ASSIGNMENT NO.5

Note:

1. This assignment is designed to practice static fields, static initializers, and static methods.
2. Understand the problem statement and use static and non-static wisely to solve the problem.
3. Use constructors, proper getter/setter methods, and `toString()` wherever required.

1. Design and implement a class named `InstanceCounter` to track and count the number of instances created from this class.

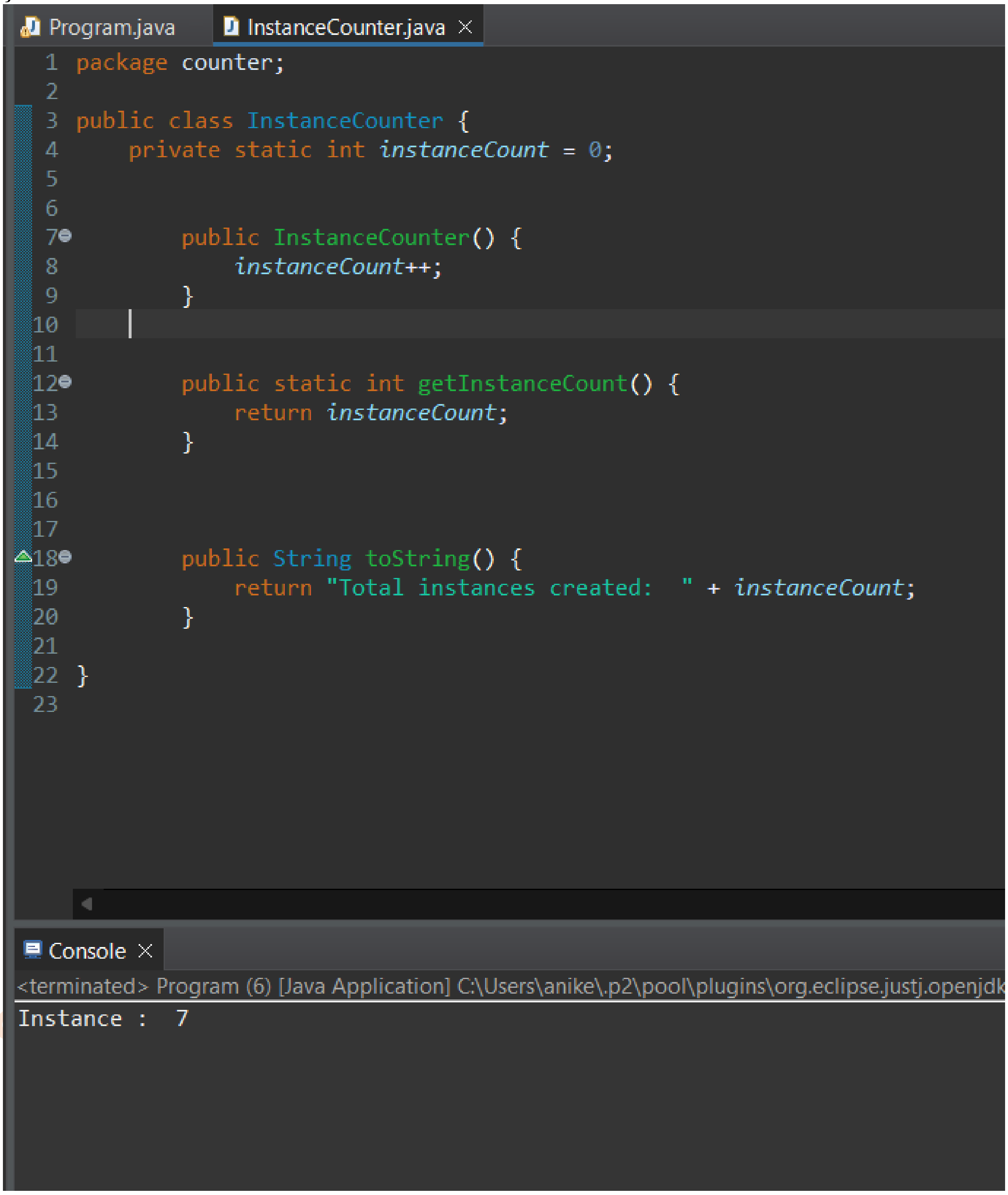
Program.java:

```
package counter;

public class Program {
        public static void main(String[] args) {

                        InstanceCounter a1 = new InstanceCounter();
                        InstanceCounter a2 = new InstanceCounter();
                        InstanceCounter a3 = new InstanceCounter();
                        InstanceCounter a4 = new InstanceCounter();
                        InstanceCounter a5 = new InstanceCounter();
                        InstanceCounter a6 = new InstanceCounter();
                        InstanceCounter a7 = new InstanceCounter();

                System.out.println("Instance :  "  +InstanceCounter.getInstanceCount());
                 }
}
```

InstanceCalculator:

```
package counter;

public class InstanceCounter {
      private static int instanceCount = 0;


            public InstanceCounter() {
                  instanceCount++;
            }


      public static int getInstanceCount() {
                  return instanceCount;
            }
```

```
        public String toString() {
                return "Total instances created:  " + instanceCount;
        }

}
```



Program.java   InstanceCounter.java

```
package counter;

public class InstanceCounter {
    private static int instanceCount = 0;


        public InstanceCounter() {
            instanceCount++;
        }
        

        public static int getInstanceCount() {
            return instanceCount;
        }



        public String toString() {
            return "Total instances created:  " + instanceCount;
        }

}

```

Console

```
<terminated> Program (6) [Java Application] C:\Users\anike\.p2\pool\plugins\org.eclipse.justj.openjdk
Instance :  7
```

2. Design and implement a class named `Logger` to manage logging messages for an application. The class should be implemented as a singleton to ensure that only one instance of the `Logger` exists throughout the application.

The class should include the following methods:

- **`getInstance()`**: Returns the unique instance of the `Logger` class.
- **`log(String message)`**: Adds a log message to the logger.
- **`getLog()`**: Returns the current log messages as a `String`.
- **`clearLog()`**: Clears all log messages.

Program.java:

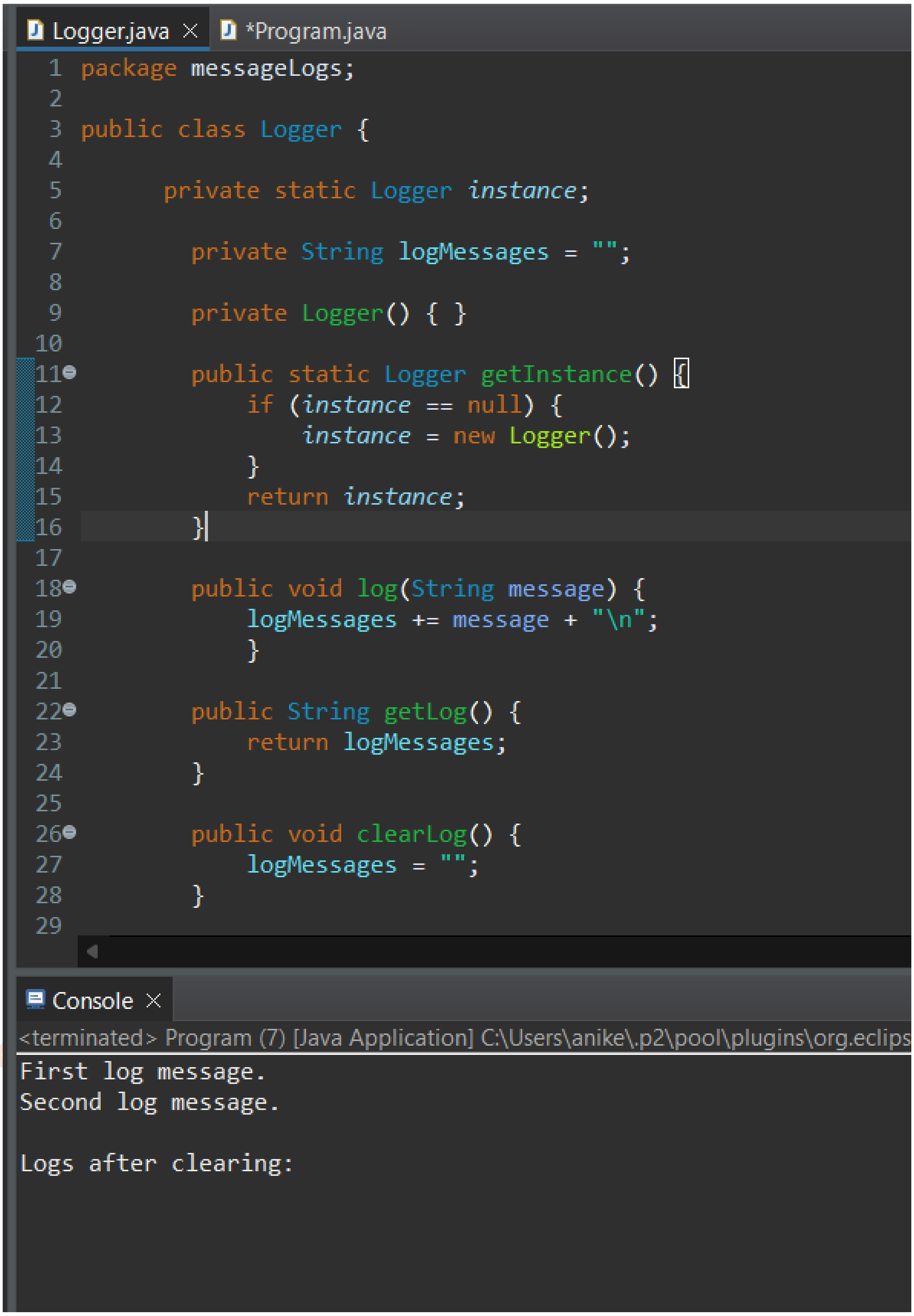
```
package messageLogs;


public class Program {

        public static void main(String[] args) {

                Logger logger = Logger.getInstance();


                logger.log("First log message.");

                logger.log("Second log message.");


                System.out.println(logger.getLog());

                logger.clearLog();


                System.out.println("Logs after clearing: " + logger.getLog());

        }

}
```

Logger.java:

```
package messageLogs;


public class Logger {
```

```
private static Logger instance;

 private String logMessages = "";

 private Logger() { }

 public static Logger getInstance() {
     if (instance == null) {
         instance = new Logger();
     }
     return instance;
 }

 public void log(String message) {
     logMessages += message + "\n";
     }

 public String getLog() {
     return logMessages;
 }

 public void clearLog() {
         logMessages = "";
 }
```

}



Logger.java ×   *Program.java

```java
package messageLogs;

public class Logger {

	private static Logger instance;

	  private String logMessages = "";

	  private Logger() { }

	  public static Logger getInstance() {
	      if (instance == null) {
	          instance = new Logger();
	      }
	      return instance;
	  }

	  public void log(String message) {
	      logMessages += message + "\n";
	  }

	  public String getLog() {
	      return logMessages;
	  }

	  public void clearLog() {
	      logMessages = "";
	  }

```

Console ×

```
<terminated> Program (7) [Java Application] C:\Users\anike\.p2\pool\plugins\org.eclips
First log message.
Second log message.

Logs after clearing:
```

3. Design and implement a class named `Employee` to manage employee data for a company. The class should include fields to keep track of the total number of employees and the total salary expense, as well as individual employee details such as their ID, name, and salary.

   The class should have methods to:

   - Retrieve the total number of employees (`getTotalEmployees()`)
   - Apply a percentage raise to the salary of all employees (`applyRaise(double percentage)`)
   - Calculate the total salary expense, including any raises (`calculateTotalSalaryExpense()`)
   - Update the salary of an individual employee (`updateSalary(double newSalary)`)

   Understand the problem statement and use static and non-static fields and methods appropriately. Implement static and non-static initializers, constructors, getter and setter methods, and a `toString()` method to handle the initialization and representation of employee data.

   Write a menu-driven program in the `main` method to test the functionalities.

Employee.java:

```
package emp;



public class Employee {

    private static int totalEmployees = 0;

      private static double totalSalaryExpense = 0;



      private int id;

      private String name;

      private double salary;



      public Employee(int id, String name, double salary) {

         this.id = id;

         this.name = name;
```

```
            this.salary = salary;

            totalEmployees++;  // Increase the total number of employees

            totalSalaryExpense += salary;  // Add to the total salary expense

        }


        public static int getTotalEmployees() {

            return totalEmployees;

        }


        public static void applyRaise(Employee[] employees, int employeeCount, double
percentage) {

            for (int i = 0; i < employeeCount; i++) {

                Employee emp = employees[i];

                double raise = emp.salary * (percentage / 100);

                emp.salary += raise;

                totalSalaryExpense += raise;  // Update total salary expense

            }

        }


        public static double calculateTotalSalaryExpense() {

            return totalSalaryExpense;

        }


        public void updateSalary(double newSalary) {

            totalSalaryExpense = totalSalaryExpense - this.salary + newSalary;  // Adjust the
total salary expense
```

```
        this.salary = newSalary;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public double getSalary() {
        return salary;
    }

    @Override
    public String toString() {
        return "Employee ID: " + id + ", Name: " + name + ", Salary: " + salary;
    }
}
```

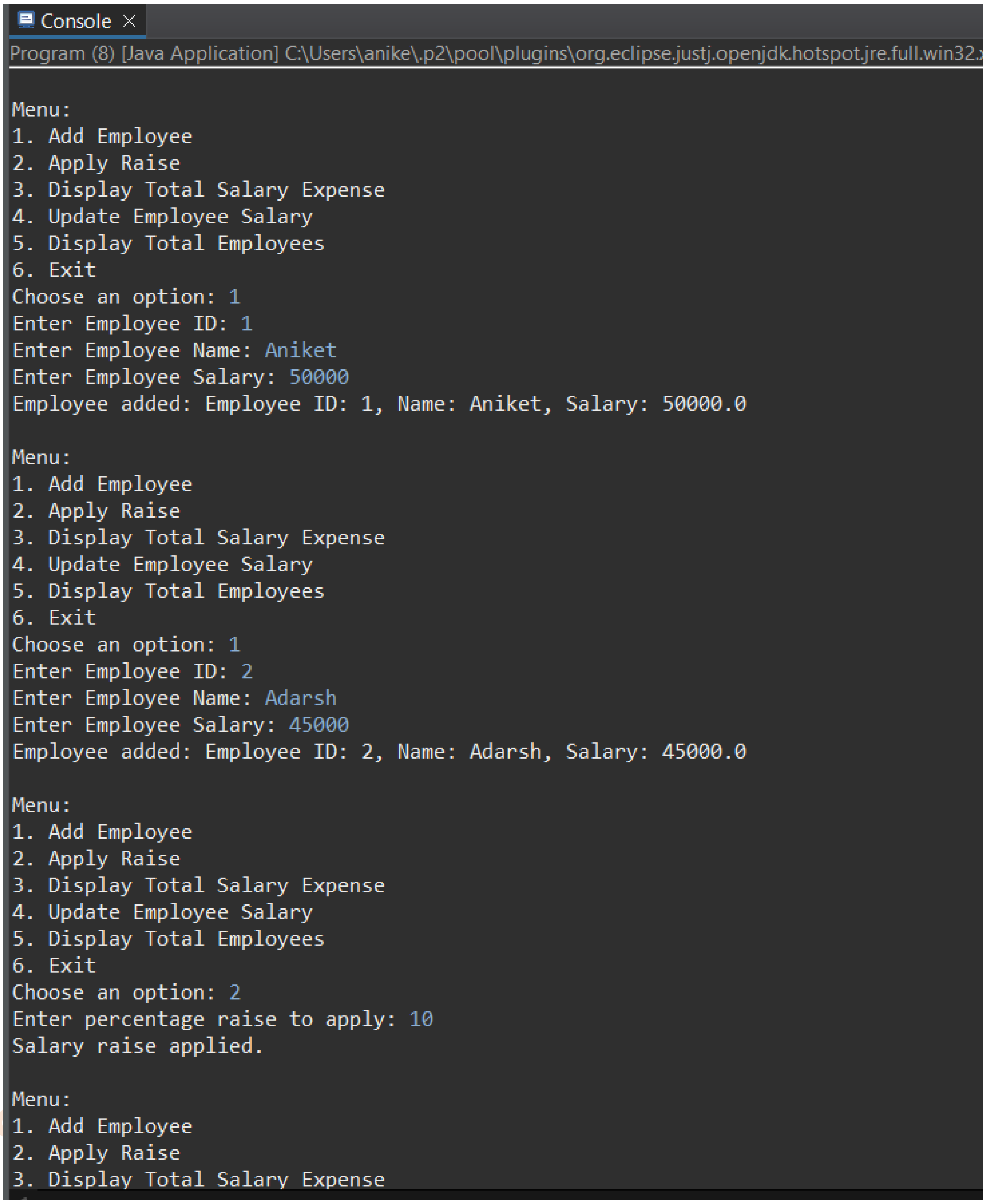
Program,java:

```
package emp;

import java.util.Scanner;
```

```
public class Program {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        Employee[] employees = new Employee[100];
        int employeeCount = 0;
        boolean exit = false;
        while (!exit) {
            System.out.println("\nMenu:");
            System.out.println("1. Add Employee");
            System.out.println("2. Apply Raise");
            System.out.println("3. Display Total Salary Expense");
            System.out.println("4. Update Employee Salary");
            System.out.println("5. Display Total Employees");
            System.out.println("6. Exit");

            System.out.print("Choose an option: ");
            int option = sc.nextInt();

            switch (option) {
                case 1:
                    System.out.print("Enter Employee ID: ");
                    int id = sc.nextInt();
                    sc.nextLine();  // Consume the newline
                    System.out.print("Enter Employee Name: ");
                    String name = sc.nextLine();
                    System.out.print("Enter Employee Salary: ");
                    double salary = sc.nextDouble();

                    employees[employeeCount] = new Employee(id, name, salary);
                    System.out.println("Employee added: " + employees[employeeCount]);
                    employeeCount++;
                    break;

                case 2:
                    System.out.print("Enter percentage raise to apply: ");
                    double percentage = sc.nextDouble();
                    Employee.applyRaise(employees, employeeCount, percentage);
                    System.out.println("Salary raise applied.");
                    break;

                case 3:
                    System.out.println("Total Salary Expense: " +
Employee.calculateTotalSalaryExpense());
                    break;

                case 4:
                    System.out.print("Enter Employee ID to update salary: ");
                    int updateId = sc.nextInt();
```

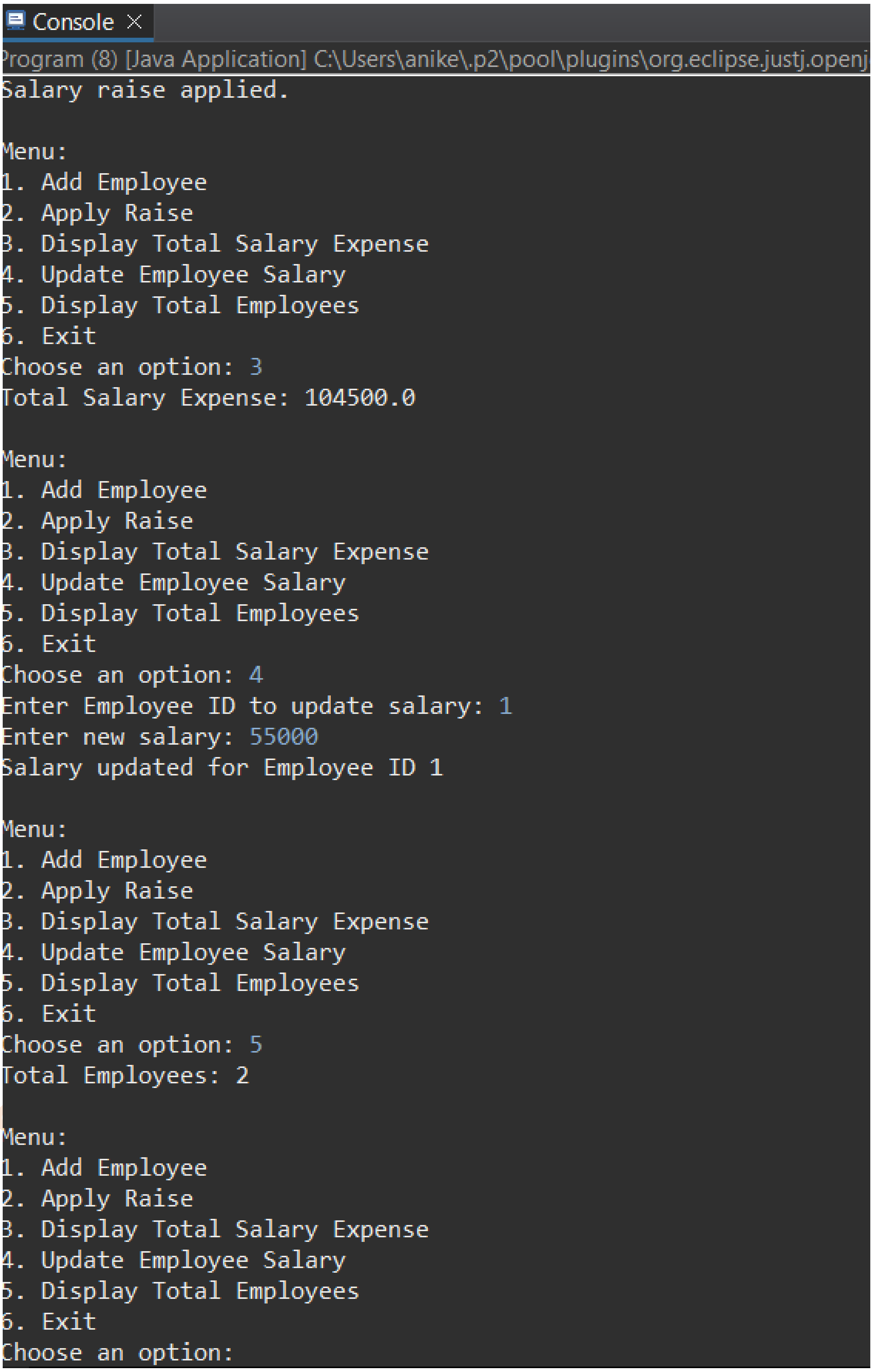
```
                    for (int i = 0; i < employeeCount; i++) {
                        if (employees[i].getId() == updateId) {
                            System.out.print("Enter new salary: ");
                            double newSalary = sc.nextDouble();
                            employees[i].updateSalary(newSalary);
                            System.out.println("Salary updated for Employee ID " + updateId);
                        }
                    }
                    break;

                case 5:
                    System.out.println("Total Employees: " +
Employee.getTotalEmployees());
                    break;

                case 6:
                    exit = true;
                    break;

                default:
                    System.out.println("Invalid option. Please try again.");
            }
        }
        sc.close();
    }
```

```
}
```



Program (8) [Java Application] C:\Users\anike\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.

```
Menu:
1. Add Employee
2. Apply Raise
3. Display Total Salary Expense
4. Update Employee Salary
5. Display Total Employees
6. Exit
Choose an option: 1
Enter Employee ID: 1
Enter Employee Name: Aniket
Enter Employee Salary: 50000
Employee added: Employee ID: 1, Name: Aniket, Salary: 50000.0

Menu:
1. Add Employee
2. Apply Raise
3. Display Total Salary Expense
4. Update Employee Salary
5. Display Total Employees
6. Exit
Choose an option: 1
Enter Employee ID: 2
Enter Employee Name: Adarsh
Enter Employee Salary: 45000
Employee added: Employee ID: 2, Name: Adarsh, Salary: 45000.0

Menu:
1. Add Employee
2. Apply Raise
3. Display Total Salary Expense
4. Update Employee Salary
5. Display Total Employees
6. Exit
Choose an option: 2
Enter percentage raise to apply: 10
Salary raise applied.

Menu:
1. Add Employee
2. Apply Raise
3. Display Total Salary Expense
```



```
Program (8) [Java Application] C:\Users\anike\.p2\pool\plugins\org.eclipse.justj.openj
Salary raise applied.

Menu:
1. Add Employee
2. Apply Raise
3. Display Total Salary Expense
4. Update Employee Salary
5. Display Total Employees
6. Exit
Choose an option: 3
Total Salary Expense: 104500.0

Menu:
1. Add Employee
2. Apply Raise
3. Display Total Salary Expense
4. Update Employee Salary
5. Display Total Employees
6. Exit
Choose an option: 4
Enter Employee ID to update salary: 1
Enter new salary: 55000
Salary updated for Employee ID 1

Menu:
1. Add Employee
2. Apply Raise
3. Display Total Salary Expense
4. Update Employee Salary
5. Display Total Employees
6. Exit
Choose an option: 5
Total Employees: 2

Menu:
1. Add Employee
2. Apply Raise
3. Display Total Salary Expense
4. Update Employee Salary
5. Display Total Employees
6. Exit
Choose an option:
```