CS4001 – Programming

Primitive Data Types in Java

Week 03: Workshop

Itahari International College

Academic Year 2025/26

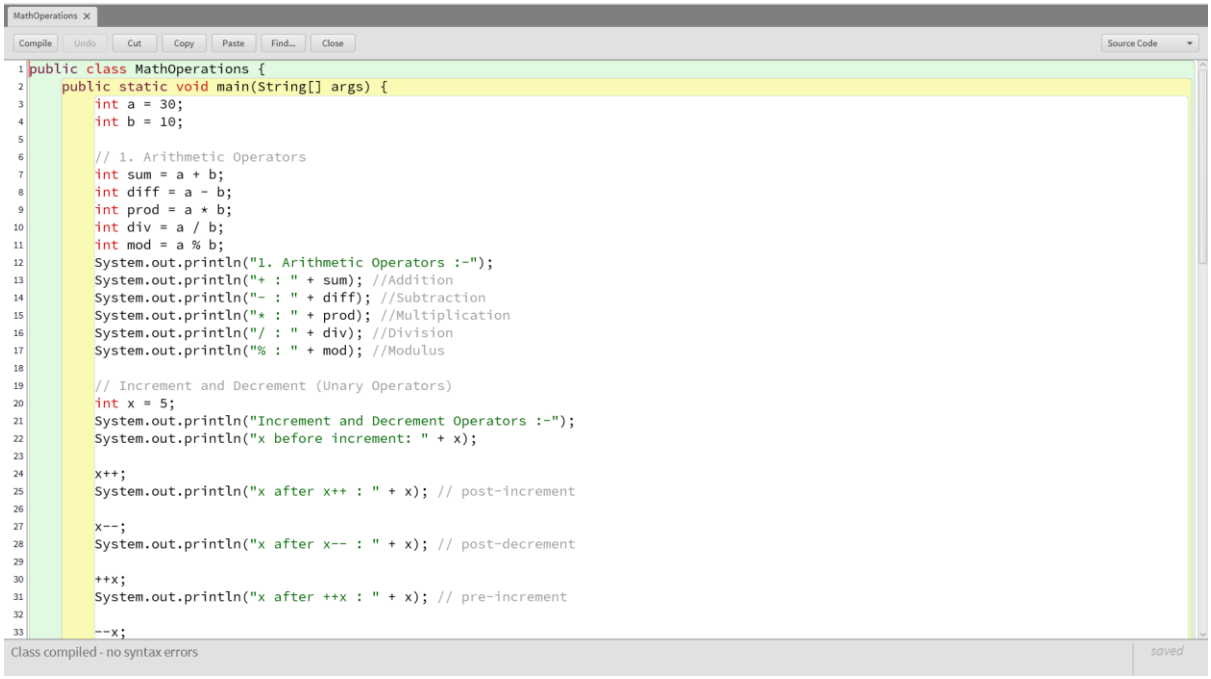
Question 1:

Create "MathOperations.java" with all operator types

Test case

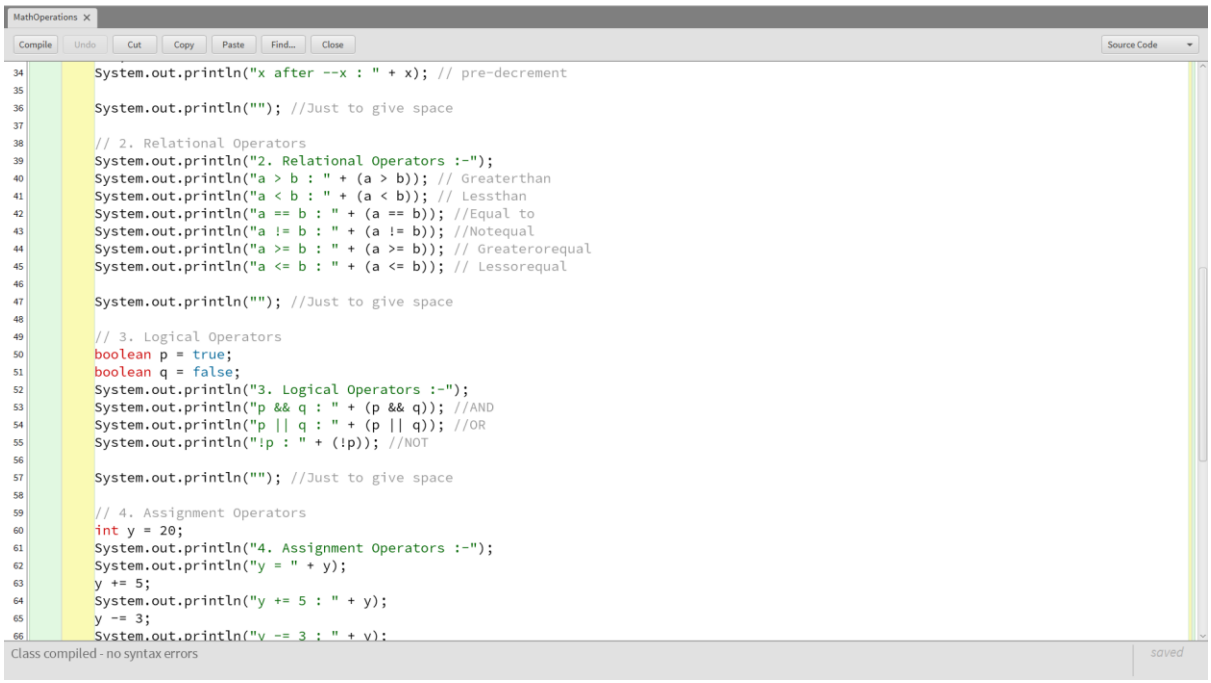
Objectives	To compile and execute the program using BlueJ
Action	• BlueJ was opened and the project containing the source code file MathOperations.java was loaded. • The program file MathOperations.java was compiled using the Compile button in BlueJ. • The complied class MathOperations was executed by right clicking the class and selecting the void main(String[] args) option.
Expected Result	The program should compile and display all operator types as programmed without any errors.
Actual Result	The program successfully compiles, and display operator types as programmed without any errors.
Conclusion	Test was successful

Screenshot of the code compilation




```
public class MathOperations {
    public static void main(String[] args) {
        int a = 30;
        int b = 10;

        // 1. Arithmetic Operators
        int sum = a + b;
        int diff = a - b;
        int prod = a * b;
        int div = a / b;
        int mod = a % b;
        System.out.println("1. Arithmetic Operators :-");
        System.out.println("+ : " + sum); //Addition
        System.out.println("- : " + diff); //Subtraction
        System.out.println("* : " + prod); //Multiplication
        System.out.println("/ : " + div); //Division
        System.out.println("% : " + mod); //Modulus

        // Increment and Decrement (Unary Operators)
        int x = 5;
        System.out.println("Increment and Decrement Operators :-");
        System.out.println("x before increment: " + x);

        x++;
        System.out.println("x after x++ : " + x); // post-increment

        x--;
        System.out.println("x after x-- : " + x); // post-decrement

        ++x;
        System.out.println("x after ++x : " + x); // pre-increment

        --x;
```

Figure 1 QN.1 Code Compilation (1)




```
        System.out.println("x after --x : " + x); // pre-decrement

        System.out.println(""); //Just to give space

        // 2. Relational Operators
        System.out.println("2. Relational Operators :-");
        System.out.println("a > b : " + (a > b)); // Greaterthan
        System.out.println("a < b : " + (a < b)); // Lessthan
        System.out.println("a == b : " + (a == b)); //Equal to
        System.out.println("a != b : " + (a != b)); //Notequal
        System.out.println("a >= b : " + (a >= b)); // Greateroequal
        System.out.println("a <= b : " + (a <= b)); // Lessorequal

        System.out.println(""); //Just to give space

        // 3. Logical Operators
        boolean p = true;
        boolean q = false;
        System.out.println("3. Logical Operators :-");
        System.out.println("p && q : " + (p && q)); //AND
        System.out.println("p || q : " + (p || q)); //OR
        System.out.println("!p : " + (!p)); //NOT

        System.out.println(""); //Just to give space

        // 4. Assignment Operators
        int y = 20;
        System.out.println("4. Assignment Operators :-");
        System.out.println("y = " + y);
        y += 5;
        System.out.println("y += 5 : " + y);
        y -= 3;
        System.out.println("y -= 3 : " + y);
```

Figure 2 QN.1 Code Compilation (2)

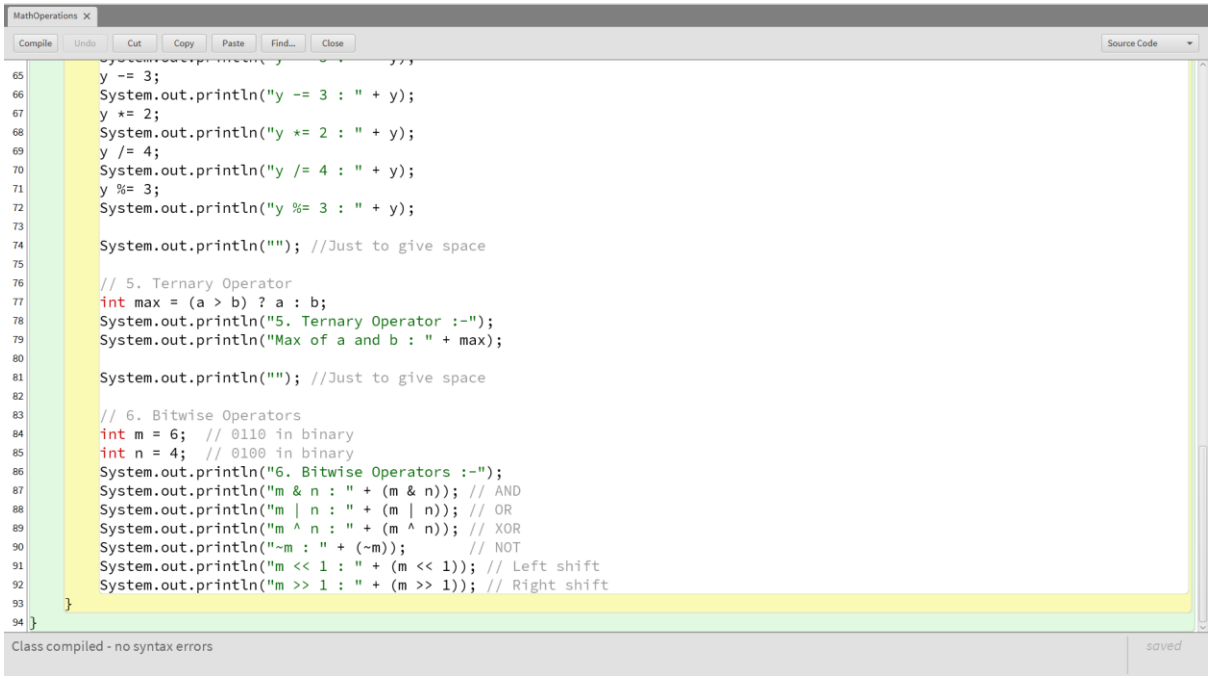


Figure 3 QN.1 Code Compilation (3)

Screenshot of the Output

Figure 4 QN.1 Output Display

Question 2: GradeEvaluator.java

Create a program that:

- Takes a numeric grade as input
- Uses the ternary operator to assign:
- "Pass" if grade ≥ 40
- "Fail" if grade < 40
  Format output using escape sequences

Test case

Objectives	To compile and execute the program using BlueJ
Action	• BlueJ was opened and the project containing the source code file GradeEvaluator.java was loaded. • The program file GradeEvaluator.java was compiled using the Compile button in BlueJ. • The complied class GradeEvaluator was executed by right clicking the class and selecting the void main(String[] args) option.
Expected Result	The program should compile and ask the user a grade and display the result (i.e. Pass/Fail) without any errors.
Actual Result	The program Successfully compile and ask user a grade and display the result (i.e. Pass/Fail) without any errors.
Conclusion	Test was successful

Screenshot of the code compilation

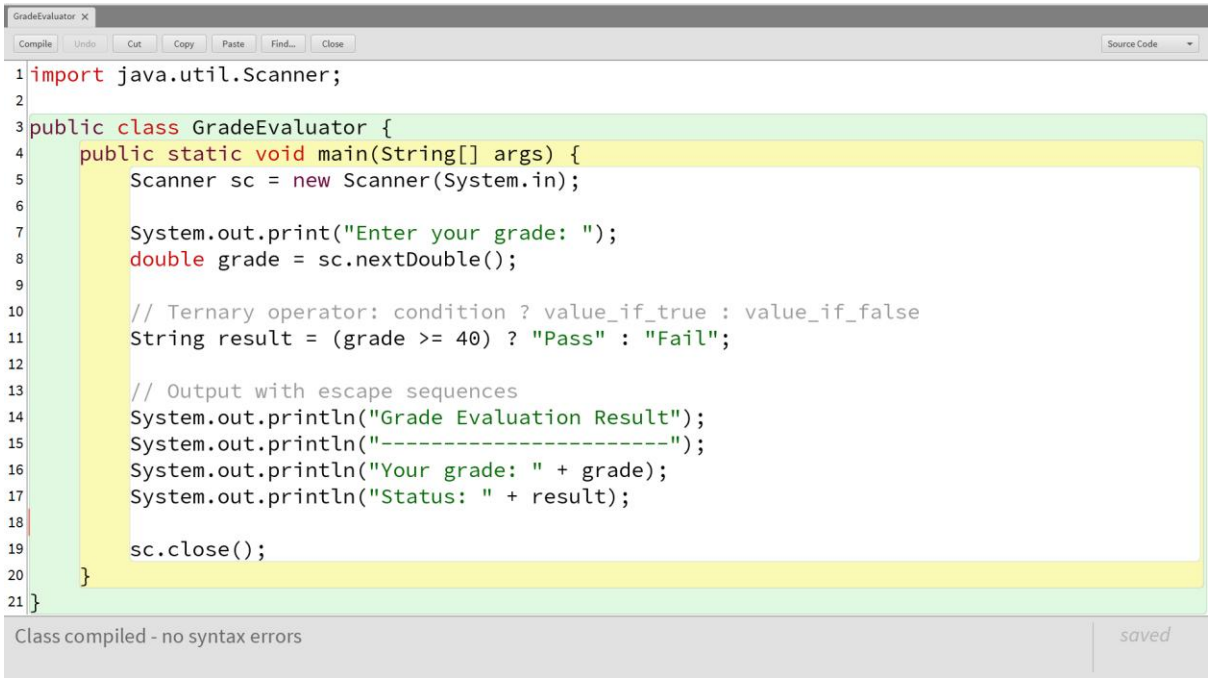


Figure 5 QN.2 Code Compilation

Screenshot of the Output

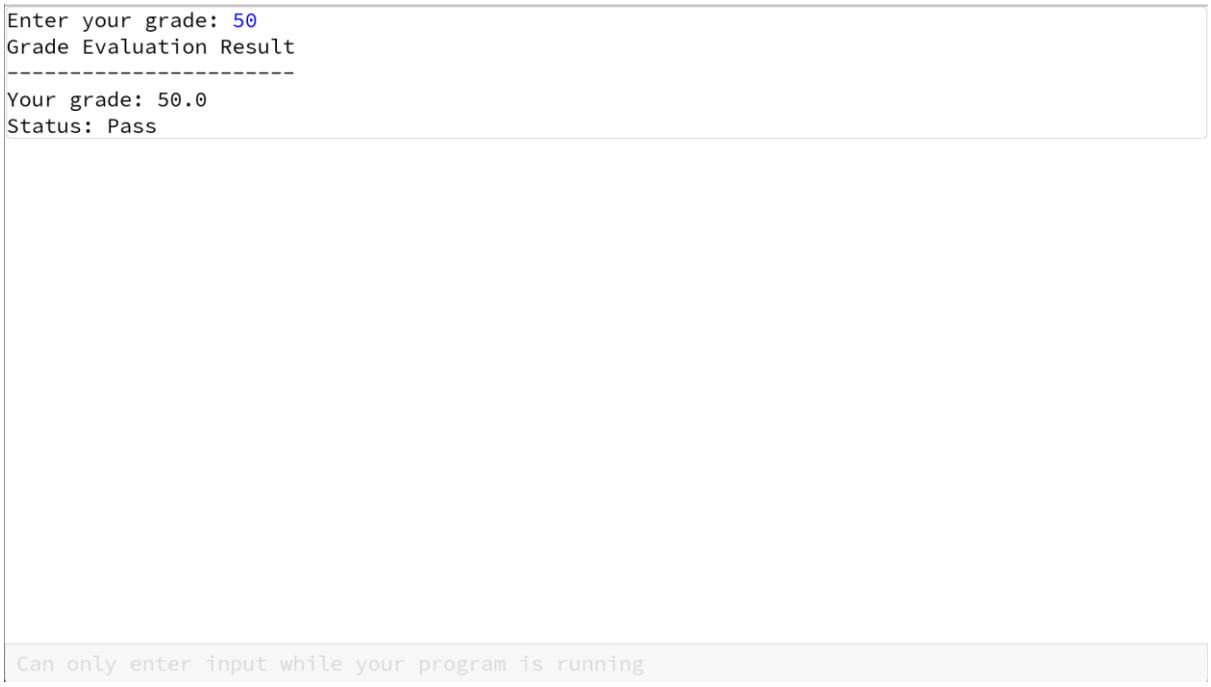


Figure 6 QN.2 Output Display

Question 3: Data Type Inspector

Create a Java program named DataTypeInspector.java that:

- Declares and initializes a variable for each of Java's 8 primitive data types.
- Uses appropriate literal values for initialization.
- Prints the value of each variable to the console, each with a descriptive label.

Test case

Objectives	To compile and execute the program using BlueJ
Action	• BlueJ was opened and the project containing the source code file DataTypeInspector.java was loaded. • The program file DataTypeInspector.java was compiled using the Compile button in BlueJ. • The complied class DataTypeInspector was executed by right clicking the class and selecting the void main(String[] args) option.
Expected Result	The program should compile and display the value of each variable as instructed without any errors.
Actual Result	The program successfully compiles and displays the value of each variable as instructed without any errors.
Conclusion	Test was successful

Screenshot of the code compilation

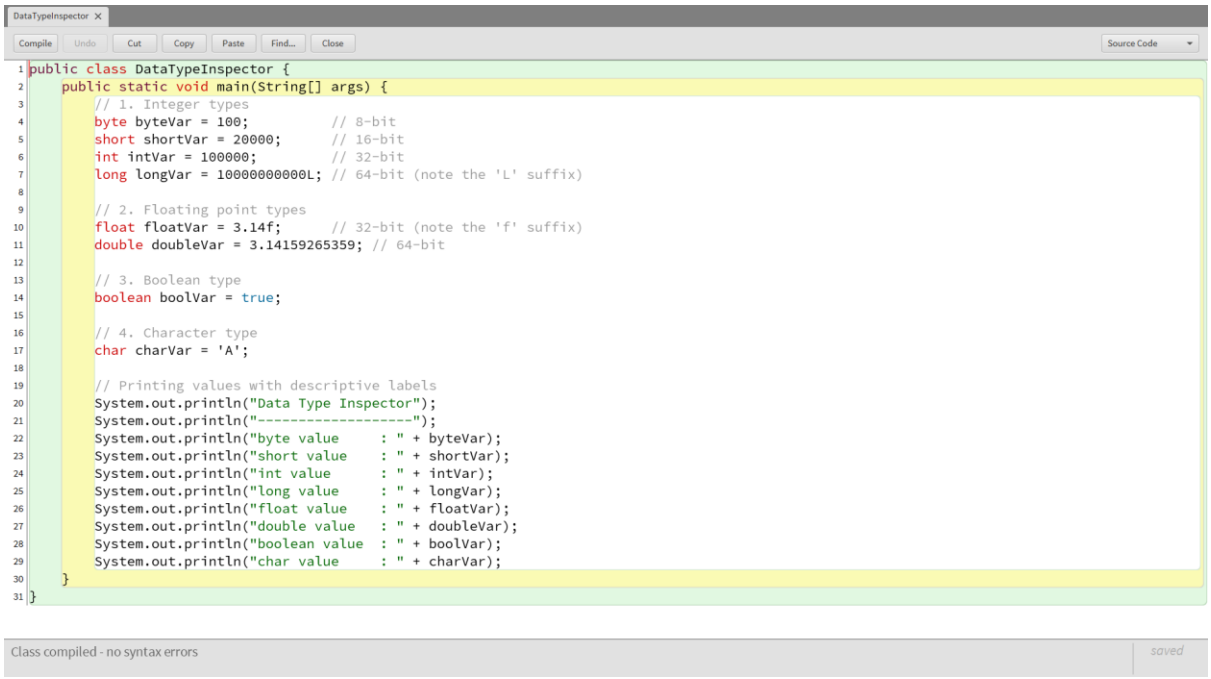


Figure 7 QN.3 Code Compilation

Screenshot of the Output

Figure 8 QN.3 Output Display

Question 4: Default Value Checker

Create a Java class named DefaultValues.java.

- Declare member variables (fields) for all 8 primitive types without initializing them.
- In the main method, create an instance of the class and print the value of each field.
- Add a comment explaining why this wouldn't work for local variables.

Test case

Objectives	To compile and execute the program using BlueJ
Action	• BlueJ was opened and the project containing the source code file DefaultValues.java was loaded. • The program file DefaultValues.java was compiled using the Compile button in BlueJ. • The complied class DefaultValues was executed by right clicking the class and selecting the void main(String[] args) option.
Expected Result	The program should compile and display the value as instructed without any errors.
Actual Result	The program successfully compiles and displays the value as instructed without any errors.
Conclusion	Test was successful

Screenshot of the code compilation

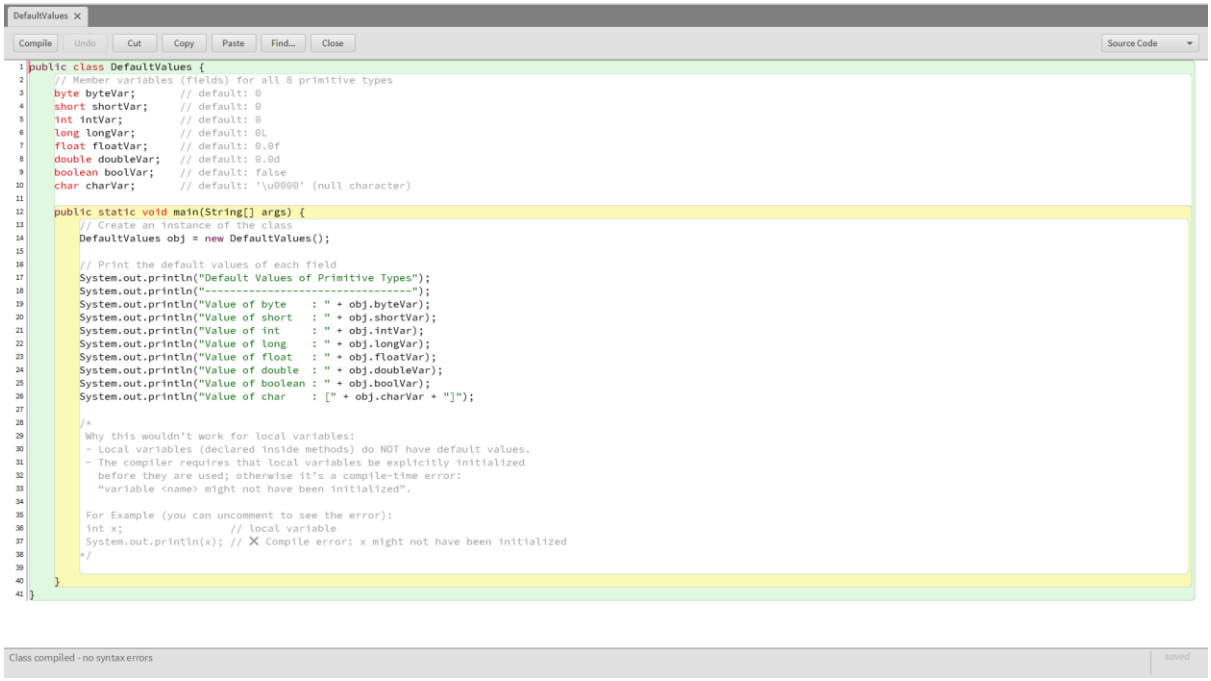


Figure 9 QN.4 Code Compilation

Screenshot of the Output

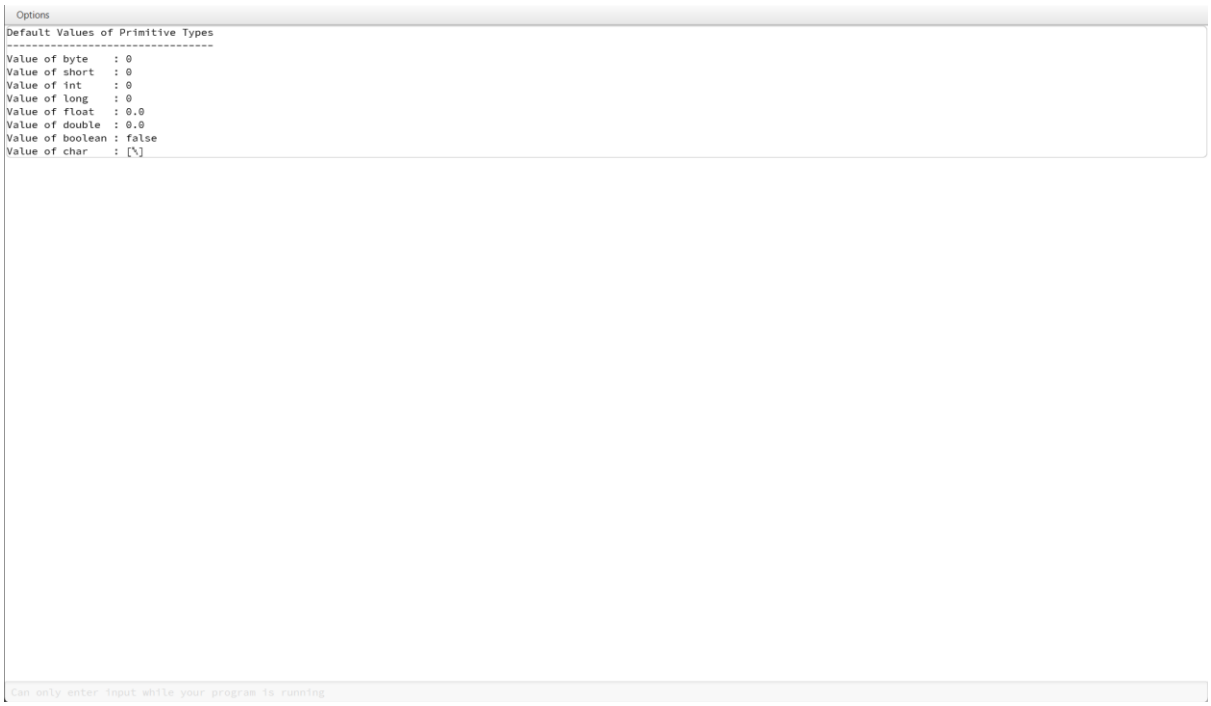


Figure 10 QN.4 Output Display

Question 5: Literal Practice

Create a program named LiteralPractice.java that demonstrates the use of specific literals:

- A long variable initialized with a value requiring the 'L' suffix.
- A float variable initialized with a value requiring the 'f' suffix.
- A char variable initialized using a Unicode escape sequence (e.g., for the copyright symbol ©).
- Print the value of each variable.

Test case

Objectives	To compile and execute the program using BlueJ
Action	• BlueJ was opened and the project containing the source code file LiteralPractice.java was loaded. • The program file LiteralPractice.java was compiled using the Compile button in BlueJ. • The complied class LiteralPractice was executed by right clicking the class and selecting the void main(String[] args) option.
Expected Result	The program should compile and print the value as instructed without any errors.
Actual Result	The program successfully compiles and prints the value as instructed without any errors.
Conclusion	Test was successful

Screenshot of the code compilation

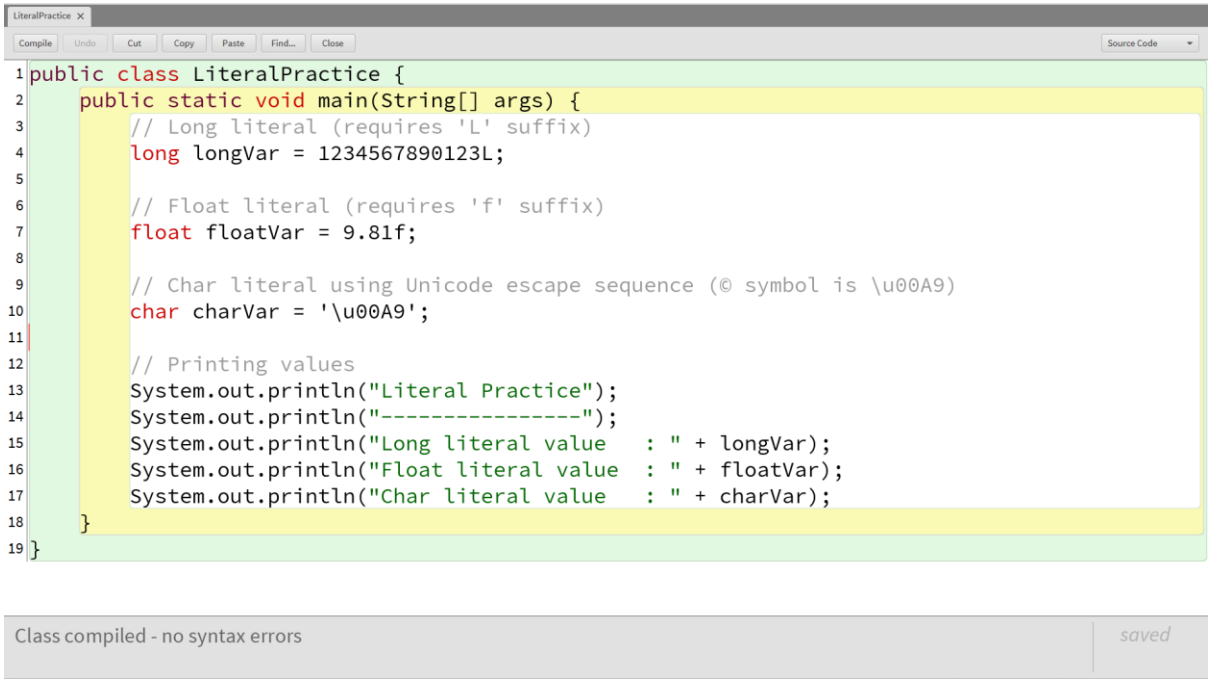


Figure 11 QN.5 Code Compilation

Screenshot of the Output

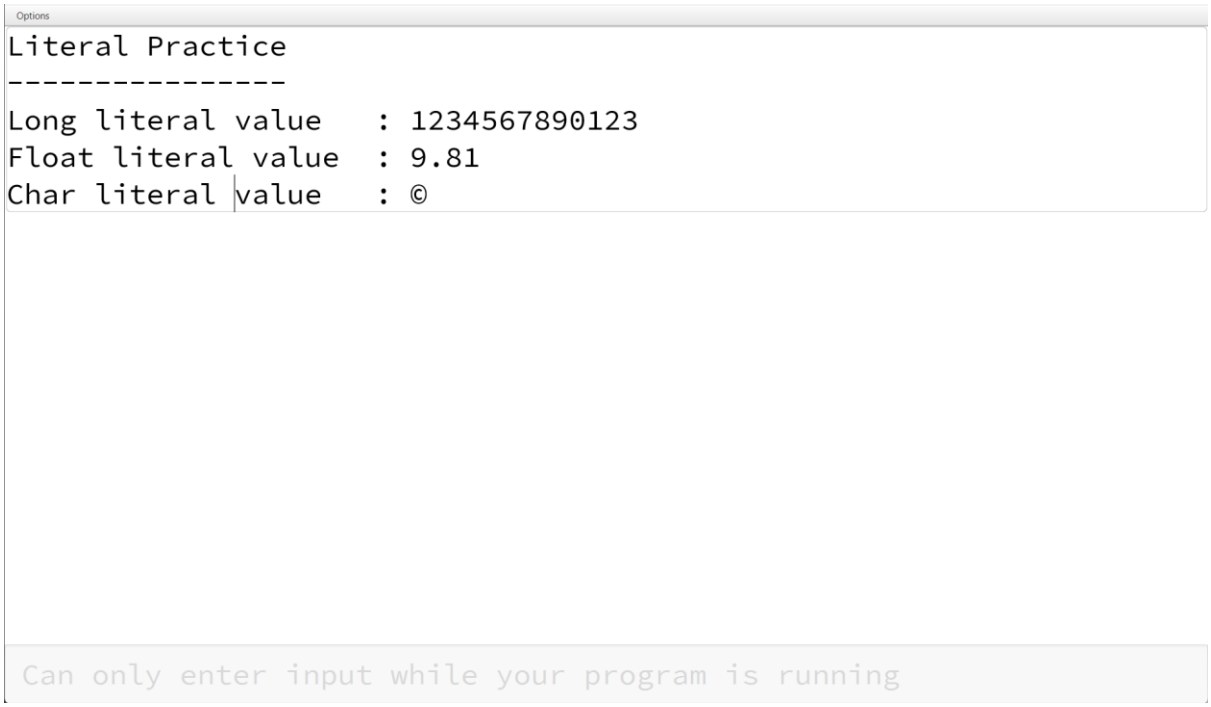


Figure 12 QN.5 Output Display

Scenario Question

Context

A local rickshaw service in Biratnagar needs a simple tool to calculate fares for their customers. The fare calculation has a few components: a base fare, a per-kilometer charge, and a per-minute charge. They also offer discounts for locals on long distances and have a surcharge for night-time travel.

Problem

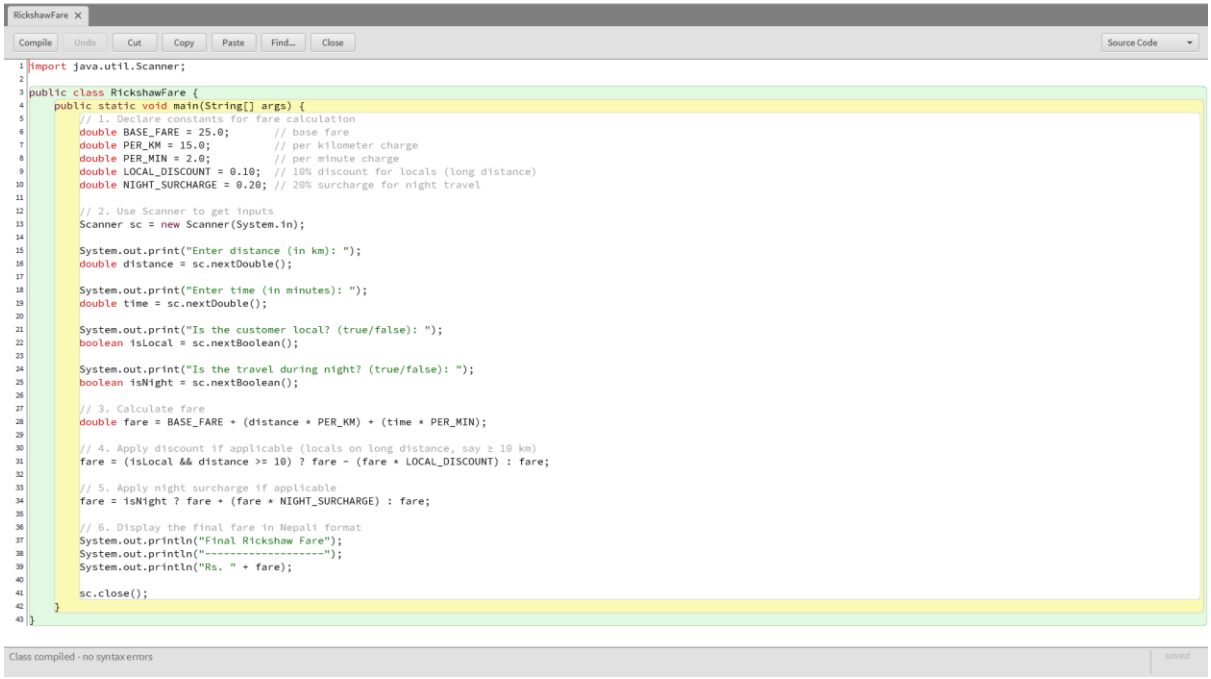
The rickshaw drivers need a program that can:

- Take distance (in km) and time (in minutes) as input.
- Ask if the customer is a local and if the travel is during the night. (Hint: use ternary operator)
- Calculate the total fare based on the rules.
- Display the final fare in a clear, Nepali format (e.g., "Rs. 550")

Test case

Objectives	To compile and execute the program using BlueJ
Action	• BlueJ was opened and the project containing the source code file RickshawFare.java was loaded. • The program file RickshawFare.java was compiled using the Compile button in BlueJ. • The complied class RickshawFare was executed by right clicking the class and selecting the void main(String[] args) option.
Expected Result	The program should compile and display the Final Rickshaw fare without any errors.
Actual Result	The program successfully compiles and displays the Final Rickshaw fare without any errors.
Conclusion	Test was successful

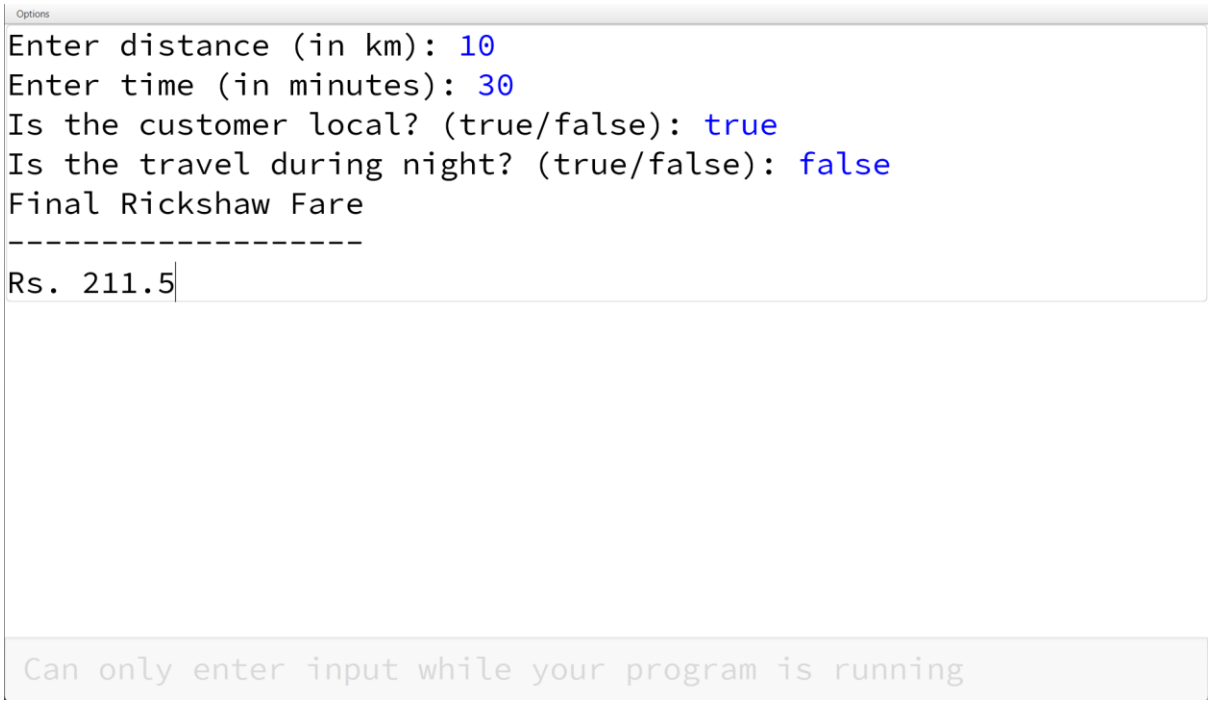
Screenshot of the code compilation



```
import java.util.Scanner;

public class RickshawFare {
    public static void main(String[] args) {
        // 1. Declare constants for fare calculation
        double BASE_FARE = 25.0;          // base fare
        double PER_KM = 15.0;             // per kilometer charge
        double PER_MIN = 2.0;             // per minute charge
        double LOCAL_DISCOUNT = 0.10;  // 10% discount for locals (long distance)
        double NIGHT_SURCHARGE = 0.20; // 20% surcharge for night travel

        // 2. Use Scanner to get inputs
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter distance (in km): ");
        double distance = sc.nextDouble();

        System.out.print("Enter time (in minutes): ");
        double time = sc.nextDouble();

        System.out.print("Is the customer local? (true/false): ");
        boolean isLocal = sc.nextBoolean();

        System.out.print("Is the travel during night? (true/false): ");
        boolean isNight = sc.nextBoolean();

        // 3. Calculate fare
        double fare = BASE_FARE + (distance * PER_KM) + (time * PER_MIN);

        // 4. Apply discount if applicable (locals on long distance, say ≥ 10 km)
        fare = (isLocal && distance >= 10) ? fare - (fare * LOCAL_DISCOUNT) : fare;

        // 5. Apply night surcharge if applicable
        fare = isNight ? fare + (fare * NIGHT_SURCHARGE) : fare;

        // 6. Display the final fare in Nepali format
        System.out.println("Final Rickshaw Fare");
        System.out.println("-------------------");
        System.out.println("Rs. " + fare);

        sc.close();
    }
}
```

Figure 13 QN.6 Code Compilation

Screenshot of the Output



```
Enter distance (in km): 10
Enter time (in minutes): 30
Is the customer local? (true/false): true
Is the travel during night? (true/false): false
Final Rickshaw Fare
-------------------
Rs. 211.5
```

Figure 14 QN.6 Output Display