# Java Programming Lecture Notes (Expanded)

## 1. Understand Programming Concepts

### Definition of Programming

Programming is giving a computer a set of precise instructions it can execute to solve problems. These instructions are written using a programming language. Good programming requires clarity, logic, and attention to detail.

### Phases of Program Development

1. **Establish Program Requirements** – Define the problem clearly. What input is needed? What output is expected? Example: A program to calculate student grades needs student marks as input and final grades as output.
2. **Design a Program** – Plan a step-by-step solution. Use pseudocode or flowcharts to represent logic before writing code. Example:

* START  
  Input marks  
  If marks >= 50 then print "Pass" else print "Fail"  
  END

1. **Coding** – Write the plan in a programming language like Java. Use proper syntax, indentation, and comments.
2. **Test and Debug** – Run the program with different inputs. Check for errors like syntax errors (typos) and logic errors (wrong results). Debugging tools like Eclipse debugger can help trace problems.
3. **Document** – Write comments inside code and prepare user manuals so others can understand how to use and maintain the program.
4. **Maintain** – Modify and update code when requirements change or to fix bugs found later.

### Key Terms

* **Algorithm** – A clear set of steps that solves a problem.
* **Source Code** – The programmer’s code before it is compiled.
* **Executable** – The file that runs on a computer after compilation.
* **Compiling** – Translating source code to bytecode or machine code.
* **Debugging** – The process of finding and fixing errors in code.

### Types of Code

* **Source Code** – Human-readable text.
* **Object Code** – Intermediate compiled form.
* **Machine Code** – Binary form executed by the CPU.

### Translators Used in Programming

* **Compiler** – Translates whole program at once. Faster execution.
* **Interpreter** – Executes line by line. Easier for testing.
* **Assembler** – Converts assembly language into machine code.

### OOP Fundamental Concepts

* **Encapsulation** – Hide data inside classes, access with methods.
* **Inheritance** – Share attributes and methods between classes.
* **Polymorphism** – Methods with the same name behave differently based on object type.
* **Abstraction** – Show only what is necessary, hide implementation details.

## 2. Understand the Java Environment

### Installation of Java

* **Download JDK** – From Oracle or OpenJDK site.
* **Install JDK** – Follow wizard; install JRE if prompted.
* **Set Environment Variables** – Add JAVA\_HOME path and update system PATH to include bin folder.

### Java Programming Environment

* **Command-Line Tools:** javac (compiler) and java (runner).
* **IDE Tools:** Eclipse, IntelliJ, NetBeans for faster development.

### Eclipse IDE Setup

1. Download from [eclipse.org](https://www.eclipse.org/downloads/).
2. Extract ZIP file and run eclipse.exe.
3. Select a workspace folder.
4. Create a new Java project.
5. Write a simple Hello World program and run it.

### Java Basics (Syntax and Rules)

* **Case Sensitivity** – Variable and variable are different.
* **Class Names** – Capitalize first letter.
* **Method Names** – Use lowerCamelCase.
* **Program File Name** – Must match class name.
* **main Method** – public static void main(String[] args) is the program entry point.
* **Identifiers** – Follow naming rules: no spaces, cannot start with numbers.
* **Modifiers** – Control access: public, private, protected.
* **Variables** – Store data values.
* **Arrays** – Store multiple values of same type.
* **Enums** – Define a fixed set of constants.
* **Keywords** – Reserved words like class, if, else, while, cannot be used as identifiers.

## 3. Perform Data Operations

### Java Data Types

* **Primitive Types:** int, byte, short, long, float, double, char, boolean.
* **Reference Types:** Strings, Arrays, Objects.

### Statements

* **Expression:** Produces a value. Example: a + b
* **Declaration:** Creates variables. Example: int x = 5;
* **Control-flow:** Direct program execution. Example: if (x > 0)

### Variables and Constants

* **Local Variables:** Exist within a method.
* **Class Variables:** Declared as static, shared by all objects.
* **Instance Variables:** Belong to a specific object.
* **Constants:** Declared with final. Example: final double PI = 3.14;

### Data Operations

* **Assignment:** x = 10;
* **Reading Input:** Using Scanner.
* **Arithmetic:** + - \* / %
* **Object Instantiation:** Student s = new Student();

### Example Programs

* Area of Circle: area = Math.PI \* r \* r;
* Quadratic Equation: Use discriminant formula (-b ± √(b²-4ac))/2a.
* Compound Interest: A = P \* Math.pow(1 + r/n, n\*t);

## 4. Use Control Structures

### Decision Making

* **if:** if (x > 0) {...}
* **if-else:** Provides alternate path.
* **switch:** Multiple case choices.

### Looping

* **for:** Definite loops.
* **while:** Condition checked before loop.
* **do-while:** Executes at least once.

### Branching

* **break:** Exit loop.
* **continue:** Skip current iteration.

### Example Program

for (int i = 1; i <= 10; i++) {  
 if (i == 5) continue;  
 System.out.println(i);  
}

## 5. Use Methods

### Definition

Methods group code that performs a task and can be reused.

### Structure

returnType methodName(parameters) {  
 // body  
}

### Key Concepts

* **Method Creation:** Define in class.
* **Method Calling:** object.methodName();
* **void:** No return value.
* **Parameter Passing:** Java passes values by copy.
* **Overloading:** Multiple methods with same name but different parameters.
* **Command-Line Args:** Read values from terminal.
* **this:** Refers to current object.
* **Variable Arguments:** int... nums
* **finalize():** Cleanup before object is destroyed.

### Example

public int add(int a, int b) {  
 return a + b;  
}

## 6. Understand Object-Oriented Programming

### Concepts

* **Class:** Blueprint. Contains attributes (fields) and methods.
* **Object:** Instance of a class.
* **Inheritance:** Use extends. Child class inherits parent features.
* **Encapsulation:** Use private + getters/setters.
* **Abstraction:** Use abstract classes/interfaces.
* **Polymorphism:** Same method behaves differently depending on class.

### Example

class Animal {  
 void sound() { System.out.println("Animal sound"); }  
}  
class Dog extends Animal {  
 @Override  
 void sound() { System.out.println("Bark"); }  
}

This demonstrates inheritance and polymorphism. Create objects of each and call sound() to see different results.

**Suggested Assessment Methods:**

* Practical lab tests with real coding tasks.
* Oral tests to explain key terms.
* Written quizzes with definitions and short code snippets.