برنامه نویسی پیشرفته

رفع اشکال: جلسه ۱

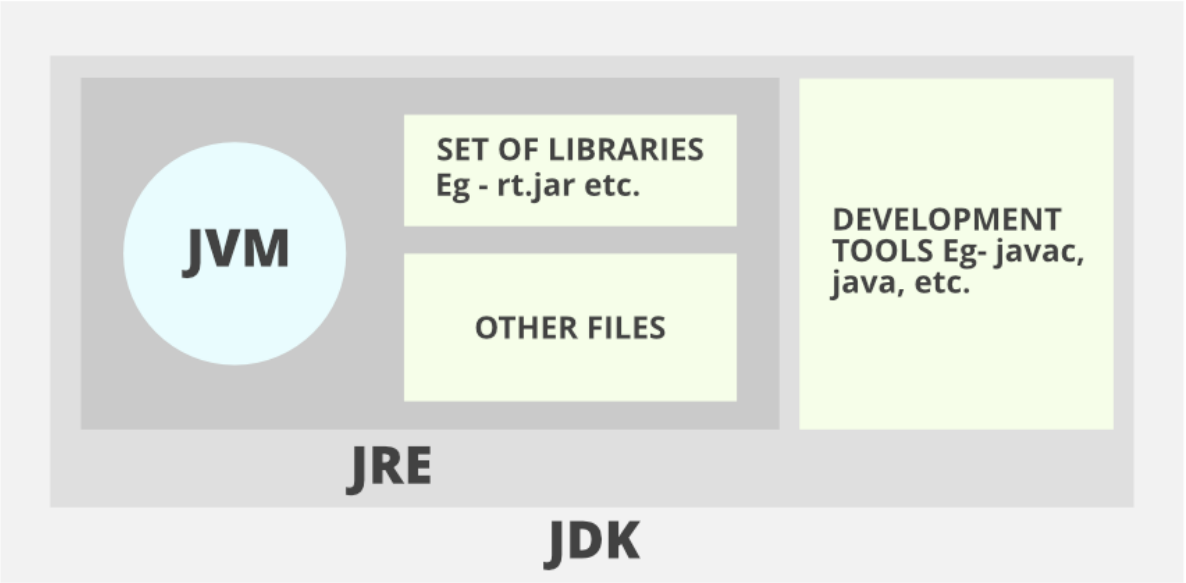
Introduction to Java and JDK
Basic syntax, control flow, classes and objects
Constructors, getters, setters, and encapsulation
Class Diagram
Examples

Java

Java is a high-level, class-based, object-oriented programming. It is intended to let application developers write once, run anywhere (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.

Java Development Kit (JDK):

- Compiler (javac): Converts Java source code into bytecode.
- Java Runtime Environment (JRE): Provides libraries, JVM, and other components.
- Java Virtual Machine (JVM): An abstract machine that enables your computer to run a Java program
- Java Archive Tool (jar): Packages multiple files into a single JAR archive.



Java execution

1. Write the Source Code:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

2. Compile to ByteCode:

```
javac HelloWorld.java
```

3. Run the Program:

```
java HelloWorld
```

4. Output:

```
Hello, World!
```

Sample Questions (1)

1. What is the difference between JVM, JRE, and JDK?

2. Explain Java Program Execution Process.

ANSWERS (1)

1. Difference between JVM, JRE, and JDK:
   - JVM: Executes Java bytecode. Platform-dependent runtime engine.
   - JRE: Includes JVM + libraries to run Java applications. Used by end-users.
   - JDK: Includes JRE + development tools (e.g., `javac`, debugger). Used by developers.

2. Java Program Execution Process:
   - Write `.java` file → Compile with `javac` → Generate `.class` (bytecode) → JVM loads, verifies, and executes bytecode.
   - Key Steps: Compilation → Loading → Verification → Execution → Output.

Basic Syntax

- Case Sensitivity: Java is case-sensitive; Variable and variable are different identifiers.
- Class Names: By convention, class names start with an uppercase letter.
- Method Names: Method names start with a lowercase letter.
- File Name: The filename must match the class name and have a .java extension.

Operators	Precedence
!, +, – (unary operators)	first (highest)
*, /, %	second
+, –	third
<, <=, >=, >	fourth
==, !=	fifth
&&	sixth
||	seventh
= (assignment operator)	last (lowest)

Sample Questions (2)

1. What error occurs if the file name is not the same as the class name?
2. Can we have multiple classes in the same file? Can we have multiple **public** classes in the same file?

ANSWERS (2)

1. Compile Error occurs: Java checks each .java file at compile time and expects the public class name to be the same as the file name. Otherwise, the class cannot be found and processed correctly.
2. No: The compiler needs to know which class to consider as the main file. As a result, the compiler gives a file naming error. But Java allows more than one non-public class in a file.

```
public class MainClass {
    public static void main(String[] args) {
        HelperClass.sayHello();
    }
}
class HelperClass {
    static void sayHello() {
        System.out.println("Hello from HelperClass!");
    }
}
```

Non-public classes are actually "helper" classes and can only be used from within the same file or package.
Non-public classes do not need to match the file name.

Conditional Statements

- **if Statement:**

```
int x = 10;
if (x > 5) {
    System.out.println("x is greater than 5");
}
```

- **if-else Statement:**

```
int x = 10;
if (x > 5) {
    System.out.println("x is greater than 5");
} else {
    System.out.println("x is less than or equal to 5");
}
```

Switch Statements

- **switch Statement:**

```
int day = 3;
switch (day) {
    case 1:
        System.out.println("Monday");
        break;
    case 2:
        System.out.println("Tuesday");
        break;
    case 3:
        System.out.println("Wednesday");
        break;
    default:
        System.out.println("Invalid day");
        break;
}
```

Enhanced switch is also supported in java with (->) notation.

Loops

- **for Loop:**

```
for (int i = 0; i < 5; i++) {
    System.out.println("i is: " + i);
}
```

- **while Loop:**

```
int i = 0;
while (i < 5) {
    System.out.println("i is: " + i);
    i++;
}
```

- **do-while Loop:**

```
int i = 0;
do {
    System.out.println("i is: " + i);
    i++;
} while (i < 5);
```

Sample Questions (3)

1. What error occurs if the wrong index (for example, a negative index or one greater than the length) is used when traversing an array?
2. Can an array be resized in Java after it has been defined?
3. What is the difference between if-else and switch statements?

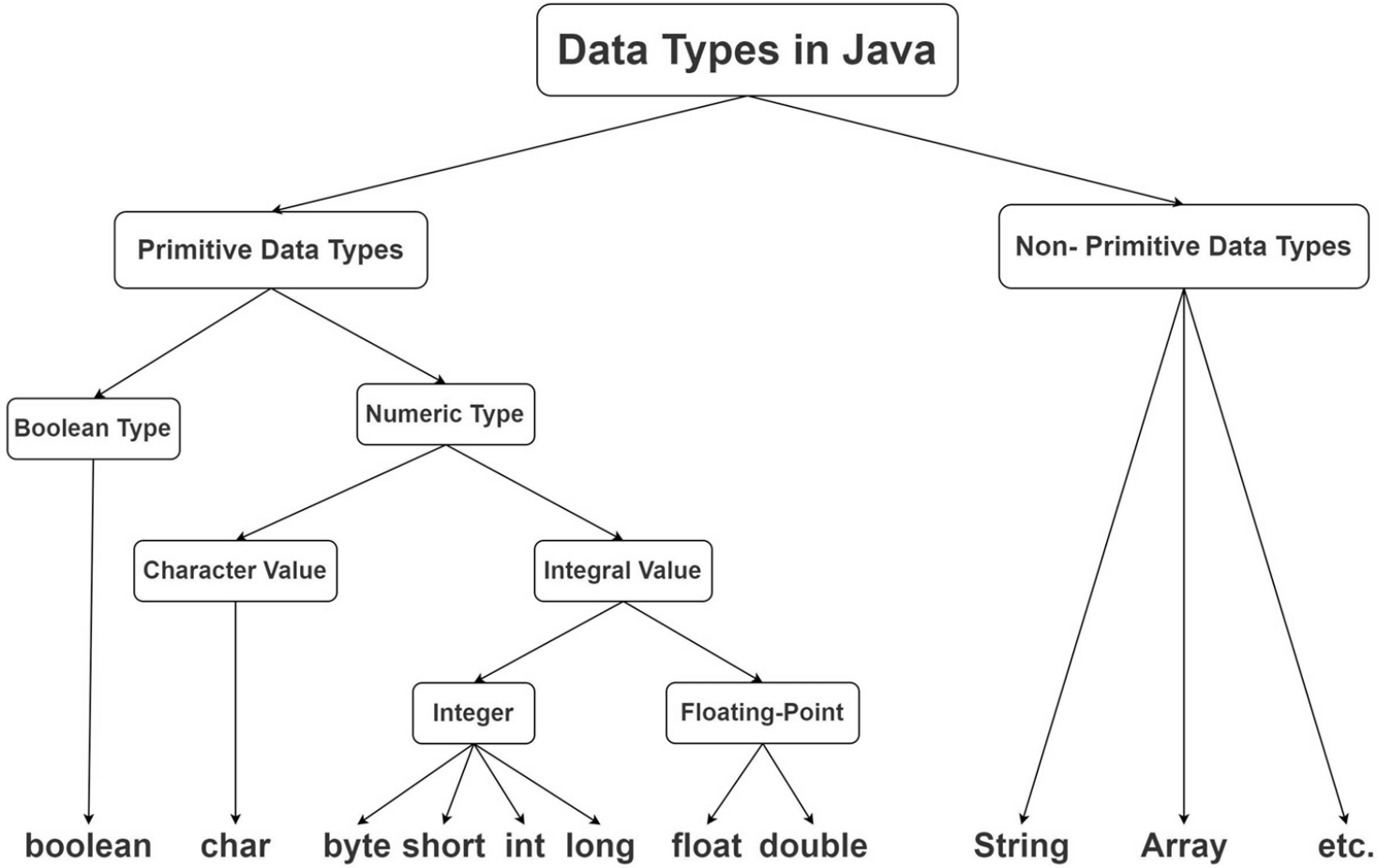
ANSWERS (3)

1. An ArrayIndexOutOfBoundsException error occurs also for negative index (not like C).

2. No, the size of an array cannot be changed after it is created . But you can use ArrayList (from java.util) to have a dynamic array (not like realloc in C)

3. Switch statements utilize a jump table which has O(1) access time, meaning 10 or 10000 choices has no impact on execution time.

Data Types


Data Types in Java
Primitive Data Types
Non- Primitive Data Types
Boolean Type
Numeric Type
Character Value
Integral Value
Integer
Floating-Point
boolean
char
byte short int long
float double
String
Array
etc.

Strings

You can create a String using string literals or the `new` keyword.

```
public class StringExample {
    public static void main(String[] args) {
        // Creating Strings
        String greeting = "Hello, Java!";
        String anotherGreeting = new String("Welcome to OOP!");
        // Using String methods
        System.out.println("Greeting: " + greeting);
        System.out.println("Length of greeting: " + greeting.length());
        System.out.println("Character at position 1: " + greeting.charAt(1));
        System.out.println("Substring (7, 11): " + greeting.substring(7, 11));  // prints "Java“
        System.out.println("Concatenated string: " + greeting.concat(" Let's code."));
    }
}
```

Arrays and For-each

An array in Java is a collection of elements of the same data type, stored in a contiguous block of memory.

```
int[] numbers;   // Preferred style
numbers_2 = new int[5];  // Creates an array with default values
int[] numbers_3 = {1, 2, 3, 4, 5};

System.out.println(numbers[0]);
System.out.println(numbers.length);

for (int num : numbers) {
        System.out.println(num);  // Outputs: 1, 2, 3, 4, 5
}
```

Scanner

The Scanner class in Java (found in the `java.util` package) is used to capture input from various sources.

```
import java.util.Scanner;

public class ScannerExample {
    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter your name: ");
        String name = scanner.nextLine();

        System.out.print("Enter your age: ");
        int age = scanner.nextInt();

        System.out.print("Enter your height in meters: ");
        double height = scanner.nextDouble();

        System.out.println("Hello, " + name + ". You are " + age + " years old and " + height + " meters tall.");

        // Close the scanner to free resources
        scanner.close();
    }
}
```

Sample Questions (4)

1. Why might unexpected behavior be seen if we immediately call nextLine() after using nextInt()?

ANSWERS (4)

1. In Java, the Scanner class processes input line by line and token by token. When reading data from the user, the Scanner uses an Input Buffer.

   When you use nextInt() to read a number, it only reads the numeric value but leaves the Enter (newline \n) in the buffer.

   Then if nextLine() is used immediately, instead of taking new input from the user, it will just read the remaining '\n' in the buffer. nextLine() does not take any new input from the user and is unexpectedly rejected.

   SOLUTION:
   1. put an extra nextLine()
   2. nextLine() for all inputs & convert the numeric value with Integer.parseInt()

Classes and Objects

A class is a blueprint that defines the properties (attributes) and behaviors (methods) common to all objects of that type.

```
public class Person {
    // attributes
    private String name;
    private int age;

    // Constructor
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    // Method
    public void displayInfo() {
        System.out.println("Name: " + name
           + ", Age: " + age);
    }
```

```
// Main method to test our class
    public static void main(String[] args) {

        // Creating objects
        Person p1 = new Person("Alice", 30);
        Person p2 = new Person("Bob", 25);

        // Calling methods
        p1.displayInfo();
        p2.displayInfo();
    }
}
```

Constructor

A constructor is a special method used to initialize new objects. It has the following key properties:

- Same name as the class.
- No return type.
- Can be overloaded: You can have multiple constructors with different parameter lists.
- The compiler automatically creates a default (no parameters) constructor unless programmed.

```
public class Car {
    private String brand;
    private int speed;
    public Car() {
        this.brand = "Unknown";
        this.speed = 0;
    }
    public Car(String brand, int speed) {
        this.brand = brand;
        this.speed = speed;
    }
}
```

Encapsulation

Encapsulation is the concept of bundling data (instance variables) and methods that operate on that data into one unit—a class. It also involves hiding the internal state of the object from the outside world. This is usually achieved by:

- Declaring instance variables as **private**.
- Providing public **getter and setter** methods to read and modify these variables.

Why Encapsulate?

- Control Access: Prevent direct access to the fields so that you can enforce constraints (e.g., a person's age cannot be negative).
- Maintainability: You can change the internal implementation later without affecting classes that use your object.
- Data Hiding: Protects the object's state from unintended interference.

access modifiers and non-access modifiers

Java provides different access modifiers to control the visibility of classes, variables, and methods:
Note that we will learn about non-access modifiers in later chapters.

Access Modifiers	Non-Access Modifiers
private default or No Modifier protected public	static final abstract synchronized transient volatile strictfp

	default	private	protected	public
same class	yes	yes	yes	yes
same package subclass	yes	no	yes	yes
same package non-subclass	yes	no	yes	yes
different package subclass	no	no	yes	yes
different package non-subclass	no	no	no	yes

STATIC FIELDS

The static keyword in Java is used to define fields (variables), methods and special blocks that belong to a class, rather than to each object (instance).
Static variables are created only once for the entire class and are shared by all objects of that class.

- When the value of a variable must be common to all objects

- When the value of a variable is independent of any object

- When we need a global variable

- When we want the initial value to be set only once

FINAL FIELDS

- final makes the value of a variable permanent (constant) after it has been initialized. final can be used for instance variables or static variables.

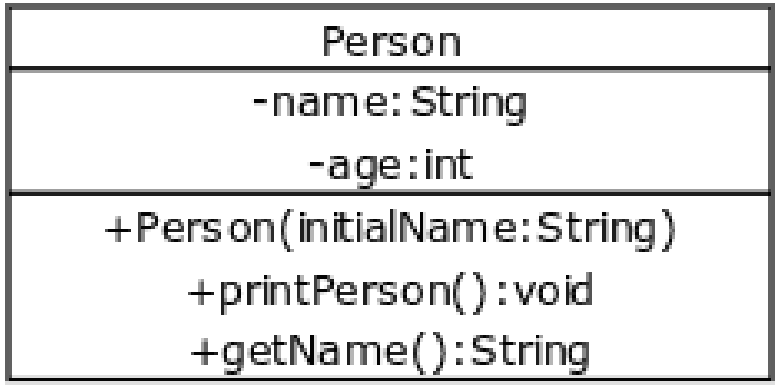
  If we want a value to be both static and final :
  is shared across all objects (static) does not change after it is initialized (final)

```
class MathUtil {
        static final double PI = 3.14159;
}
```

Class Diagrams

A class diagram is a diagram used in designing and modeling software to describe classes and their relationships. Class diagrams enable us to model software in a high level of abstraction and without having to look at the source code.
Note that we will learn about UML in later chapters.

- Classes: Represented as rectangles.
- Attributes (fields): Listed in the top section (often with a "–" sign if they're private).
- Methods (operations): Listed in the bottom section (with a "+" for public methods).
- Object Composition: User-defined types can be used in a class.



Time to Code

1. **Bank:**
   Develop a Java application that simulates a simple banking system. The system should manage customers and their bank accounts, allowing for basic operations such as account creation, deposits, withdrawals, and fund transfers between accounts.

2. **Library**
   Develop a Library Management System in Java that manages books and members, and allows users to search and borrow books.

PRACTICE

Medical Clinic Management System

Design a Java application that simulates a simple medical clinic management system.
The system should manage patients, doctors, and appointments, allowing for the following

- **Register Patients** with details such as **name, age, medical record number, and medical history**.
- **Register Doctors** with details like **name, specialization, and doctor ID**.
- **Schedule Appointments** between patients and doctors with a **specific date and time**.
- **Manage Prescriptions** including **medicine names, dosage, and doctor's recommendations**.
- **Search and view a patient's medical history** based on their medical record number.

پایان