UNIVERSITY OF NIGERIA, NSUKKA

FACULTY OF PHYSICAL SCIENCES

DEPARTMENT OF COMPUTER SCIENCE

AN ASSIGNMENT ON

COS 261

BY

EKEH IHECHUKWU BRIGHT

REG NUMBER: 2023/257470

LECTURER: MRS. AMINAT ATANDA

DATE: MAY 2025

Basics & Syntax

1. Write a Java program to print "Hello, World!".

2. Explain the difference between == and .equals() in Java. Show with code examples and outputs.

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

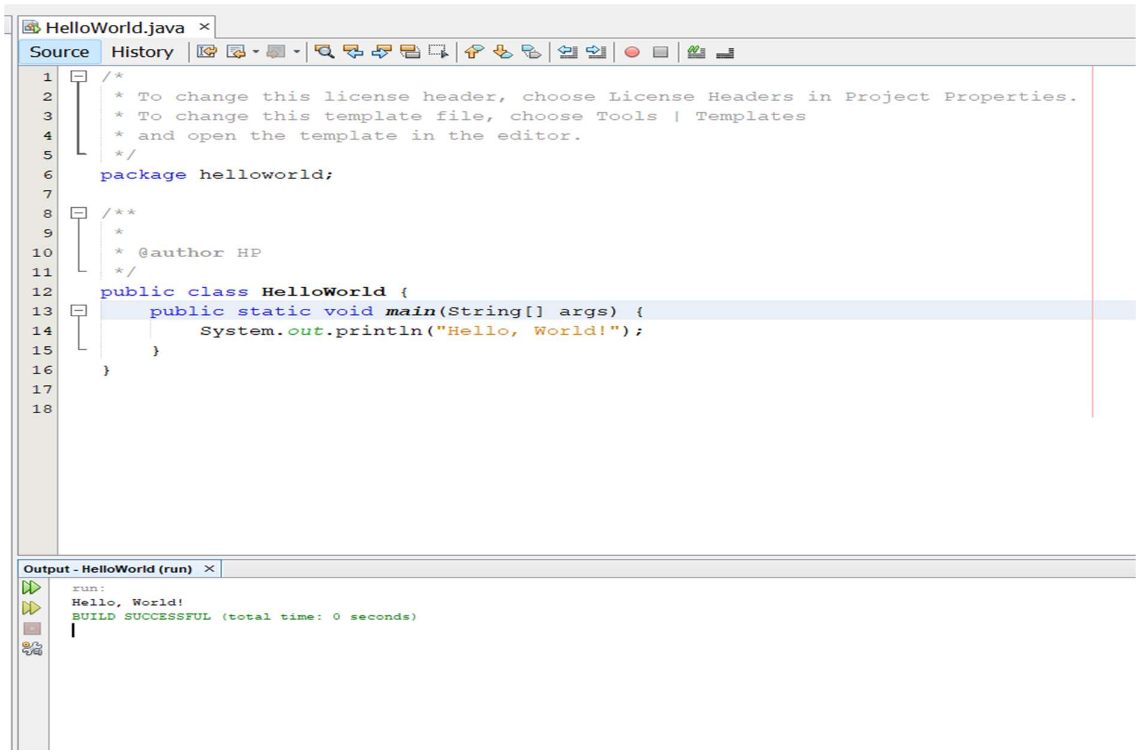
4. Write a Java program to add two numbers entered by the user.

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

ANSWER

1.

```
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package helloworld;
 @author HP
 public class HelloWorld {
   public static void main(String[] args) {
       System.out.println("Hello, World!");
   }
```



```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package helloworld;

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

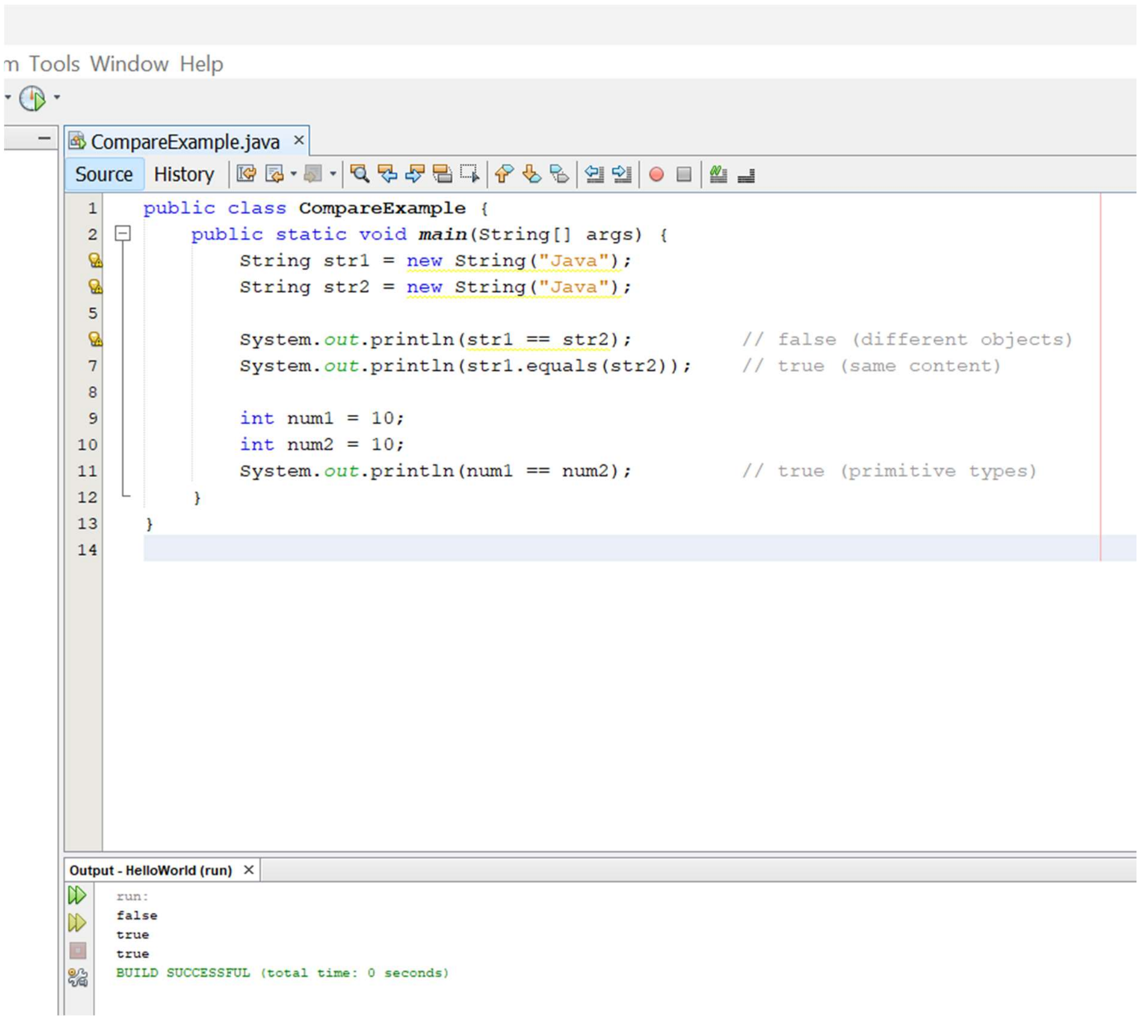
```
run:
Hello, World!
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. **Difference Between == and .equals() in Java**

- ==: Compares *references* (memory addresses), i.e., whether two references point to the same object.
- .equals(): Compares *contents* of objects, i.e., whether two objects have the same value.

Example:

```
public class CompareExample {
   public static void main(String[] args) {
      String str1 = new String("Java");
      String str2 = new String("Java");

      System.out.println(str1 == str2);         // false (different objects)
      System.out.println(str1.equals(str2));    // true (same content)

      int num1 = 10;
      int num2 = 10;
      System.out.println(num1 == num2);         // true (primitive types)
   }
}
```



```
public class CompareExample {
    public static void main(String[] args) {
        String str1 = new String("Java");
        String str2 = new String("Java");

        System.out.println(str1 == str2);          // false (different objects)
        System.out.println(str1.equals(str2));     // true (same content)

        int num1 = 10;
        int num2 = 10;
        System.out.println(num1 == num2);          // true (primitive types)
    }
}
```

```
run:
false
true
true
BUILD SUCCESSFUL (total time: 0 seconds)
```

Explanation:

- str1 == str2: false because they are two different objects in memory.
- str1.equals(str2): true because their contents are the same.
- num1 == num2: true because primitive values are compared directly.

3. Use of the main Method in Java

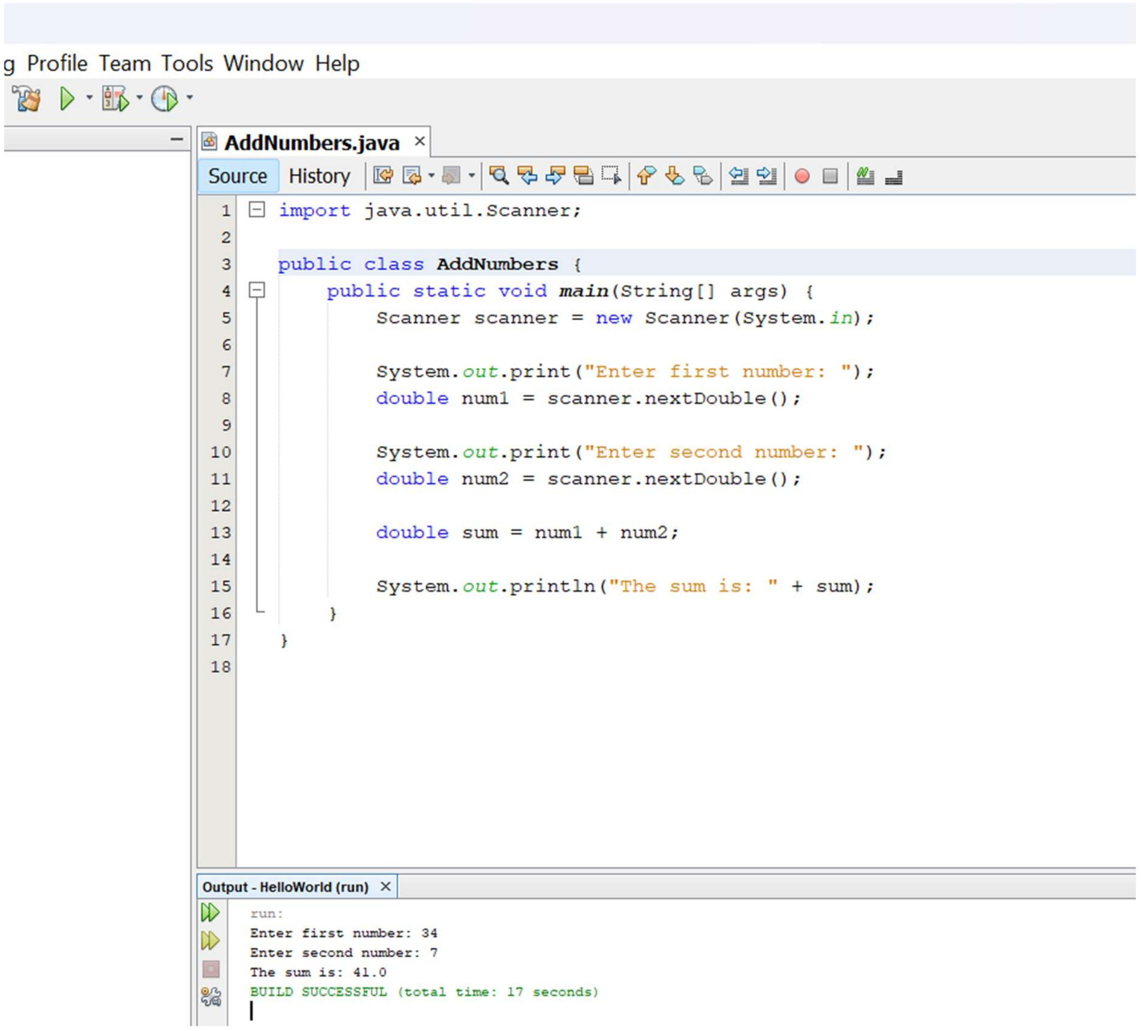
The main method is the *entry point* of any standalone Java application. It tells the JVM where to start running your program.

Signature:
public static void main(String[] args)

Why?
    public: JVM needs to access it from outside the class.
    static: Can run without creating an object.
    void: Doesn't return anything.
    String[] args: Receives command-line arguments

4. Java Program to Add Two Numbers Entered by the User

```
import java.util.Scanner;

public class AddNumbers {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter first number: ");
        double num1 = scanner.nextDouble();

        System.out.print("Enter second number: ");
        double num2 = scanner.nextDouble();

        double sum = num1 + num2;

        System.out.println("The sum is: " + sum);
    }
}
```



```
import java.util.Scanner;

public class AddNumbers {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter first number: ");
        double num1 = scanner.nextDouble();

        System.out.print("Enter second number: ");
        double num2 = scanner.nextDouble();

        double sum = num1 + num2;

        System.out.println("The sum is: " + sum);
    }
}
```

```
run:
Enter first number: 34
Enter second number: 7
The sum is: 41.0
BUILD SUCCESSFUL (total time: 17 seconds)
```

5. Difference Between `int`, `Integer`, and `String`

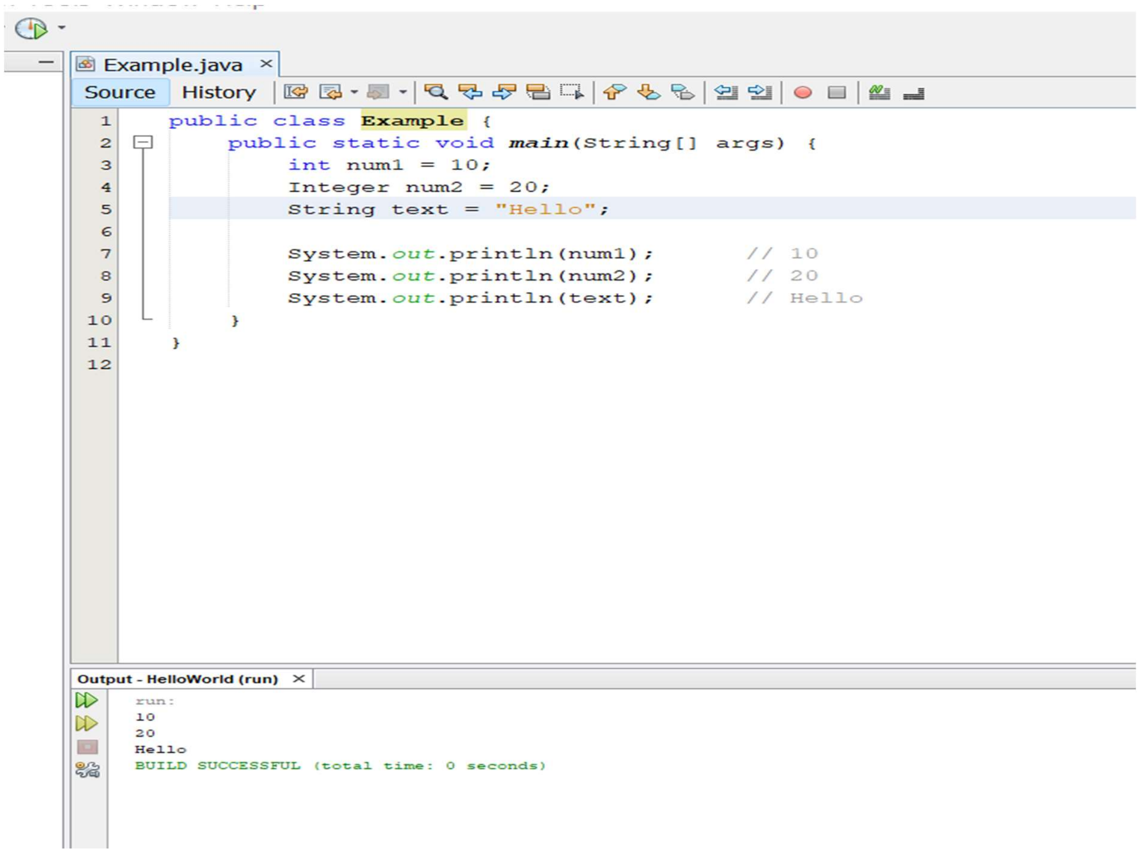
Type	Description	Example
`int`	A **primitive data type** for integers.	`int x = 5;`
`Integer`	A **wrapper class** for `int` (object version).	`Integer y = 5;`
`String`	An **object** that represents a sequence of characters (text).	`String s = "Java";`

Key Differences:

- `int` is primitive and faster but cannot be used with objects/collections directly.
- `Integer` is an object, useful when working with collections or when nullability is needed.
- `String` holds sequences of characters, not numbers.

Example

```
public class Example {
    public static void main(String[] args) {
        int num1 = 10;
        Integer num2 = 20;
        String text = "Hello";

        System.out.println(num1);      // 10
        System.out.println(num2);      // 20
        System.out.println(text);      // Hello
    }
}
```



```
public class Example {
    public static void main(String[] args) {
        int num1 = 10;
        Integer num2 = 20;
        String text = "Hello";

        System.out.println(num1);        // 10
        System.out.println(num2);        // 20
        System.out.println(text);        // Hello
    }
}
```

```
run:
10
20
Hello
BUILD SUCCESSFUL (total time: 0 seconds)
```

Control Structures

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

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

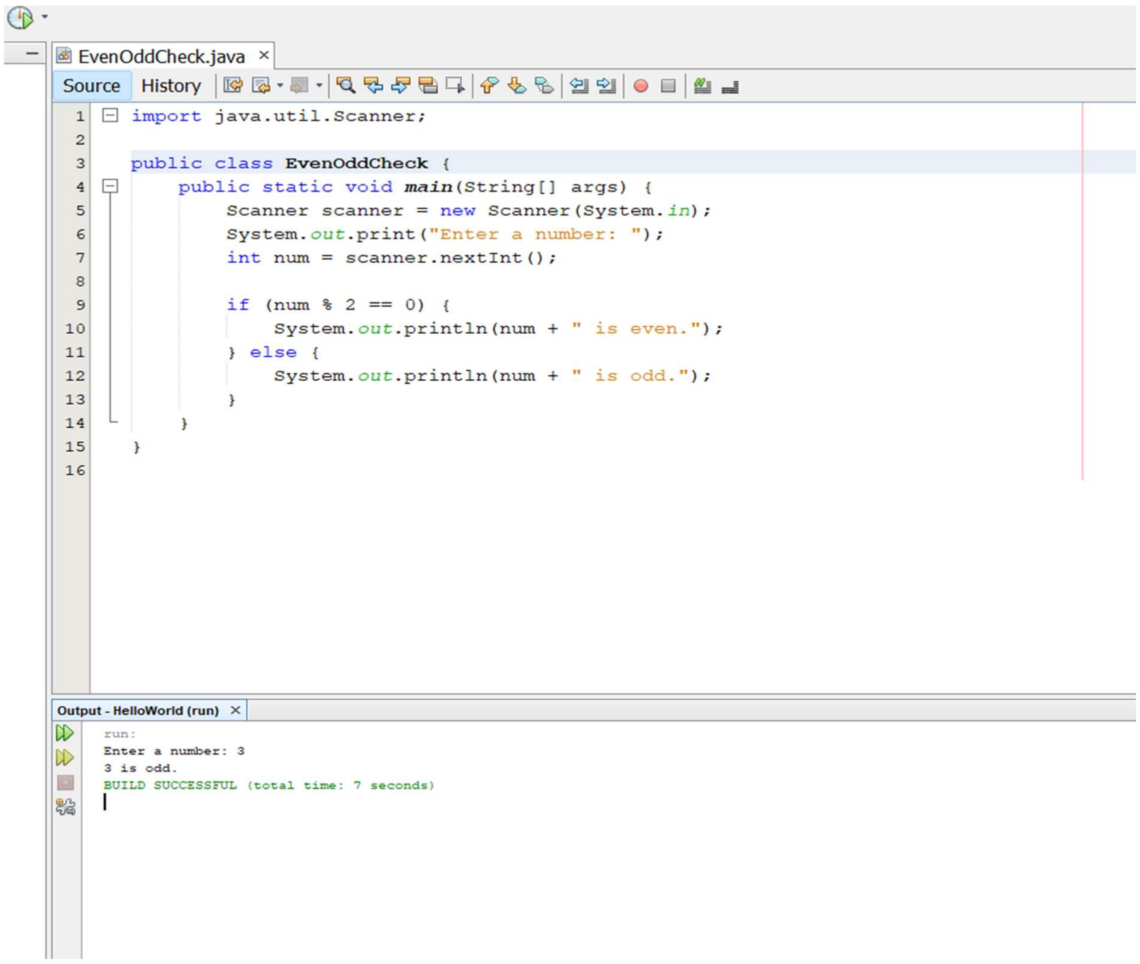
8. Explain the difference between while, for, and do-while loops in Java.

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

ANSWER

6. Program to check if a number is even or odd:

```java
import java.util.Scanner;

public class EvenOddCheck {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = scanner.nextInt();

        if (num % 2 == 0) {
            System.out.println(num + " is even.");
        } else {
            System.out.println(num + " is odd.");
        }
    }
}
```

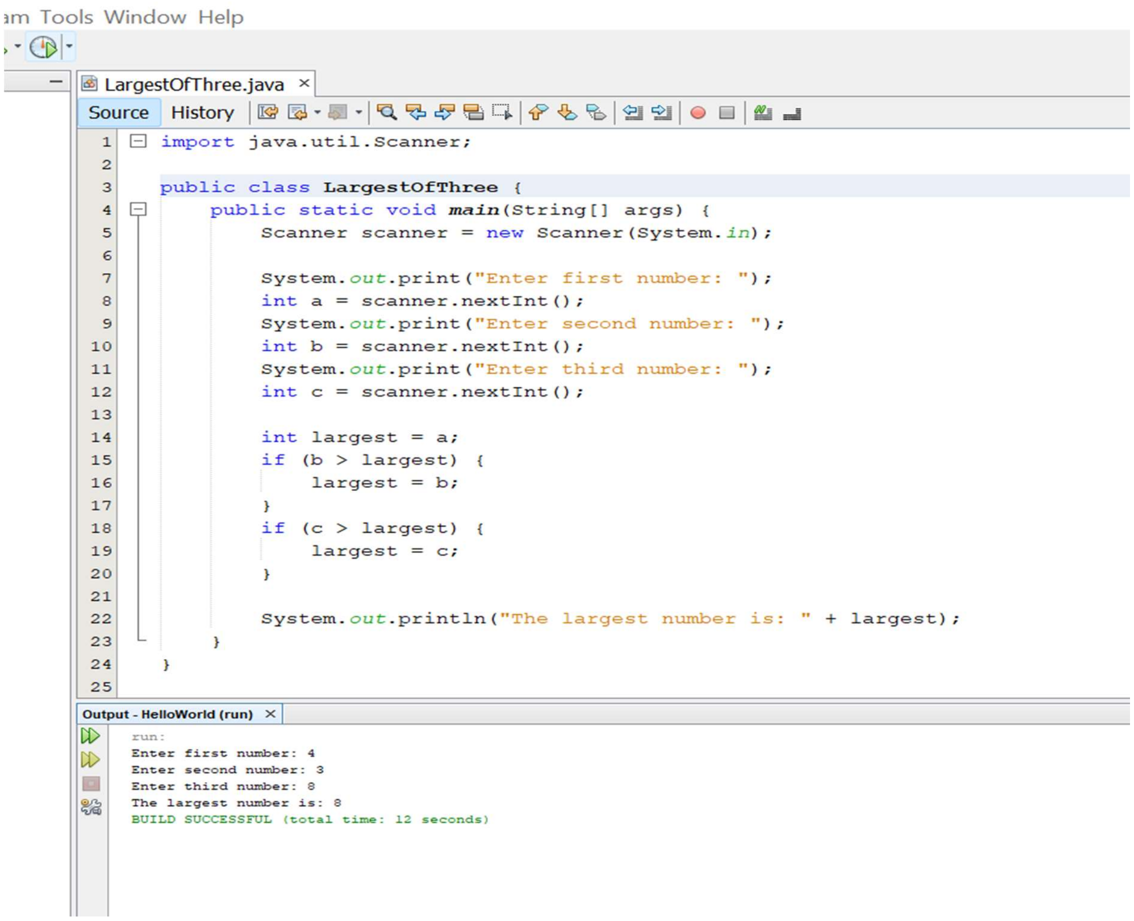


```java
import java.util.Scanner;

public class EvenOddCheck {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = scanner.nextInt();

        if (num % 2 == 0) {
            System.out.println(num + " is even.");
        } else {
            System.out.println(num + " is odd.");
        }
    }
}
```

```
run:
Enter a number: 3
3 is odd.
BUILD SUCCESSFUL (total time: 7 seconds)
```

7. Program to find the largest among three numbers:

```
import java.util.Scanner;

public class LargestOfThree {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter first number: ");
        int a = scanner.nextInt();
        System.out.print("Enter second number: ");
        int b = scanner.nextInt();
        System.out.print("Enter third number: ");
        int c = scanner.nextInt();

        int largest = a;
        if (b > largest) {
            largest = b;
        }
        if (c > largest) {
            largest = c;
        }

        System.out.println("The largest number is: " + largest);
    }
}
```



```
import java.util.Scanner;

public class LargestOfThree {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter first number: ");
        int a = scanner.nextInt();
        System.out.print("Enter second number: ");
        int b = scanner.nextInt();
        System.out.print("Enter third number: ");
        int c = scanner.nextInt();

        int largest = a;
        if (b > largest) {
            largest = b;
        }
        if (c > largest) {
            largest = c;
        }

        System.out.println("The largest number is: " + largest);
    }
}
```

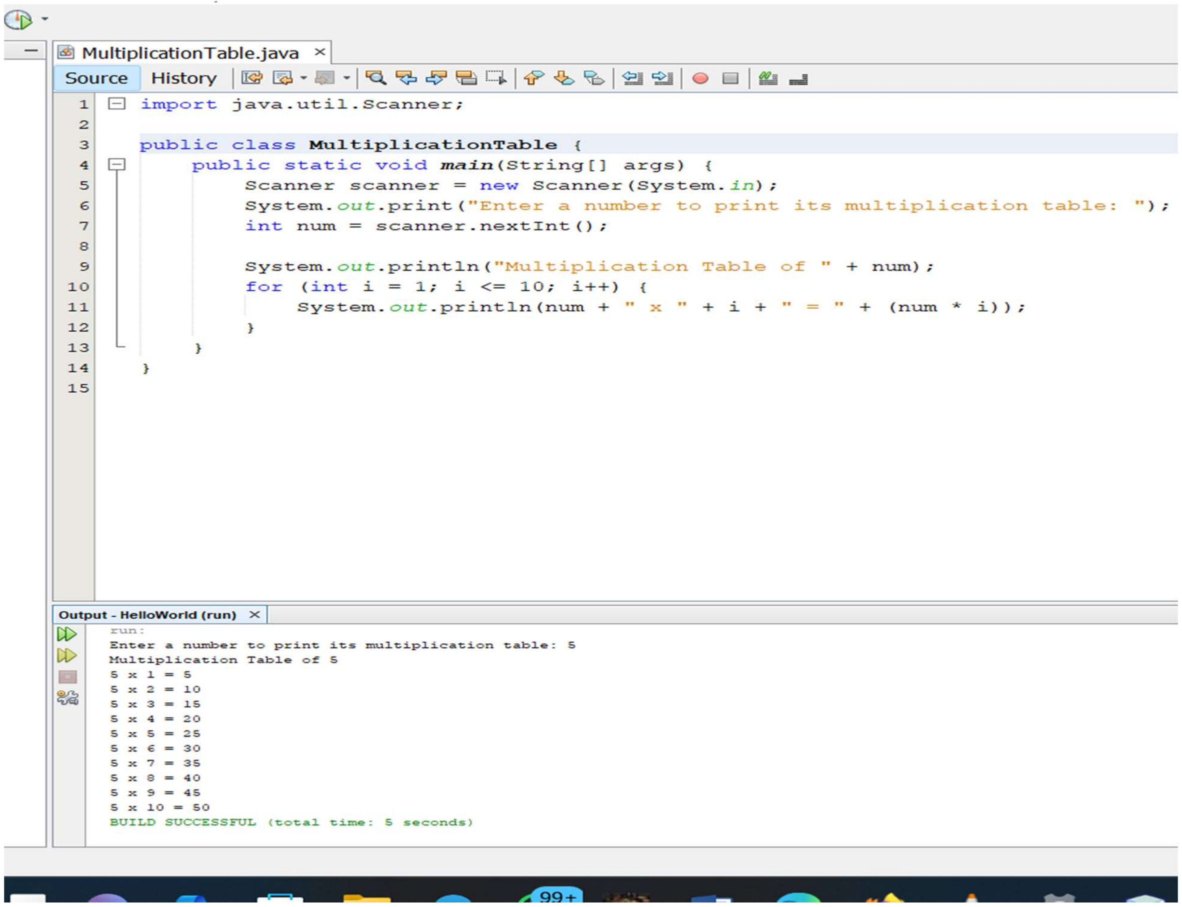
```
run:
Enter first number: 4
Enter second number: 3
Enter third number: 8
The largest number is: 8
BUILD SUCCESSFUL (total time: 12 seconds)
```

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

Loop Type	**Syntax Example**	**Condition Check**	**Use Case**
while	while(condition) { ... }	Before loop	Use when number of iterations is not known in advance.
for	for(init; condition; update) { ... }	Before loop	Use when number of iterations is known or for counting loops.
do-while	do { ... } while(condition);	After loop	Use when loop should run at least once regardless of condition.

9. Java program to print the multiplication table of any number:

```
import java.util.Scanner;

public class MultiplicationTable {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number to print its multiplication table: ");
        int num = scanner.nextInt();

        System.out.println("Multiplication Table of " + num);
        for (int i = 1; i <= 10; i++) {
            System.out.println(num + " x " + i + " = " + (num * i));
        }
```



```
import java.util.Scanner;

public class MultiplicationTable {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number to print its multiplication table: ");
        int num = scanner.nextInt();

        System.out.println("Multiplication Table of " + num);
        for (int i = 1; i <= 10; i++) {
            System.out.println(num + " x " + i + " = " + (num * i));
        }
    }
}
```

```
run:
Enter a number to print its multiplication table: 5
Multiplication Table of 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
BUILD SUCCESSFUL (total time: 5 seconds)
```

Intermediate-Level Questions

OOP Concepts

10. Explain the four pillars of OOP in Java.

11. Create a class Student with properties name, matricNo, and score, and add methods to display the student's info.

12. What is method overloading? Give a code example.

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

General Practices

14. What does it mean to write "clean code"? Give 3 practices that make code clean and maintainable.

15. Why should you avoid writing very long methods in Java programs?

16. What naming conventions should be followed in Java for: Classes, Variables, Methods.

Give examples with screenshot of code and output.

17. What is the importance of breaking your Java program into methods?

18. Explain the concept of DRY (Don't Repeat Yourself) with a Java code example.

19. What are the benefits of using classes and objects instead of writing all logic in the main method?

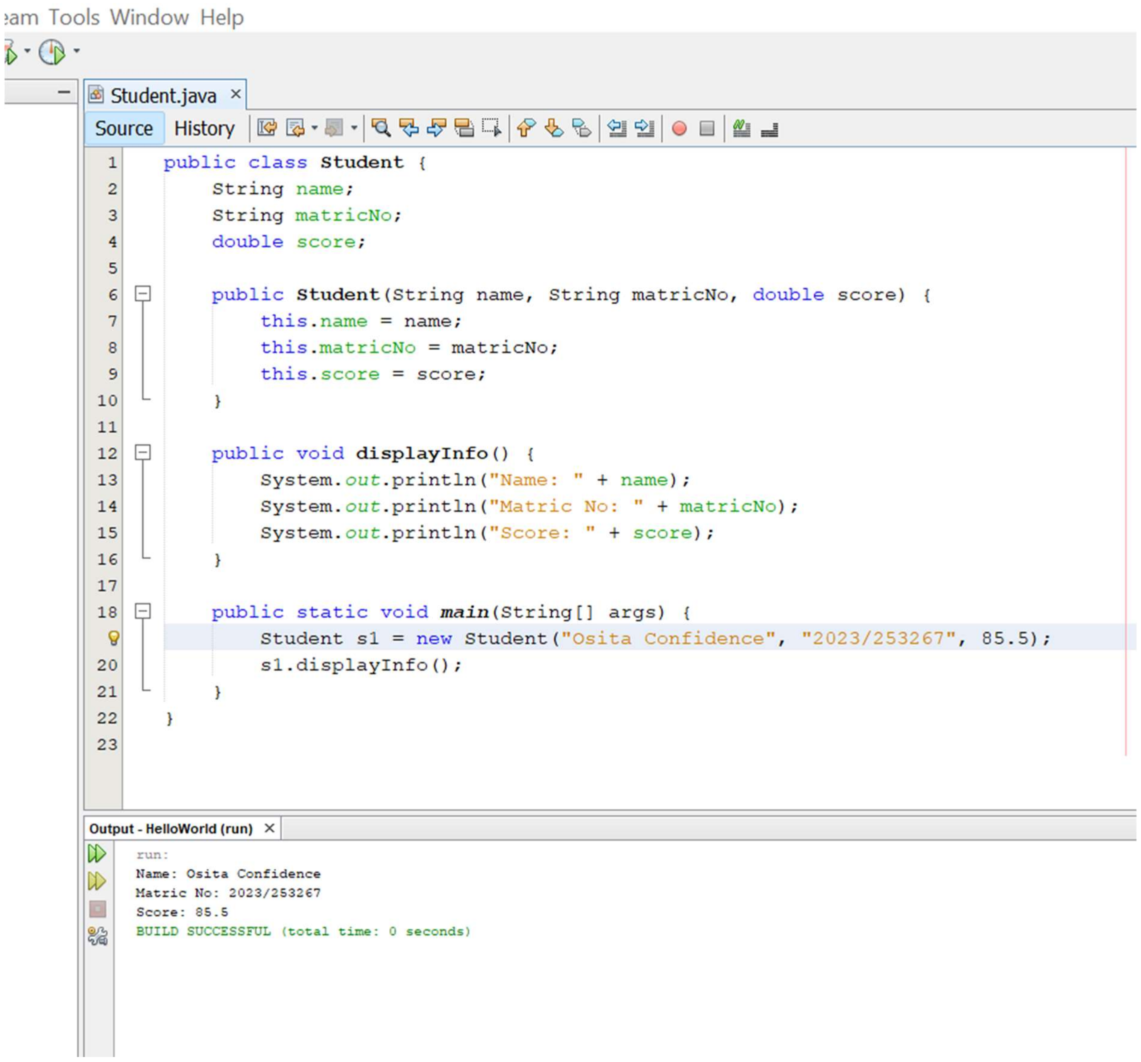
ANSWER

10. Four Pillars of OOP in Java

1. **Encapsulation**: Wrapping data (variables) and code (methods) together inside a class.
2. **Abstraction**: Hiding complex implementation and showing only the relevant details to the user.
3. **Inheritance**: Acquiring properties and behavior (methods) from another class.
4. **Polymorphism**: Performing a single action in different ways, like method overloading/overriding.

11. Class Example

```
public class Student {
   String name;
   String matricNo;
   double score;

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

   public void displayInfo() {
        System.out.println("Name: " + name);
        System.out.println("Matric No: " + matricNo);
        System.out.println("Score: " + score);
    }

    public static void main(String[] args) {
        Student s1 = new Student("Osita Confidence", "2023/253267", 85.5);
        s1.displayInfo();
    }
}
```



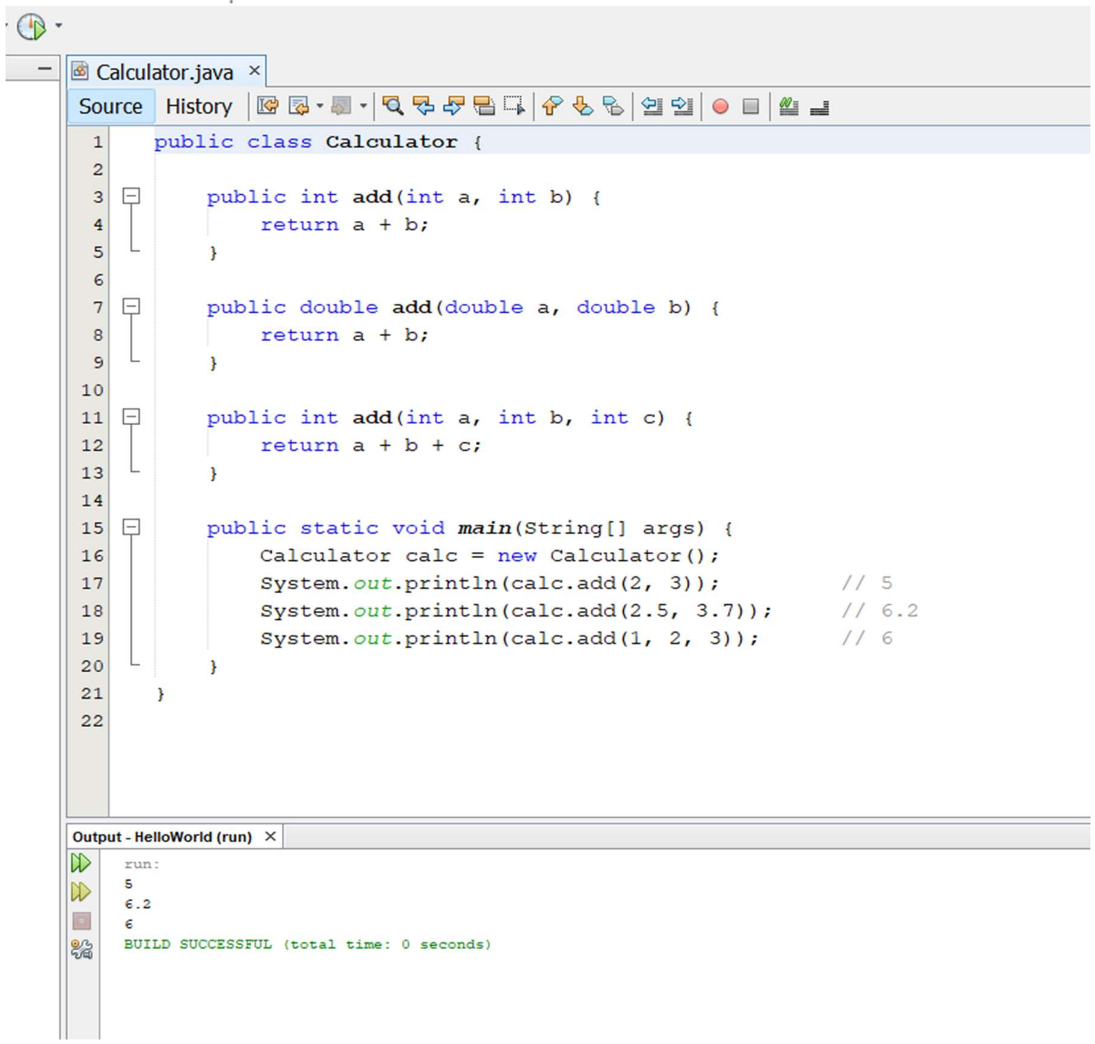
```
public class Student {
    String name;
    String matricNo;
    double score;

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

    public void displayInfo() {
        System.out.println("Name: " + name);
        System.out.println("Matric No: " + matricNo);
        System.out.println("Score: " + score);
    }

    public static void main(String[] args) {
        Student s1 = new Student("Osita Confidence", "2023/253267", 85.5);
        s1.displayInfo();
    }
}
```

```
run:
Name: Osita Confidence
Matric No: 2023/253267
Score: 85.5
BUILD SUCCESSFUL (total time: 0 seconds)
```

12. Method Overloading

Definition: Method overloading allows multiple methods with the same name but different parameters in the same class.

```
public class Calculator {

    public int add(int a, int b) {

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

    public int add(int a, int b, int c) {
        return a + b + c;
    }
    public static void main(String[] args) {
        Calculator calc = new Calculator();
        System.out.println(calc.add(2, 3));         // 5
        System.out.println(calc.add(2.5, 3.7));     // 6.2
        System.out.println(calc.add(1, 2, 3));      // 6
    }
}
```



```
public class Calculator {

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

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

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

    public static void main(String[] args) {
        Calculator calc = new Calculator();
        System.out.println(calc.add(2, 3));         // 5
        System.out.println(calc.add(2.5, 3.7));     // 6.2
        System.out.println(calc.add(1, 2, 3));      // 6
    }
}
```

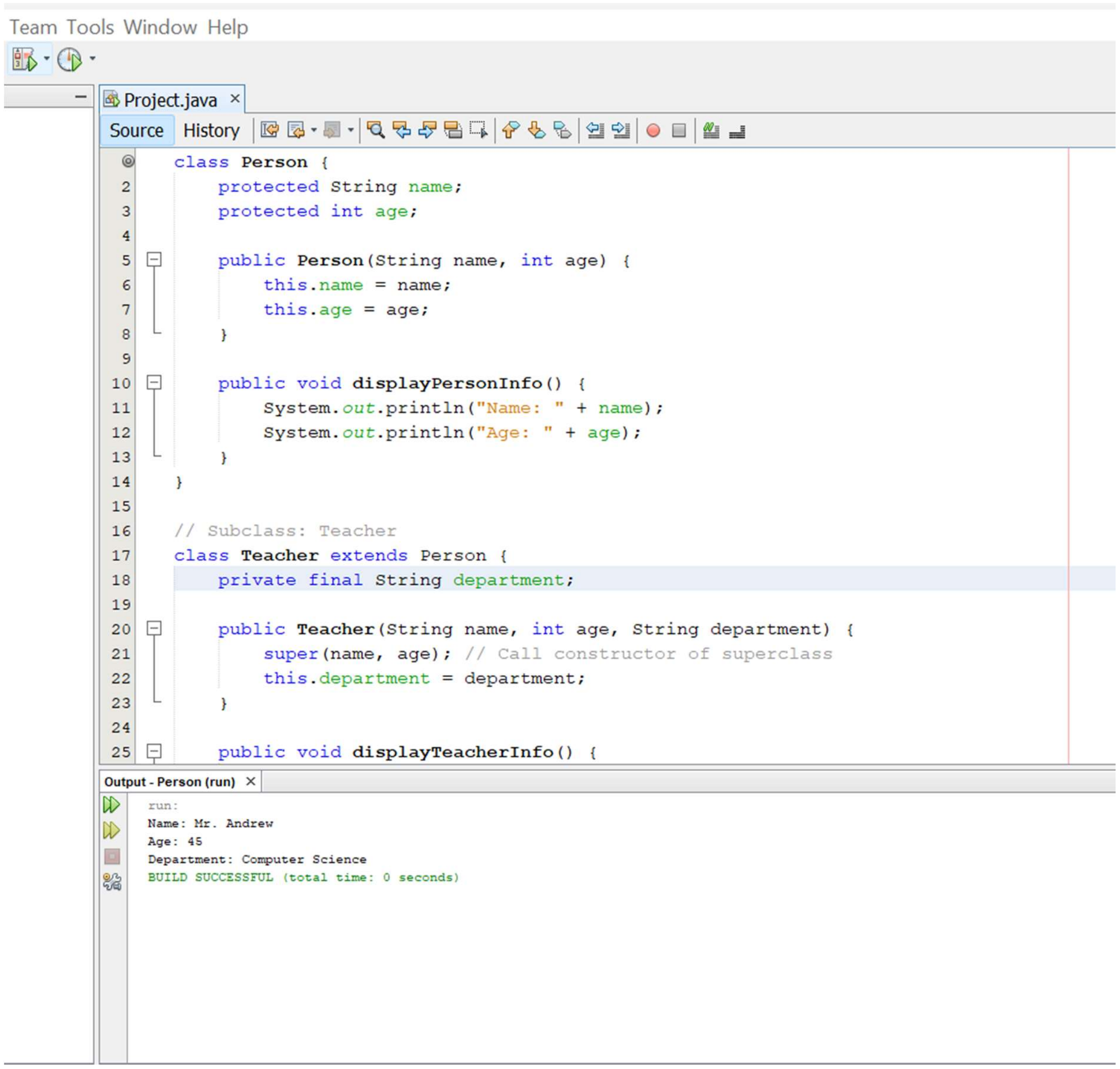
```
run:
5
6.2
6
BUILD SUCCESSFUL (total time: 0 seconds)
```

13. What is Inheritance?

Inheritance is a fundamental concept of Object-Oriented Programming (OOP) that allows a class (called a subclass or child class) to inherit fields and methods from another class (called a superclass or parent class). This promotes code reuse and establishes a natural "is-a" relationship between classes.

Example Using Inheritance in Javaclass Person {

```
    protected String name;

    protected int age;

    public Person(String name, int age) {

        this.name = name;

        this.age = age;

    }

    public void displayPersonInfo() {

        System.out.println("Name: " + name);

        System.out.println("Age: " + age);

    }

}

// Subclass: Teacher

class Teacher extends Person {

    private final String department;


    public Teacher(String name, int age, String department) {

        super(name, age); // Call constructor of superclass

        this.department = department;

    }

    public void displayTeacherInfo() {

        displayPersonInfo(); // Inherited method

        System.out.println("Department: " + department);

    }

    public static void main(String[] args) {

        Teacher teacher = new Teacher("Mr. Andrew", 45, "Computer Science");
        teacher.displayTeacherInfo();
    }
}
```



```
class Person {
    protected String name;
    protected int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void displayPersonInfo() {
        System.out.println("Name:  " + name);
        System.out.println("Age:  " + age);
    }
}

// Subclass: Teacher
class Teacher extends Person {
    private final String department;

    public Teacher(String name, int age, String department) {
        super(name, age); // Call constructor of superclass
        this.department = department;
    }

    public void displayTeacherInfo() {
```

```
run:
Name: Mr. Andrew
Age: 45
Department: Computer Science
BUILD SUCCESSFUL (total time: 0 seconds)
```

This demonstrates how Teacher inherits common properties (name, age) and behavior (displayPersonInfo) from Person, while also introducing its own property (department).

Would you like to see this extended with more subclasses (e.g., Student, Admin)?

14. Clean Code Practices

"Clean code" means code that is easy to read, understand, and maintain.

Three Practices:

- Use meaningful variable and method names.
- Write short, focused methods that do one thing.

- Add comments and follow consistent formatting.

15. Avoiding Long Methods

Long methods are harder to read, debug, and test. They often try to do too much and break the Single Responsibility Principle. Breaking code into smaller methods improves modularity and reusability.

16. Java Naming Conventions

Type	**Convention**	**Example**
Class	PascalCase	StudentRecord
Variable	camelCase	studentName
Method	camelCase with verb prefix	calculateAverage()

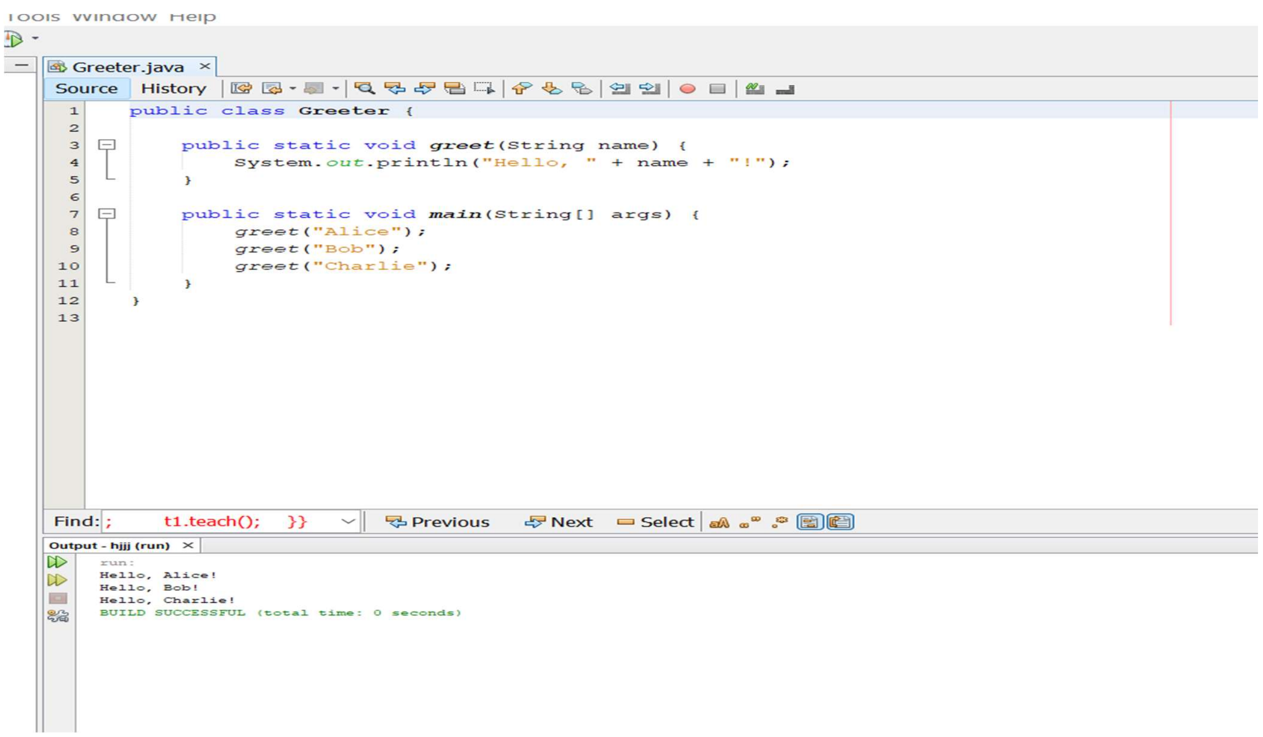
17. Breaking Programs into Methods

It promotes modularity, code reuse, and easier debugging. Each method can be tested independently and reused across different parts of your program.

18. DRY Principle ("Don't Repeat Yourself")

Avoid duplicating code by extracting common logic into methods.

```
public class Greeter {
  public static void greet(String name) {
    System.out.println("Hello, " + name + "!");
  }
  public static void main(String[] args) {
    greet("Alice");
    greet("Bob");
    greet("Charlie");
  }
}
```



```
public class Greeter {

    public static void greet(String name) {
        System.out.println("Hello, " + name + "!");
    }

    public static void main(String[] args) {
        greet("Alice");
        greet("Bob");
        greet("Charlie");
    }
}
```

```
run:
Hello, Alice!
Hello, Bob!
Hello, Charlie!
BUILD SUCCESSFUL (total time: 0 seconds)
```

Testing & Debugging

20. Why is testing important during program development?

21. What is the difference between syntax error, runtime error, and logic error?

22. How would you test a method that calculates the average of five numbers?

Documentation & Comments

23. Why should Java developers write comments in their code?

24. What are JavaDoc comments and how are they different from regular comments?

25. Write a sample Java method with JavaDoc comments.

Versioning & Collaboration

26. What is version control and why is it important in team projects?

27. How would you explain the concept of "code refactoring" to a junior developer?

28. What tools can Java developers use to collaborate on large projects? Attach screenshots of 3 examples.

ANSWER

20. Why is testing important during program development?

Testing ensures the software works as intended, helps identify and fix bugs early, improves code reliability, and reduces the risk of failure when deployed.

21. What is the difference between syntax error, runtime error, and logic error?

Syntax error: Violations of the language's grammar rules, caught at compile time (e.g., missing semicolon).

Runtime error: Errors that occur while the program is running (e.g., division by zero, null pointer exception).

Logic error: The program runs without crashing but produces incorrect results due to flawed logic (e.g., incorrect formula).

22. How would you test a method that calculates the average of five numbers?

Create test cases with known values and expected results.

Test normal values (e.g., 10, 20, 30, 40, 50).

Test edge cases (e.g., all zeros, all same number, negative numbers).

Test with decimal values (if applicable).

Validate that the method returns the correct average and handles unexpected inputs gracefully (if applicable).

Documentation & Comments

23. Why should Java developers write comments in their code?

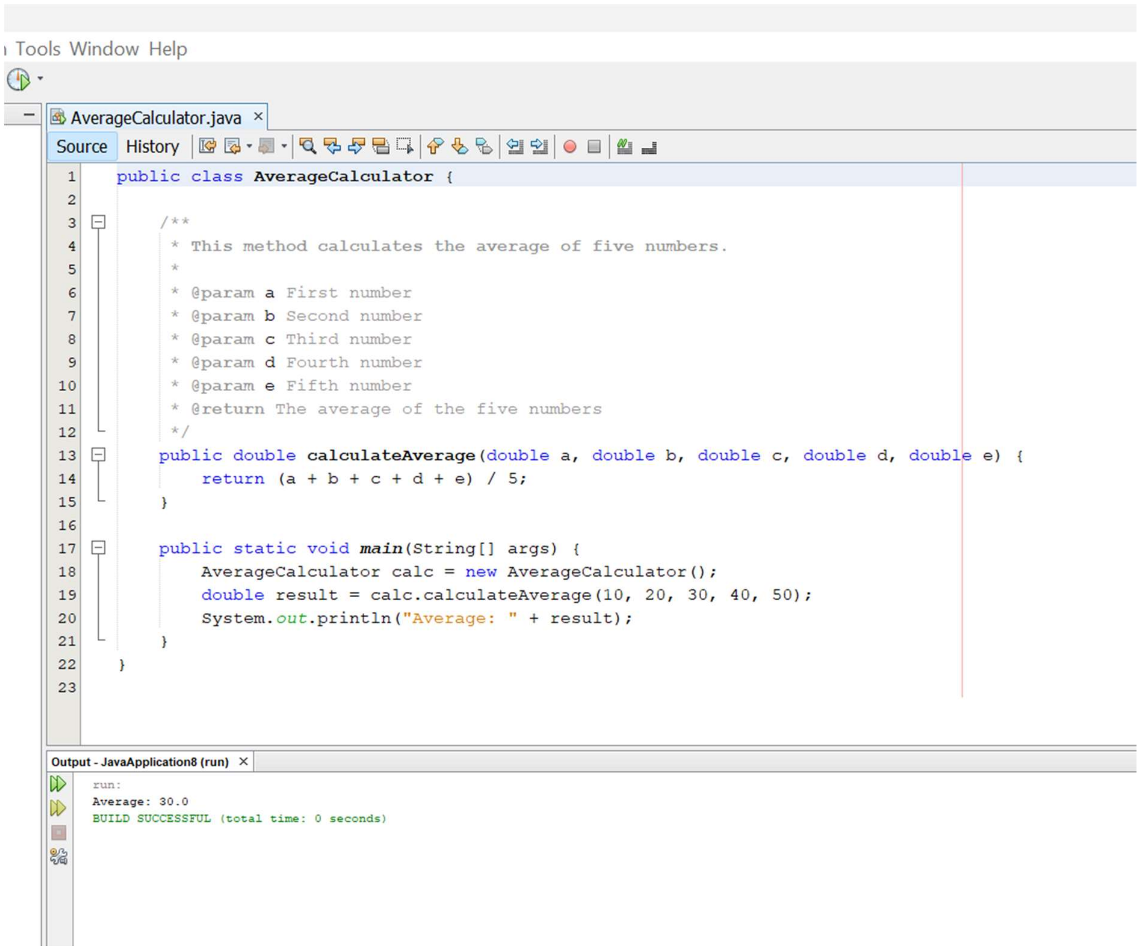
Comments help others (and future you) understand the code’s purpose, logic, and usage. They improve maintainability, aid debugging, and are especially important in collaborative projects.

24. What are JavaDoc comments and how are they different from regular comments?

JavaDoc comments (/** ... */) are special comments used to generate HTML documentation for classes, methods, and fields. Regular comments (// or /* ... */) are for internal notes and are not included in documentation.

25. Write a sample Java method with JavaDoc comments.

```
/**

 * This method calculates the average of five numbers.

 *

 * @param a First number

 * @param b Second number

 * @param c Third number
 * @param d Fourth number

 * @param e Fifth number

 * @return The average of the five numbers

 */

public double calculateAverage(double a, double b, double c, double d, double e) {

   return (a + b + c + d + e) / 5;

}
```



```
public class AverageCalculator {

    /**
     * This method calculates the average of five numbers.
     *
     * @param a First number
     * @param b Second number
     * @param c Third number
     * @param d Fourth number
     * @param e Fifth number
     * @return The average of the five numbers
     */
    public double calculateAverage(double a, double b, double c, double d, double e) {
        return (a + b + c + d + e) / 5;
    }

    public static void main(String[] args) {
        AverageCalculator calc = new AverageCalculator();
        double result = calc.calculateAverage(10, 20, 30, 40, 50);
        System.out.println("Average: " + result);
    }
}
```

```
run:
Average: 30.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Versioning & Collaboration

26. What is version control and why is it important in team projects?

Version control systems (VCS) like Git track changes to code over time, allowing multiple developers to collaborate efficiently. They help manage different versions, resolve conflicts, and maintain a history of changes, which is vital for teamwork and project management.

27. How would you explain the concept of “code refactoring” to a junior developer?

Code refactoring is the process of restructuring existing code without changing its external behavior. The goal is to improve code readability, reduce complexity, and enhance maintainability, making it easier to add new features and fix bugs.

28. What tools can Java developers use to collaborate on large projects?

Here are three tools commonly used for collaboration:

GitHub: A platform for hosting and reviewing code, managing projects, and collaborating with teams. It provides version control using Git, issue tracking, and CI/CD integration.

Jira: A project management tool used for bug tracking, issue tracking, and project management. It helps teams plan, track, and release software.

IntelliJ IDEA: An integrated development environment (IDE) that supports collaborative coding through features like Code With Me, which allows real-time collaboration within the IDE.

These tools facilitate version control, project management, and real-time collaboration, making them essential for Java developers working on large projects.

Good Practices Summary

29. Mention 5 best practices you follow when developing a Java program.

30. What is code readability, and why is it more important than “smart” code?

ANSWERS

29. Mention 5 best practices you follow when developing a Java program:

1. Use meaningful variable and method names – Names should clearly describe the purpose of the variable or method (e.g., calculateAverage, not ca).
2. Follow consistent indentation and formatting – Makes your code easier to read and maintain.
3. Write comments and JavaDocs – Explain complex logic and provide documentation for methods and classes.
4. Avoid hardcoding values – Use constants or configuration files to make your code more flexible and maintainable.
5. Use exception handling properly – Catch exceptions where appropriate and give meaningful error messages.

30. What is code readability, and why is it more important than “smart” code?

Code readability means writing code that is easy for other developers (and your future self) to understand. This includes clear naming, consistent formatting, and avoiding overly complex or "clever" logic.

Why it's more important than “smart” code:

Readable code is easier to debug, maintain, and update. “Smart” code might be shorter or more efficient, but if no one can understand it, it becomes a maintenance nightmare. Clear code leads to fewer bugs and better teamwork.

Advanced-Level Questions

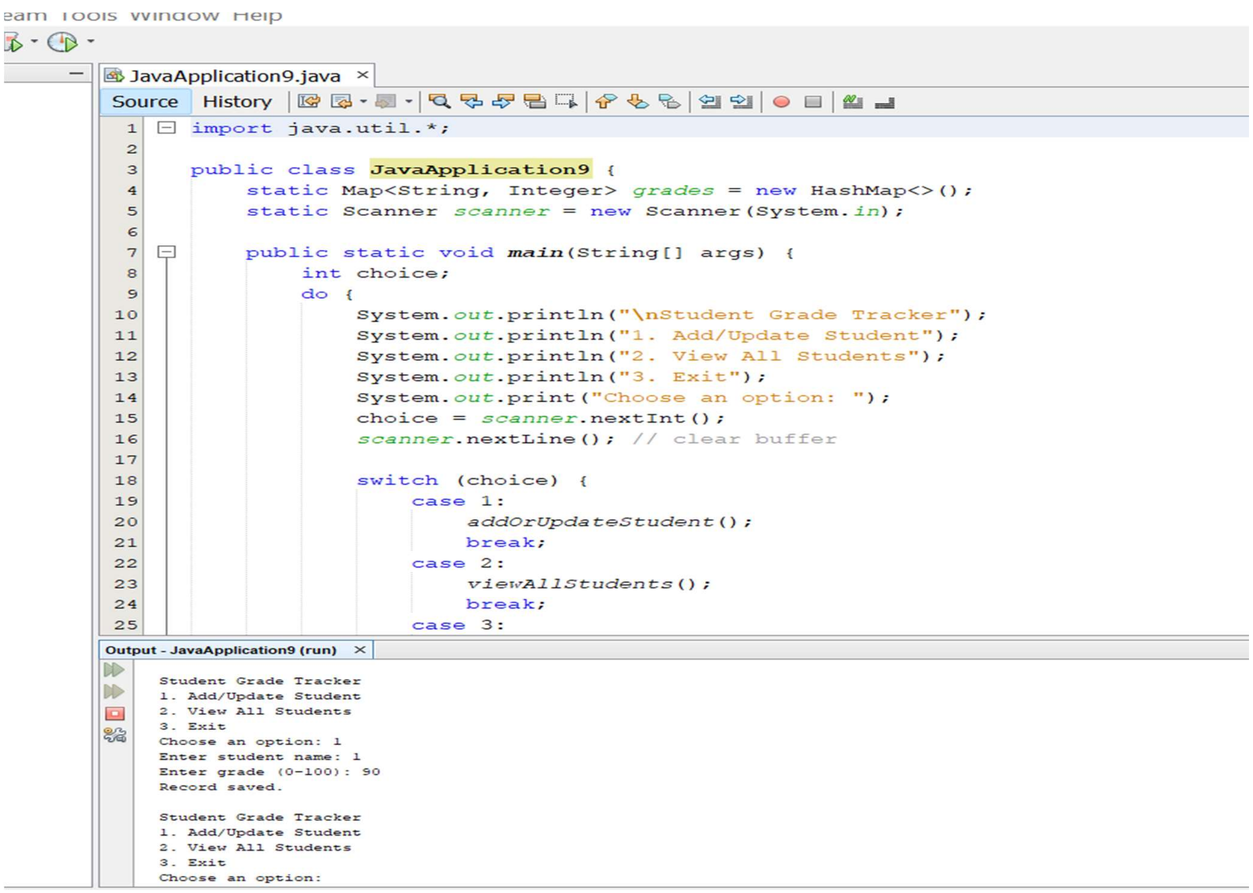
Mini Projects / Logic Building

31. Build a command-line application that keeps track of student grades and allows adding, updating, and viewing records.

32. Write a program that simulates a basic ATM system (check balance, deposit, withdraw).

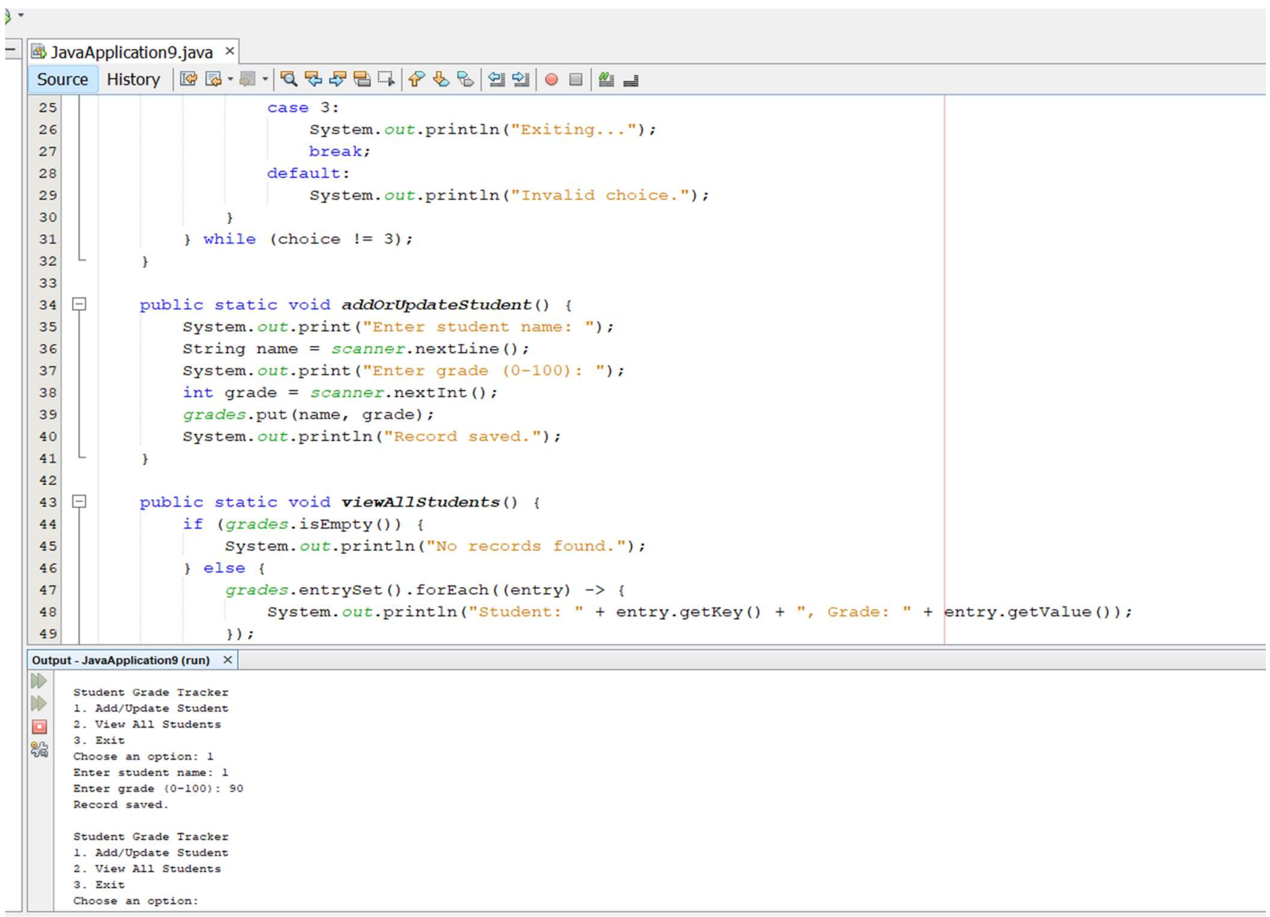
ANSWERS

31.
```
import java.util.*;
public class JavaApplication9 {
    static Map<String, Integer> grades = new HashMap<>();
    static Scanner scanner = new Scanner(System.in);
    public static void main(String[] args) {
        int choice;
        do {
            System.out.println("\nStudent Grade Tracker");
            System.out.println("1. Add/Update Student");
            System.out.println("2. View All Students");
            System.out.println("3. Exit");
            System.out.print("Choose an option: ");
            choice = scanner.nextInt();
            scanner.nextLine(); // clear buffer
            switch (choice) {
                case 1:
                    addOrUpdateStudent();
                    break;
                case 2:
                    viewAllStudents();
                    break;
                case 3:
                    System.out.println("Exiting...");
                    break;
                default:
                    System.out.println("Invalid choice.");
            }
        } while (choice != 3);
    }
    public static void addOrUpdateStudent() {
        System.out.print("Enter student name: ");
        String name = scanner.nextLine();
        System.out.print("Enter grade (0-100): ");
        int grade = scanner.nextInt();
        grades.put(name, grade);
        System.out.println("Record saved.");
    }
    public static void viewAllStudents() {
        if (grades.isEmpty()) {
            System.out.println("No records found.");
        } else {
            for (Map.Entry<String, Integer> entry : grades.entrySet()) {
                System.out.println("Student: " + entry.getKey() + ", Grade: " + entry.getValue());
            }
        }
    }
}
```



```java
import java.util.*;

public class JavaApplication9 {
    static Map<String, Integer> grades = new HashMap<>();
    static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        int choice;
        do {
            System.out.println("\nStudent Grade Tracker");
            System.out.println("1. Add/Update Student");
            System.out.println("2. View All Students");
            System.out.println("3. Exit");
            System.out.print("Choose an option: ");
            choice = scanner.nextInt();
            scanner.nextLine(); // clear buffer

            switch (choice) {
                case 1:
                    addOrUpdateStudent();
                    break;
                case 2:
                    viewAllStudents();
                    break;
                case 3:
```

```
Student Grade Tracker
1. Add/Update Student
2. View All Students
3. Exit
Choose an option: 1
Enter student name: 1
Enter grade (0-100): 90
Record saved.

Student Grade Tracker
1. Add/Update Student
2. View All Students
3. Exit
Choose an option:
```

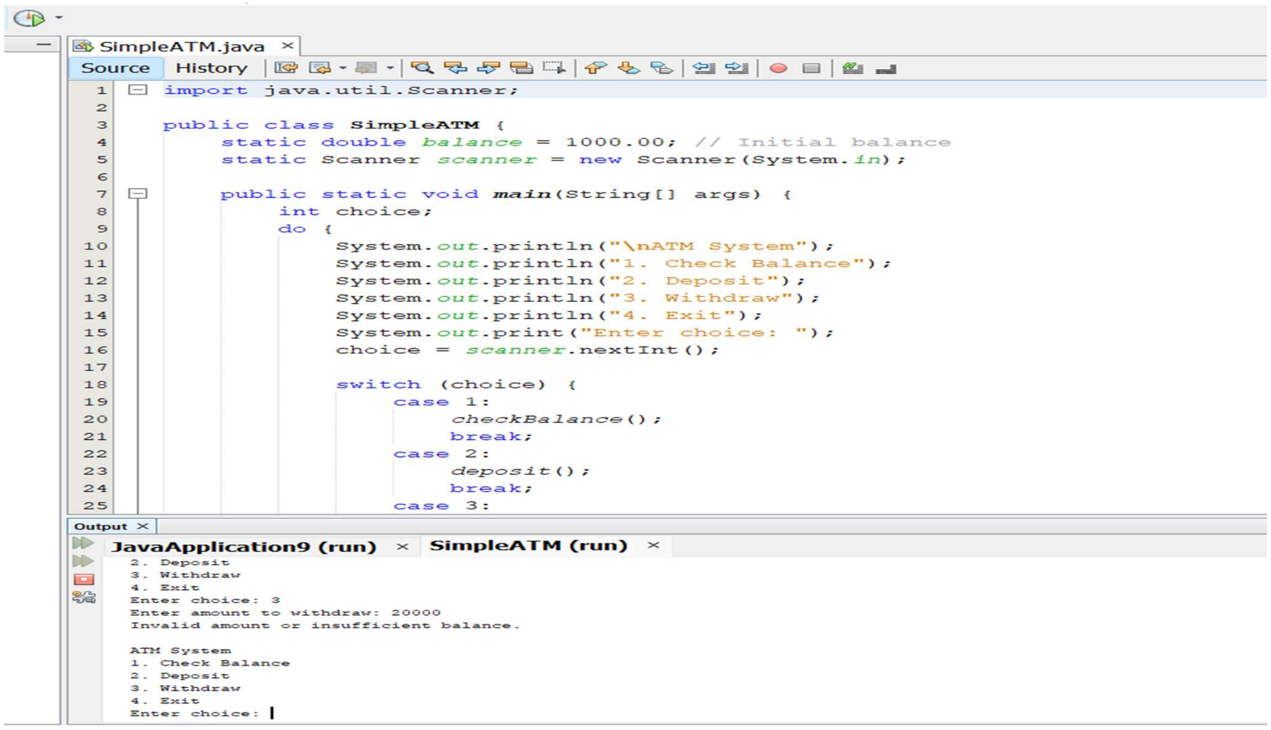


```java
                case 3:
                    System.out.println("Exiting...");
                    break;
                default:
                    System.out.println("Invalid choice.");
            }
        } while (choice != 3);
    }

    public static void addOrUpdateStudent() {
        System.out.print("Enter student name: ");
        String name = scanner.nextLine();
        System.out.print("Enter grade (0-100): ");
        int grade = scanner.nextInt();
        grades.put(name, grade);
        System.out.println("Record saved.");
    }

    public static void viewAllStudents() {
        if (grades.isEmpty()) {
            System.out.println("No records found.");
        } else {
            grades.entrySet().forEach((entry) -> {
                System.out.println("Student: " + entry.getKey() + ", Grade: " + entry.getValue());
            });
```

```
Student Grade Tracker
1. Add/Update Student
2. View All Students
3. Exit
Choose an option: 1
Enter student name: 1
Enter grade (0-100): 90
Record saved.

Student Grade Tracker
1. Add/Update Student
2. View All Students
3. Exit
Choose an option:
```

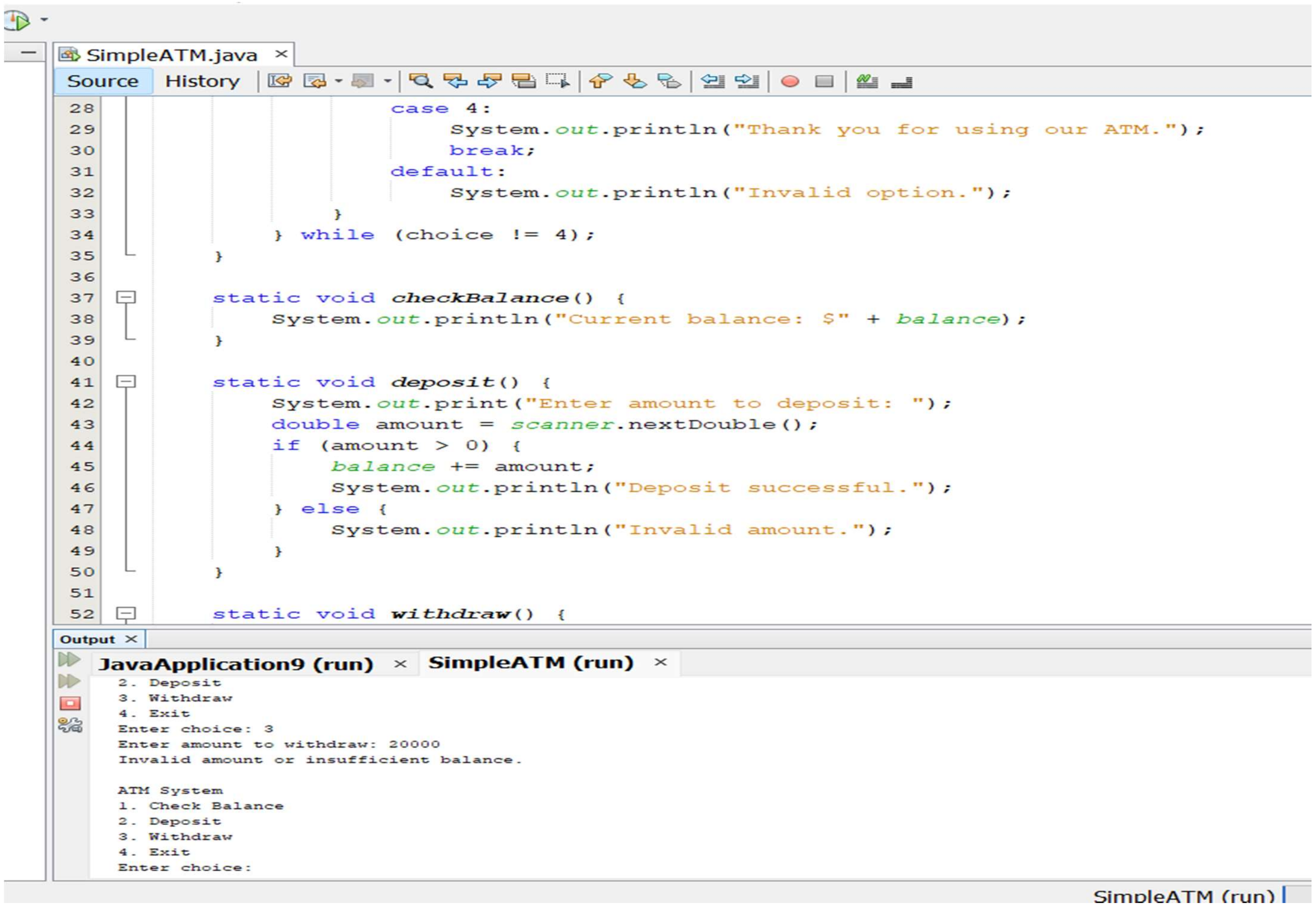
32.

```
import java.util.Scanner;
public class SimpleATM {
    static double balance = 1000.00; // Initial balance
    static Scanner scanner = new Scanner(System.in);
    public static void main(String[] args) {
        int choice;
        do {
            System.out.println("\nATM System");
            System.out.println("1. Check Balance");
            System.out.println("2. Deposit");
            System.out.println("3. Withdraw");
            System.out.println("4. Exit");
            System.out.print("Enter choice: ");
            choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    checkBalance();
                    break;
                case 2:
                    deposit();
                    break;
                case 3:
                    withdraw();
                    break;
                case 4:
                    System.out.println("Thank you for using our ATM.");
                    break;
                default:
                    System.out.println("Invalid option.");
            }
        } while (choice != 4);
    }
    static void checkBalance() {
        System.out.println("Current balance: $" + balance);
    }
    static void deposit() {
        System.out.print("Enter amount to deposit: ");
        double amount = scanner.nextDouble();
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposit successful.");
        } else {
            System.out.println("Invalid amount.");
        }
    }

    static void withdraw() {
        System.out.print("Enter amount to withdraw: ");
        double amount = scanner.nextDouble();
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            System.out.println("Withdrawal successful.");
        } else {
            System.out.println("Invalid amount or insufficient balance.");
        }
    }
}
```



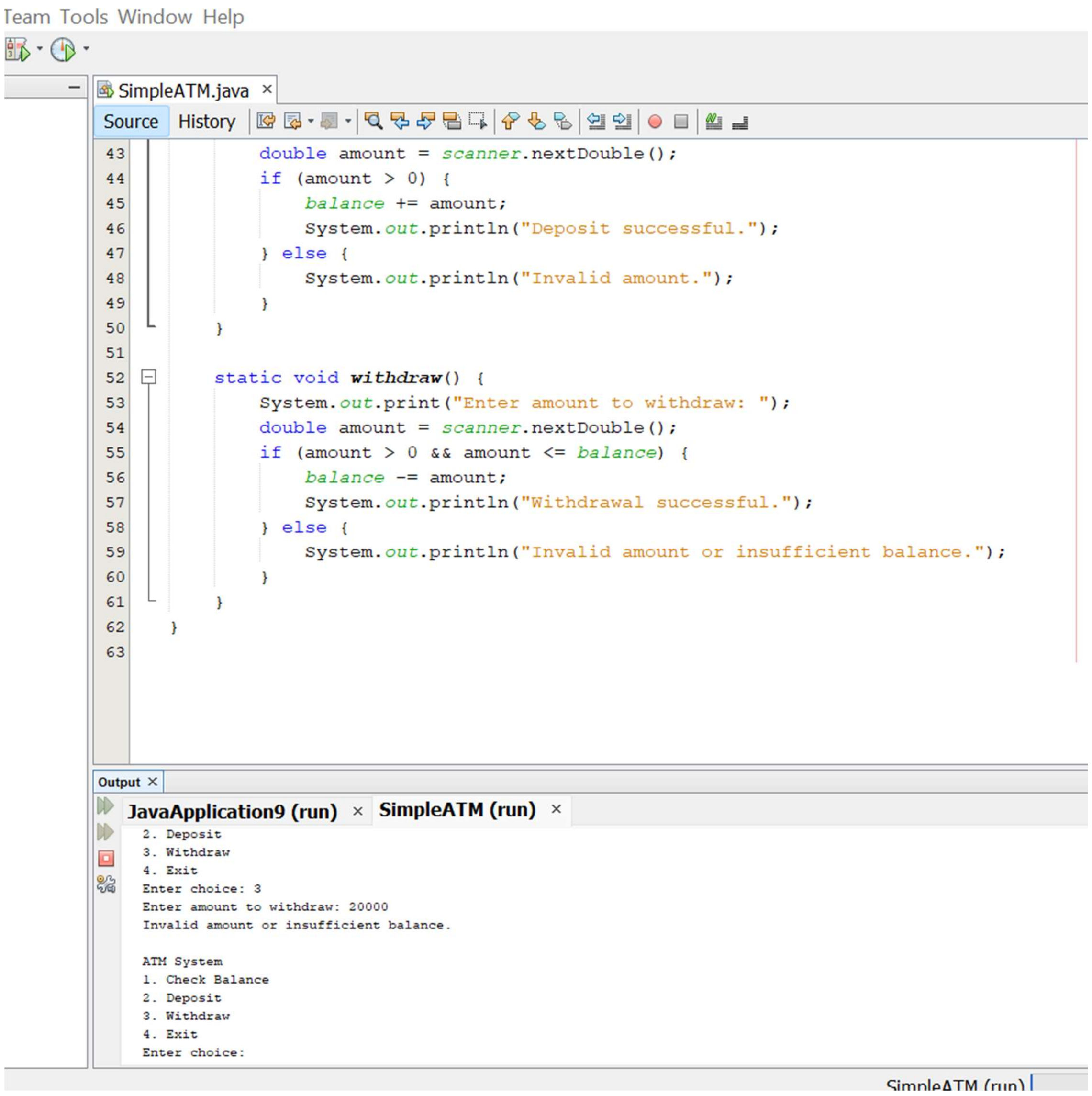
```java
import java.util.Scanner;

public class SimpleATM {
    static double balance = 1000.00; // Initial balance
    static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        int choice;
        do {
            System.out.println("\nATM System");
            System.out.println("1. Check Balance");
            System.out.println("2. Deposit");
            System.out.println("3. Withdraw");
            System.out.println("4. Exit");
            System.out.print("Enter choice: ");
            choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    checkBalance();
                    break;
                case 2:
                    deposit();
                    break;
                case 3:
```

```
2. Deposit
3. Withdraw
4. Exit
Enter choice: 3
Enter amount to withdraw: 20000
Invalid amount or insufficient balance.

ATM System
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Enter choice:
```



```java
                case 4:
                    System.out.println("Thank you for using our ATM.");
                    break;
                default:
                    System.out.println("Invalid option.");
            }
        } while (choice != 4);
    }

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

    static void deposit() {
        System.out.print("Enter amount to deposit: ");
        double amount = scanner.nextDouble();
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposit successful.");
        } else {
            System.out.println("Invalid amount.");
        }
    }

    static void withdraw() {
```

```
2. Deposit
3. Withdraw
4. Exit
Enter choice: 3
Enter amount to withdraw: 20000
Invalid amount or insufficient balance.

ATM System
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Enter choice:
```



```java
            double amount = scanner.nextDouble();
            if (amount > 0) {
                balance += amount;
                System.out.println("Deposit successful.");
            } else {
                System.out.println("Invalid amount.");
            }
        }

        static void withdraw() {
            System.out.print("Enter amount to withdraw: ");
            double amount = scanner.nextDouble();
            if (amount > 0 && amount <= balance) {
                balance -= amount;
                System.out.println("Withdrawal successful.");
            } else {
                System.out.println("Invalid amount or insufficient balance.");
            }
        }
    }
```

```
2. Deposit
3. Withdraw
4. Exit
Enter choice: 3
Enter amount to withdraw: 20000
Invalid amount or insufficient balance.

ATM System
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Enter choice:
```