ASSIGNMENT -5

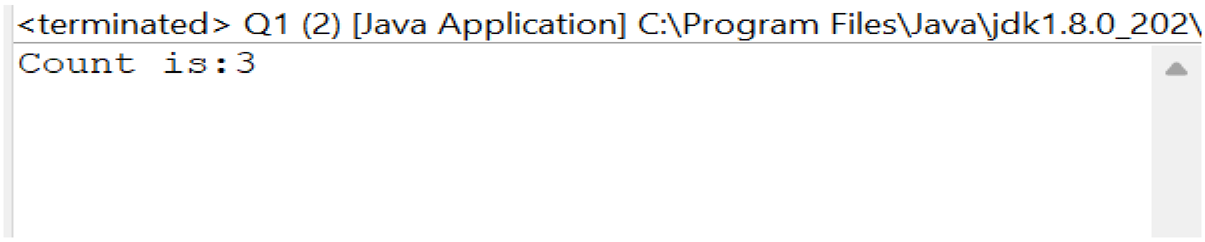
Note:

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.

```
package Assignment_5;
class InstanceCounter{  //class employee extends Object
private static int count;
public InstanceCounter() {
InstanceCounter.count=InstanceCounter.count+1;
}
public static int getCount() {
return count;
}
}
public class Q1 {
public static void main(String[] args) {
InstanceCounter ic1=new InstanceCounter();
InstanceCounter ic2=new InstanceCounter();
InstanceCounter ic3=new InstanceCounter();
System.out.println("Count is:" +InstanceCounter.getCount());
}
}
```

Output:


```
<terminated> Q1 (2) [Java Application] C:\Program Files\Java\jdk1.8.0_202\
Count is: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:

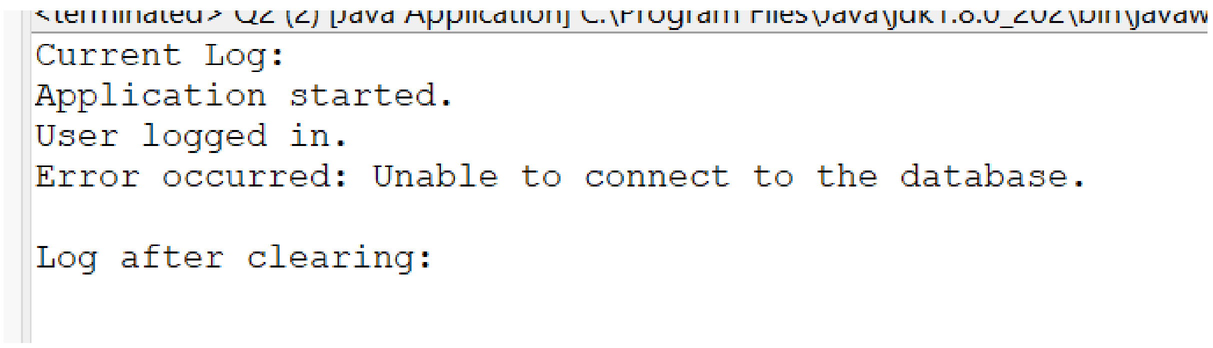
getInstance(): Returns the unique instance of the Logger class.

log(String message): Adds a log message to the logger.

getLog(): Returns the current log messages as a String.

clearLog(): Clears all log messages.

```
package Assignment_5;
class Logger {
private static Logger instance;
private StringBuilder logMessages;
// Private constructor to prevent instantiation
private Logger() {
logMessages = new StringBuilder();
}
// Public method to provide access to the singleton instance
public static synchronized Logger getInstance() {
if (instance == null) {
instance = new Logger();
}
return instance;
}
// Method to add a log message
public void log(String message) {
logMessages.append(message).append("\n");
}
// Method to get all log messages
public String getLog() {
return logMessages.toString();
}
// Method to clear all log messages
public void clearLog() {
logMessages.setLength(0);
}
}
public class Q2 {
public static void main(String[] args) {
Logger logger = Logger.getInstance();
// Logging some messages
logger.log("Application started.");
logger.log("User logged in.");
logger.log("Error occurred: Unable to connect to the database.");
// Getting and printing the log messages
System.out.println("Current Log:");
System.out.println(logger.getLog());
// Clearing the log messages
logger.clearLog();
System.out.println("Log after clearing:");
System.out.println(logger.getLog());
}
}
```



```
<terminated> Q2 (2) [Java Application] C:\Program Files\Java\jdk1.8.0_202\bin\javaw
Current Log:
Application started.
User logged in.
Error occurred: Unable to connect to the database.

Log 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())

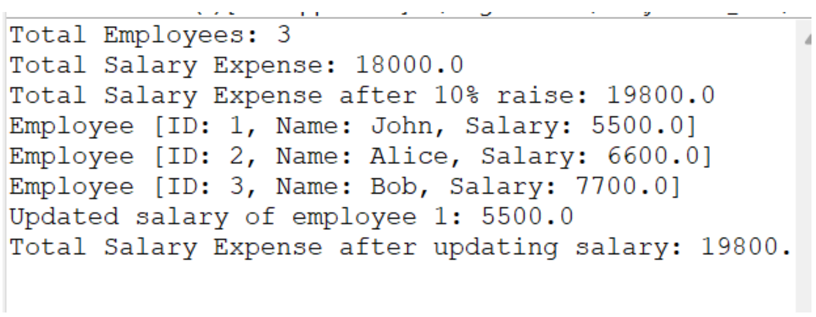
• Update the salary of an individual employee (updateSalary(double newSalary)) Understand the problem statement and use static and non-static fields and methods appropriately. Implement static and non-static initializers, constructors, getter and setter methods, and a toString() method to handle the initialization and representation of employee data. Write a menu-driven program in the main method to test the functionalities.

```
package Assignment_5;

import java.util.ArrayList;

import java.util.List;

class Employee {

private static int totalEmployees = 0;

private static double totalSalaryExpense = 0.0;

private int id;

private String name;

private float salary;

// List to store all employee objects

private static List<Employee> allEmployees = new ArrayList<>();

// Default constructor

public Employee() {
```

```
this(0, "", 0.0f);
}
// Constructor to initialize employee data
public Employee(int id, String name, float salary) {
this.id = id;
this.name = name;
this.salary = salary;
totalEmployees++;
totalSalaryExpense += salary;
allEmployees.add(this); // Add the employee to the list
}
// Getter and Setter methods
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 float getSalary() {
return salary;
}
public void setSalary(float salary) {
// Update total salary expense by subtracting old salary and adding new salary
totalSalaryExpense -= this.salary;
this.salary = salary;
```

```
totalSalaryExpense += salary;
}
public static int getTotalEmployees() {
return totalEmployees;
}
public static double getTotalSalaryExpense() {
return totalSalaryExpense;
}
// Method to apply raise to all employees
public static void applyRaise(double percentage) {
for (Employee employee : allEmployees) {
double raiseAmount = employee.salary * (percentage / 100);
employee.salary += raiseAmount;
totalSalaryExpense += raiseAmount;
}
}
// Method to update the salary of an individual employee
public void updateSalary(float newSalary) {
setSalary(newSalary);
}
// toString method to display employee details
@Override
public String toString() {
return "Employee [ID: " + id + ", Name: " + name + ", Salary: " + salary + "]";
}
// Method to return all employees (needed for applying raise)
public static List<Employee> getAllEmployees() {
return allEmployees;
}
}
public class Q3 {
```

```
public static void main(String[] args) {

// Create a few employee objects

Employee emp1 = new Employee(1, "John", 5000.0f);

Employee emp2 = new Employee(2, "Alice", 6000.0f);

Employee emp3 = new Employee(3, "Bob", 7000.0f);

// Display total number of employees and total salary expense

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

System.out.println("Total Salary Expense: " + Employee.getTotalSalaryExpense());

// Apply a 10% raise to all employees

Employee.applyRaise(10.0);

// Display total salary expense after raise

System.out.println("Total Salary Expense after 10% raise: " + Employee.getTotalSalaryExpense());

// Display all employee details after the raise

for (Employee e : Employee.getAllEmployees()) {

System.out.println(e);

}

// Update salary of an individual employee

emp1.updateSalary(5500.0f);

System.out.println("Updated salary of employee 1: " + emp1.getSalary());

// Display total salary expense after updating salary

System.out.println("Total Salary Expense after updating salary: " +
Employee.getTotalSalaryExpense());

}

}
```



```
Total Employees: 3
Total Salary Expense: 18000.0
Total Salary Expense after 10% raise: 19800.0
Employee [ID: 1, Name: John, Salary: 5500.0]
Employee [ID: 2, Name: Alice, Salary: 6600.0]
Employee [ID: 3, Name: Bob, Salary: 7700.0]
Updated salary of employee 1: 5500.0
Total Salary Expense after updating salary: 19800.
```