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.

QUESTIONS

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

Solution:

```
package com.example;

public class InstanceCounter {

// Static field to track the number of instances
private static int instanceCount = 0;

// Constructor increments instance count
public InstanceCounter() {
instanceCount++;
}

// Static method to get the number of instances
public static int getInstanceCount() {
return instanceCount;
}

@Override
public String toString() {
```

```
return "InstanceCounter [Current Instance Count = " + instanceCount +
"]";
}
}
```

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:

```
package com.example;

public class Logger {

// Static field to hold the single instance of Logger
private static Logger instance;

// StringBuilder to store log messages
private StringBuilder logMessages;

// Private constructor to prevent instantiation
private Logger() {
logMessages = new StringBuilder();
}


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

// to add a log message
public void log(String message) {
logMessages.append(message).append("\n");
}

// to get all log messages
public String getLog() {
return logMessages.toString();
}

// to clear all log messages
public void clearLog() {
logMessages.setLength(0);
}

@Override
public String toString() {
return "Logger [Log Messages = " + logMessages.toString() + "]";
}
}
```

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:

```
package com.example;

public class Employee {

// Static fields to track total employees and total salary
private static int totalEmployees = 0;
private static double totalSalaryExpense = 0.0;
```

```java
// Non-static fields for individual employee details
private int id;
private String name;
private double salary;

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

// Static method to get total number of employees
public static int getTotalEmployees() {
return totalEmployees;
}

// Static method to apply a raise to all employees
public static void applyRaise(double percentage) {
totalSalaryExpense += totalSalaryExpense * (percentage / 100);
}

// Static method to calculate total salary expense
public static double calculateTotalSalaryExpense() {
return totalSalaryExpense;
}

// Method to update the salary of an individual employee
public void updateSalary(double newSalary) {
totalSalaryExpense -= this.salary;
this.salary = newSalary;
totalSalaryExpense += newSalary;
}

@Override
public String toString() {
```

```
return "Employee [ID = " + id + ", Name = " + name + ", Salary = ₹" +
salary + "]";
}
}
```

```
package com.example;

import java.util.Scanner;

public class Program {

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

// Testing InstanceCounter
System.out.println("Testing InstanceCounter:");
InstanceCounter obj1 = new InstanceCounter();
InstanceCounter obj2 = new InstanceCounter();
System.out.println("Number of InstanceCounter instances: " +
InstanceCounter.getInstanceCount());

// Testing Logger
System.out.println("\nTesting Logger:");
Logger logger = Logger.getInstance();
logger.log("Application started.");
logger.log("Processing data.");
System.out.println("Current Log:\n" + logger.getLog());
logger.clearLog();
System.out.println("Log after clearing:\n" + logger.getLog());

// Testing Employee
System.out.println("\nTesting Employee:");
Employee emp1 = new Employee(1, "Alice", 50000);
Employee emp2 = new Employee(2, "Bob", 60000);
System.out.println("Total Employees: " +
Employee.getTotalEmployees());
System.out.println("Total Salary Expense: ₹" +
Employee.calculateTotalSalaryExpense());
Employee.applyRaise(10); // Apply 10% raise
System.out.println("Total Salary Expense after 10% raise: ₹" +
Employee.calculateTotalSalaryExpense());
emp1.updateSalary(55000);
System.out.println("Total Salary Expense after salary update: ₹" +
Employee.calculateTotalSalaryExpense());

scanner.close();
}
}
```