

Assignment 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.

Solution:

Class File 1:

```
package org.instancecounter;

public class InstanceCounter {
    private static int instanceCount = 0;

    static {
        instanceCount = 0;
    }

    public InstanceCounter() {
        instanceCount++;
    }

    public static int getInstanceCount() {
        return instanceCount;
    }

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

Class File 2:

```
package org.instancecounter;

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

        InstanceCounter instance1 = new InstanceCounter();
    }
}
```

```

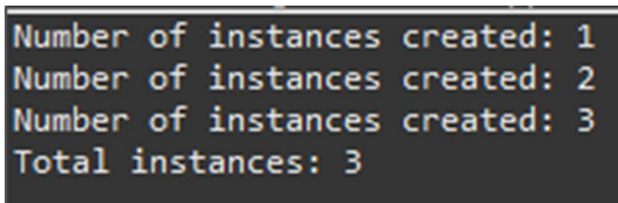
        System.out.println(instance1);

        InstanceCounter instance2 = new InstanceCounter();
        System.out.println(instance2);

        InstanceCounter instance3 = new InstanceCounter();
        System.out.println(instance3);

        System.out.println("Total instances: " + InstanceCounter.getInstanceCount());
    }
}

```



```

Number of instances created: 1
Number of instances created: 2
Number of instances created: 3
Total instances: 3

```

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.

Solution:

Class File 1:

```

package com.logger;

public class Logger {
    private static Logger loggerInstance = null;
    private StringBuilder logMessages;

    private Logger() {
        logMessages = new StringBuilder();
    }

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

```



```

        System.out.println(logger.getLog());
        break;
    case 3:
        logger.clearLog();
        System.out.println("Log cleared.");
        break;
    case 4:
        System.out.println("Exiting...");
        break;
    default:
        System.out.println("Invalid choice! Please try again.");
    }
} while (choice != 4);

scanner.close();
}
}

```

```

Menu:
1. Add log message
2. Display log messages
3. Clear log messages
4. Exit
Enter your choice: 1
Enter log message: Error!
Log message added.
Menu:
1. Add log message
2. Display log messages
3. Clear log messages
4. Exit
Enter your choice: 2
Current log:
Error!

Menu:
1. Add log message
2. Display log messages
3. Clear log messages
4. Exit
Enter your choice: 3
Log cleared.

```

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.

Solution:

Class File 1:

```
package com.employee;

public class Employee {
    private static int totalEmployees = 0;
    private static double totalSalaryExpense = 0;

    private int id;
    private String name;
    private double salary;

    static {
        totalEmployees = 0;
        totalSalaryExpense = 0;
    }

    public Employee(int id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
        totalEmployees++;
        totalSalaryExpense += salary;
    }

    public static int getTotalEmployees() {
        return totalEmployees;
    }
}
```

```

    public static void applyRaise(double percentage) {
        totalSalaryExpense += totalSalaryExpense * (percentage / 100);
    }

    public static double calculateTotalSalaryExpense() {
        return totalSalaryExpense;
    }

    public void updateSalary(double newSalary) {
        totalSalaryExpense -= this.salary;
        this.salary = newSalary;
        totalSalaryExpense += newSalary;
    }

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

```

Class File 2:

```

package com.employee;

import java.util.Scanner;

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

        do {
            System.out.println("Menu:");

            System.out.println("1. Create new employee");
            System.out.println("2. Display total number of employees");
            System.out.println("3. Apply raise to all employees");
            System.out.println("4. Display total salary expense");
            System.out.println("5. Exit");
            System.out.print("Enter your choice: ");

            choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
                case 1:
                    System.out.print("Enter Employee ID: ");
                    int id = scanner.nextInt();
                    scanner.nextLine();

```

```

        System.out.print("Enter Employee Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Employee Salary: ");
        double salary = scanner.nextDouble();
        new Employee(id, name, salary);
        System.out.println("Employee created.");
        break;

    case 2:
        System.out.println("Total Employees: " + Employee.getTotalEmployees());
        break;

    case 3:
        System.out.print("Enter raise percentage: ");
        double percentage = scanner.nextDouble();
        Employee.applyRaise(percentage);
        System.out.println("Salary raise applied.");
        break;

    case 4:
        System.out.println("Total Salary Expense: ₹" +
Employee.calculateTotalSalaryExpense());
        break;

    case 5:
        System.out.println("Exiting...");
        break;

    default:
        System.out.println("Invalid choice! Please try again.");
    }
} while (choice != 5);

scanner.close();
}
}

```

```
Menu:
1. Create new employee
2. Display total number of employees
3. Apply raise to all employees
4. Display total salary expense
5. Exit
Enter your choice: 1
Enter Employee ID: 00210
Enter Employee Name: Divya
Enter Employee Salary: 400000
Employee created.
Menu:
1. Create new employee
2. Display total number of employees
3. Apply raise to all employees
4. Display total salary expense
5. Exit
Enter your choice: 2
Total Employees: 1
Menu:
1. Create new employee
2. Display total number of employees
3. Apply raise to all employees
4. Display total salary expense
5. Exit
```

```
Enter your choice: 3
Enter raise percentage: 30
Salary raise applied.
Menu:
1. Create new employee
2. Display total number of employees
3. Apply raise to all employees
4. Display total salary expense
5. Exit
Enter your choice: 4
Total Salary Expense: ₹520000.0
Menu:
1. Create new employee
2. Display total number of employees
3. Apply raise to all employees
4. Display total salary expense
5. Exit
```