **For-each Loop**:

java

for (int num : numbers) {

System.out.println(num);

}

**Tools & Best Practices**

* **Tools**: Android Studio, Java SDK.
* **APIs**: System.out, String methods.
* **Best Practices**:
  + Use double over float for precision.
  + Prefer for loops when iterations are known.
  + Avoid magic numbers; use constants.
  + Always handle array bounds (index 0 to length-1).
* **Alternatives**: Use StringBuilder for heavy string manipulation.

**Part B: Important Topics Not Covered**

1. **Object-Oriented Programming (OOP)**
   * Classes, objects, inheritance, polymorphism.
   * **Why**: Core of Java; enables modular code.
2. **Methods**
   * Defining/calling methods, parameters, return types.
   * **Why**: Reusability and code organization.
3. **Exception Handling**
   * try-catch, custom exceptions.
   * **Why**: Critical for robust error management.
4. **Collections Framework**
   * ArrayList, HashMap (dynamic data structures).
   * **Why**: More flexible than arrays for real-world apps.
5. **Input/Output (I/O)**
   * Reading user input (Scanner class), file I/O.
   * **Why**: Essential for interactive applications.
6. **Java Standard Library**
   * Utilities like Math, Arrays, Collections.
   * **Why**: Prebuilt tools for common tasks.
7. **Enums**
   * Defining fixed sets of constants (e.g., enum Day { MON, TUE }).
   * **Why**: Type safety for predefined values.
8. **Packages**
   * Organizing classes into namespaces.
   * **Why**: Avoid naming conflicts; modular design.
9. **Java Memory Management**
   * Stack vs. heap, garbage collection.
   * **Why**: Optimize performance and avoid leaks.
10. **Modern Java Features**
    * Lambdas, streams (Java 8+).
    * **Why**: Concise syntax for functional programming.

**Note**: These topics are foundational for advanced Java/Android development and should be covered in subsequent sections.