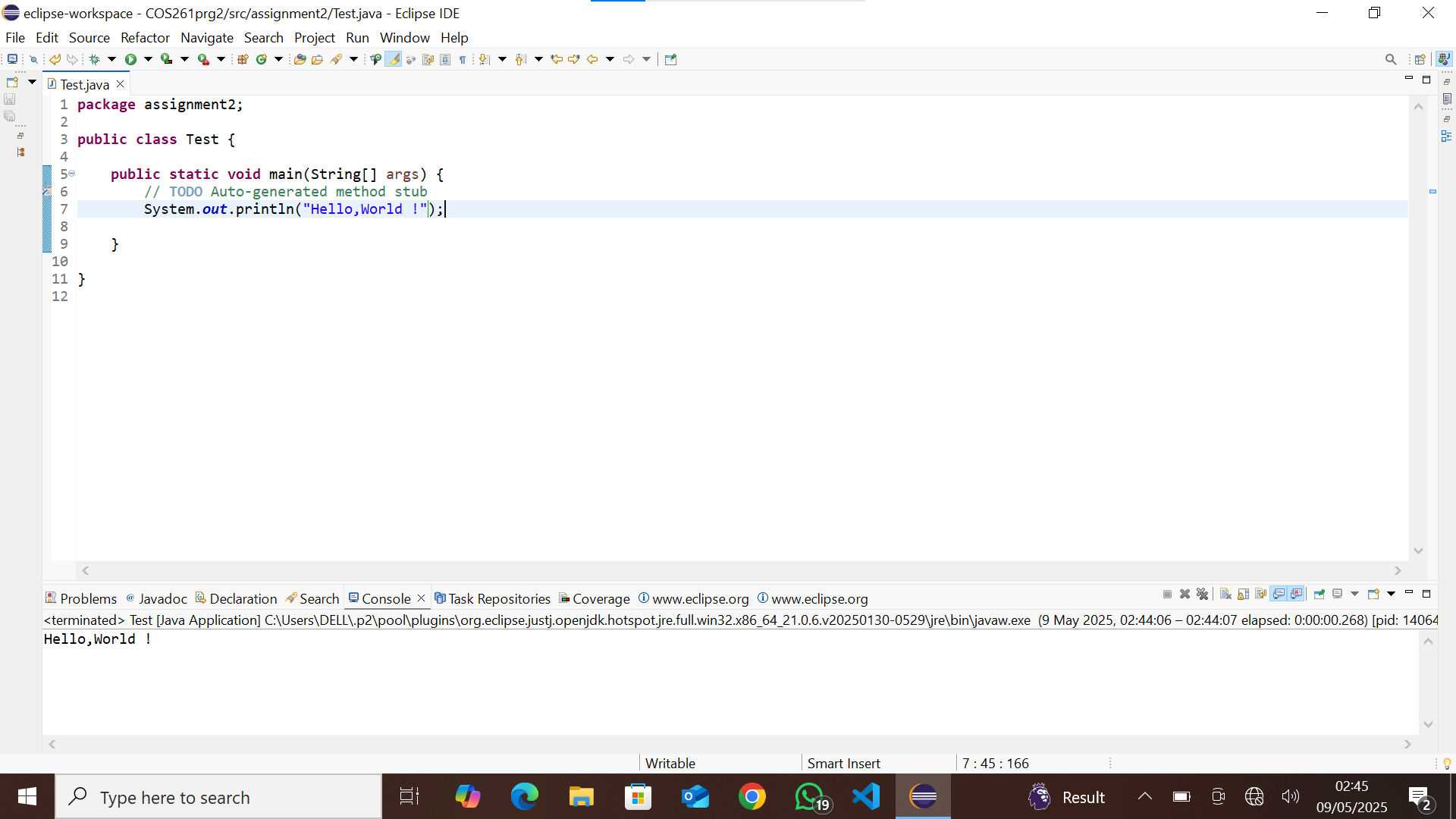
Proof of answer

1. Write a java program to print “Hello, World!”.



2.Explain the difference between == and .equals() in java.

== **(Equality Operator):**

* Compares **references** (memory addresses) rather than actual content.
* Used for primitive data types (int, char, boolean, etc.) and to check if two objects point to the same memory location.

.equals() **(Method for Content Comparison):**

* Compares the **content** of objects rather than memory addresses.
* Used for objects like String, List, or custom objects where logical equality matters.

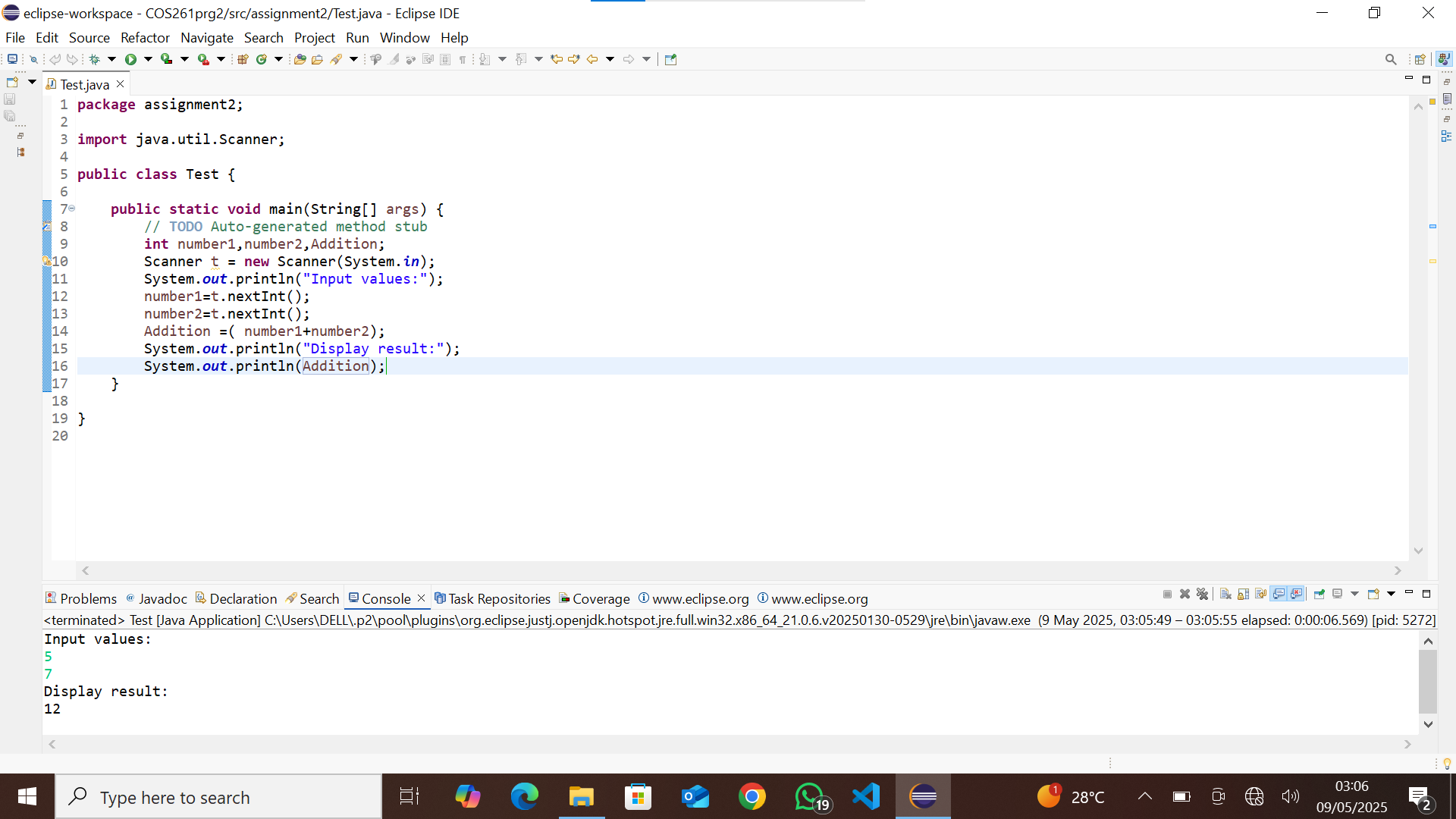
3. What is the main method in java?

In Java, the main **method** is the entry point of any Java application. When you run a Java program, the JVM (Java Virtual Machine) looks for this method and starts executing the code from there.

### **Breaking it Down:**

1. public – The method is accessible from anywhere.
2. static – No object of the class is needed to call this method.
3. void – The method doesn't return anything.
4. main – This is the special name recognized by the JVM.

4.Write a java program to add two numbers entered by the user.



5. What is the difference between int, integer, and String?

### **int (Primitive Data Type)**

* A primitive data type used to store whole numbers.
* More memory-efficient because it directly stores values.
* Cannot be null.

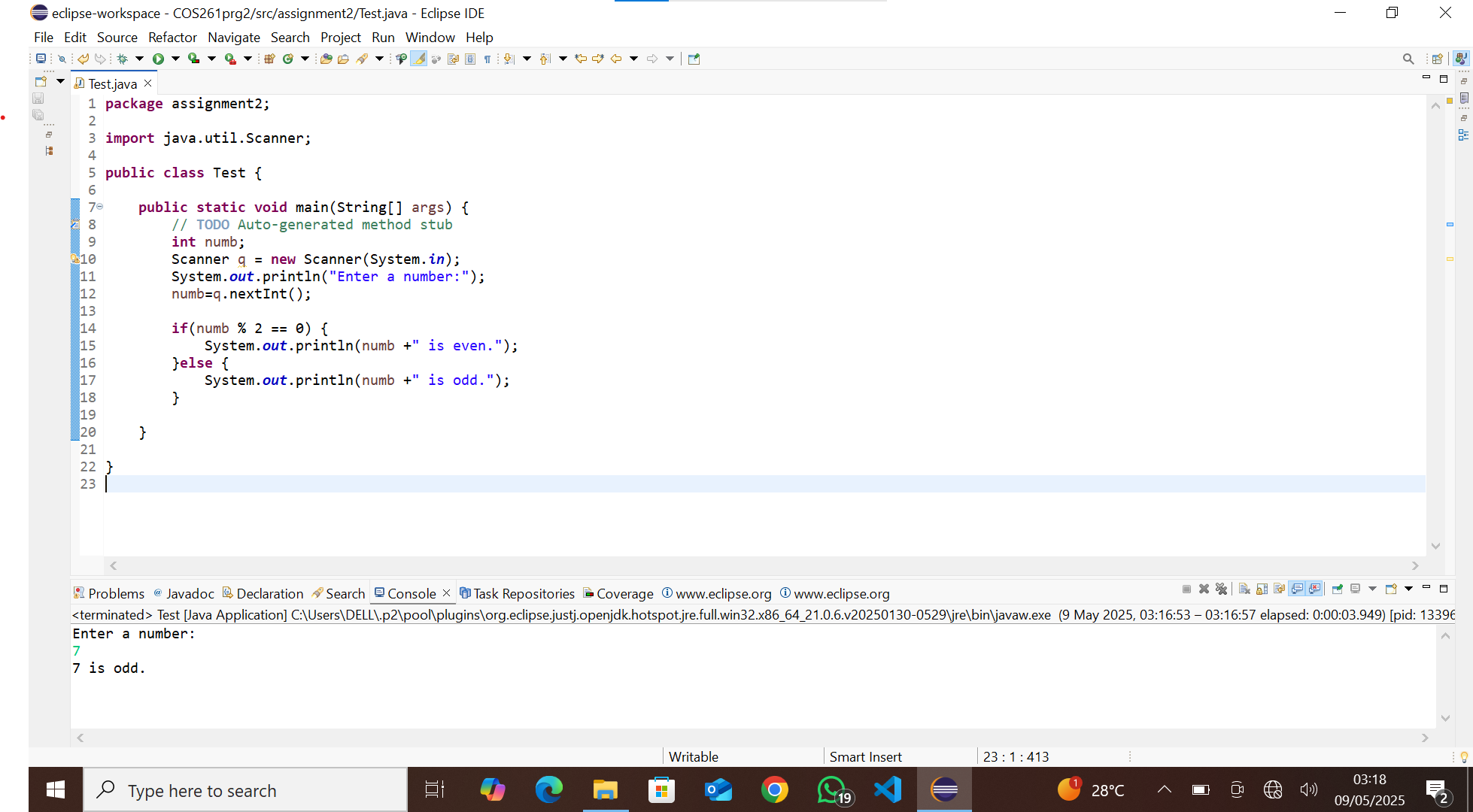
### **Integer (Wrapper Class)**

* A **wrapper class** for int found in java.lang.Integer.
* Allows use of objects and methods (like parseInt(), toString()).
* Can be null.
* Supports auto-boxing & unboxing (automatic conversion between int and Integer).

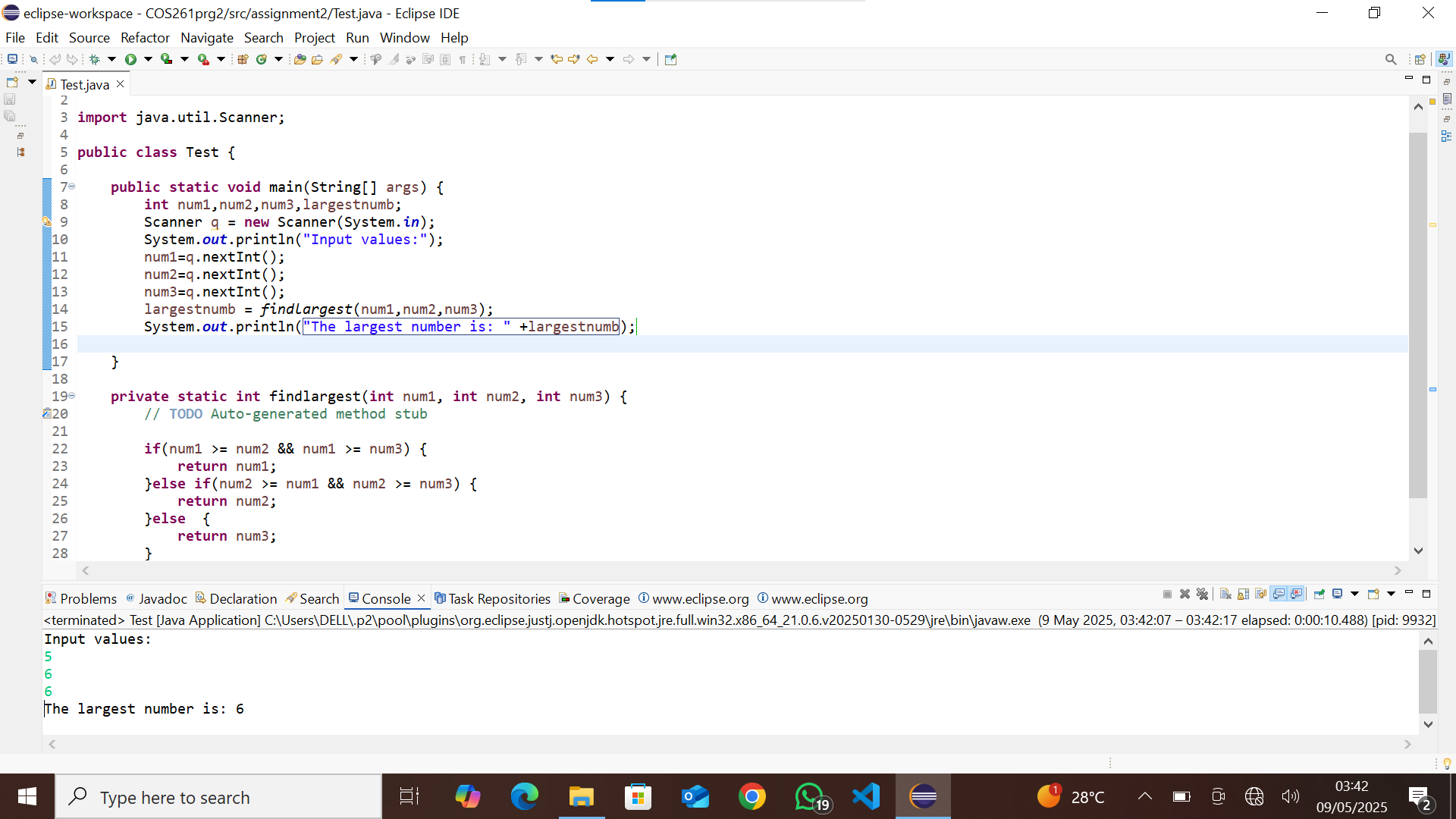
### **String (Sequence of Characters)**

* Represents a sequence of characters (text).
* **Immutable** (cannot be changed after creation).
* Can store numbers but treats them as text.

6. Write a program to check if a number is even or odd



7. Write a program to find the largest among three numbers



8. Explain the difference between while, for,and do while loops in java

### **while Loop (Entry-Controlled)**

* **Condition is checked first** before executing the loop body.
* If the condition is **false initially**, the loop **will not execute** at all.
* Used when the number of iterations is **not known beforehand**

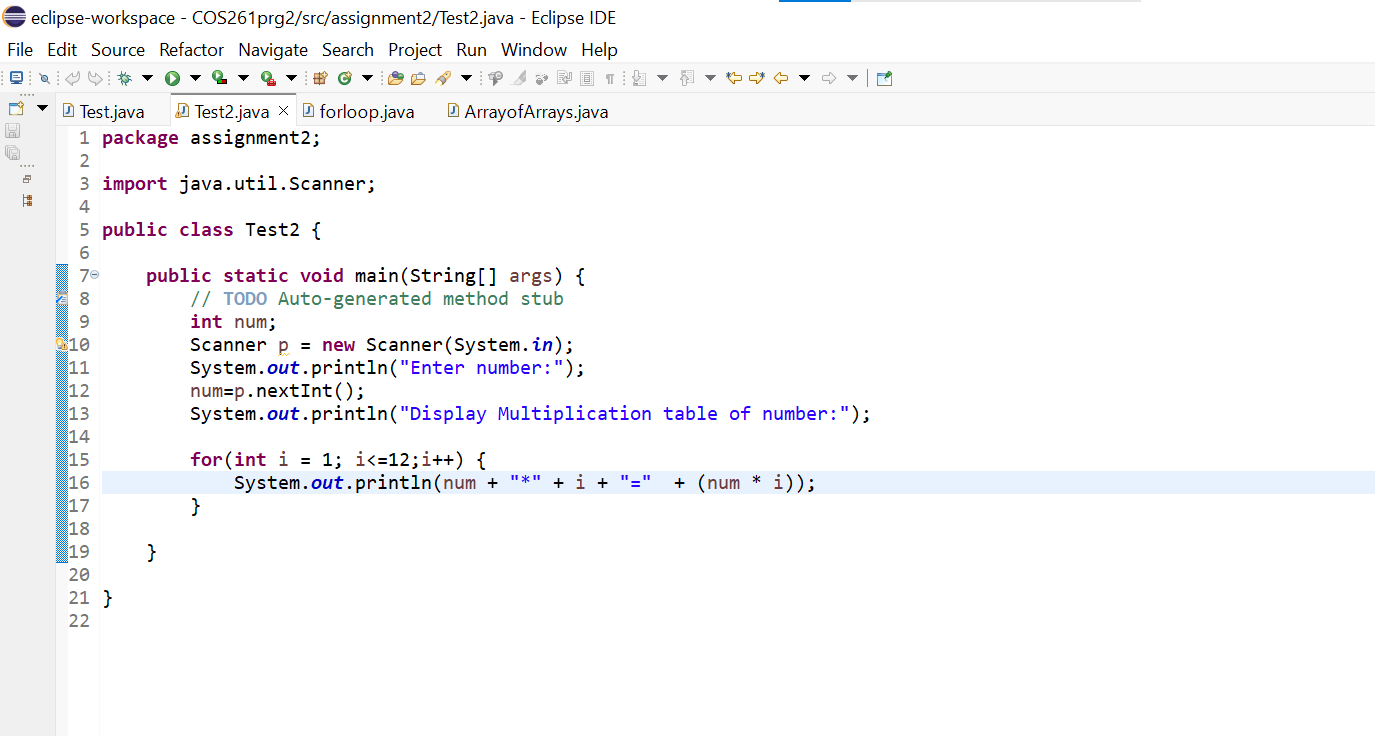
### **for Loop (Entry-Controlled)**

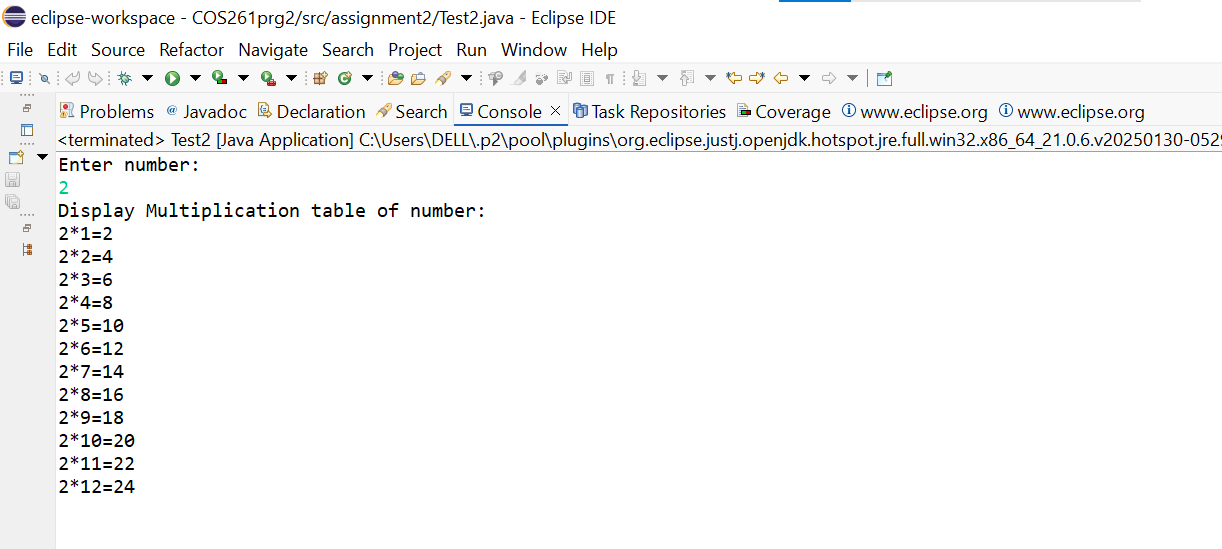
* Used when the number of iterations is **known beforehand**.
* Has a built-in initialization, condition check, and increment/decrement in one line.

### **do-while Loop (Exit-Controlled)**

* Executes **at least once**, since the condition is checked **after** executing the loop body.
* Used when you **need to guarantee at least one execution**.

9. Write a java program to print the multiplication table of a number .





10. Explain the pillars of OOP in java

Object-Oriented Programming (OOP) is the foundation of Java, making code **modular**, **reusable**, and **organized**. OOP is built on **four key pillars**:

### **1. Encapsulation (Data Hiding)**

* Restricts direct access to class fields by **wrapping data and methods together**.
* Provides **controlled access** via getters/setters.
* Helps protect data from unintended modification.

### **Inheritance (Code Reusability)**

* Allows a class (child) to **inherit** properties & methods from another class (parent).
* Promotes **code reuse** and reduces redundancy.
* Uses extends keyword.

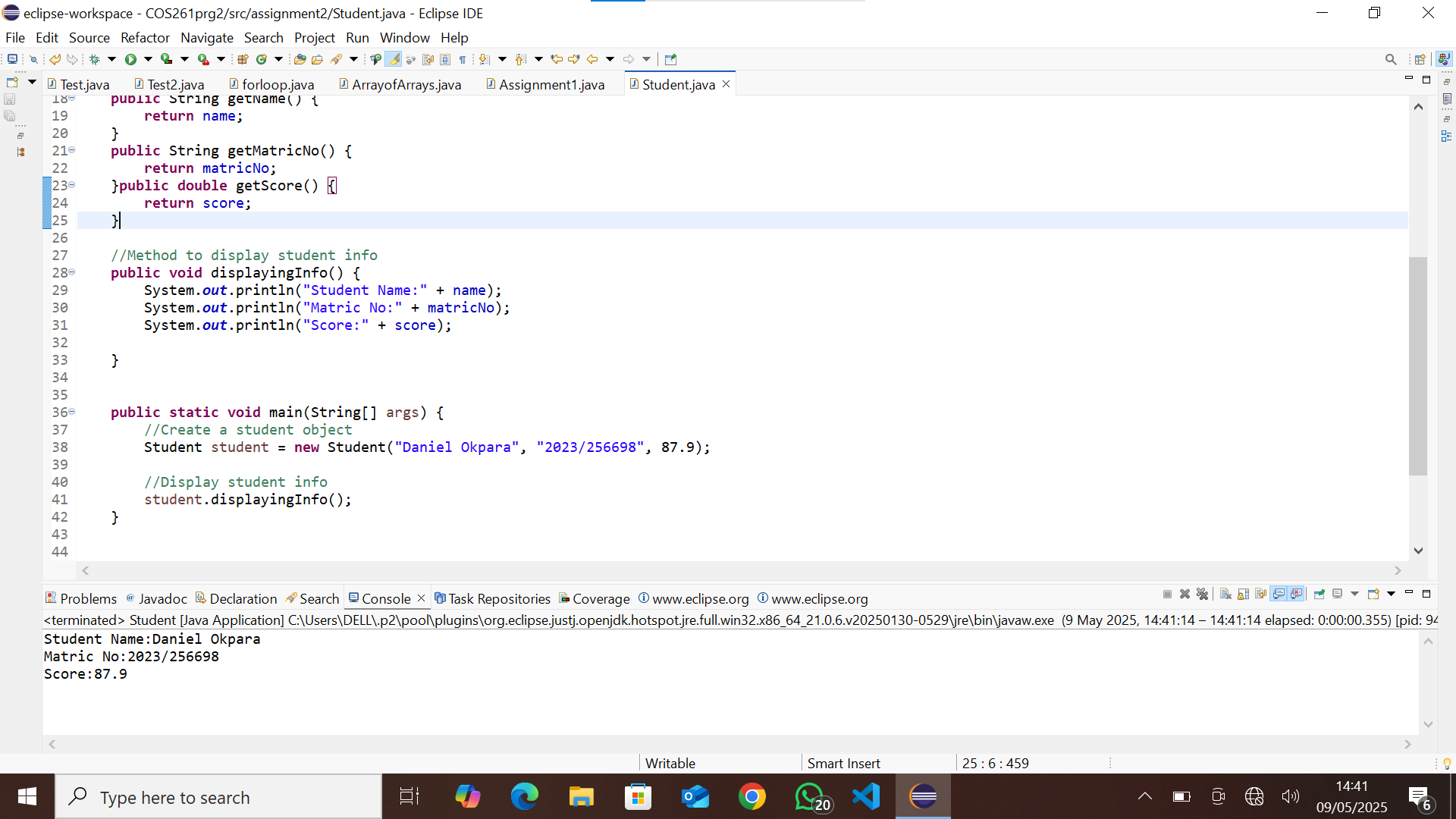
### **Polymorphism (Multiple Forms)**

* Allows **one interface to be used for different types**.
* Can be achieved via **method overriding** (changing inherited method behavior) or **method overloading** (same method name with different parameters).

### **Abstraction (Hiding Complexity)**

* Defines **essential details** while hiding implementation.
* Achieved via **abstract classes** or **interfaces**.
* Helps focus on **what** a class should do, rather than **how** it does it.

11. Create a class Student with properties name, matric no, and score and add methods to display the students info.



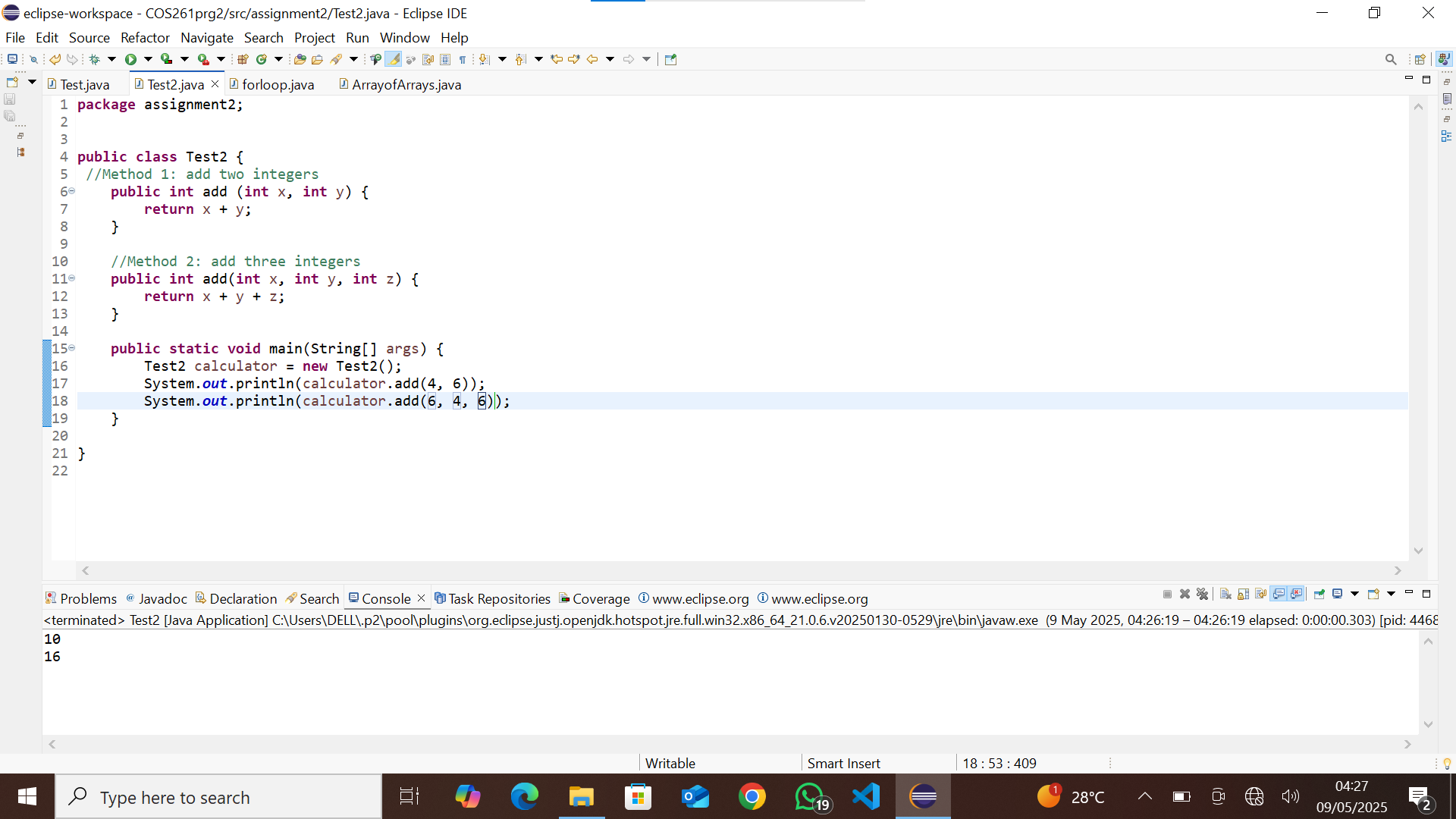
12. What is method over loading ?. Give a code example

Method **overloading** in Java allows multiple methods in the same class to share the same name but differ in their **parameter lists** (number, type, or order of parameters). It enables flexibility and cleaner code.

### **How It Works**

* Same method name, but **different parameters**.
* Helps avoid creating separate methods for similar operations.
* Compiler determines the correct method based on **arguments provided**.

### **Example of Method Overloading**



13. What is inheritance ?. Create a base class person and a subclass Teacher

Inheritance is one of the **four pillars of Object-Oriented Programming (OOP)** in Java. It allows one class (**child/subclass**) to **inherit** properties and behaviors from another class (**parent/superclass**), promoting **code reusability** and a hierarchical structure.

14.Writing **clean code** means producing **well-structured, readable, and maintainable** code that is easy to understand and modify. It follows best practices to improve efficiency and collaboration.

Practices that make a code clean and maintainable

1️⃣ **Use Meaningful Names** – Choose **clear and descriptive** names for variables, methods, and classes.

2️⃣ **Keep Methods Short & Focused** – Each method should have **one purpose** to enhance readability.

. 3️⃣ **Avoid Repetitive Code (DRY Principle)** – Use reusable functions instead of duplicating logic.

15. Writing very long methods in a Java program can lead to several problems that negatively impact **readability, maintainability, and performance**. Here’s why you should avoid them:

### **1️⃣ Reduced Readability**

* Long methods make code **harder to understand** and follow.
* Other developers (or your future self!) will struggle to grasp the logic quickly.
* A reader should be able to **scan a method** and immediately know what it does.

### **2️⃣ Difficult to Maintain & Debug**

* When a method does too much, debugging becomes complex.
* If errors occur, identifying the issue in a **massive block of code** is time-consuming.
* Small, focused methods make debugging **easy and efficient**.

### **3️⃣ Violation of the Single Responsibility Principle (SRP)**

* **Each method should have ONE purpose**.
* Long methods usually handle **multiple tasks**, making them harder to modify.

16. **1️⃣ Classes (PascalCase)**

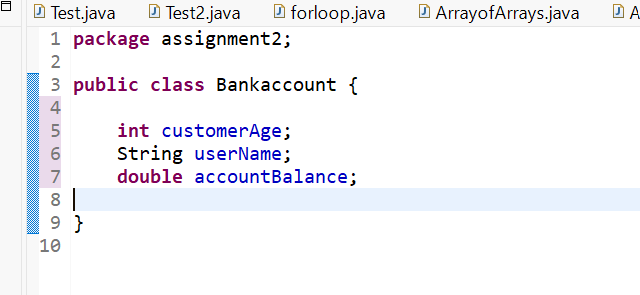
✔ Start class names with **uppercase letters** ✔ Use **PascalCase** (each word capitalized) ✔ Should be **nouns**, representing objects or entities

#### **✅ Example (Good Naming)**

### **2️⃣ Variables (camelCase)**

✔ Start variable names with **lowercase letters** ✔ Use **camelCase** (first word lowercase, next words capitalized) ✔ Should be **descriptive**

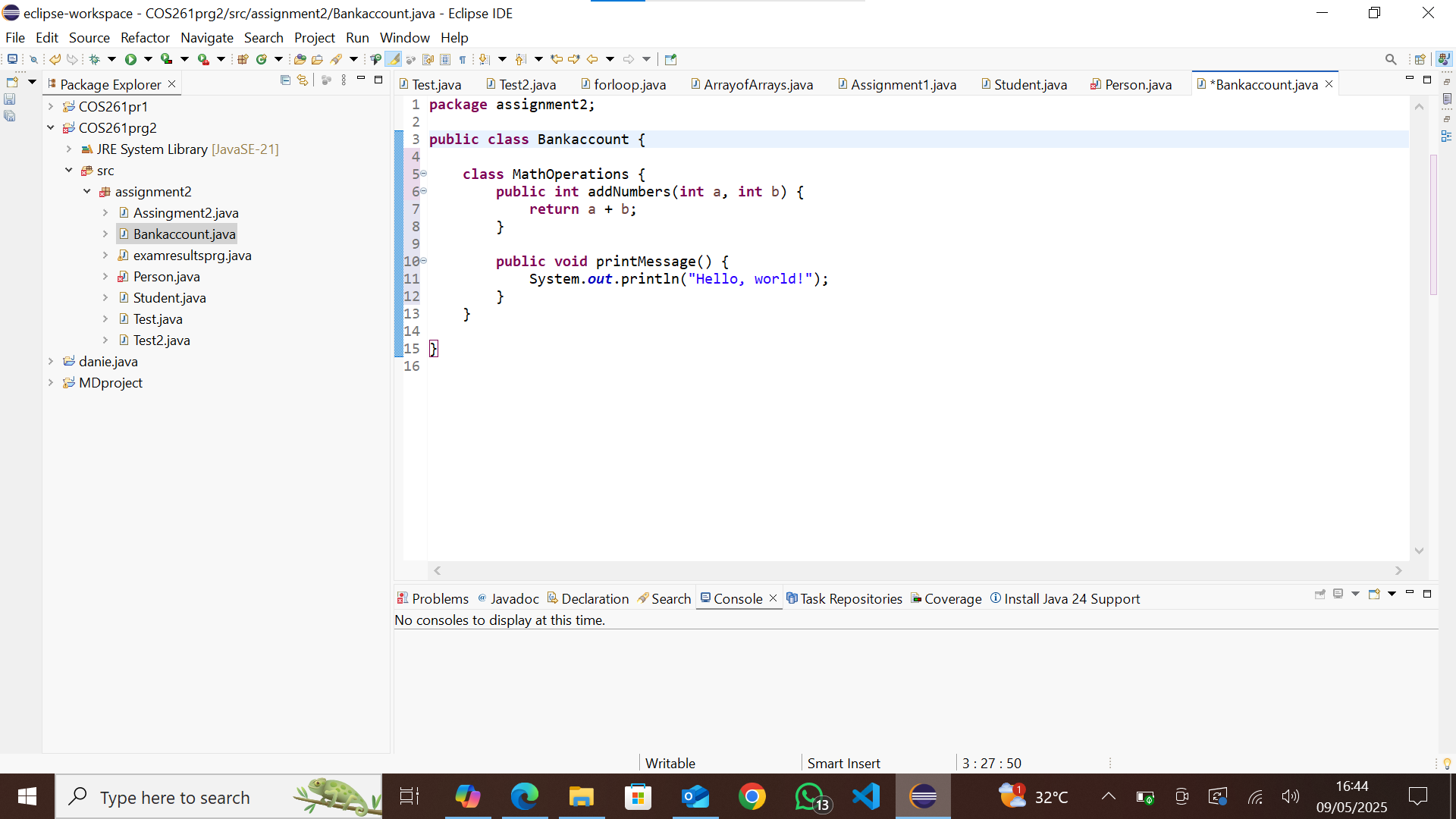
#### **✅ Example (Good Naming)**



### **3️⃣ Methods (camelCase)**

✔ Start method names with **lowercase letters** ✔ Use **camelCase** (same as variables) ✔ Should be **verbs**, describing an action

#### **✅ Example (Good Naming)**



17. Breaking your Java program into **methods** is crucial for writing clean, organized, and efficient code. Here’s why:

### **1️⃣ Improves Readability & Maintainability**

* Code becomes easier to **understand** when divided into logical blocks.
* Helps future developers (or yourself) quickly **grasp the logic** without digging through a giant method.

### **2️⃣ Promotes Code Reusability (DRY Principle)**

* **Avoids duplication** by reusing functions instead of rewriting logic.
* Saves time and effort when modifying or updating functionality.

### **3️⃣ Enhances Debugging & Testing**

* Small, isolated methods make it **easier to test and debug** errors.
* Unit testing becomes more efficient since you can **test individual components** separately.

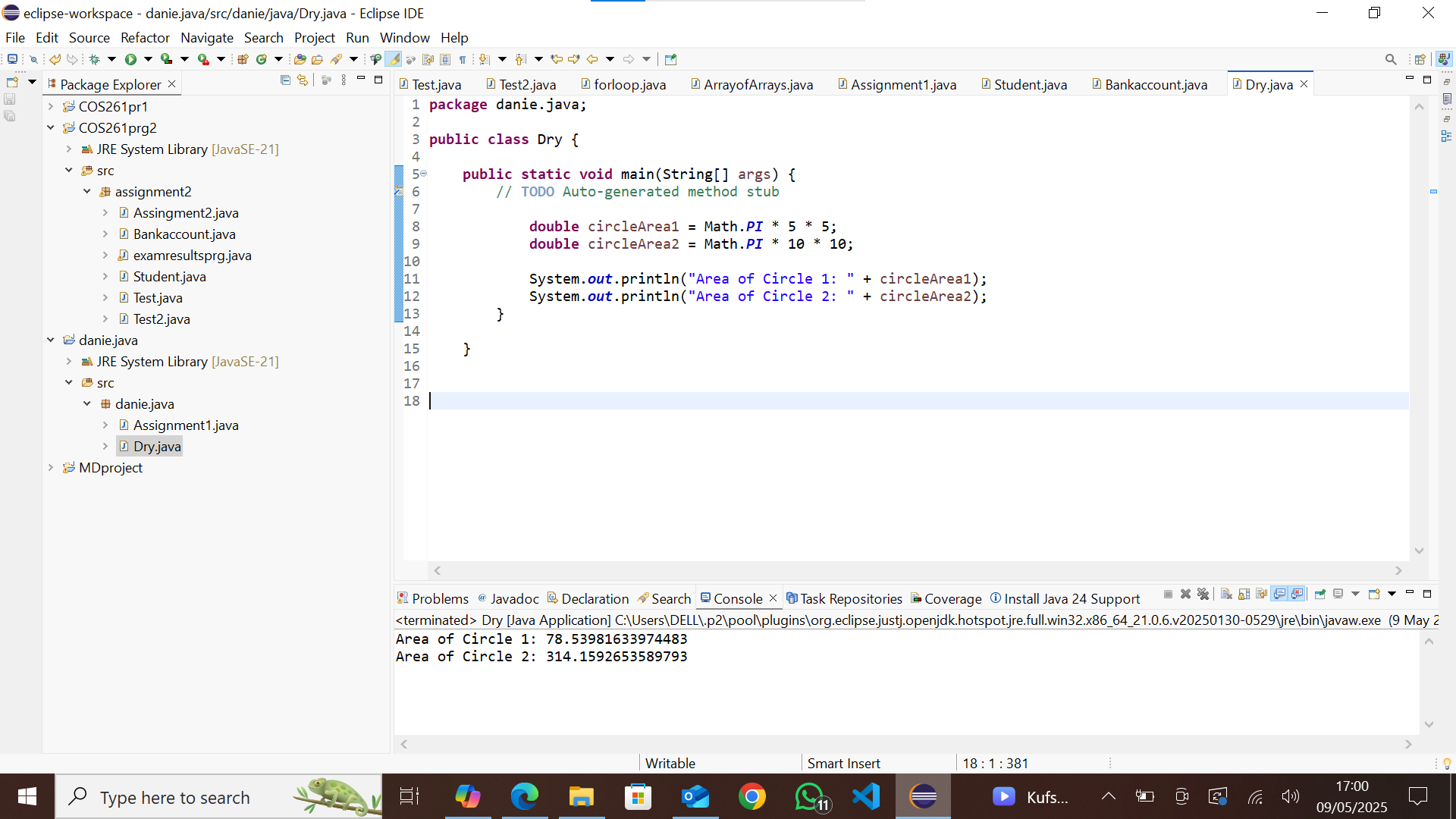
### **4️⃣ Supports Modularity & Scalability**

* Makes programs **modular**, allowing easy updates and feature additions.
* Example: If you build a banking system, separate methods for **deposit**, **withdraw**, and **transfer** make expanding the program much simpler.

18. The **DRY (Don't Repeat Yourself)** principle is a fundamental concept in software development that emphasizes reducing duplication in code. The goal is to improve maintainability, readability, and reusability by ensuring that any given piece of functionality is defined only once in a system.

### **Why DRY?**

* Reduces redundancy in code
* Makes code easier to maintain and update
* Improves readability and efficiency
* Helps avoid inconsistencies



19. Testing is crucial during program development because it helps ensure that software functions correctly, meets requirements, and remains reliable. Here are some key reasons why testing is important:

### **1. Detecting Bugs Early**

* Identifies errors before they cause major issues.
* Saves time and resources by catching problems early.

### **2. Ensuring Correct Functionality**

* Confirms that the program performs as expected.
* Helps developers verify that all features work correctly.

### **3. Improving Security**

* Identifies vulnerabilities that hackers could exploit.
* Helps prevent security breaches and data leaks.

20. Using **classes and objects** in programming, rather than writing all logic in the main method, offers several advantages, especially in **object-oriented programming (OOP)**. Here are some key benefits:

### **1. Encapsulation**

* Groups related data and methods together inside a class.
* Prevents unintended interference by restricting access to private data.
* Helps organize code in a logical and structured manner.

### **2. Reusability**

* Allows you to reuse existing classes without rewriting code.
* Promotes efficient coding by utilizing already-defined behaviors.

21. **1. Syntax Error**

* **Occurs when:** The code violates the language’s grammatical rules.
* **Detected when:** The compiler or interpreter processes the code.

### **2. Runtime Error**

* **Occurs when:** The code is syntactically correct but fails during execution.
* **Detected when:** The program runs and encounters an issue (like dividing by zero).
* **Example in Java:**

### **3. Logic Error**

* **Occurs when:** The program runs but does not behave as intended.
* **Detected when:** The output is incorrect despite no syntax or runtime errors.

22. To test a method that calculates the average of five numbers, you would typically use **unit testing** to ensure that the method behaves correctly under various scenarios. In **Java**, you can use **JUnit** to write test cases. Here’s how you can do it:

Step 1: Implement the Method

Step 2: Write Unit Tests

Step 3: Run the Tests:

Save the test file inside the test directory of your project.

Run JUnit from your IDE or using Maven/Gradle.

Check if all tests **pass**. If any **fail**, review and fix the method.

23. **1. Improves Code Readability**

* Helps others (and your future self) understand complex logic.
* Makes it easier to follow the flow of the program.

### **2. Assists in Debugging**

* Explains why certain logic exists, helping identify issues quickly.
* Prevents confusion when modifying or debugging code later.

### **3. Facilitates Collaboration**

* Helps teams work together by making code more understandable.
* Avoids wasting time figuring out unfamiliar sections.

### **4. Documents Important Information**

* Provides insight into the purpose and expected behavior of methods.
* Helps new developers onboard faster.

### **5. Helps with Maintenance & Updates**

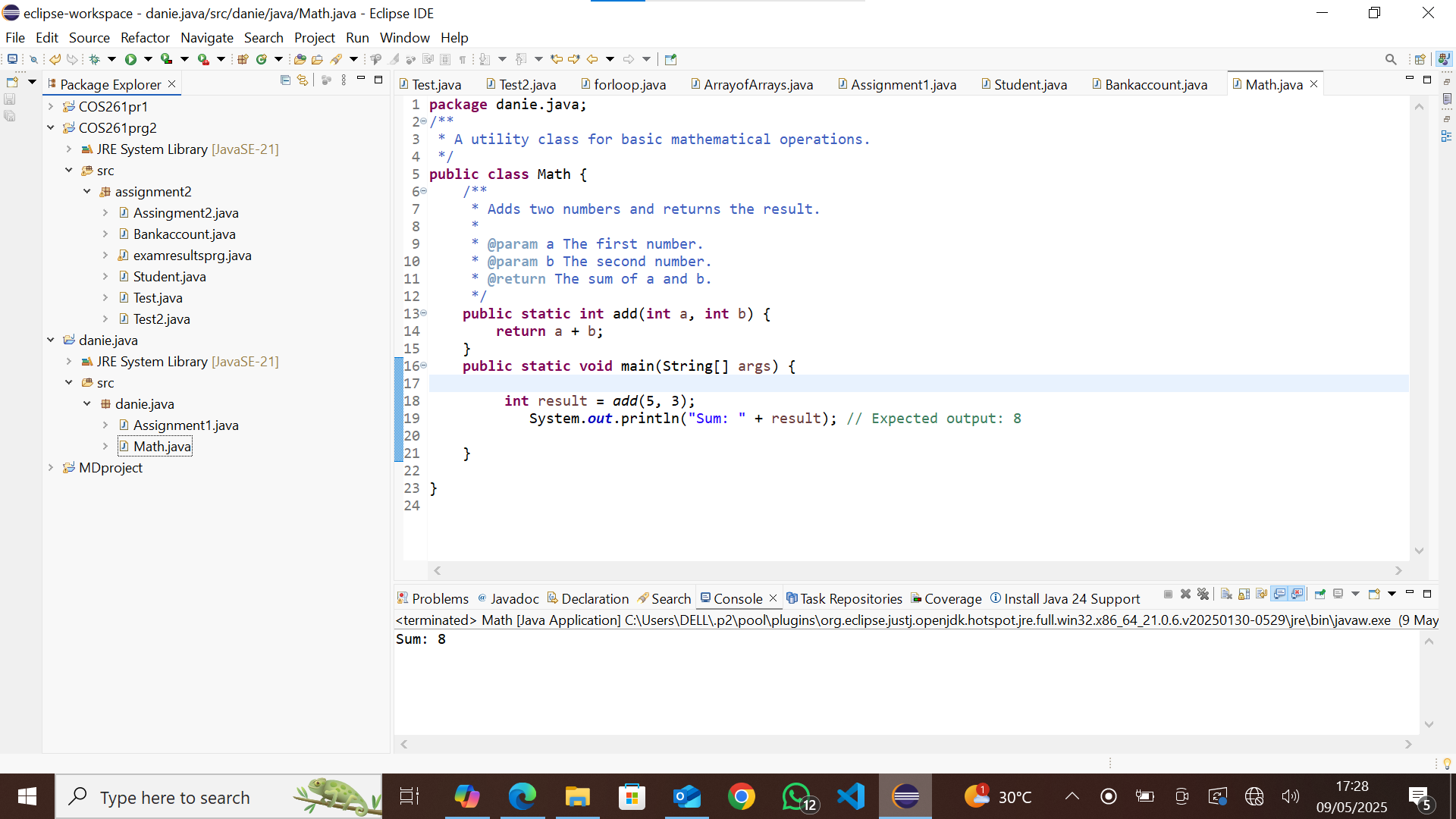
* Makes future modifications easier by clarifying functionality.
* Reduces errors when adjusting old code

24. **1. Javadoc Comments (/\*\* ... \*/)**

* Used to create **automatic documentation** for Java code.
* Typically placed **above** classes, methods, and fields to describe their purpose.
* Supports **special annotations** like @param, @return, and @author.
* Can be processed using the javadoc tool to generate HTML documentation.

### **2. Regular Comments (// and /\* ... \*/)**

* Used **within the code** for explanation.
* Does **not** generate documentation.
* Helps developers understand complex logic.

25. 

26. **Version control** is a system that tracks changes to files over time, allowing multiple people to collaborate, manage updates, and revert to previous versions if necessary. It is **especially crucial in team projects** because it helps developers work together efficiently without overwriting each other's work.

### **Why Version Control is Important in Team Projects**

1. **Tracks Changes Over Time**
   * Keeps a history of modifications, making it easy to review past versions.
   * Allows rollback to previous versions if mistakes occur.
2. **Enables Collaboration**
   * Multiple team members can work on the same project without conflicts.
   * Helps merge contributions from different developers seamlessly.

27. **How to Explain Code Refactoring Simply**

Imagine you wrote code that works but is **hard to read, modify, or maintain**. Over time, messy code can lead to bugs, slow development, and frustration. **Refactoring** helps by:

* Removing unnecessary complexity.
* Simplifying logic while keeping the behavior intact.
* Making the code more **understandable, reusable, and efficient**.

28. **1. Git & GitHub/GitLab/Bitbucket – Version Control**

Git is essential for tracking changes and collaborating with teams. Developers use GitHub, GitLab, or Bitbucket for hosting repositories.

### **2. Maven & Gradle – Build Automation**

Maven and Gradle help manage dependencies, compile Java code, and automate builds.

### **3. Docker – Containerization**

Docker enables teams to package applications into lightweight containers for easy deployment.

29. 1. Follow Object-Oriented Principles (OOP)

2. Use Meaningful Variable and Method Names

3. Handle Exceptions Properly

4. Optimize Memory Management

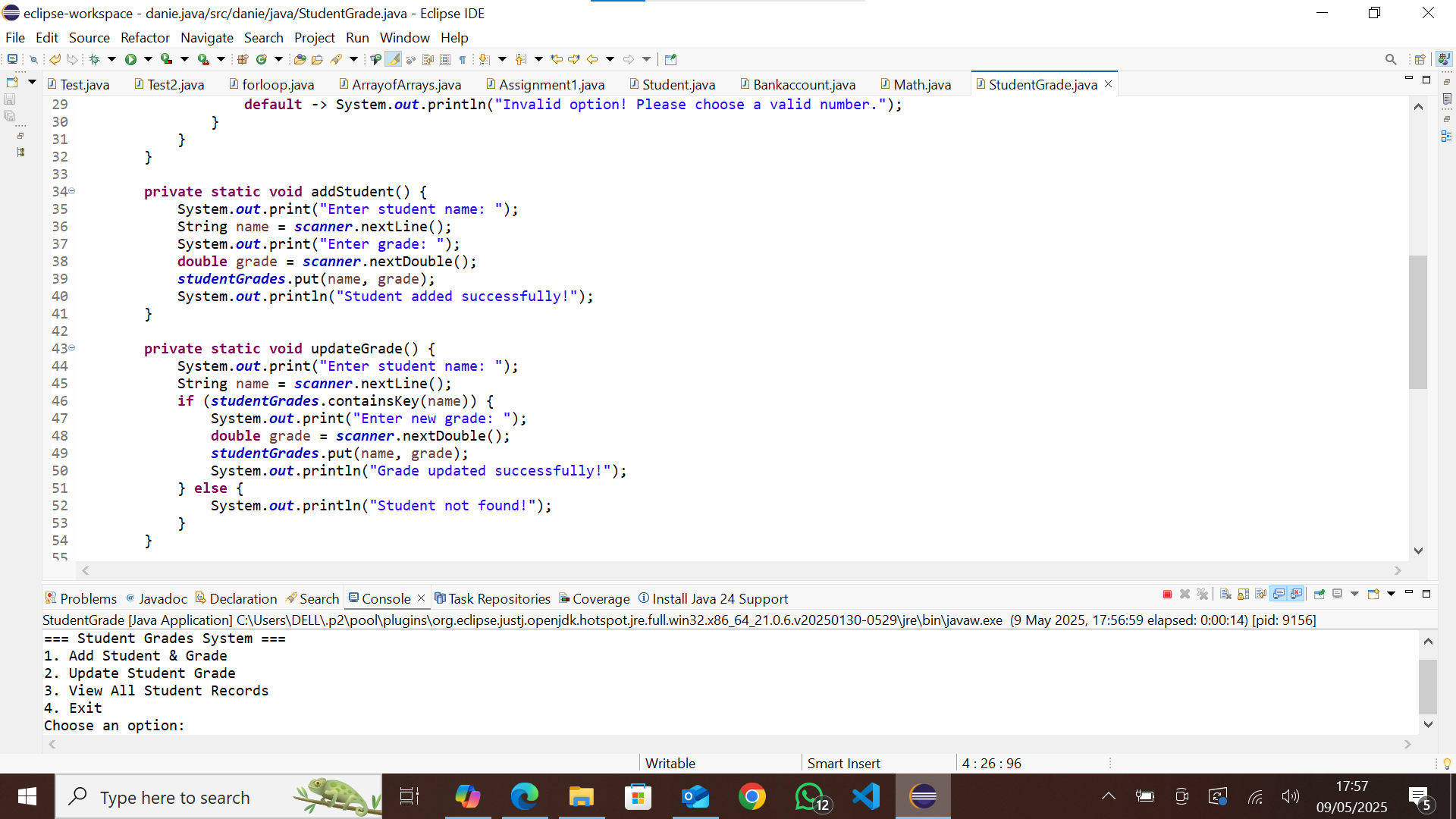
5. Write Unit Tests

30. Code readability is the ease with which a developer can understand, navigate, and modify code. Readable code follows consistent formatting, uses meaningful names, avoids unnecessary complexity, and includes comments or documentation where needed.

### **Why is Readability More Important Than "Smart" Code?**

"Smart" code often refers to overly clever, compact, or complex solutions that might be impressive but difficult to understand. Readability is more important because:

1. **Maintains Clarity** – Simple, well-structured code is easier to debug and extend.
2. **Improves Collaboration** – Other developers (or even future you) can quickly grasp readable code.
3. **Reduces Bugs** – Clear logic minimizes misunderstandings that lead to errors.
4. **Eases Maintenance** – Readable code ensures long-term maintainability without unnecessary complexity.
5. **Enhances Performance & Scalability** – While "smart" code may look elegant, if it’s hard to read, it can become a bottleneck for improvement.

31. 

32. 