2024

[OUbs033213] OO Programming

Doosan PAGOOAH

Learner ID: 202306292

10/30/2024

Table of Contents

PART A

Description of the Application:

The ***MauriBankLoanCalculator*** is a Java-based console application that helps users calculate the monthly payments for different types of loans: Home Loan, Car Loan, and Personal Loan. The application collects user inputs such as age, loan amount, and loan term, and based on these details, calculates the monthly payment using the specific interest rate associated with each loan type. The results are then displayed in an ASCII art table for better readability.

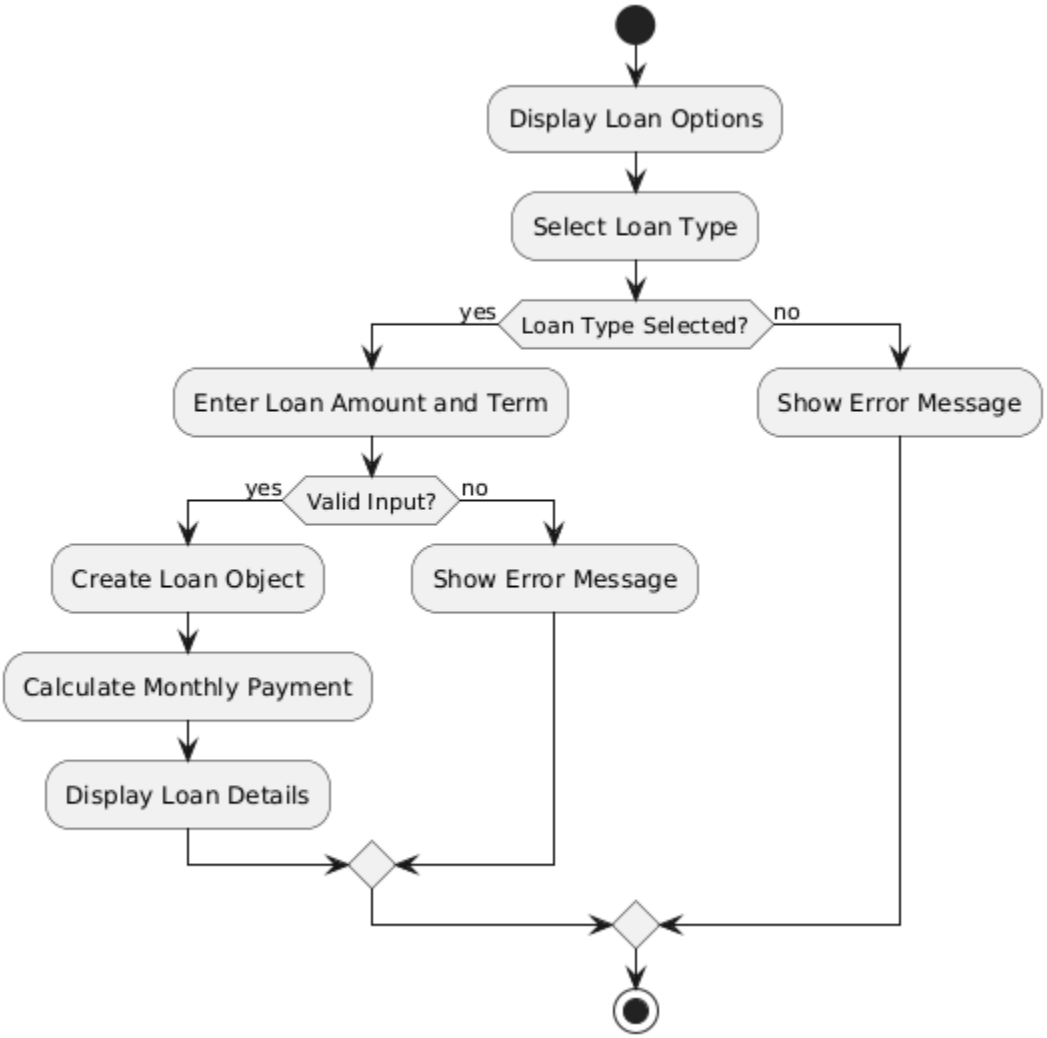


Figure 1: Activity diagram for *MauriBankLoanCalculator*

As depicted in Figure 1, the loan selection process starts by displaying loan options to the user. The user then selects a loan type, and the system immediately checks if the selection is valid. If the loan type is valid, the user is prompt to enter the loan amount and the loan term. After the user inputs these details, the program verifies if the input is valid. When everything checks out, it creates the loan object based on the user's choices, calculate the monthly payment, and display all the relevant loan details. However, if the user selects an invalid loan type or enters incorrect input, it displays an error message and guide the user back to the appropriate step to correct their input. This ensures the process flows smoothly while handling any mistakes efficiently.

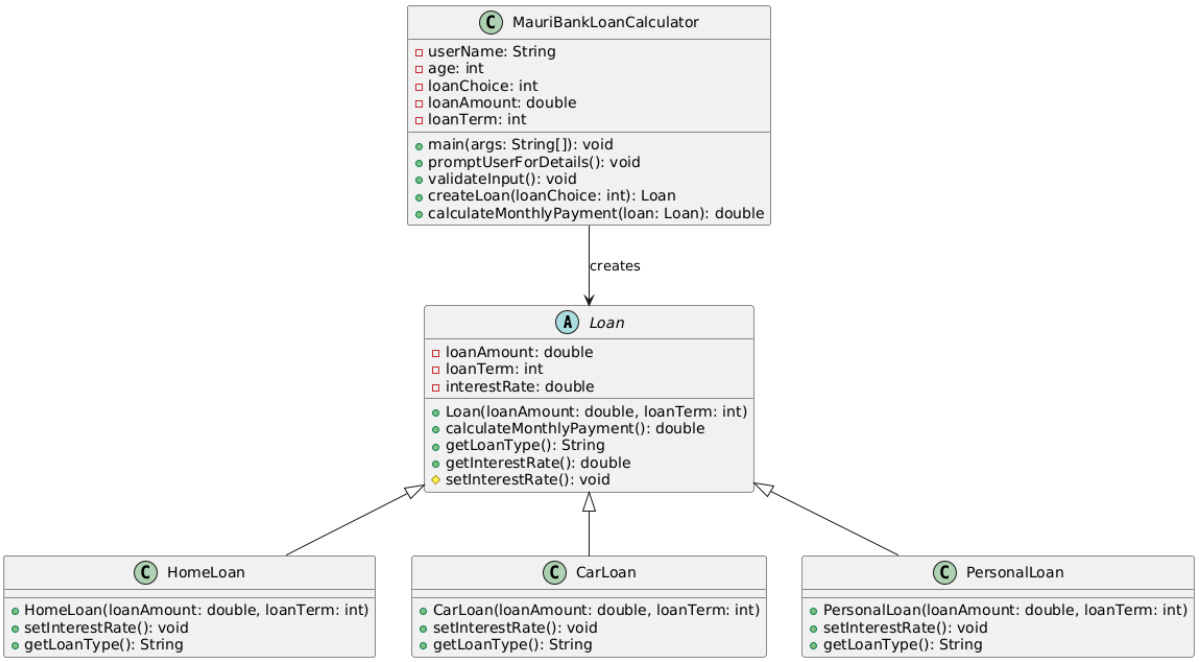


Figure 2: Class diagram for *MauriBankLoanCalculator*

The class diagram depicts the structure of a loan calculation system, where the main driver class is ***MauriBankLoanCalculator***, which interacts with an abstract class ***Loan*** and its concrete subclasses *HomeLoan*, *CarLoan*, and *PersonalLoan*. Figure 2 illustrates how the system encapsulates loan-related functionality and handles different loan types using inheritance and polymorphism.

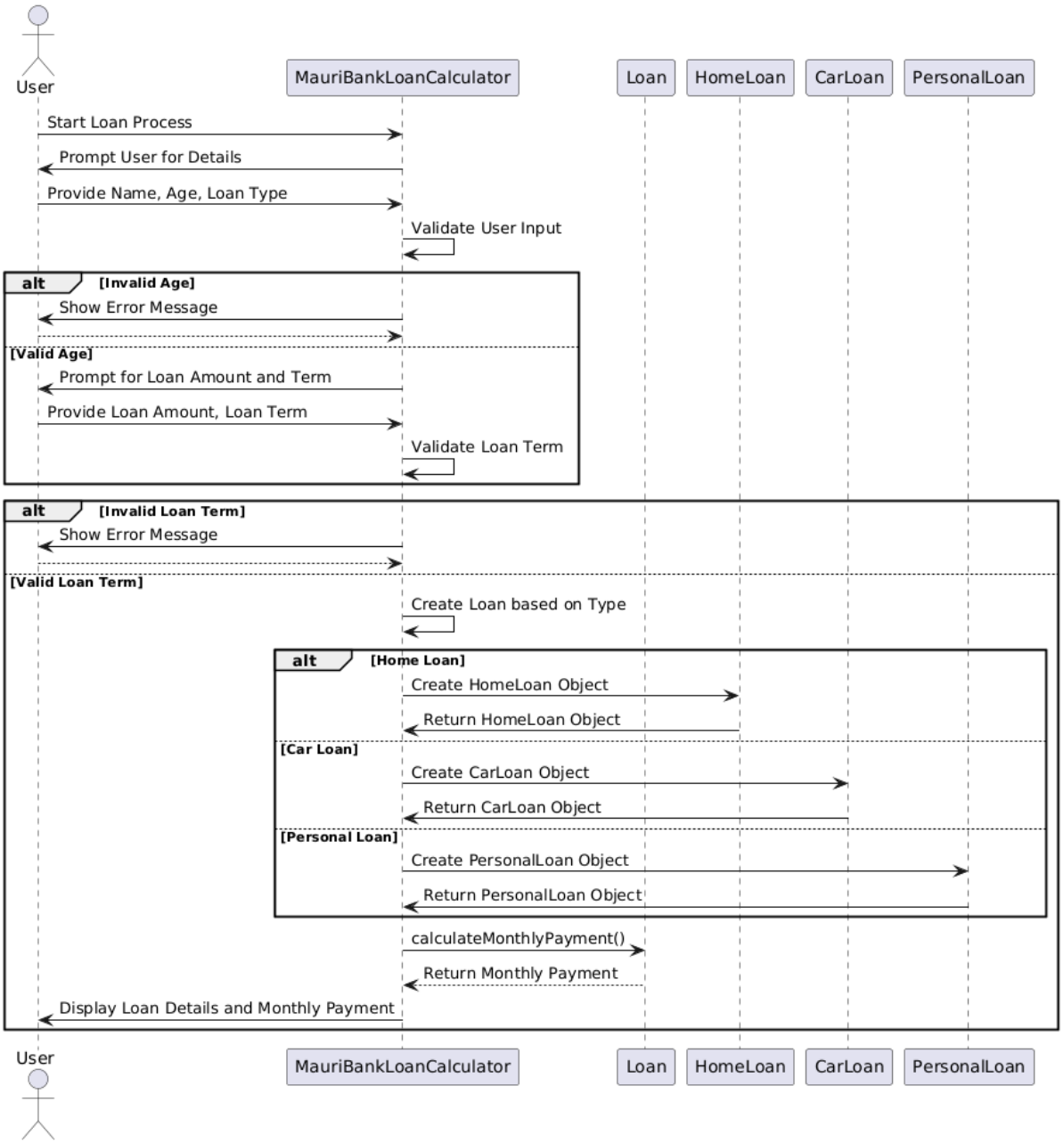


Figure 3: Sequence diagram for *MauriBankLoanCalculator*

Figure 3 shows how the ***MauriBankLoanCalculator*** interacts with the user and the loan classes (*Loan*, *HomeLoan*, *CarLoan*, *PersonalLoan*) during the loan process. It further illustrates how the system handles input validation (for both age and loan term) and dynamically creates a loan object based on the user's selection. It also shows how different loan types are processed and how the system calculates and displays the final monthly payment.

Object-Oriented Programming (OOP) Features in the Application:

1. Abstraction:

• **Definition**: Abstraction focuses on hiding complex implementation details and exposing only the necessary aspects. Java supports abstraction using abstract classes and interfaces (NUS, 2023).

• **Application in the Code**: The ***Loan*** class is an abstract class that provides the basic structure for different types of loans. It contains attributes and methods that are common to all loan types (such as ***loanAmount*** , ***loanTerm*** and ***calculateMonthlyPayment()***), but the specific details, like setting the interest rate and the loan type, are abstracted and left to the subclasses (e.g., ***HomeLoan***, ***CarLoan*** , and ***PersonalLoan***).

```
public abstract class Loan {
        protected double loanAmount;
        protected int loanTerm; // in years
        protected double interestRate;
        public Loan(double loanAmount, int loanTerm) {
            this.loanAmount = loanAmount;
            this.loanTerm = loanTerm;
            setInterestRate(); // Interest rate set by the bank for each loan type
        }

        // Abstract method implemented by subclasses
        protected abstract void setInterestRate();
        public abstract String getLoanType();
    }
```

2. Encapsulation:

• **Definition**: Encapsulation groups data and methods operating on that data together, restricting access to the inner workings of an object (NUS, 2023)

• **Application in the Code**: The fields ***loanAmount*** , ***loanTerm*** , and ***interestRate*** are declared as protected in the Loan class. The users of the class cannot directly access or modify these fields. Instead, they interact with the class through public methods like ***getInterestRate()*** and ***calculateMonthlyPayment()*** , which control how the internal data is accessed and modified.

```
protected double loanAmount;
  protected int loanTerm;
  protected double interestRate;

  public double getInterestRate() {
      return interestRate;
  }
```

3. Inheritance:

• **Definition**: Inheritance allows a class to inherit methods and properties from another class, facilitating code reusability (Brusca, 2023).

• **Application in the Code**: The classes ***HomeLoan*** , ***CarLoan*** , and ***PersonalLoan*** all inherit from the abstract Loan class. They reuse the common functionality provided by the ***Loan*** class (such as calculating monthly payments) and override specific behaviours like setting the interest rate and defining the loan type.

• This approach reduces code duplication since all loans share common features like loan amount and term but differ in their specific characteristics (e.g., interest rates).

```
class HomeLoan extends Loan {
    public HomeLoan(double loanAmount, int loanTerm) {
        super(loanAmount, loanTerm);
    }

    protected void setInterestRate() {
        this.interestRate = 5.0; // interest rate for home loan
    }

    public String getLoanType() {
        return "Home Loan";
    }

```

4. Polymorphism:

• **Definition**: Polymorphism enables objects to take multiple forms, allowing the same interface to be used for different data types (Brusca, 2023).

• **Application in the Code**: The Loan object is polymorphic, meaning it can reference objects of any subclass (e.g., ***HomeLoan*** , ***CarLoan*** , or ***PersonalLoan***). The specific implementation of ***setInterestRate()*** and ***getLoanType()*** is chosen at runtime based on the type of loan object created. For example, when the user selects a loan type, the system dynamically creates the corresponding loan object (either ***HomeLoan*** , ***CarLoan*** , or ***PersonalLoan***) and calculates the monthly payment using the overridden methods of that specific subclass.

```
Loan loan = null;
  switch (loanChoice) {
      case 1:
          loan = new HomeLoan(loanAmount, loanTerm);
          break;
      case 2:
          loan = new CarLoan(loanAmount, loanTerm);
          break;
      case 3:
          loan = new PersonalLoan(loanAmount, loanTerm);
          break;
  }
```

5. Method Overriding:

• **Definition**: Method-overriding occurs when a subclass provides a specific implementation for a method in its superclass to achieve runtime polymorphism (Brusca, 2023).

• **Application in the Code**: The ***setInterestRate***() and ***getLoanType***() methods are overridden in each subclass (***HomeLoan*** , ***CarLoan*** , and ***PersonalLoan***). Each subclass provides its own implementation of these methods based on the specific type of loan.

```

    protected void setInterestRate() {
        this.interestRate = 5.0; // Bank-defined interest rate for home loan
    }


    public String getLoanType() {
        return "Home Loan";
    }

```

6. Error Handling:

• **Definition**: Java handles runtime exceptions (error handling) using try-catch-finally blocks, ensuring smooth program execution (NUS, 2023).

• **Application in the Code**: The program uses ***try-catch*** blocks to handle exceptions like ***InputMismatchException*** (for invalid data types) and ***IllegalArgumentException*** (for invalid loan terms or loan amounts). This ensures the program can gracefully handle errors and give feedback to the user when incorrect input is provided.

```
try {
        // Input and processing code
    } catch (InputMismatchException e) {
        System.out.println("Invalid input. Please enter the correct data type.");
    } catch (IllegalArgumentException e) {
        System.out.println(e.getMessage());
    }

```

Summary of OOP Principles Applied:

1. **Abstraction**: The abstract class ***Loan*** provides a general structure for different loan types, leaving specific details to the subclasses.
2. **Encapsulation**: The class restricts direct access to its data and exposes it via public methods.
3. **Inheritance**: ***HomeLoan*** , ***CarLoan*** , and ***PersonalLoan*** inherit from Loan to reuse the common loan logic.
4. **Polymorphism**: A ***Loan*** object can reference any loan subclass, and method calls are resolved based on the actual object type at runtime.
5. **Method Overriding**: Subclasses provide their own implementation of certain methods like ***setInterestRate()*** and ***getLoanType()*** .
6. **Error Handling**: The program handles invalid inputs and exceptions gracefully, ensuring robust and user-friendly behaviour.

PART B

Loan.java

```java
// Base Loan class (abstraction and inheritance)
public abstract class Loan {
    protected double loanAmount;
    protected int loanTerm; // in years
    protected double interestRate;

    public Loan(double loanAmount, int loanTerm) {
        this.loanAmount = loanAmount;
        this.loanTerm = loanTerm;
        setInterestRate(); // Interest rate is set by the bank for each loan type
    }

    // Set interest rate
    protected abstract void setInterestRate();

    // Method to calculate monthly payment
    public double calculateMonthlyPayment() {
        int totalMonths = loanTerm * 12;
        double monthlyInterestRate = (interestRate / 100) / 12;
        return (loanAmount * monthlyInterestRate * Math.pow(1 + monthlyInterestRate, totalMonths)) /
                (Math.pow(1 + monthlyInterestRate, totalMonths) - 1);
    }

    // Getter for loan type
    public abstract String getLoanType();

    // Getter for interest rate to display it to the user
    public double getInterestRate() {
        return interestRate;
    }
}
```

MauriBankLoanCalculator.java

```java
import java.util.InputMismatchException;
import java.util.Scanner;

// Home Loan class (inheritance and polymorphism)
class HomeLoan extends Loan {
    public HomeLoan(double loanAmount, int loanTerm) {
        super(loanAmount, loanTerm);
    }


    protected void setInterestRate() {
        this.interestRate = 5.0; // Bank-defined interest rate for home loan
    }


    public String getLoanType() {
        return "Home Loan";
    }
}

// Car Loan class (inheritance and polymorphism)
class CarLoan extends Loan {
    public CarLoan(double loanAmount, int loanTerm) {
        super(loanAmount, loanTerm);
    }


    protected void setInterestRate() {
        this.interestRate = 6.5; // Bank-defined interest rate for car loan
    }


    public String getLoanType() {
        return "Car Loan";
    }
}

// Personal Loan class (inheritance and polymorphism)
class PersonalLoan extends Loan {
    public PersonalLoan(double loanAmount, int loanTerm) {
        super(loanAmount, loanTerm);
    }


    protected void setInterestRate() {
        this.interestRate = 8.0; // interest rate for personal loan
    }
```

```java


    public String getLoanType() {
        return "Personal Loan";
    }
}

// Main application class
public class MauriBankLoanCalculator {

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

        try {
            // User inputs
            System.out.println("Welcome to MauriBank Loan Department!");

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

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

            // Check if user age is valid for loan eligibility
            if (age < 18 || age > 65) {
                throw new IllegalArgumentException("You must be between 18 and 65 years old to be eligible for a loan.");
            }

            // Calculate maximum loan term based on age (up to 65 years)
            int maxLoanTerm = 65 - age;

            System.out.println("Hello " + userName + ", please choose a loan type:");
            System.out.println("1. Home Loan");
            System.out.println("2. Car Loan");
            System.out.println("3. Personal Loan");

            int loanChoice = scanner.nextInt();

            if (loanChoice < 1 || loanChoice > 3) {
                throw new IllegalArgumentException("Invalid loan type selected. Please choose a valid option (1, 2, or 3).");
            }

            System.out.print("Enter the loan amount you wish to take: ");
            double loanAmount = scanner.nextDouble();

            if (loanAmount <= 0) {
```

```
                throw new IllegalArgumentException("Loan amount must be greater than zero.");
            }

            System.out.printf("Enter the loan term in years (maximum %d years): ", maxLoanTerm);
            int loanTerm = scanner.nextInt();

                // Validate the loan term entered by the user
                if (loanTerm <= 0 || loanTerm > maxLoanTerm) {
                    throw new IllegalArgumentException(
                            String.format("Invalid loan term. You can only repay your loan over a maximum of %d years.", maxLoanTerm));
                }

                Loan loan = null;

                // Determine loan type based on user's choice
                switch (loanChoice) {
                    case 1:
                        loan = new HomeLoan(loanAmount, loanTerm);
                        break;
                    case 2:
                        loan = new CarLoan(loanAmount, loanTerm);
                        break;
                    case 3:
                        loan = new PersonalLoan(loanAmount, loanTerm);
                        break;
                }

                // Display loan details including interest rate and calculate monthly payment
                double monthlyPayment = loan.calculateMonthlyPayment();

                // ASCII Art Table to Display Loan Details
                System.out.println("+----------------+--------------------+");
                System.out.println("| Loan Detail    | Value              |");
                System.out.println("+----------------+--------------------+");
                System.out.printf("| Loan Type      | %-20s |\n", loan.getLoanType());
                System.out.println("+----------------+--------------------+");
                System.out.printf("| Loan Amount    | %.2f               |\n", loanAmount);
                System.out.println("+----------------+--------------------+");
```

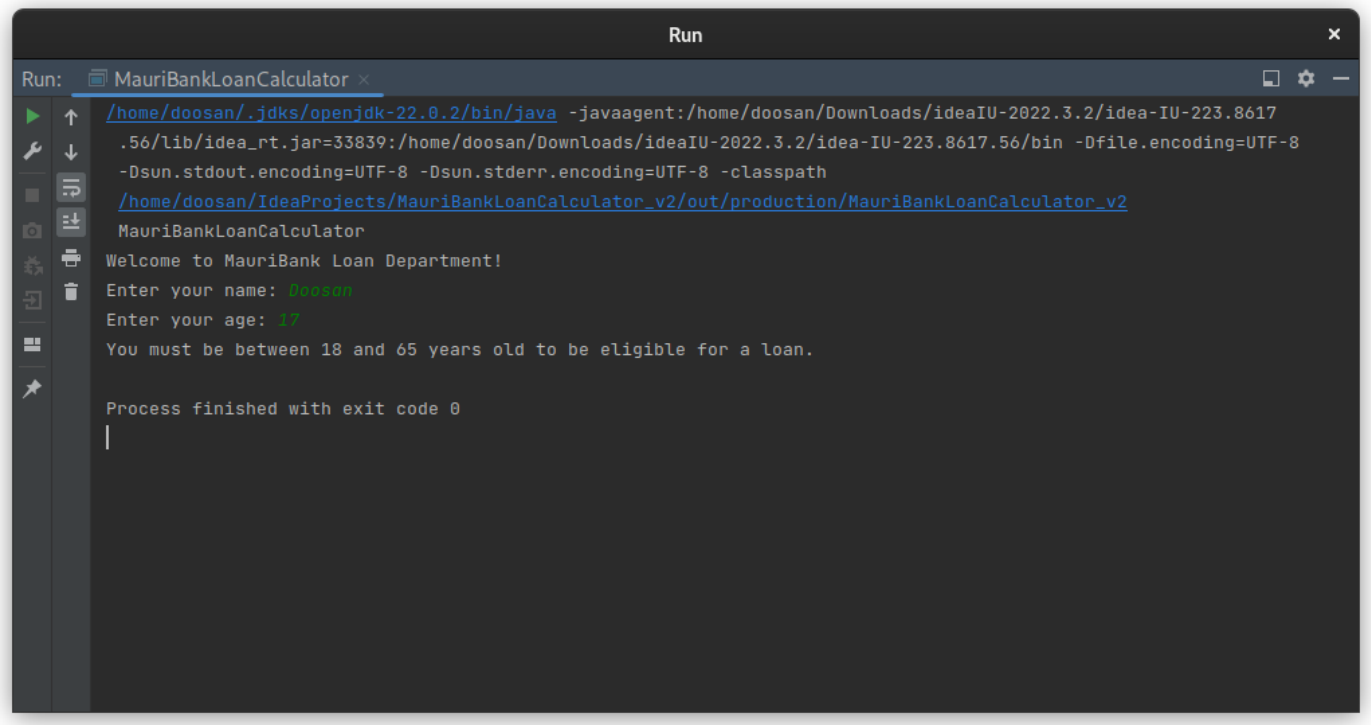
```
                System.out.printf("| Loan Term      | %d years      |\n", loanTerm);
                System.out.println("+----------------+--------------------+");
                System.out.printf("| Interest Rate  | %.2f%%        |\n", loan.getInterestRate());
                System.out.println("+----------------+--------------------+");
                System.out.printf("| Monthly Payment| %.2f               |\n", monthlyPayment);
                System.out.println("+----------------+--------------------+");

            } catch (InputMismatchException e) {
                System.out.println("Invalid input. Please enter the correct data type.");
            } catch (IllegalArgumentException e) {
                System.out.println(e.getMessage());
            } finally {
                scanner.close();
            }
        }
    }
```

PART C

Test Case Scenarios for MauriBankLoanCalculator:

1. User Input Validation

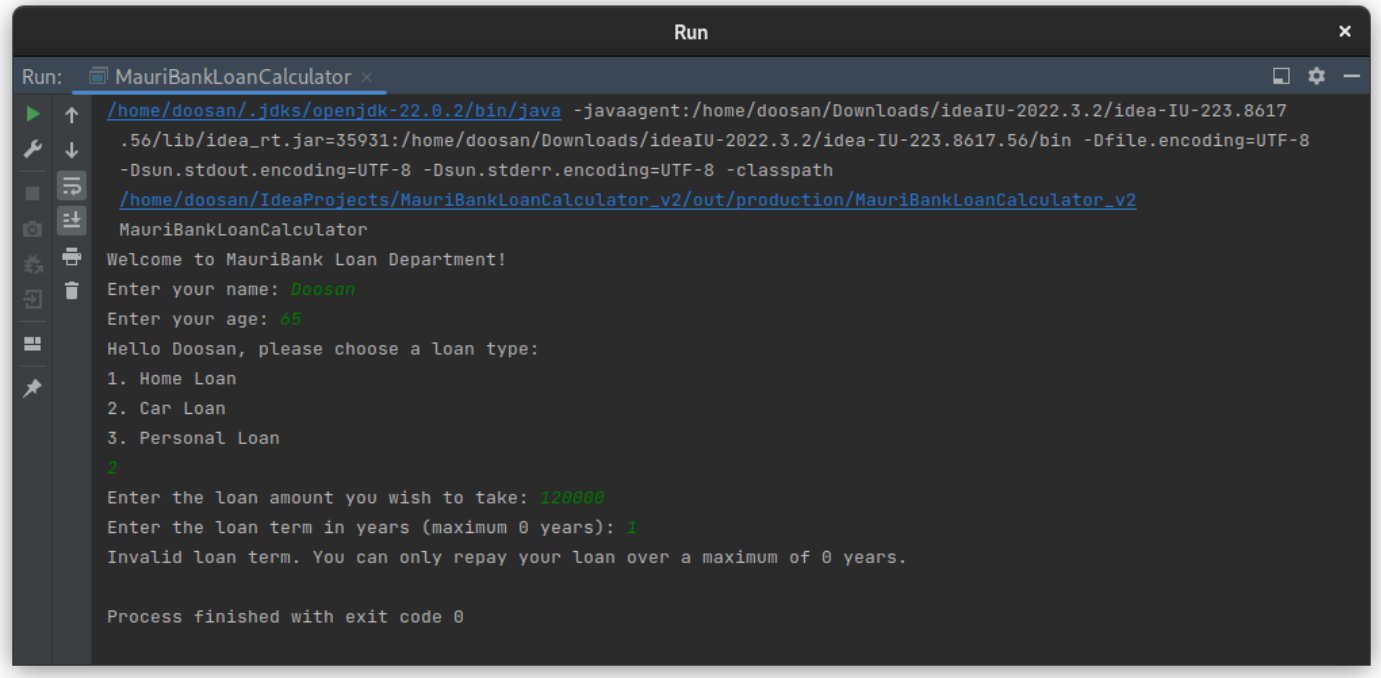
- **Test Case 1.1**: Validate that the user's age is between 18 and 65.
    - **Input**: Age = 17
    - **Expected Output**: Error message: "You must be between 18 and 65 years old to be eligible for a loan."



- **Test Case 1.2**: Validate that the user's age is exactly 18.
    - **Input**: Age = 18
    - **Expected Output**: Proceed with the loan application process.

```
Run
Run: MauriBankLoanCalculator
/home/doosan/.jdks/openjdk-22.0.2/bin/java -javaagent:/home/doosan/Downloads/ideaIU-2022.3.2/idea-IU-223.8617
 .56/lib/idea_rt.jar=39129:/home/doosan/Downloads/ideaIU-2022.3.2/idea-IU-223.8617.56/bin -Dfile.encoding=UTF-8
 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath
 /home/doosan/IdeaProjects/MauriBankLoanCalculator_v2/out/production/MauriBankLoanCalculator_v2
 MauriBankLoanCalculator
Welcome to MauriBank Loan Department!
Enter your name: Doosan
Enter your age: 18
Hello Doosan, please choose a loan type:
1. Home Loan
2. Car Loan
3. Personal Loan
1
Enter the loan amount you wish to take: 100000
Enter the loan term in years (maximum 47 years): 10
+----------------+----------------------+
| Loan Detail    | Value                |
+----------------+----------------------+
| Loan Type      | Home Loan            |
+----------------+----------------------+
| Loan Amount    | 100000.00            |
+----------------+----------------------+
| Loan Term      | 10 years             |
+----------------+----------------------+
| Interest Rate  | 5.00%                |
+----------------+----------------------+
| Monthly Payment| 1060.66              |
+----------------+----------------------+

Process finished with exit code 0
```

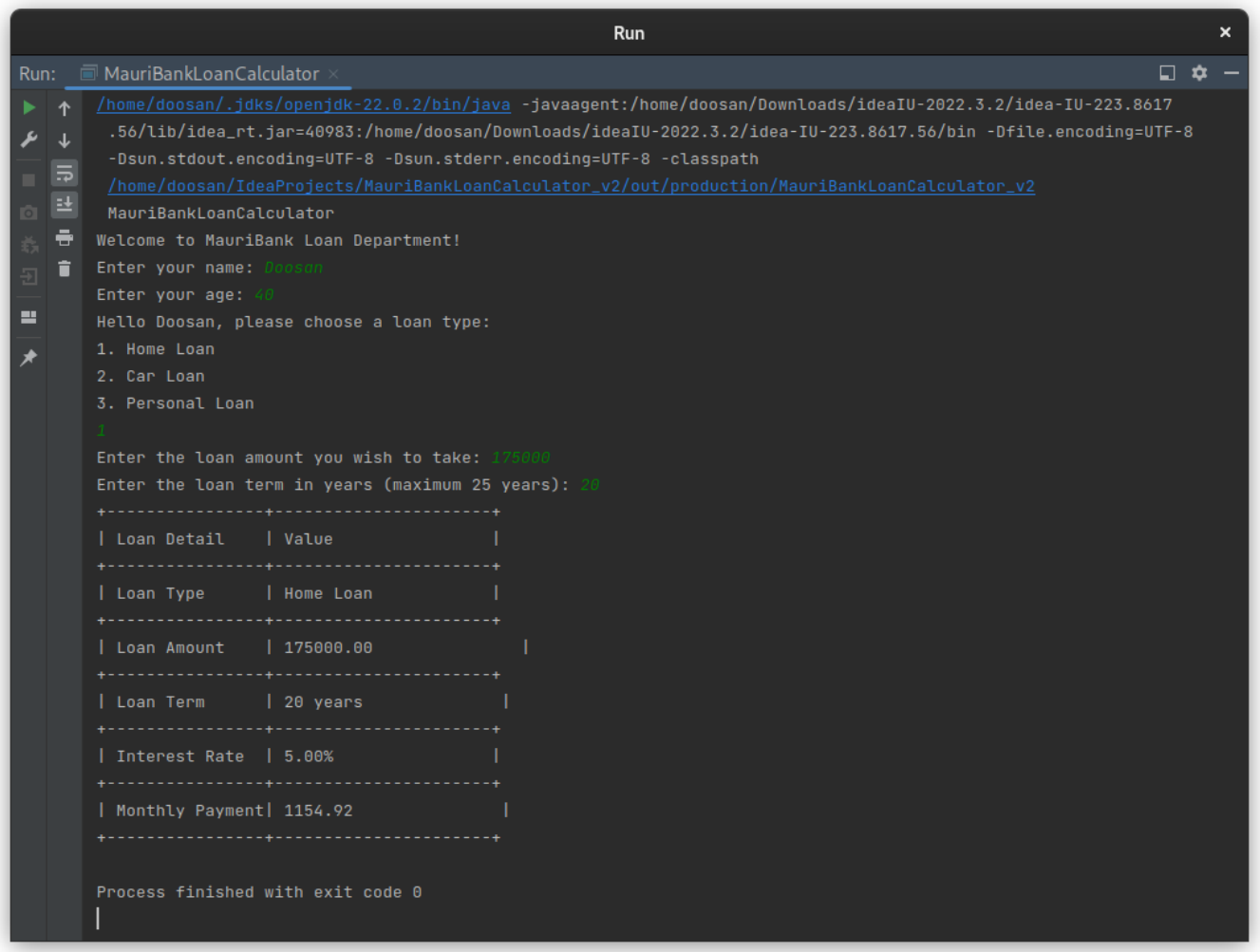
- **Test Case 1.3**: Validate that the user's age is exactly 65.
    - **Input**: Age = 65
    - **Expected Output**: Proceed with the loan application process.



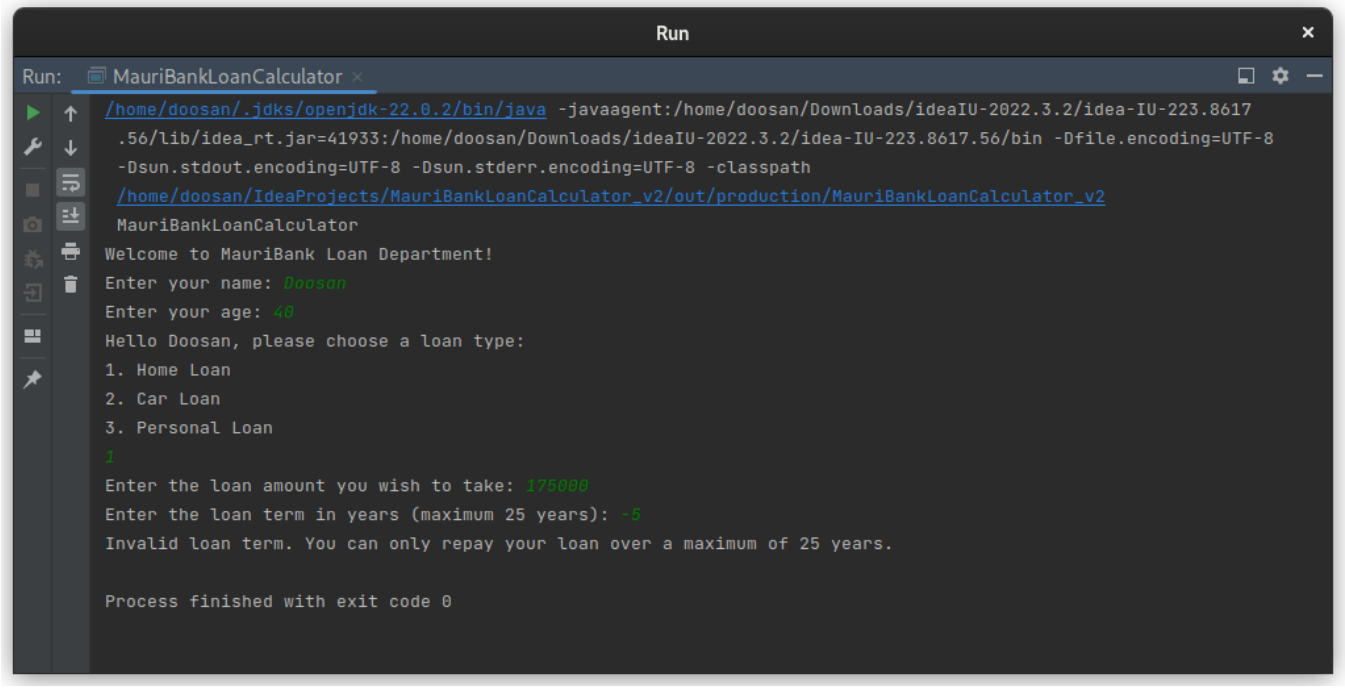
- **Test Case 1.4**: Validate that the user's age is over 65.
    - **Input**: Age = 66
    - **Expected Output**: Error message: "You must be between 18 and 65 years old to be eligible for a loan."

2. Loan Term Validation

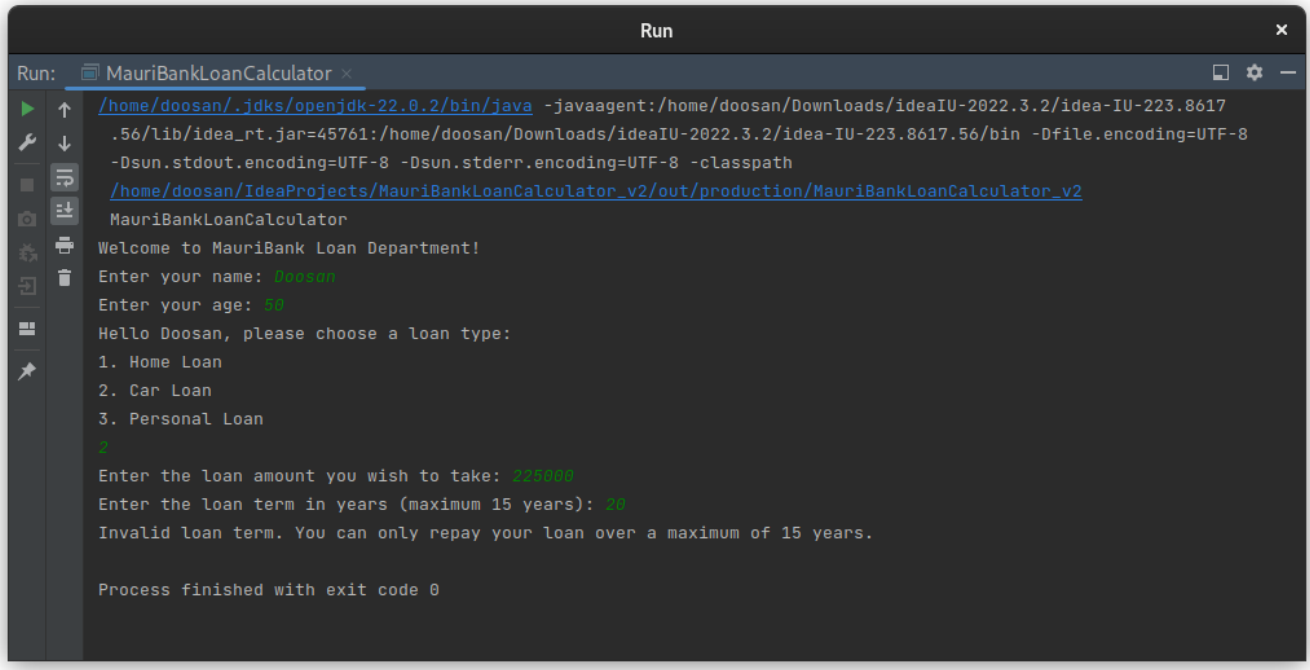
- **Test Case 2.1**: Validate that the loan term is greater than 0 years and less than or equal to the maximum loan term based on the user's age.
    - **Input**: Age = 40, Loan Term = 20 years
    - **Expected Output**: Proceed with the loan application process.



- **Test Case 2.2**: Validate that the loan term is negative.
    - **Input**: Loan Term = -5 years
    - **Expected Output**: Error message: "Invalid loan term. Loan term must be positive."

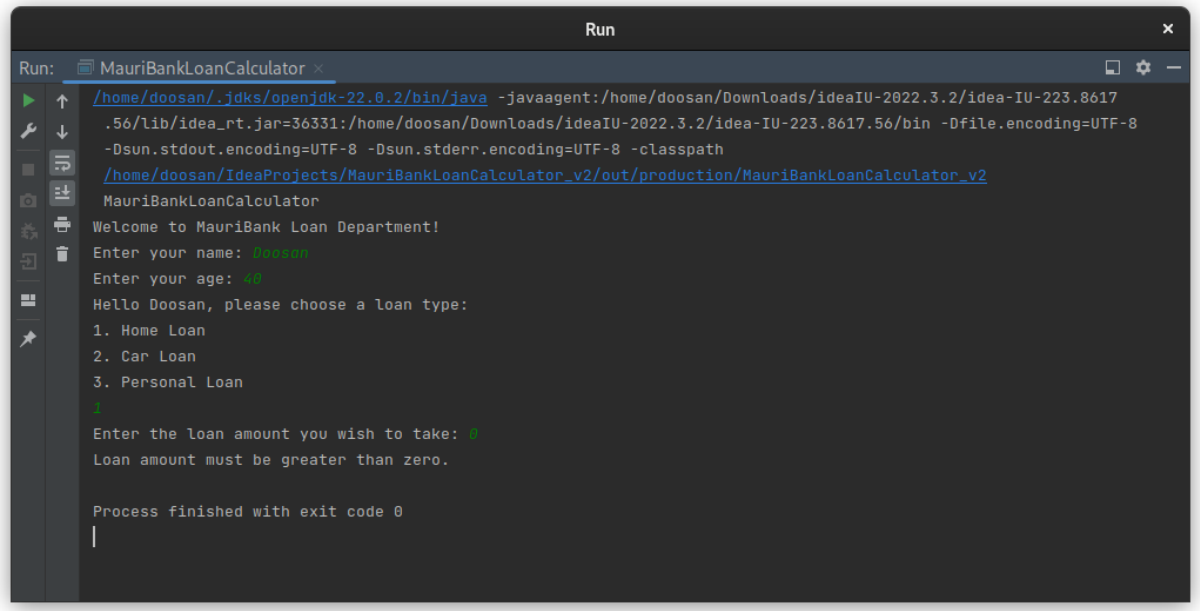


- **Test Case 2.3**: Validate that the loan term exceeds the maximum allowed based on the user's age.
    - **Input**: Age = 50, Loan Term = 20 years
    - **Expected Output**: Error message: "Invalid loan term. You can only repay your loan over a maximum of 15 years."

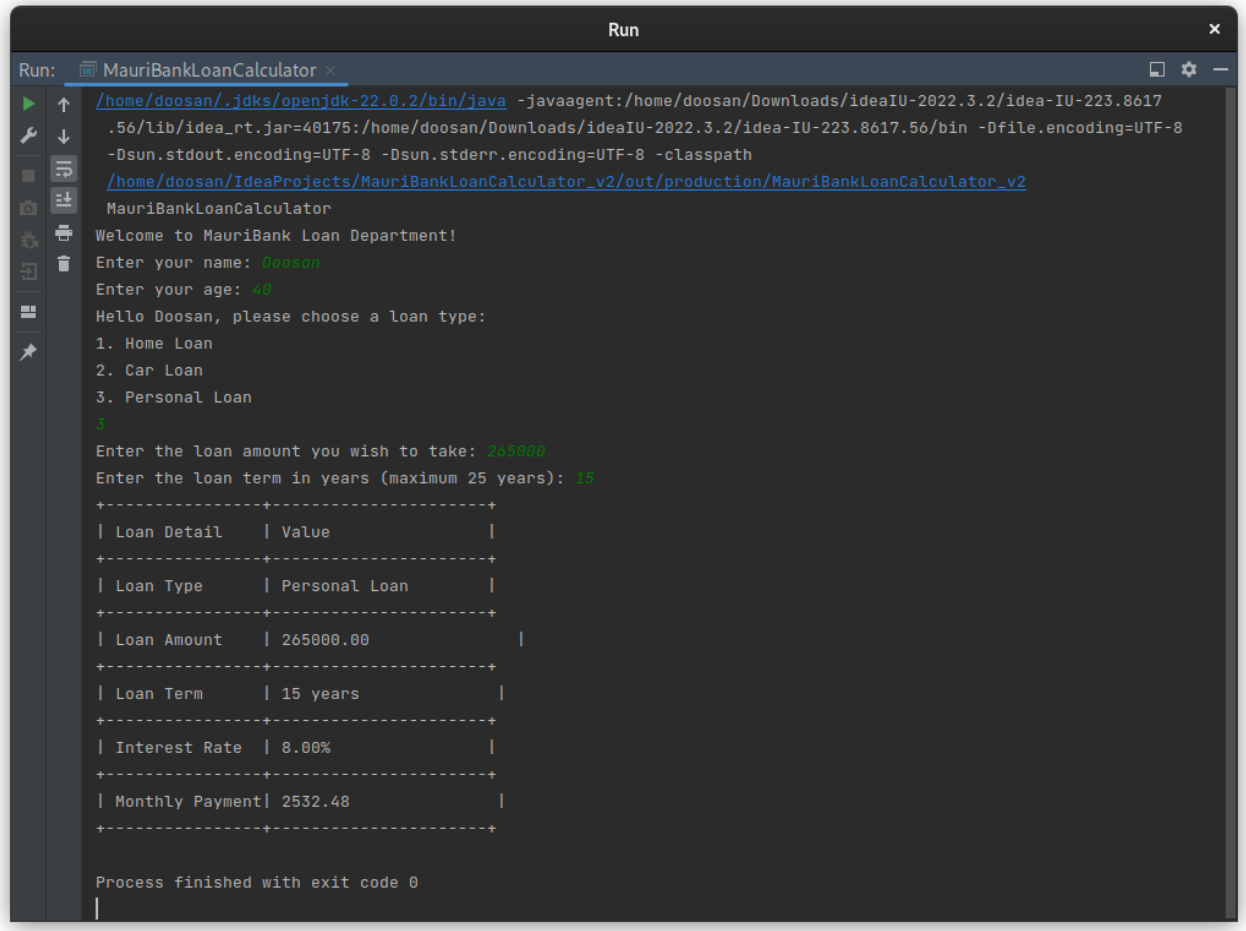


3. Loan Amount Validation

- **Test Case 3.1**: Validate that the loan amount is greater than 0.
    - **Input**: Loan Amount = 0
    - **Expected Output**: Error message: "Loan amount must be greater than zero."

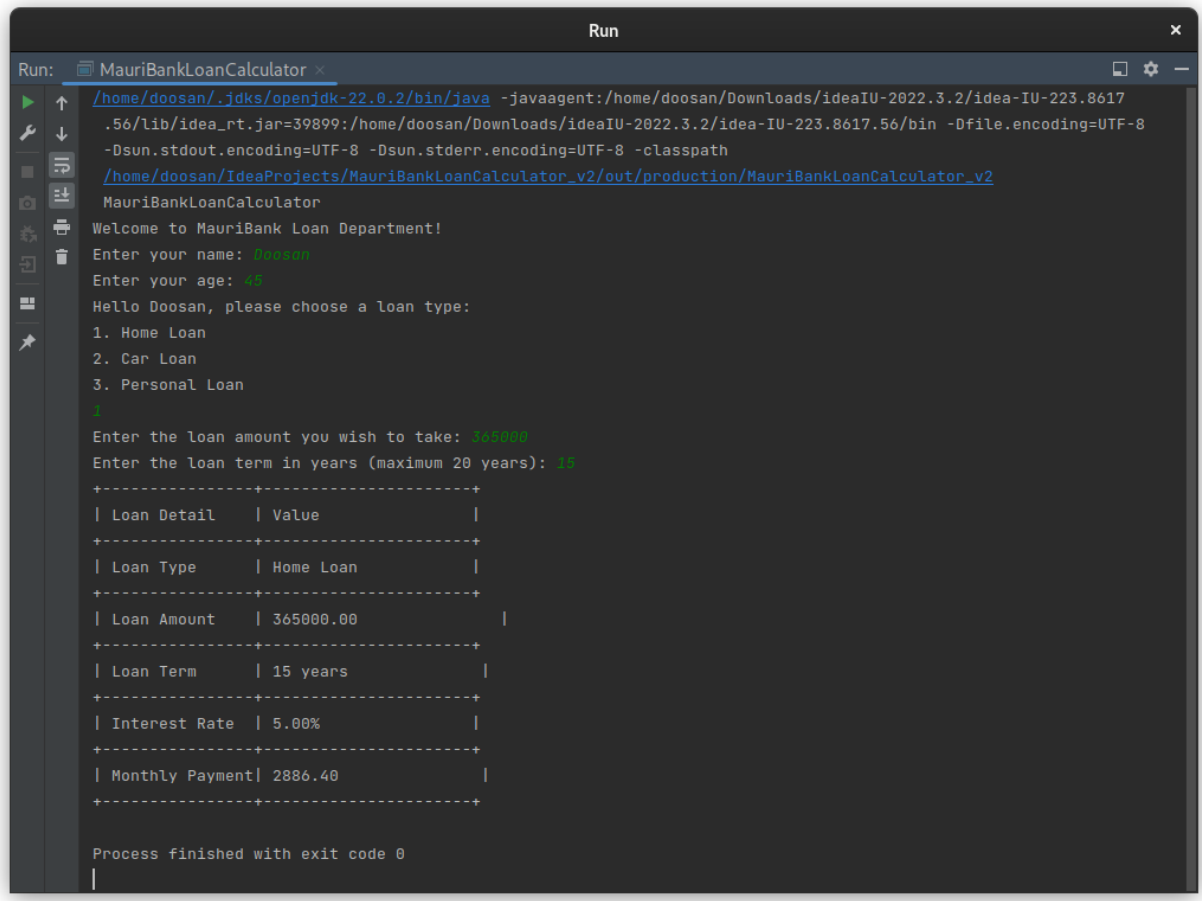


- **Test Case 3.2**: Validate that the loan amount is a valid positive number.
    - **Input**: Loan Amount = 265000
    - **Expected Output**: Proceed with the loan application process.



4. Loan Type Selection

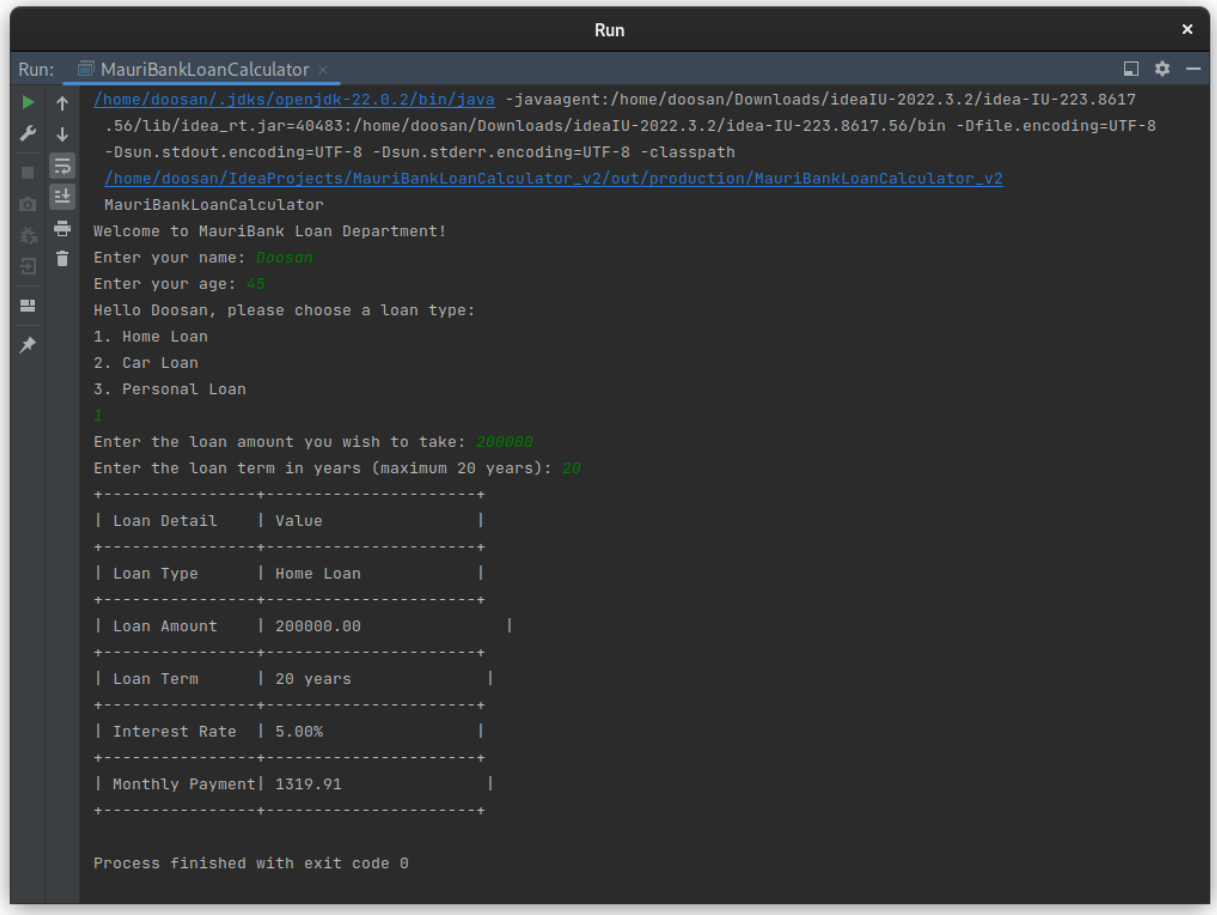
- **Test Case 4.1**: Validate that the user selects a valid loan type (1 for Home Loan, 2 for Car Loan, 3 for Personal Loan).
    - **Input**: Loan Type = 1 (Home Loan)
    - **Expected Output**: Proceed with the creation of a Home Loan object.



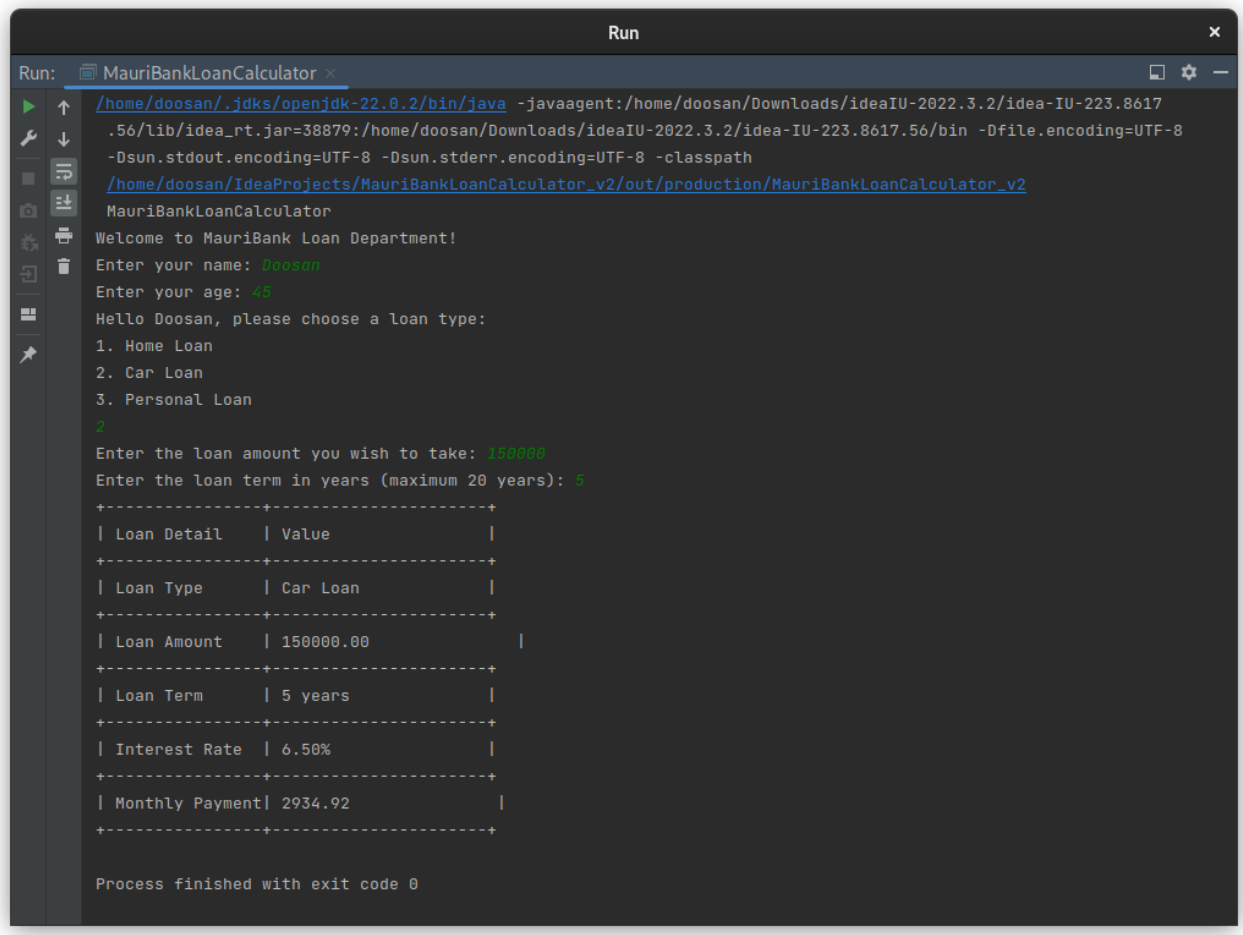
- **Test Case 4.2**: Validate that the user selects an invalid loan type.
    - **Input**: Loan Type = 4
    - **Expected Output**: Error message: "Invalid loan type selected. Please choose a valid option (1, 2, or 3)."

5. Loan Creation

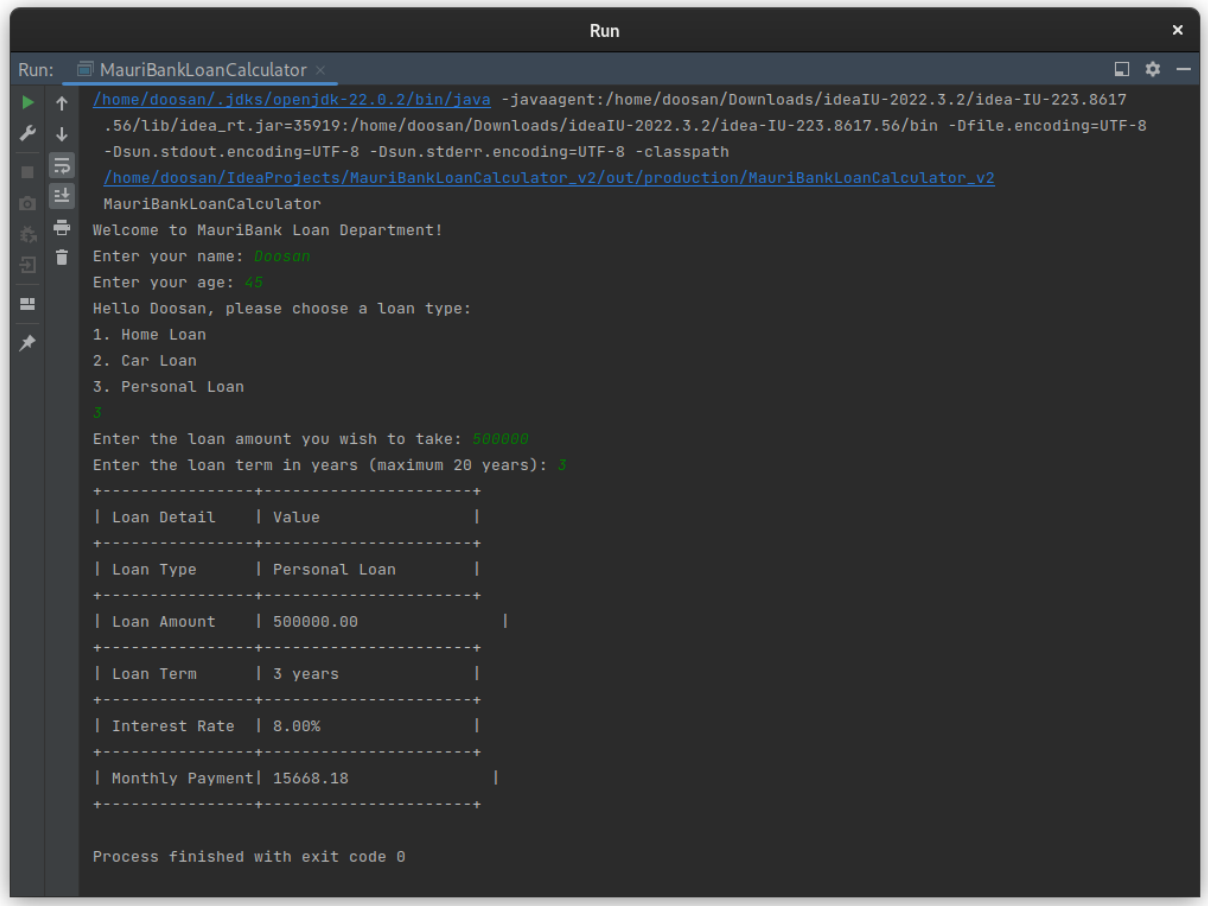
- **Test Case 5.**1: Validate that a **HOMELOAN** object is created successfully when the user selects Home Loan.
    - **Input**: Loan Type = 1, Loan Amount = 200000, Loan Term = 20 years
    - **Expected Output**: A **HOMELOAN** object is created with the correct loan amount, loan term, and a 5.0% interest rate.



- **Test Case 5.2**: Validate that a **CARLOAN** object is created successfully when the user selects Car Loan.
    - **Input**: Loan Type = 2, Loan Amount = 150000, Loan Term = 5 years
    - **Expected Output**: A **CARLOAN** object is created with the correct loan amount, loan term, and a 6.5% interest rate.

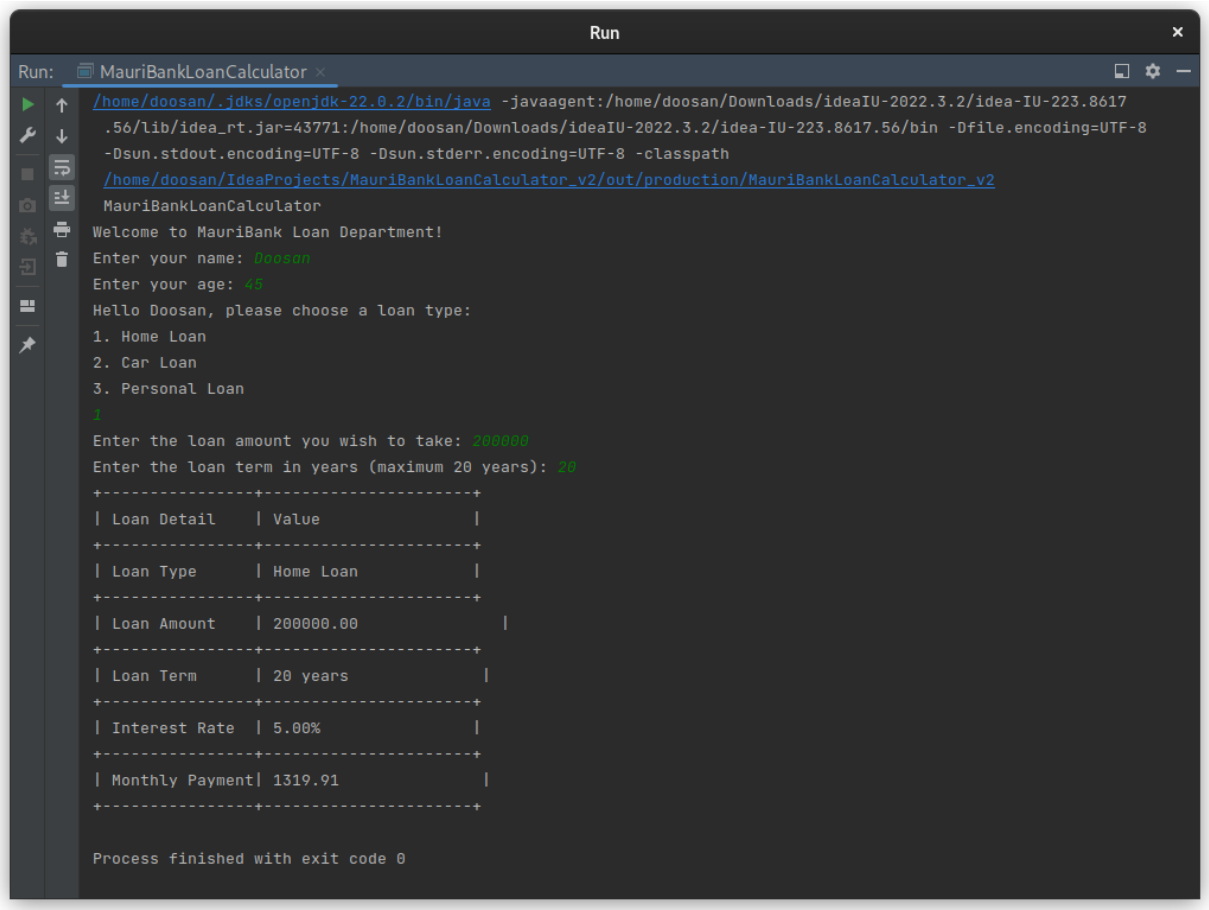


- **Test Case 5.3**: Validate that a **PERSONALOAN** object is created successfully when the user selects Personal Loan.
  - **Input**: Loan Type = 3, Loan Amount = 50000, Loan Term = 3 years
  - **Expected Output**: A **PERSONALOAN** object is created with the correct loan amount, loan term, and an 8.0% interest rate.

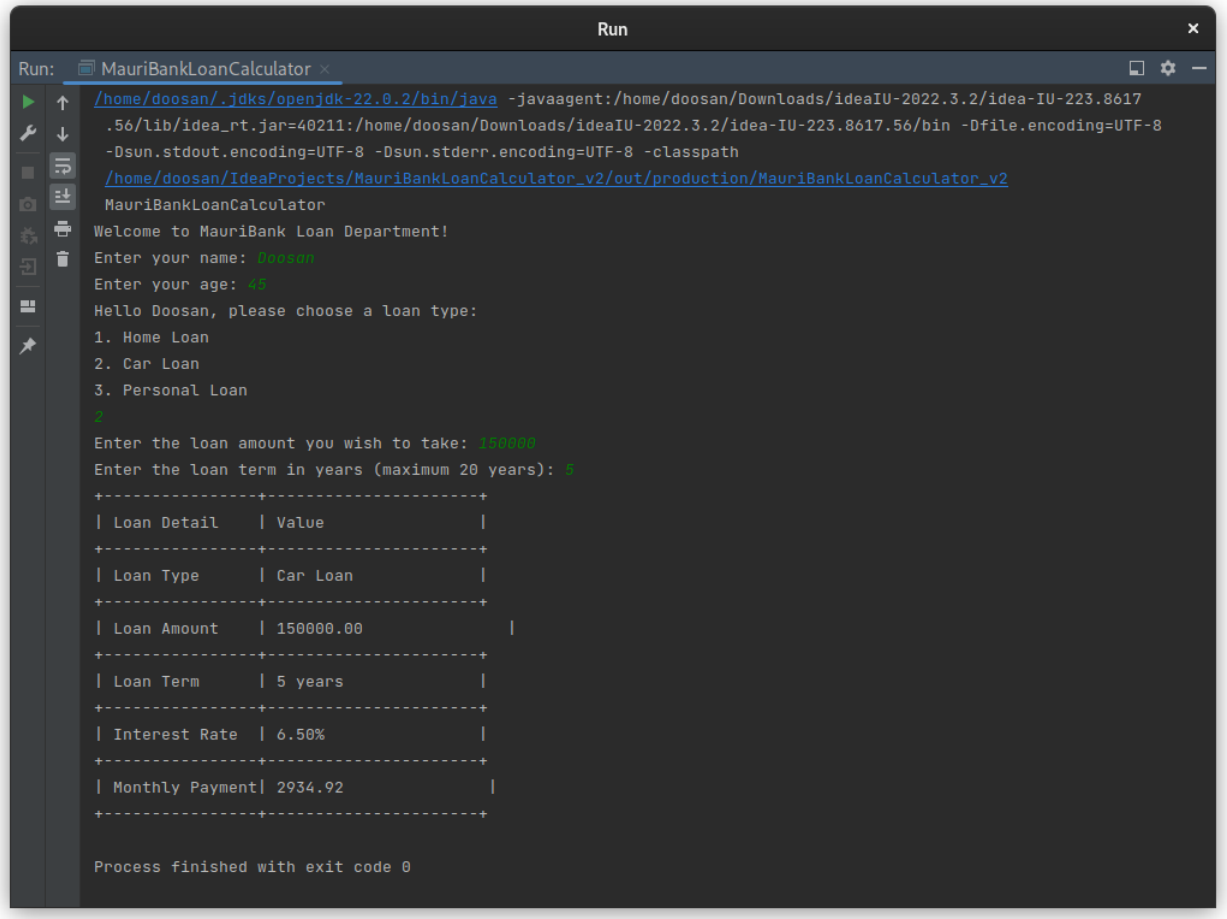


6. Monthly Payment Calculation

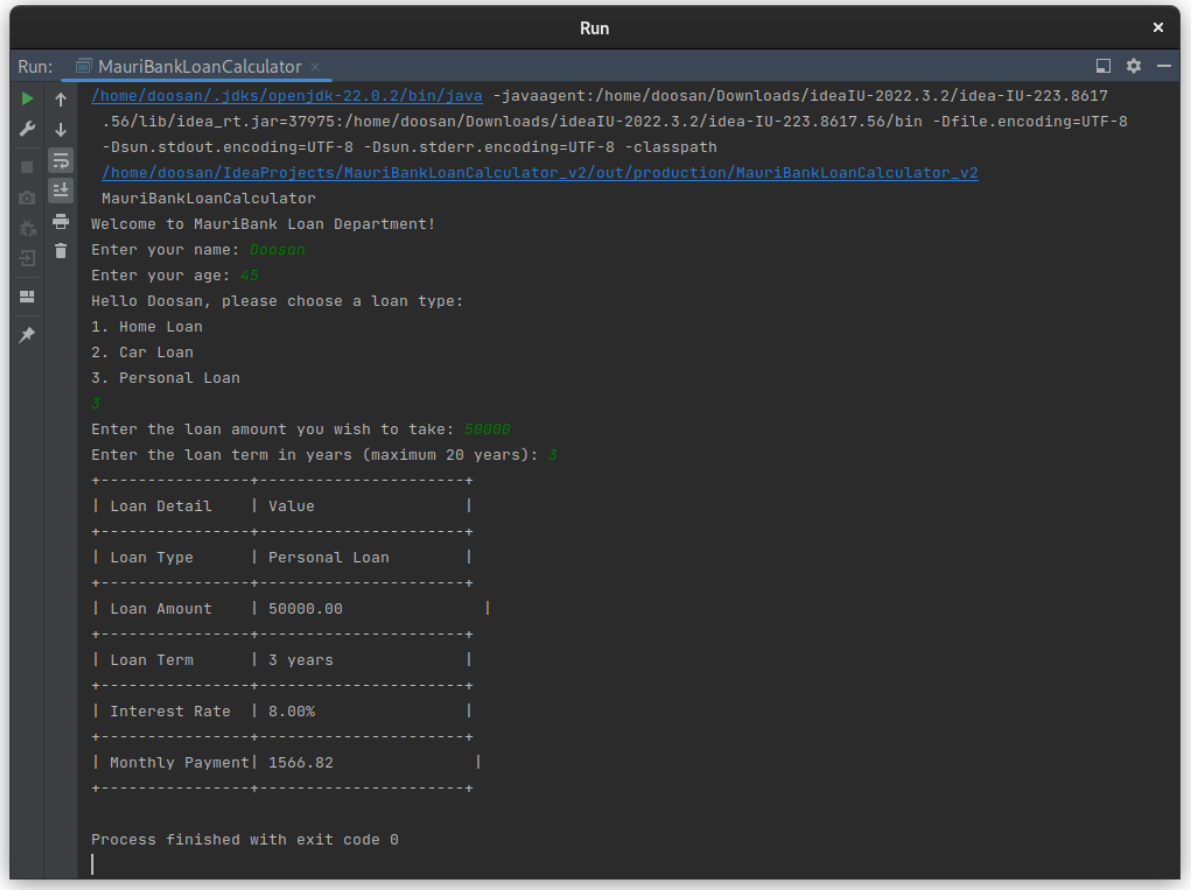
- **Test Case 6.1**: Validate that the monthly payment is calculated correctly for a Home Loan.
  - **Input**: Loan Type = 1 (Home Loan), Loan Amount = 200000, Loan Term = 20 years
  - **Expected Output**: Correct monthly payment based on the 5.0% interest rate.



- **Test Case 6.2**: Validate that the monthly payment is calculated correctly for a Car Loan.
  - **Input**: Loan Type = 2 (Car Loan), Loan Amount = 150000, Loan Term = 5 years
  - **Expected Output**: Correct monthly payment based on the 6.5% interest rate.

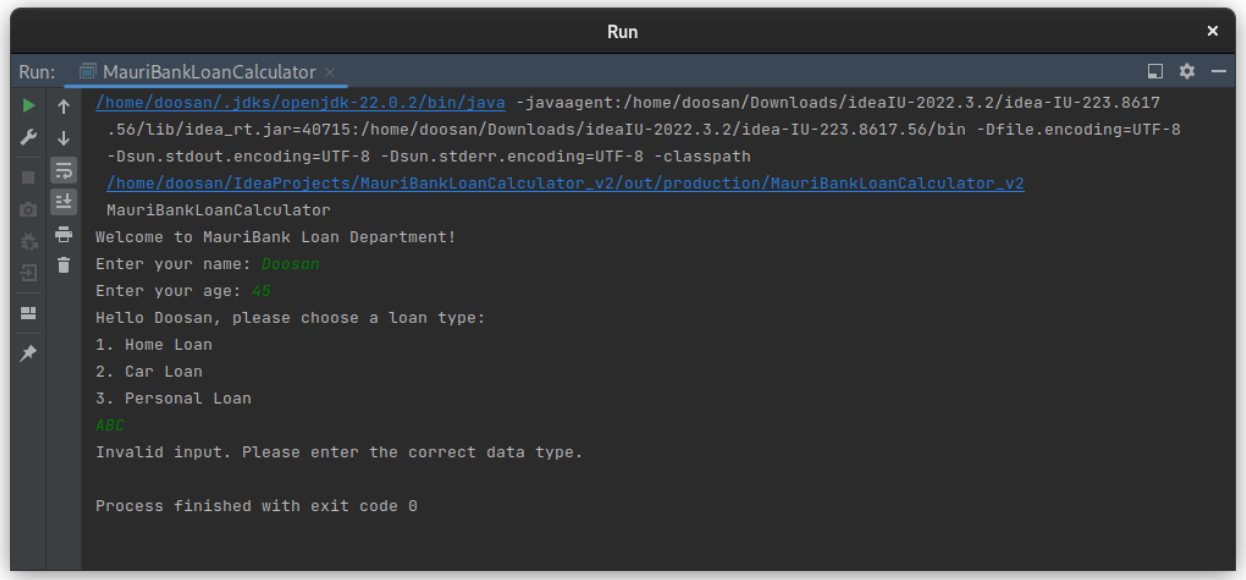


- **Test Case 6.3**: Validate that the monthly payment is calculated correctly for a Personal Loan.
  - **Input**: Loan Type = 3 (Personal Loan), Loan Amount = 50000, Loan Term = 3 years
  - **Expected Output**: Correct monthly payment based on the 8.0% interest rate.

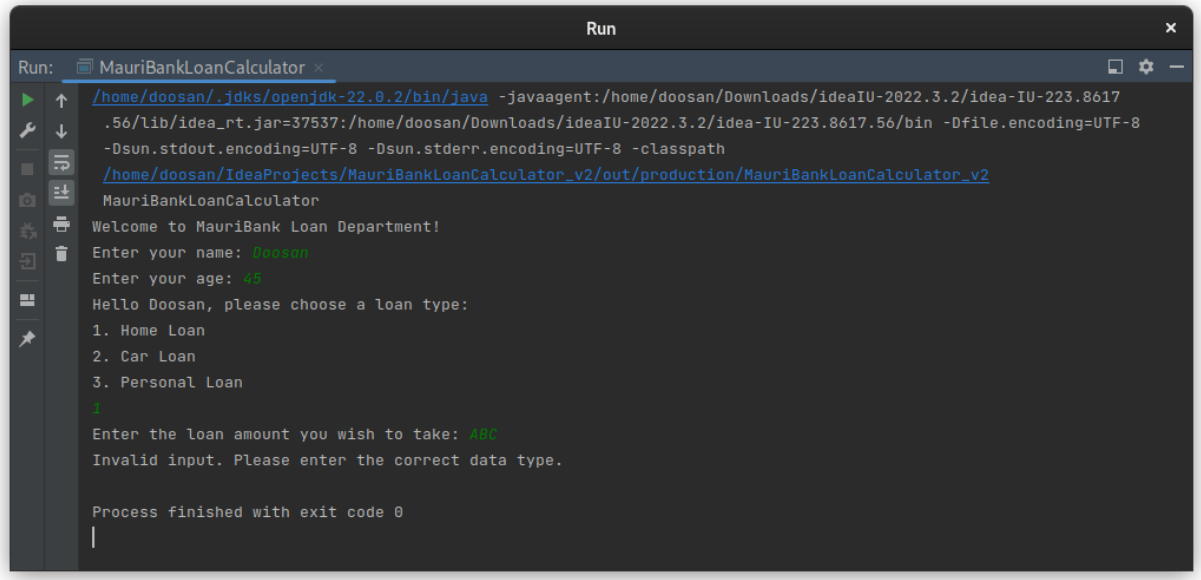


7. Error Handling

- **Test Case 7.1**: Validate that the system handles invalid input for loan type (non-integer values).
  - **Input**: Loan Type = "ABC" (non-integer input)
  - **Expected Output**: Error message: "Invalid input. Please enter the correct data type."

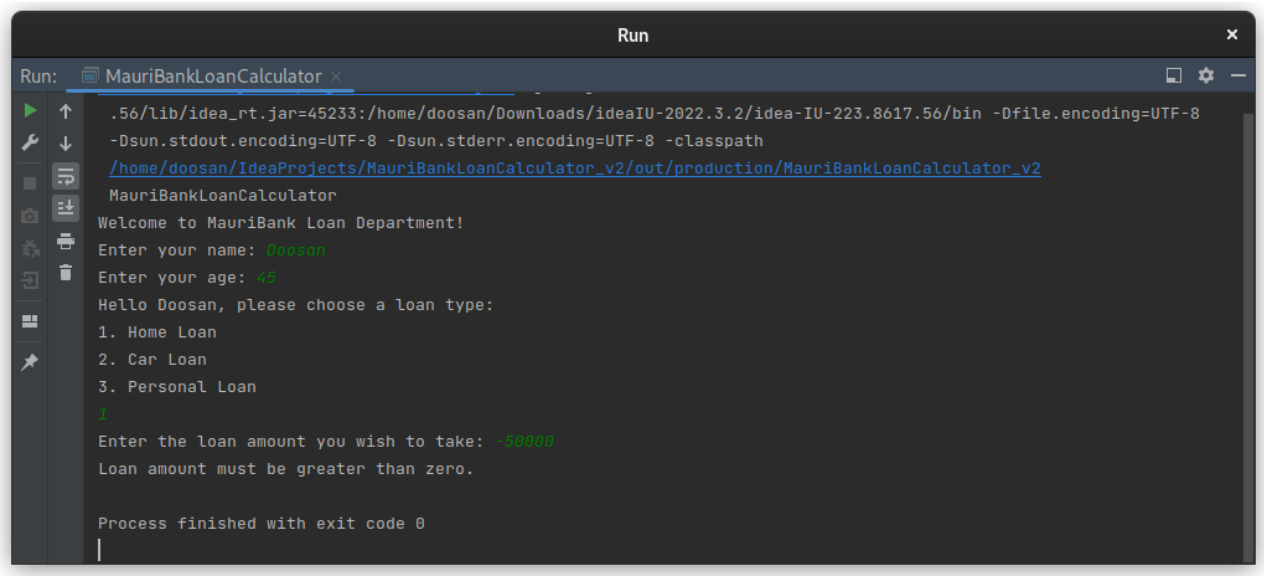


- **Test Case 7.2**: Validate that the system handles invalid input for loan amount (non- numeric values).
    - **Input**: Loan Amount = "ABC" (non-numeric input)
    - **Expected Output**: Error message: "Invalid input. Please enter the correct data type."



8. General System Behaviour

- **Test Case 8.1**: Validate that the system terminates properly when invalid input is provided multiple times.
    - **Input**: Age = 66, Loan Type = 4, Loan Amount = -50000
    - **Expected Output**: The system displays multiple error messages and terminates the loan process.



- **Test Case 8.2**: Validate that the system successfully completes the loan process from start to finish with valid input.
    - **Input**: Age = 30, Loan Type = 1, Loan Amount = 200000, Loan Term = 15 years
    - **Expected Output**: The system successfully calculates and displays the monthly payment and loan details.

```
Run
Run: MauriBankLoanCalculator
/home/doosan/.jdks/openjdk-22.0.2/bin/java -javaagent:/home/doosan/Downloads/ideaIU-2022.3.2/idea-IU-223.8617
 .56/lib/idea_rt.jar=38807:/home/doosan/Downloads/ideaIU-2022.3.2/idea-IU-223.8617.56/bin -Dfile.encoding=UTF-8
 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath
 /home/doosan/IdeaProjects/MauriBankLoanCalculator_v2/out/production/MauriBankLoanCalculator_v2
 MauriBankLoanCalculator
Welcome to MauriBank Loan Department!
Enter your name: Doosan
Enter your age: 30
Hello Doosan, please choose a loan type:
1. Home Loan
2. Car Loan
3. Personal Loan
1
Enter the loan amount you wish to take: 200000
Enter the loan term in years (maximum 35 years): 15
+----------------+----------------------+
| Loan Detail    | Value                |
+----------------+----------------------+
| Loan Type      | Home Loan            |
+----------------+----------------------+
| Loan Amount    | 200000.00            |
+----------------+----------------------+
| Loan Term      | 15 years             |
+----------------+----------------------+
| Interest Rate  | 5.00%                |
+----------------+----------------------+
| Monthly Payment| 1581.59              |
+----------------+----------------------+

Process finished with exit code 0
```

References

1. National University of Singapore (2023) CS2030S Programming Methodology II. Available at: https://nus-cs2030s.github.io (Accessed: 10 October 2024).
2. Brusca, V.G. (2023) 'Encapsulation, Inheritance, and Polymorphism', in Introduction to Java Through Game Development. Berkeley, CA: Apress. Available at: https://doi.org/10.1007/978-1-4842-8951-8_8 (Accessed: 22 October 2024).