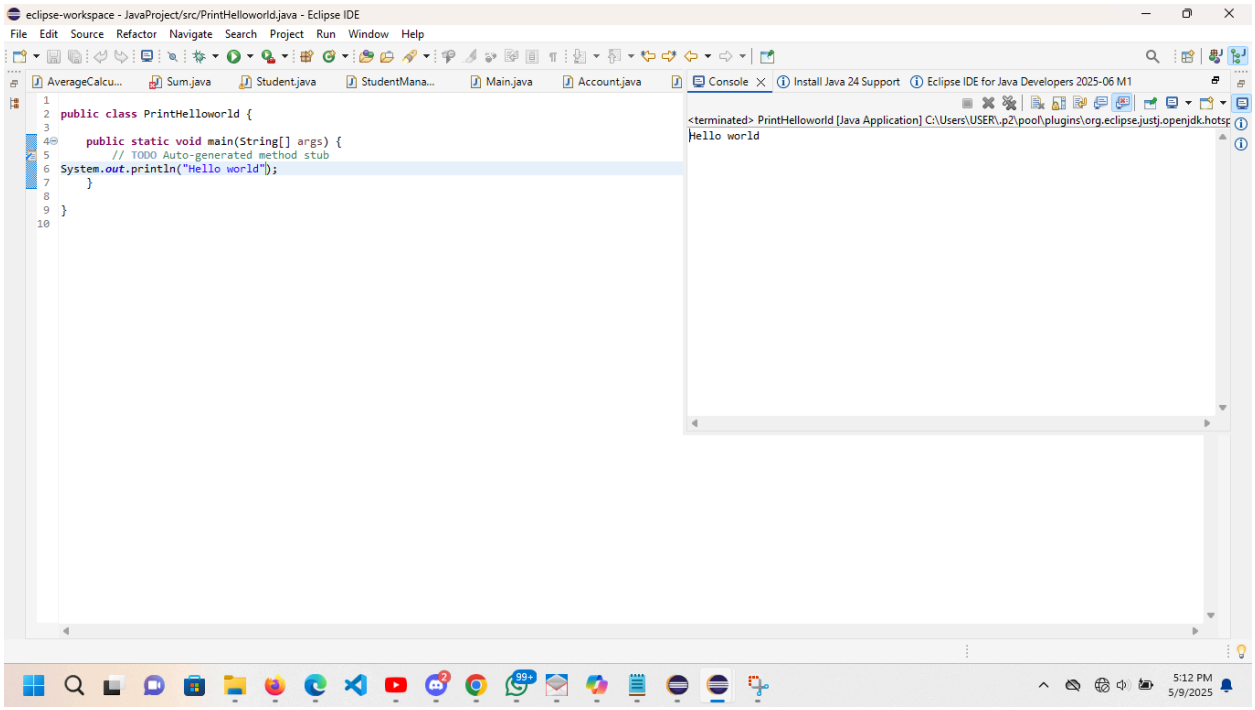
JavaAssignments

1)



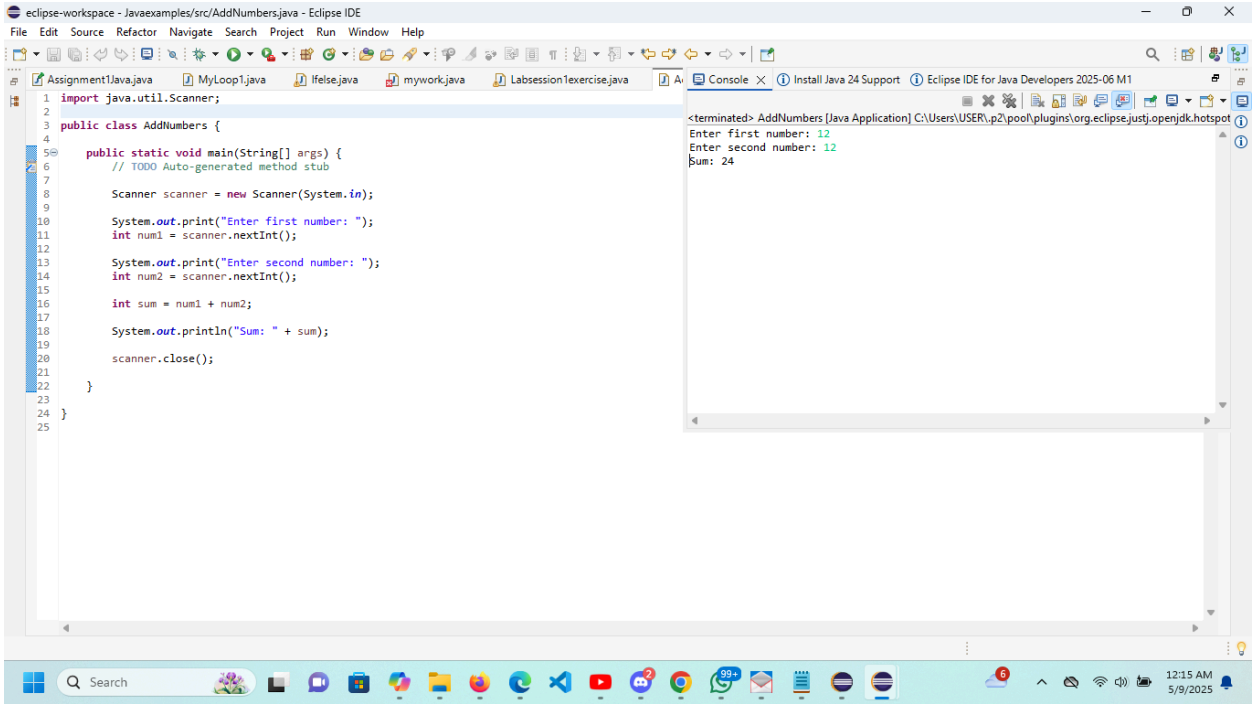
2) Difference between == and .equals() in Java

In Java:

- == is used for reference comparison, meaning it checks if two variables point to the same memory location.
- .equals() is used for content comparison, meaning it checks if two objects have the same values.

3) What is the use of the main method in Java?

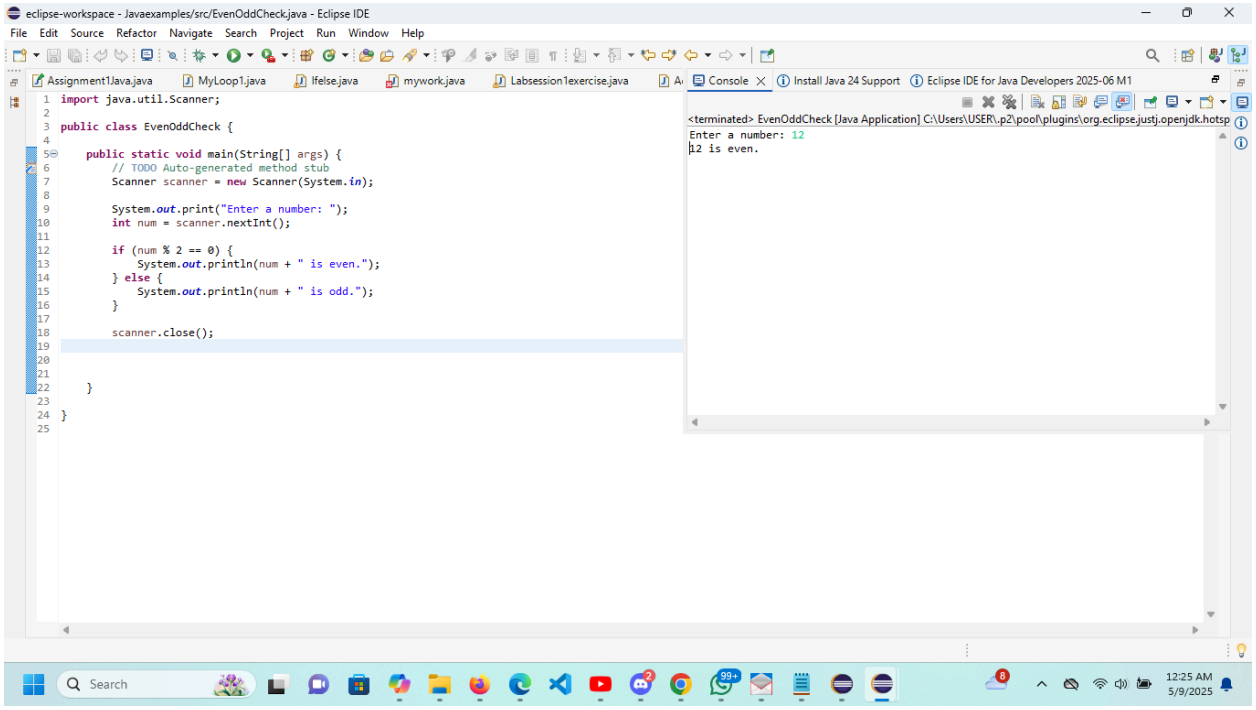
The main method is the entry point of any Java application. It allows the JVM (Java Virtual Machine) to start program execution.



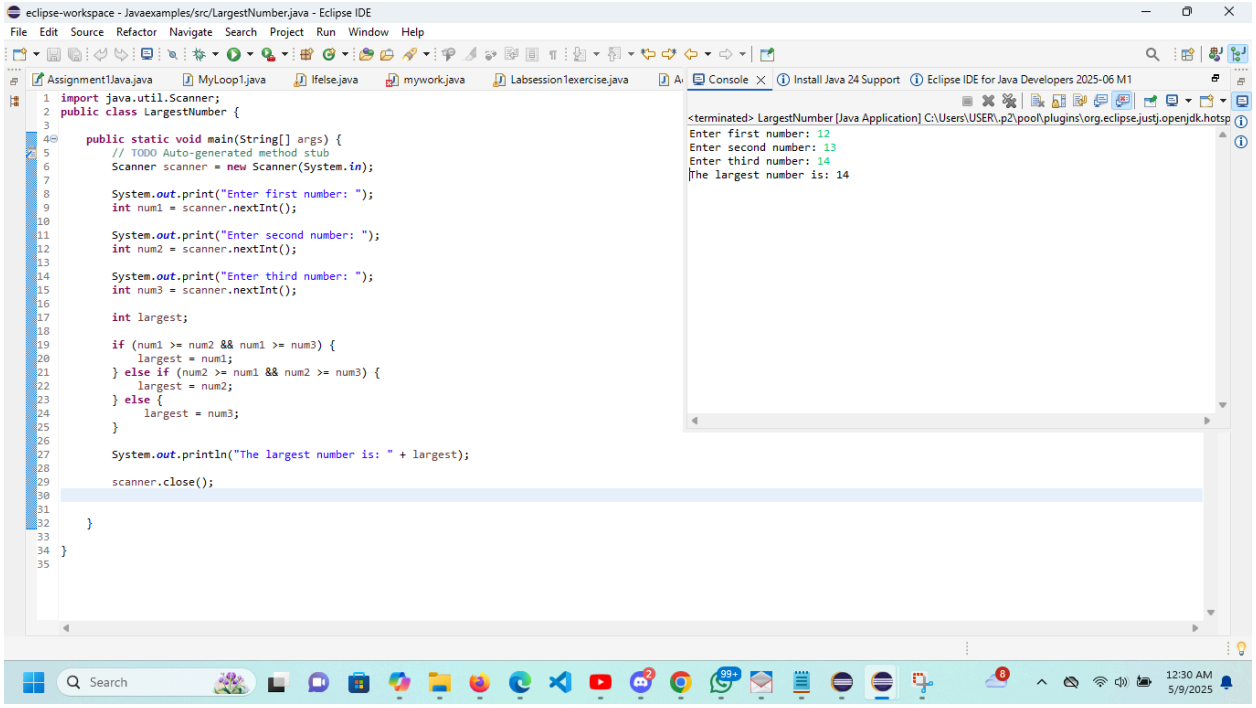
5) Difference between int, Integer, and String in Java

- int: A primitive data type that stores numerical values (e.g., 5).
- Integer: A wrapper class for int, providing useful methods (e.g., Integer.parseInt()).
- String: A sequence of characters, used for text data (e.g., "Hello")

6) The program for checking if the a number is even or not



7)the program to find the largest among three number



.

8) Difference between while, for, and do-while loops in Java

Loops help execute a block of code multiple times. Here's how they differ:

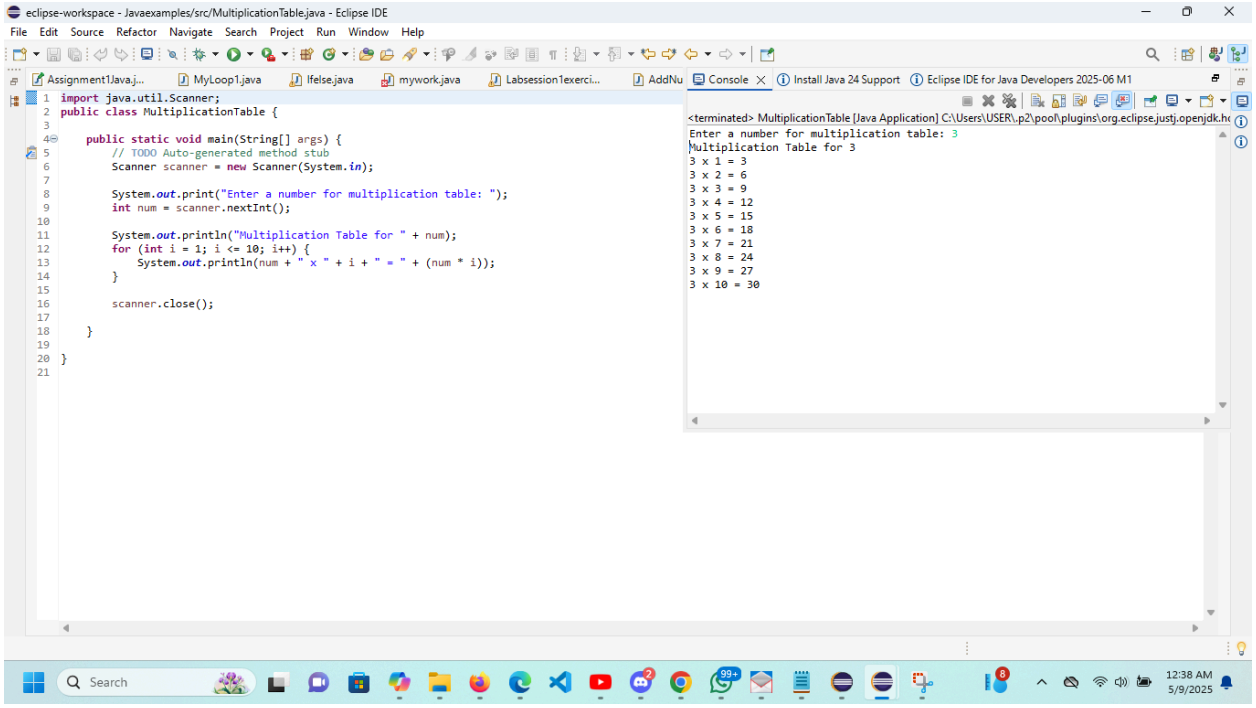
| Loop Type | Description | Example Usage |

| while loop | Runs while a condition is true. If the condition is false from the start, it won't execute. | Checking user input, infinite loops |

| for loop | Best for known number of iterations (e.g., counting, arrays). | Looping through arrays, fixed iterations |

| do-while loop | Always runs at least once, even if the condition is false. | Ensuring an action occurs at least once |

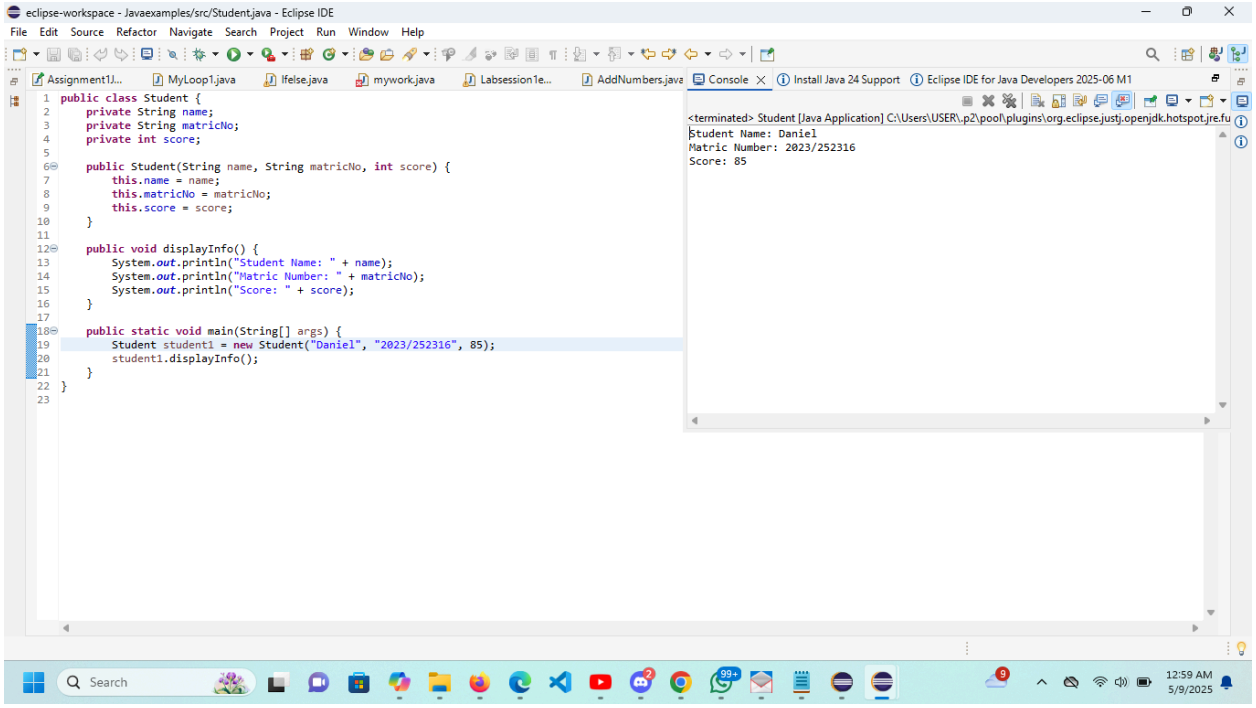
9)the code for the multiplication table of any number



10. The Four Pillars of Object-Oriented Programming (OOP) in Java

- Encapsulation – Hiding the internal details of a class and exposing only the necessary functionalities through public methods.
- Inheritance – Allowing one class to derive properties and behaviors from another, enabling code reuse.
- Polymorphism – Enabling methods or objects to take multiple forms, such as method overloading or overriding.
- Abstraction – Hiding complex implementation details and providing a simplified interface.

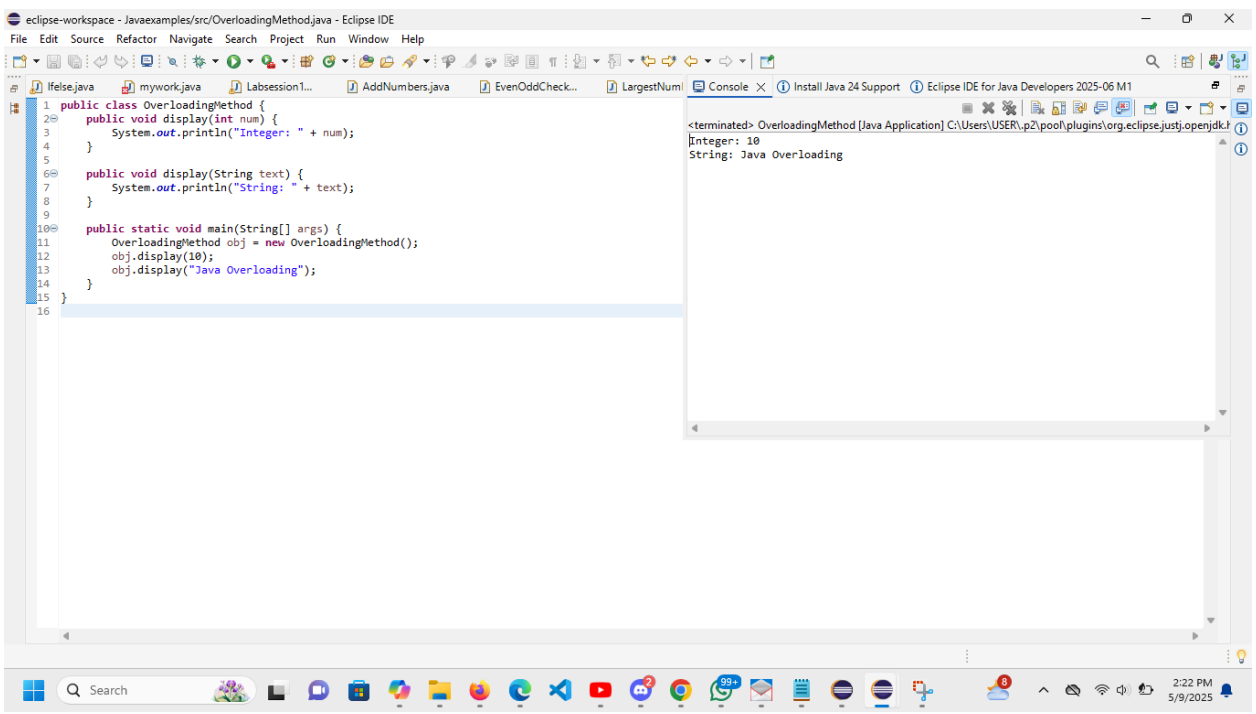
11)the code for class of student with it properties name,marticNo,and score.




```java
public class Student {
    private String name;
    private String matricNo;
    private int score;

    public Student(String name, String matricNo, int score) {
        this.name = name;
        this.matricNo = matricNo;
        this.score = score;
    }

    public void displayInfo() {
        System.out.println("Student Name: " + name);
        System.out.println("Matric Number: " + matricNo);
        System.out.println("Score: " + score);
    }

    public static void main(String[] args) {
        Student student1 = new Student("Daniel", "2023/252316", 85);
        student1.displayInfo();
    }
}
```

12. What is Method Overloading?

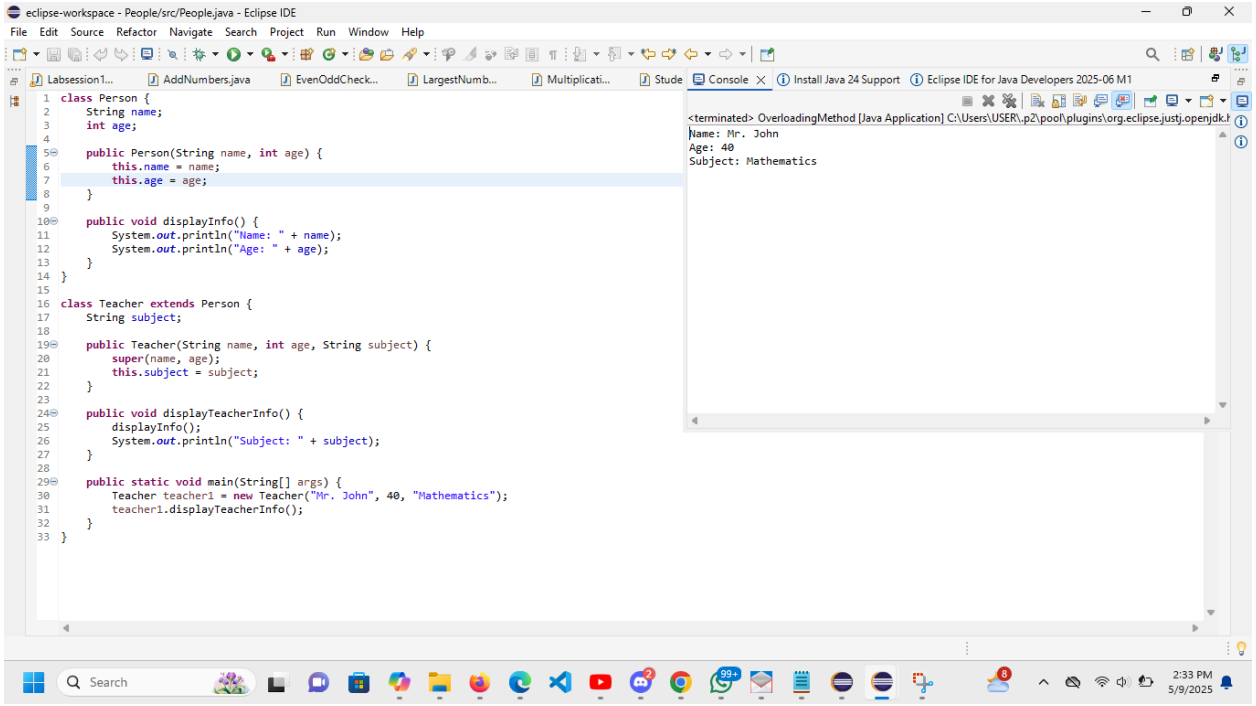
Method overloading allows multiple methods in a class to have the same name but different parameters.




```java
public class OverloadingMethod {
    public void display(int num) {
        System.out.println("Integer: " + num);
    }

    public void display(String text) {
        System.out.println("String: " + text);
    }

    public static void main(String[] args) {
        OverloadingMethod obj = new OverloadingMethod();
        obj.display(10);
        obj.display("Java Overloading");
    }
}
```

13. What is Inheritance? (Base class Person and Subclass Teacher)

Inheritance allows a subclass to inherit attributes and methods from a superclass.



14. What Does It Mean to Write “Clean Code”?

Clean code makes software easier to read, maintain, and scale. Three key practices:

- Use meaningful variable & method names – Instead of x = 5;, write studentAge = 5;.
- Follow proper indentation & formatting – Code should be visually structured for easy readability.
- Write modular & reusable functions – Avoid duplicating code; use methods to improve maintainability.

15. Why Should You Avoid Writing Very Long Methods?

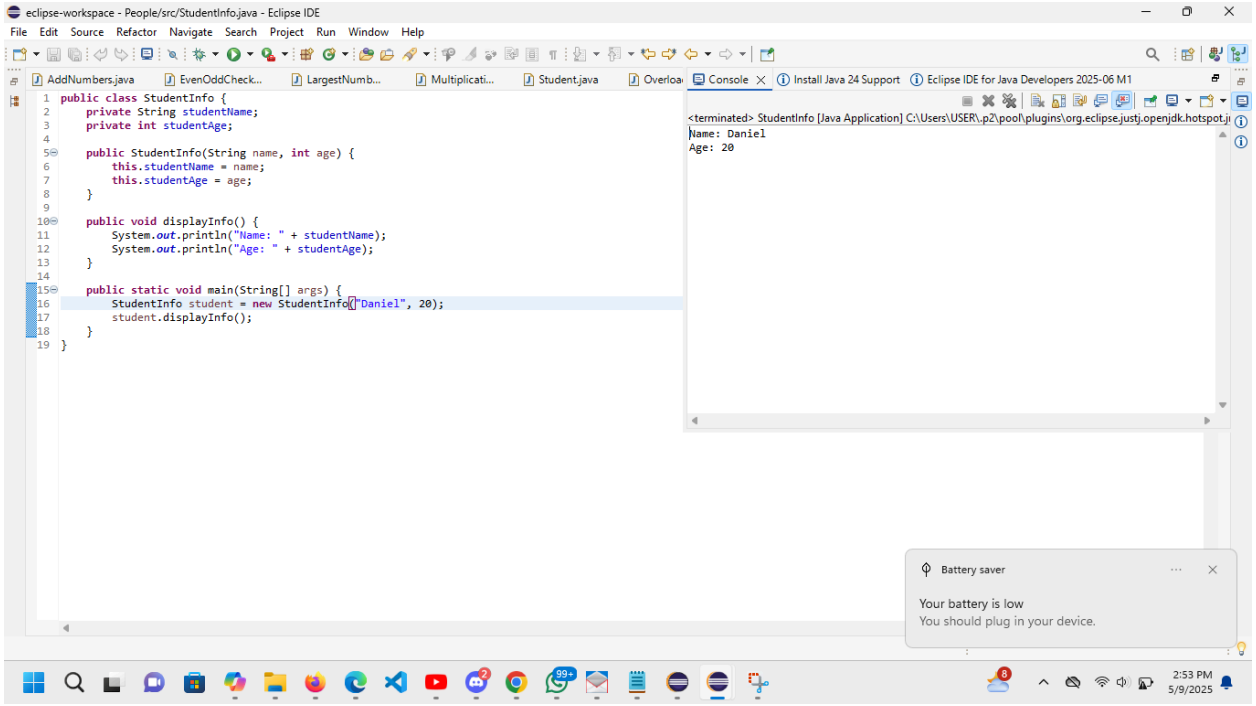
Long methods can:

- Reduce readability – Harder for other developers to understand the logic.
- Increase complexity – Debugging becomes challenging.
- Limit reusability – Difficult to reuse code blocks effectively.

16. Java Naming Conventions for Classes, Variables, and Methods

Following proper naming conventions improves readability and maintainability in Java.

Element	Convention	Example
Classes	Capitalized first letter and use CamelCase.	class StudentInfo {}
Variables	Start with lowercase, use CamelCase.	int studentAge = 20;
Methods	Start with lowercase, use verbs to describe action.	public void displayInfo() {}



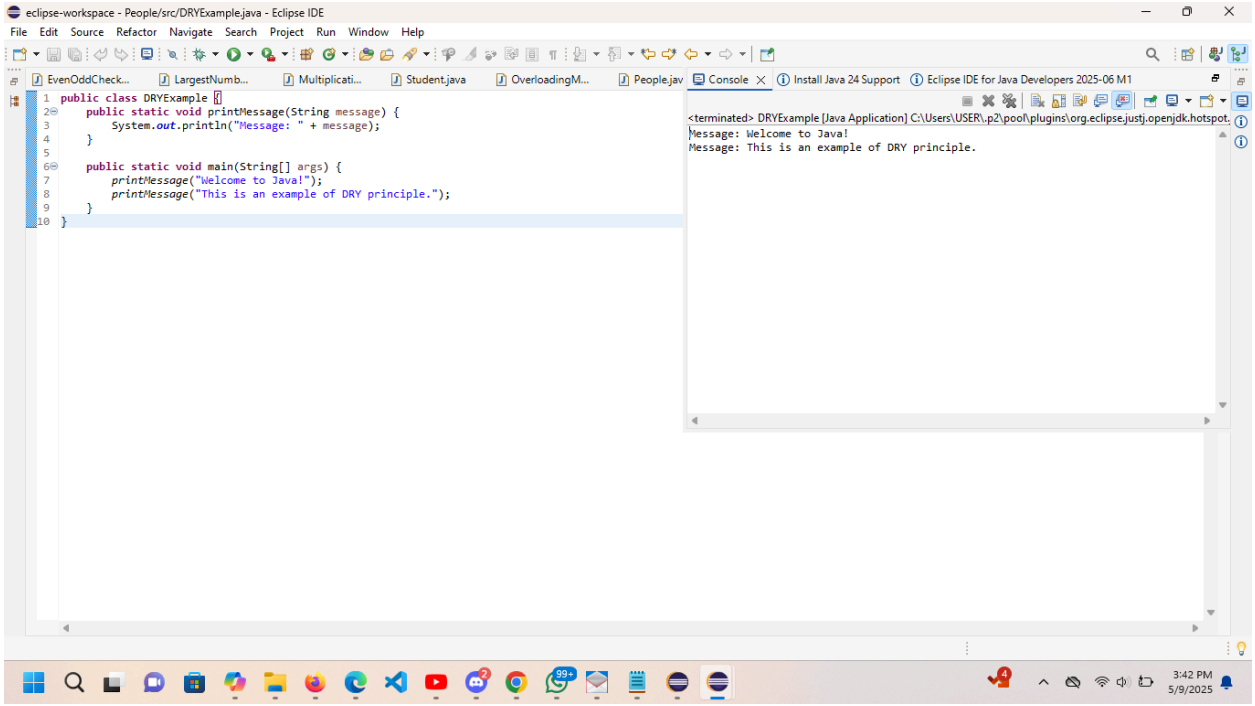
17. Why Break a Java Program into Methods?

Methods help:

- Improve readability – Code becomes easier to understand.
- Enhance reusability – Avoid duplicate code by calling methods multiple times.
- Simplify debugging – Errors are isolated within specific functions.
- Encourage modular programming – Each function handles a specific task.

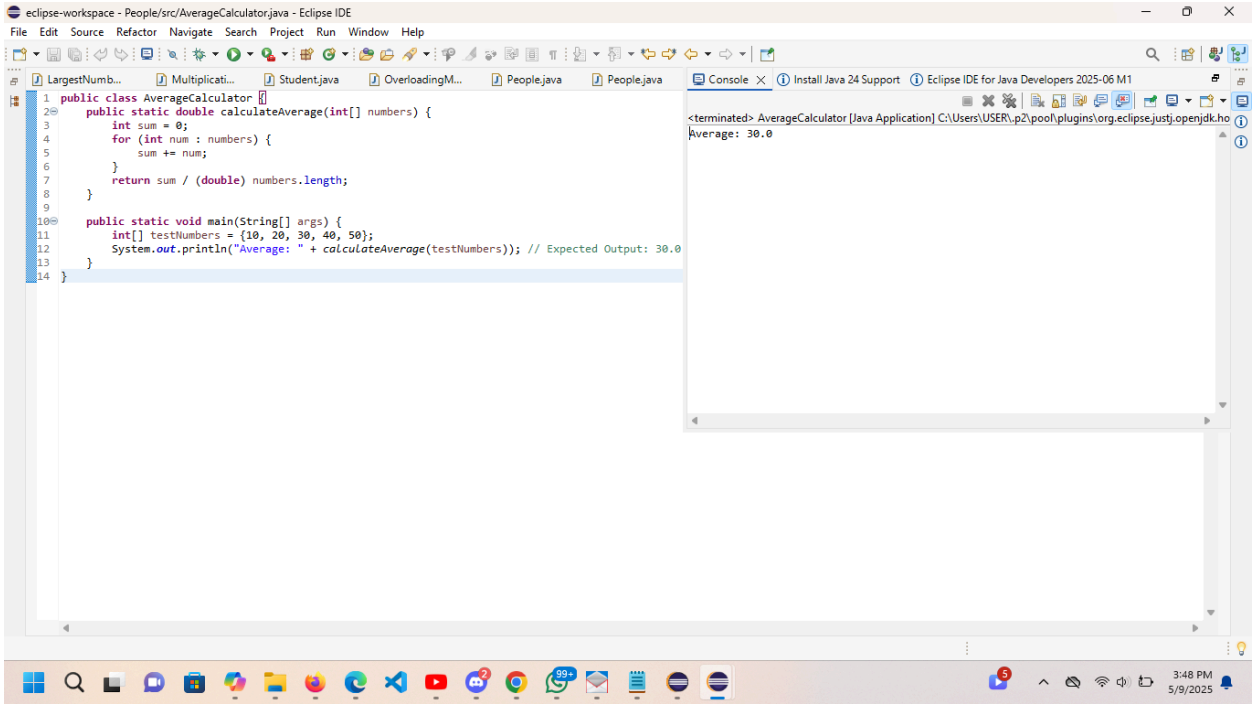
18. DRY (Don't Repeat Yourself) in Java

The DRY principle ensures that logic isn't repeated in multiple places. Instead of duplicating code, use reusable methods.



19. Benefits of Using Classes and Objects
Using classes and objects instead of writing everything in the main method:
- Encapsulation – Keeps data private and controlled.
- Code reusability – Objects can be reused across multiple classes.
- Maintainability – Code is easier to update and modify.

20. Why is Testing Important?
Testing ensures:
- Bug-free functionality
- Expected performance
- Valid user input handling
- Security and stability
21. Types of Errors in Java
Error Type	Description	Example
Syntax Error	Compiler detects invalid syntax.	Missing semicolon (int x = 10)
Runtime Error	Program crashes during execution.	Division by zero (int y = 10 / 0;)
Logic Error	Incorrect output but no crash.	Wrong calculation formula
22)Method that calculates the averages of five numbers




```
public class AverageCalculator {
    public static double calculateAverage(int[] numbers) {
        int sum = 0;
        for (int num : numbers) {
            sum += num;
        }
        return sum / (double) numbers.length;
    }

    public static void main(String[] args) {
        int[] testNumbers = {10, 20, 30, 40, 50};
        System.out.println("Average: " + calculateAverage(testNumbers)); // Expected Output: 30.0
    }
}
```

23. Why Write Comments in Java?
Comments help:
- Explain complex logic
- Guide future developers
- Simplify debugging

24. JavaDoc vs Regular Comments

Type	Description	Example
Single-line comment	Used for brief notes.	// This calculates sum
Multi-line comment	Used for longer explanations.	/* This method does XYZ */
JavaDoc	Used for automatic documentation.	/** This method returns student info */

25)
```
/**
 * Calculates the sum of two numbers.
 * @param a First number
 * @param b Second number
 * @return Sum of a and b
 */
public int sum(int a, int b) {
    return a + b;
}
```

26. Version Control Importance
Version control helps teams:
- Track changes

- Collaborate effectively
- Manage rollbacks (undo changes)

Popular tools: Git, GitHub, Bitbucket

27. Explaining Code Refactoring
Refactoring improves existing code without changing its functionality. Example: simplifying a large method into smaller ones.

28. Java Collaboration Tools
Tools like:
- GitHub – Code versioning & repository hosting.
- JIRA – Project management.
- Slack – Team communication.

26. Version Control Importance
Version control helps teams:
- Track changes
- Collaborate effectively
- Manage rollbacks (undo changes)

Popular tools: Git, GitHub, Bitbucket

27. Explaining Code Refactoring
Refactoring improves existing code without changing its functionality. Example: simplifying a large method into smaller ones.

28. Java Collaboration Tools
Tools like:
- GitHub – Code versioning & repository hosting.
- JIRA – Project management.
- Slack – Team communication.

29. 5 Best Practices in Java
- Follow naming conventions
- Use proper indentation
- Write modular code
- Handle exceptions properly
- Optimize performance

30. Why Code Readability Matters More Than "Smart" Code
Readable code:

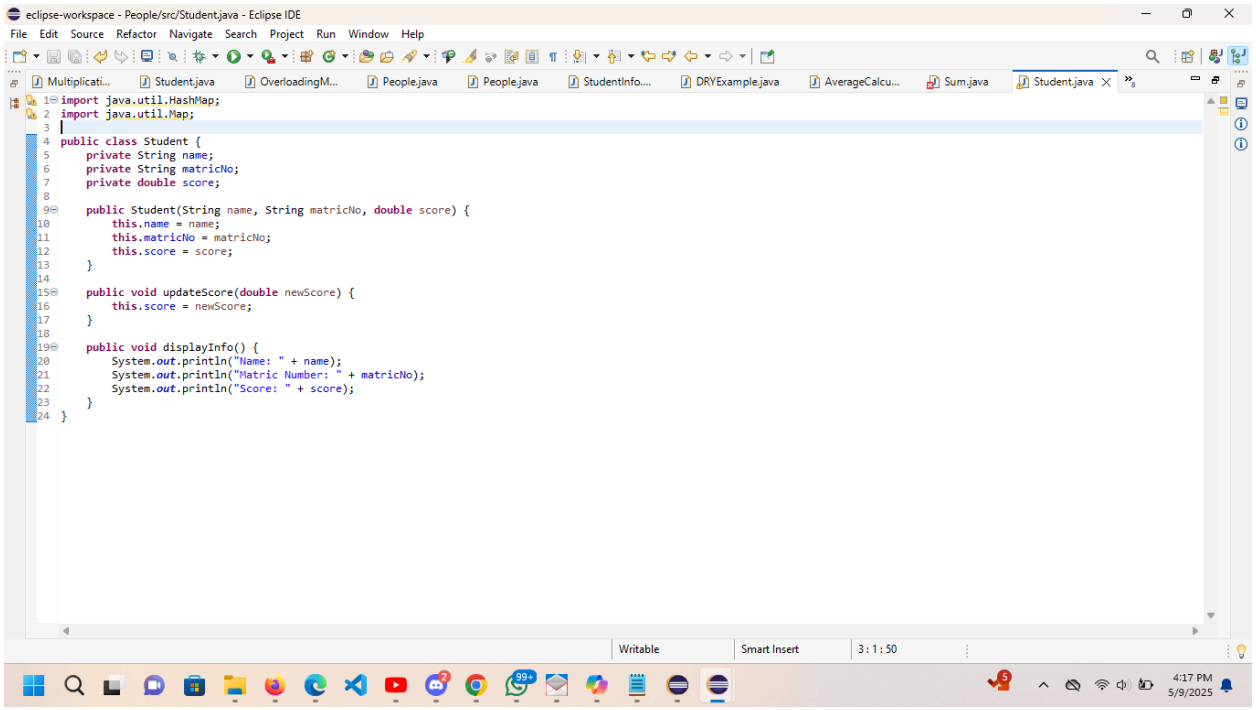
- Is easier to maintain
- Reduces debugging time
- Enhances teamwork

31)Mini Project – Student Grades Management System
Objective:
A simple command-line program to:
- Add student grades

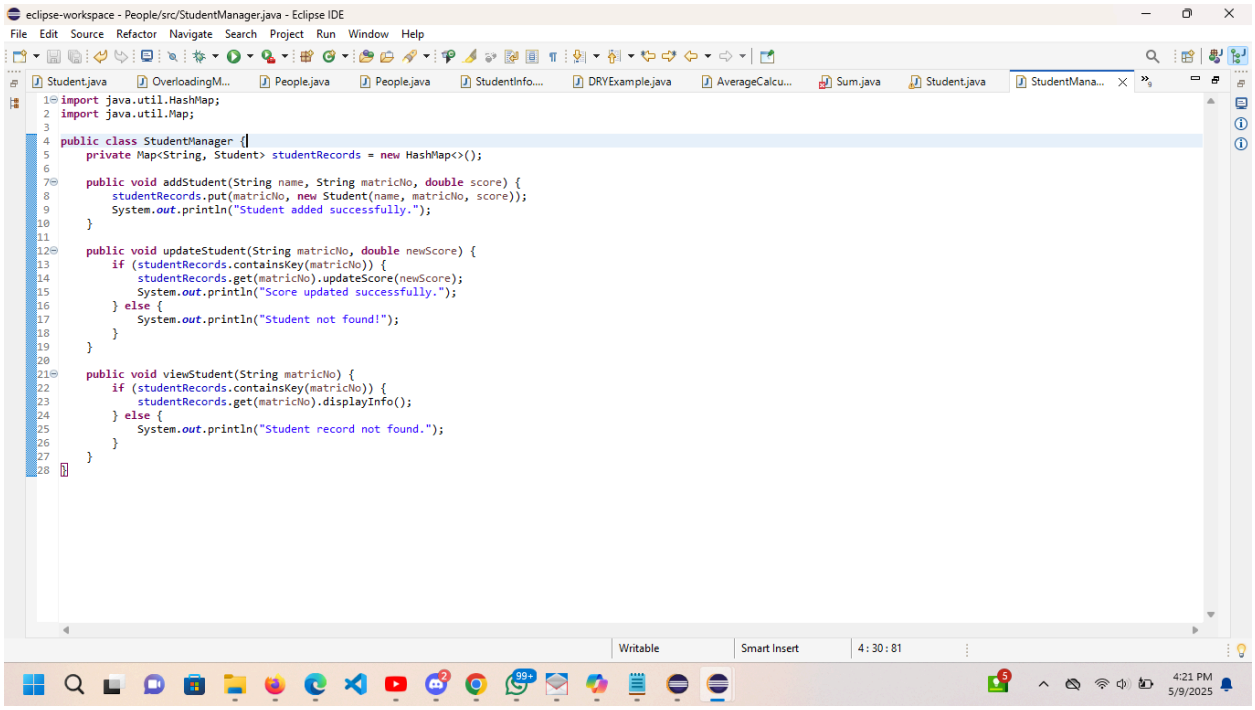
- Update existing records

- View student details

Code Structure:
- A Student class to represent students

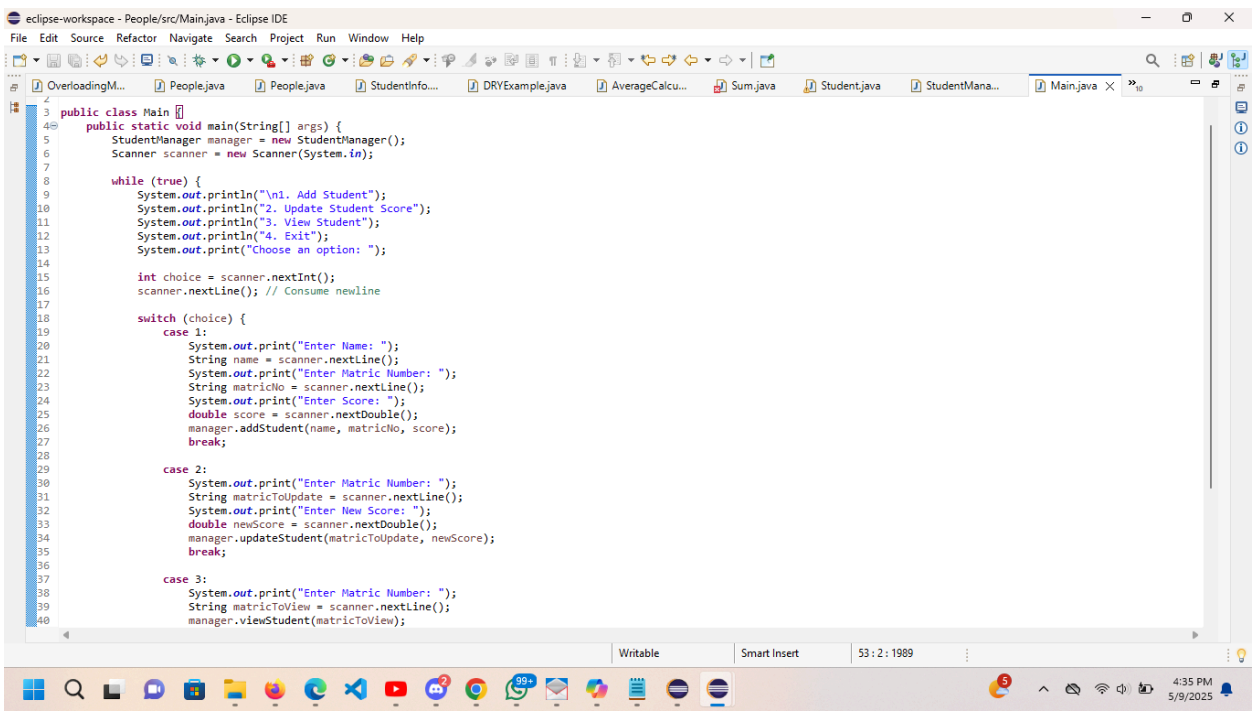


- A StudentManager class to handle records




```java
import java.util.HashMap;
import java.util.Map;

public class StudentManager {
    private Map<String, Student> studentRecords = new HashMap<>();

    public void addStudent(String name, String matricNo, double score) {
        studentRecords.put(matricNo, new Student(name, matricNo, score));
        System.out.println("Student added successfully.");
    }

    public void updateStudent(String matricNo, double newScore) {
        if (studentRecords.containsKey(matricNo)) {
            studentRecords.get(matricNo).updateScore(newScore);
            System.out.println("Score updated successfully.");
        } else {
            System.out.println("Student not found!");
        }
    }

    public void viewStudent(String matricNo) {
        if (studentRecords.containsKey(matricNo)) {
            studentRecords.get(matricNo).displayInfo();
        } else {
            System.out.println("Student record not found.");
        }
    }
}
```

- A Main class to interact with users




```java
public class Main {
    public static void main(String[] args) {
        StudentManager manager = new StudentManager();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\n1. Add Student");
            System.out.println("2. Update Student Score");
            System.out.println("3. View Student");
            System.out.println("4. Exit");
            System.out.print("Choose an option: ");

            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    System.out.print("Enter Name: ");
                    String name = scanner.nextLine();
                    System.out.print("Enter Matric Number: ");
                    String matricNo = scanner.nextLine();
                    System.out.print("Enter Score: ");
                    double score = scanner.nextDouble();
                    manager.addStudent(name, matricNo, score);
                    break;

                case 2:
                    System.out.print("Enter Matric Number: ");
                    String matricToUpdate = scanner.nextLine();
                    System.out.print("Enter New Score: ");
                    double newScore = scanner.nextDouble();
                    manager.updateStudent(matricToUpdate, newScore);
                    break;

                case 3:
                    System.out.print("Enter Matric Number: ");
                    String matricToView = scanner.nextLine();
                    manager.viewStudent(matricToView);
```

```
1. Add Student
2. Update Student Score
3. View Student
4. Exit
Choose an option: 1
Enter Name: Daniel
Enter Matric Number: 2023/252316




Enter Score: 85
Student added successfully.

Choose an option: 3
Enter Matric Number: 2023/252316
Name: Daniel
Matric Number: 2023/252316
Score: 85
```

32) Mini Project – Basic ATM System

Objective:

A simple command-line ATM system that allows users to:

Check balance

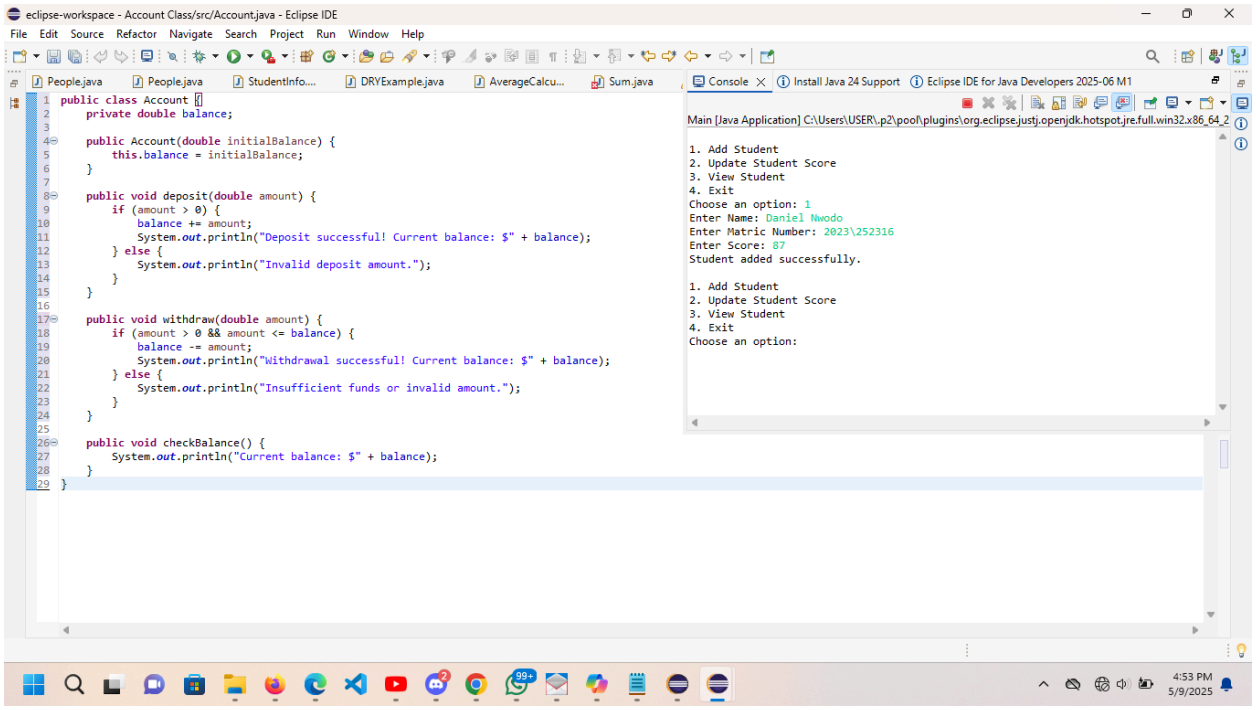
Deposit money

Withdraw money

Code Structure:

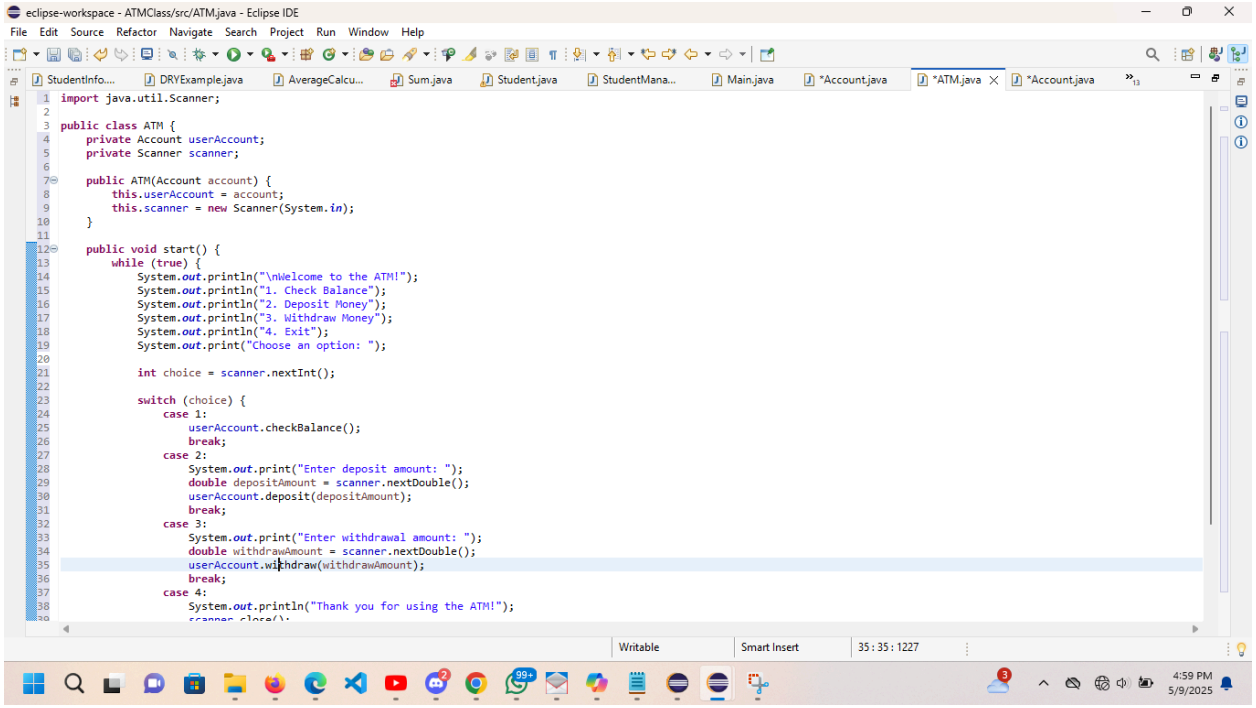
We'll create:

- An Account class to handle balance operations.



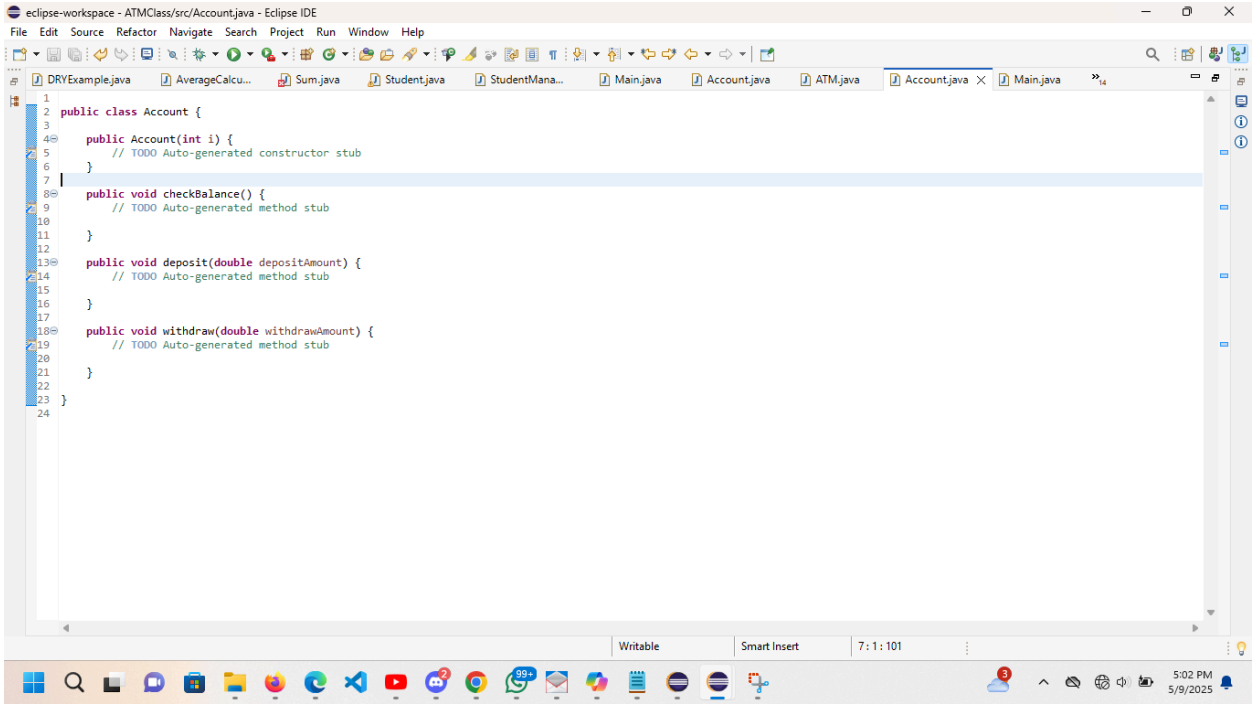

```java
public class Account {
    private double balance;

    public Account(double initialBalance) {
        this.balance = initialBalance;
    }

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposit successful! Current balance: $" + balance);
        } else {
            System.out.println("Invalid deposit amount.");
        }
    }

    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            System.out.println("Withdrawal successful! Current balance: $" + balance);
        } else {
            System.out.println("Insufficient funds or invalid amount.");
        }
    }

    public void checkBalance() {
        System.out.println("Current balance: $" + balance);
    }
}
```

- An ATM class for the user interface.




```java
import java.util.Scanner;

public class ATM {
    private Account userAccount;
    private Scanner scanner;

    public ATM(Account account) {
        this.userAccount = account;
        this.scanner = new Scanner(System.in);
    }

    public void start() {
        while (true) {
            System.out.println("\nWelcome to the ATM!");
            System.out.println("1. Check Balance");
            System.out.println("2. Deposit Money");
            System.out.println("3. Withdraw Money");
            System.out.println("4. Exit");
            System.out.print("Choose an option: ");

            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    userAccount.checkBalance();
                    break;
                case 2:
                    System.out.print("Enter deposit amount: ");
                    double depositAmount = scanner.nextDouble();
                    userAccount.deposit(depositAmount);
                    break;
                case 3:
                    System.out.print("Enter withdrawal amount: ");
                    double withdrawAmount = scanner.nextDouble();
                    userAccount.withdraw(withdrawAmount);
                    break;
                case 4:
                    System.out.println("Thank you for using the ATM!");
```

- A Main class to run the program.




```
Welcome to the ATM!
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Exit
Choose an option: 1
Current balance: $1000

Choose an option: 2
Enter deposit amount: 500
Deposit successful! Current balance: $1500

Choose an option: 3
Enter withdrawal amount: 300
Withdrawal successful! Current balance: $1200
```