

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.

```
public class InstanceCounter {

    private static int instanceCount = 0;

    public InstanceCounter() {
        instanceCount++;
    }

    public static int getInstanceCount() {
        return instanceCount;
    }

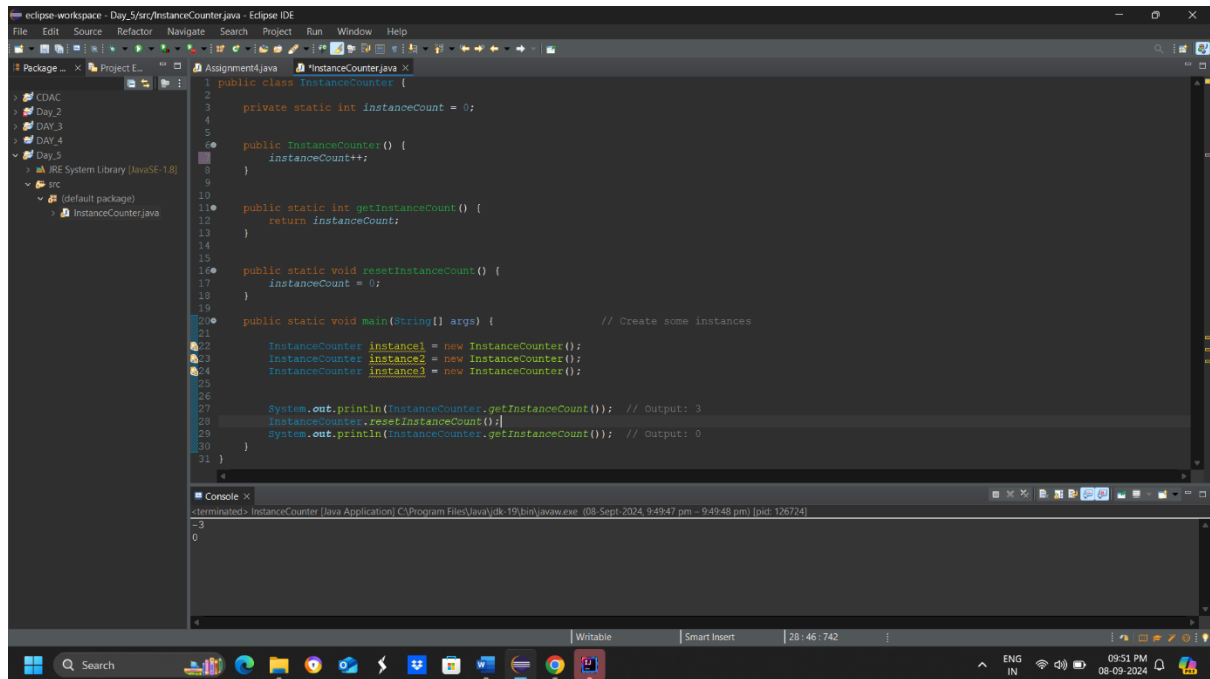
    public static void resetInstanceCount() {
        instanceCount = 0;
    }

    public static void main(String[] args) { // Create some instances

        InstanceCounter instance1 = new InstanceCounter();
        InstanceCounter instance2 = new InstanceCounter();
        InstanceCounter instance3 = new InstanceCounter();

        System.out.println(InstanceCounter.getInstanceCount()); // Output: 3
        InstanceCounter.resetInstanceCount();
        System.out.println(InstanceCounter.getInstanceCount()); // Output: 0
    }
}
```

ASSIGNMENT NO.6



The screenshot shows the Eclipse IDE with a project named 'Assignment4.java'. The main editor displays the 'InstanceCounter.java' file with the following code:

```
1 public class InstanceCounter {
2
3     private static int instanceCount = 0;
4
5
6     public InstanceCounter() {
7         instanceCount++;
8     }
9
10
11     public static int getInstanceCount() {
12         return instanceCount;
13     }
14
15
16     public static void resetInstanceCount() {
17         instanceCount = 0;
18     }
19
20     public static void main(String[] args) { // Create some instances
21
22         InstanceCounter instance1 = new InstanceCounter();
23         InstanceCounter instance2 = new InstanceCounter();
24         InstanceCounter instance3 = new InstanceCounter();
25
26
27         System.out.println(InstanceCounter.getInstanceCount()); // Output: 3
28         InstanceCounter.resetInstanceCount();
29         System.out.println(InstanceCounter.getInstanceCount()); // Output: 0
30     }
31 }
```

The console output shows the execution of the main method, resulting in the following output:

```
<terminated> InstanceCounter [Java Application] C:\Program Files\Java\jdk-19\bin\java.exe. (08-Sept-2024, 9:49:47 pm - 9:49:48 pm) [pid: 126724]
3
0
```

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.

```
public class Logger {

    private static Logger instance;

    private StringBuilder logMessages;

    private Logger() {

        logMessages = new StringBuilder();

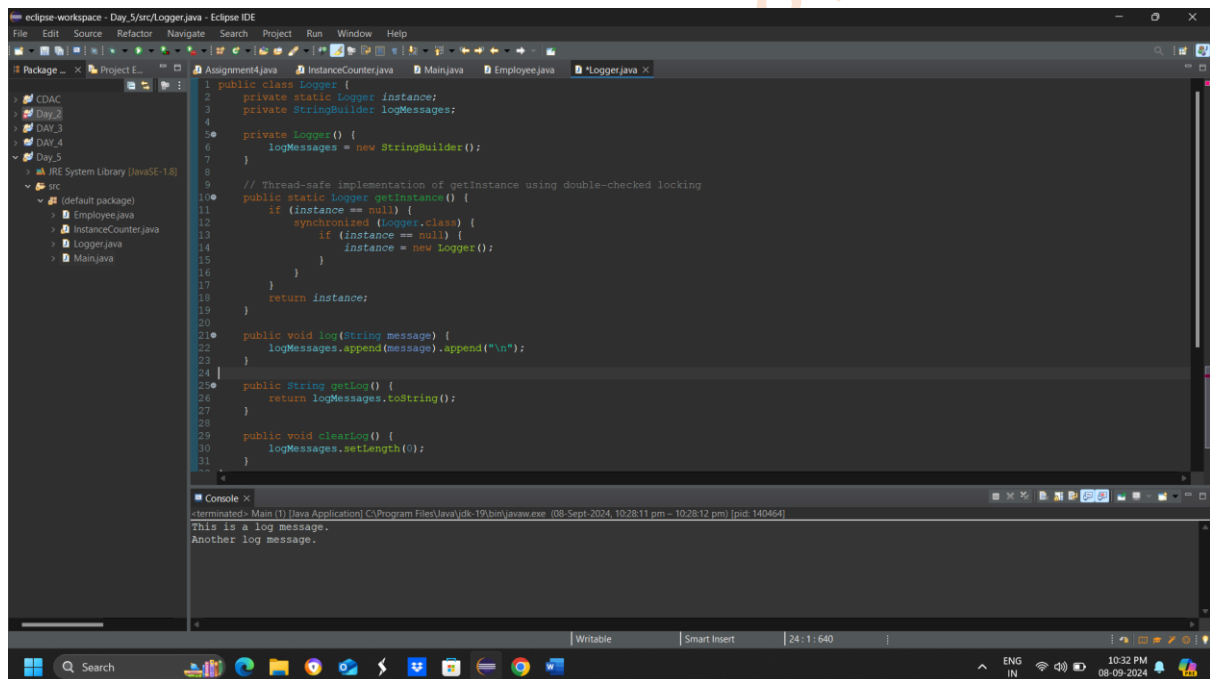
    }
```

// Thread-safe implementation of `getInstance` using double-checked locking

```
public static Logger getInstance() {  
    if (instance == null) {  
        synchronized (Logger.class) {  
            if (instance == null) {  
                instance = new Logger();  
            }  
        }  
    }  
    return instance;  
}  
  
public void log(String message) {  
    logMessages.append(message).append("\n");  
}  
  
public String getLog() {  
    return logMessages.toString();  
}  
  
public void clearLog() {  
    logMessages.setLength(0);  
}  
}
```

ASSIGNMENT NO.6

```
public class Main {  
  
    public static void main(String[] args) {  
  
        Logger logger = Logger.getInstance();  
  
        logger.log("This is a log message.");  
  
        logger.log("Another log message.");  
  
        System.out.println(logger.getLog());  
  
        logger.clearLog();  
  
        System.out.println(logger.getLog());  
  
    }  
  
}
```



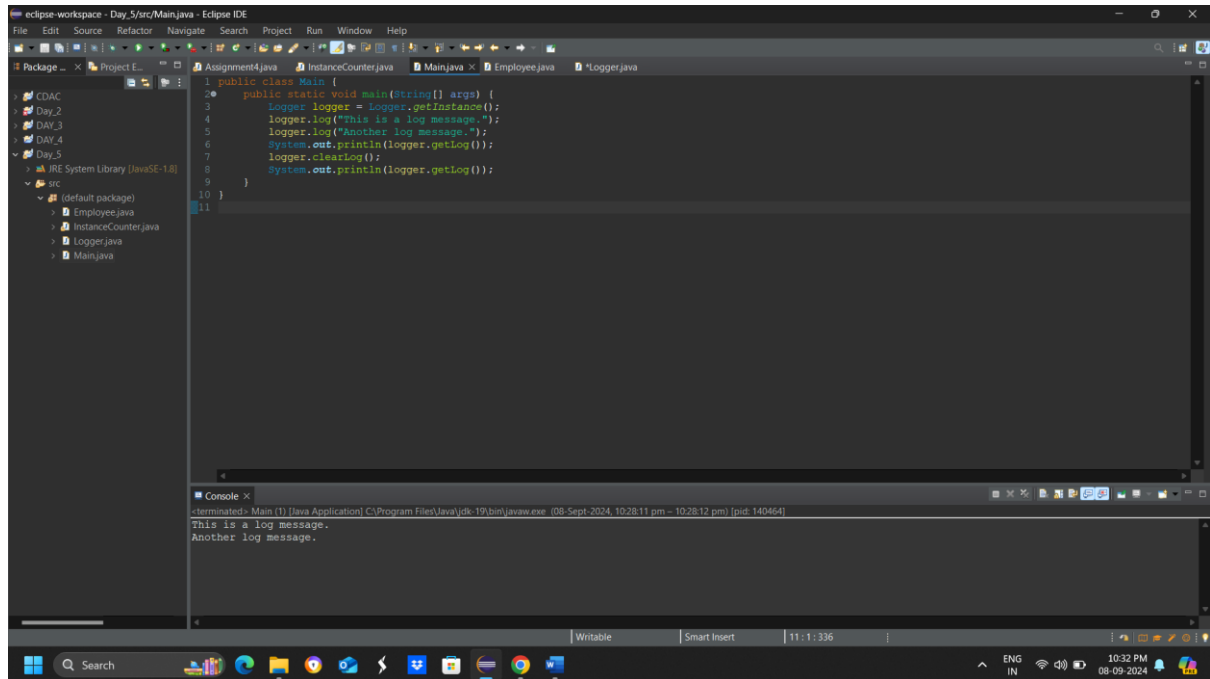
The screenshot shows the Eclipse IDE with the following details:

- Package Explorer:** Shows a project named 'Day_5' with sub-packages 'Day_2', 'Day_3', 'Day_4', and 'Day_5'. The 'Day_5' package contains 'Employee.java', 'InstanceCounter.java', 'Logger.java', and 'Main.java'.
- Editor:** Displays the 'Logger.java' file with the following code:

```
1 public class Logger {  
2     private static Logger instance;  
3     private StringBuilder logMessages;  
4  
5     private Logger() {  
6         logMessages = new StringBuilder();  
7     }  
8  
9     // Thread-safe implementation of getInstance using double-checked locking  
10    public static Logger getInstance() {  
11        if (instance == null) {  
12            synchronized (Logger.class) {  
13                if (instance == null) {  
14                    instance = new Logger();  
15                }  
16            }  
17        }  
18        return instance;  
19    }  
20  
21    public void log(String message) {  
22        logMessages.append(message).append("\n");  
23    }  
24  
25    public String getLog() {  
26        return logMessages.toString();  
27    }  
28  
29    public void clearLog() {  
30        logMessages.setLength(0);  
31    }  
32 }
```
- Console:** Shows the output of the 'Main' class:

```
<terminated> Main (1) [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (08-Sept-2024, 10:28:11 pm - 10:28:12 pm) [pid: 140464]  
This is a log message.  
Another log message.
```
- Status Bar:** Shows 'Writable', 'Smart Insert', and '24:1: 640'.

ASSIGNMENT NO.6



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.

```
public class Employee {
```

```
    private static int totalEmployees = 0;
```

```
    private static double totalSalaryExpense = 0.0;
```

```
private int id;
```

```
private String name;
```

```
private double salary;
```

```
static {
```

```
    totalEmployees = 0;
```

```
    totalSalaryExpense = 0.0;
```

```
}
```

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

```
    this.id = id;
```

```
    this.name = name;
```

```
    this.salary = salary;
```

```
    totalEmployees++;
```

```
    totalSalaryExpense += salary;
```

```
}
```

```
public int getId() {
```

```
    return id;
```

```
}
```

```
public void setId(int id) {
```

```
    this.id = id;
```

```
}
```

```
public String getName() {
```

```
    return name;
```

```
}
```

```
public void setName(String name) {
```

```
    this.name = name;
```

```
}
```

```
public double getSalary() {
```

```
    return salary;
```

```
}
```

```
public void setSalary(double salary) {
```

```
    this.salary = salary;
```

```
    totalSalaryExpense -= this.salary;
```

```
    totalSalaryExpense += salary;
```

```
}
```

```
public static int getTotalEmployees() {
```

```
return totalEmployees;
```

```
}
```

```
public static void applyRaise(double percentage) {
```

```
    totalSalaryExpense *= (1 + percentage / 100);
```

```
    for (Employee employee : Employee.getEmployees()) {
```

```
        employee.setSalary(employee.getSalary() * (1 + percentage / 100));
```

```
    }
```

```
}
```

```
public static double calculateTotalSalaryExpense() {
```

```
    return totalSalaryExpense;
```

```
}
```

```
public void updateSalary(double newSalary) {
```

```
    setSalary(newSalary);
```

```
}
```

```
public String toString() {
```

```
    return "Employee{" +
```

```
        "id=" + id +
```



```
        ", name=" + name + "\" +
```

```
        ", salary=" + salary +
```

```
        '}';
```

```
    }
```

```
    private static Employee[] employees = new Employee[0];
```

```
    public static Employee[] getEmployees() {
```

```
        return employees;
```

```
    }
```

```
    public static void addEmployee(Employee employee) {
```

```
        Employee[] newEmployees = new Employee[employees.length + 1];
```

```
        System.arraycopy(employees, 0, newEmployees, 0, employees.length);
```

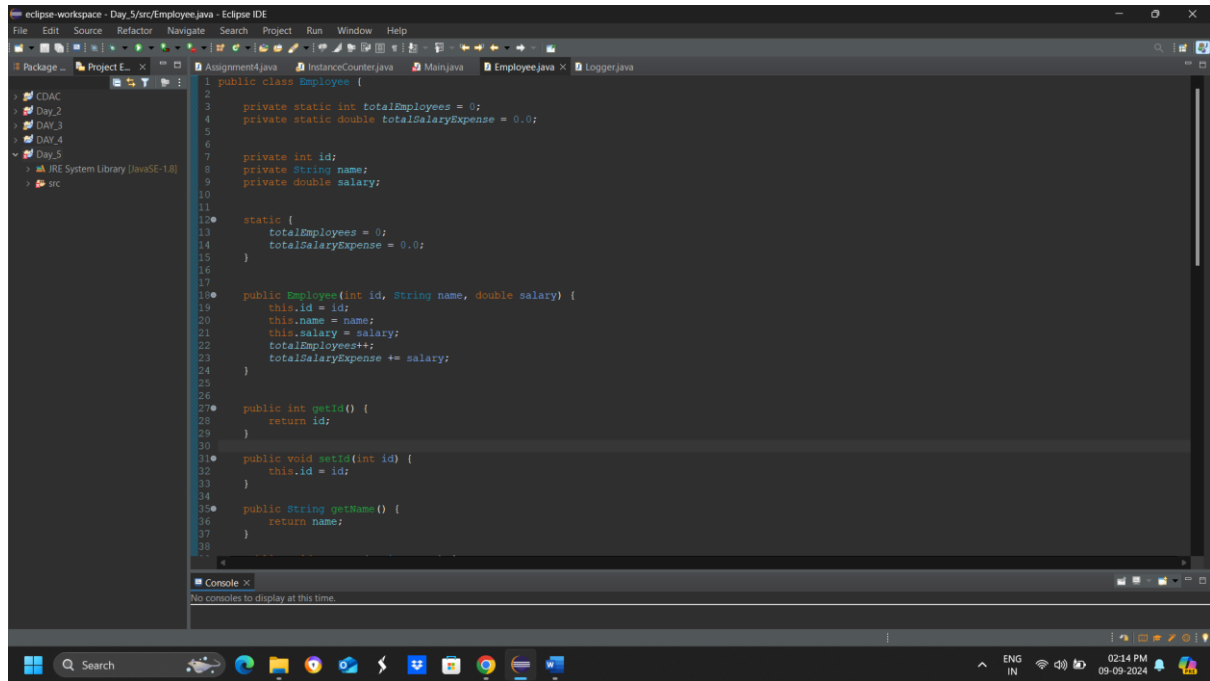
```
        newEmployees[employees.length] = employee;
```

```
        employees = newEmployees;
```

```
    }
```

```
}
```

ASSIGNMENT NO.6



```
1 public class Employee {
2
3     private static int totalEmployees = 0;
4     private static double totalSalaryExpense = 0.0;
5
6     private int id;
7     private String name;
8     private double salary;
9
10
11
12     static {
13         totalEmployees = 0;
14         totalSalaryExpense = 0.0;
15     }
16
17
18     public Employee(int id, String name, double salary) {
19         this.id = id;
20         this.name = name;
21         this.salary = salary;
22         totalEmployees++;
23         totalSalaryExpense += salary;
24     }
25
26
27     public int getId() {
28         return id;
29     }
30
31     public void setId(int id) {
32         this.id = id;
33     }
34
35     public String getName() {
36         return name;
37     }
38 }
```

Console ×
No consoles to display at this time.