

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

```
package ques1;
public class InstanceCounter {

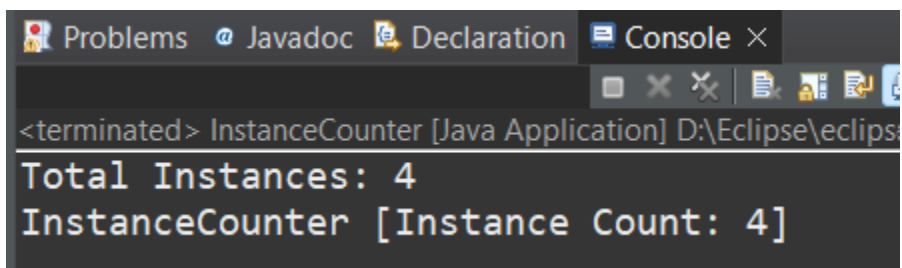
    private static int instanceCount = 0;

    // Constructor
    public InstanceCounter() {
        // Increment the instance count each time a new instance is created
        instanceCount++;
    }

    public static int getInstanceCount() {
        return instanceCount;
    }

    @Override
    public String toString() {
        return "InstanceCounter [Instance Count: " + instanceCount + "]";
    }

    // Main method for testing the InstanceCounter class
    public static void main(String[] args) {
        // Create instances of InstanceCounter
        InstanceCounter obj1 = new InstanceCounter();
        InstanceCounter obj2 = new InstanceCounter();
        InstanceCounter obj3 = new InstanceCounter();
        InstanceCounter obj4 = new InstanceCounter();
        // Print number of instances
        System.out.println("Total Instances: " + InstanceCounter.getInstanceCount());
        // Print the string representation of an instance
        System.out.println(obj1);
    }
}
```

A screenshot of the Eclipse IDE's console window. The window has tabs for 'Problems', 'Javadoc', 'Declaration', and 'Console'. The 'Console' tab is active, showing the output of the Java application. The output consists of two lines: 'Total Instances: 4' and 'InstanceCounter [Instance Count: 4]'. The first line is preceded by a '<terminated>' status.

```
<terminated> InstanceCounter [Java Application] D:\Eclipse\eclipse
Total Instances: 4
InstanceCounter [Instance Count: 4]
```

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 ques2;
public class Logger {

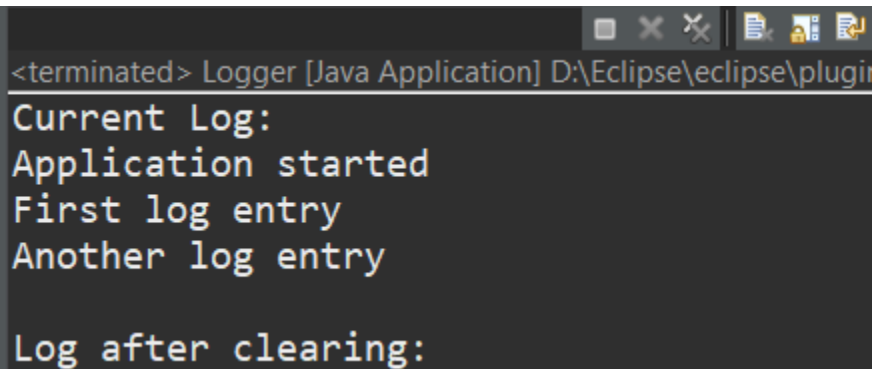
    private static Logger instance;

    // A StringBuilder to store log messages
    private StringBuilder logMessages;
    // Private constructor to prevent instantiation from outside
    private Logger() {
        logMessages = new StringBuilder();
    }
    // Static method to provide access to the single instance of the Logger
    public static synchronized Logger getInstance() {
        // Create the instance if it doesn't exist yet
        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 the current log messages
    public String getLog() {
```

```

        return logMessages.toString();
    }
    // Method to clear all log messages
    public void clearLog() {
        logMessages.setLength(0);
    }
    // Main method for testing the Logger class
    public static void main(String[] args) {
        // Get the unique Logger instance
        Logger logger = Logger.getInstance();
        // Log some messages
        logger.log("Application started");
        logger.log("First log entry");
        logger.log("Another log entry");
        // Retrieve and print log messages
        System.out.println("Current Log:");
        System.out.println(logger.getLog());
        // Clear the log
        logger.clearLog();
        // Print the log again to show it's cleared
        System.out.println("Log after clearing:");
        System.out.println(logger.getLog());
    }
}

```



```

<terminated> Logger [Java Application] D:\Eclipse\eclipse\plugin
Current Log:
Application started
First log entry
Another log entry

Log after clearing:

```

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 ques3;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
public class Employee {
    private static int totalEmployees = 0;
    private static double totalSalaryExpense = 0.0;
    // Non-static fields for individual employee
    private int id;
    private String name;
    private double salary;
    // List to hold all employee instances
    private static List<Employee> employees = new ArrayList<>();
    // Constructor to initialize an employee
    public Employee(int id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
        employees.add(this);
        totalEmployees++;
        totalSalaryExpense += salary;
    }
    public static int getTotalEmployees() {
        return totalEmployees;
    }
}
```

```

    }
    // Static method to apply a percentage raise to all employees
    public static void applyRaise(double percentage) {
        for (Employee emp : employees) {
            double newSalary = emp.salary * (1 + percentage / 100);
            emp.salary = newSalary;
        }
        updateTotalSalaryExpense();
    }
    // Static method to calculate the total salary expense
    public static double calculateTotalSalaryExpense() {
        return totalSalaryExpense;
    }
    // Non-static method to update the salary of an individual employee
    public void updateSalary(double newSalary) {
        totalSalaryExpense = totalSalaryExpense - this.salary + newSalary;
        this.salary = newSalary;
    }
    // Static method to update the total salary expense
    private static void updateTotalSalaryExpense() {
        totalSalaryExpense = 0;
        for (Employee emp : employees) {
            totalSalaryExpense += emp.salary;
        }
    }
    // Override toString() method for employee details
    @Override
    public String toString() {
        return "Employee [ID: " + id + ", Name: " + name + ", Salary: $" + salary +
        "]\n";
    }
    // Main method for testing
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int choice;
        do {
            System.out.println("Menu:");
            System.out.println("1. Add Employee");
            System.out.println("2. Apply Raise ");
            System.out.println("3. Calculate Total Salary");

```

```

System.out.println("4. Update Salary ");
System.out.println("5. Display Total Number of Employees");
System.out.println("6. Display All Employees");
System.out.println("7. Exit");
System.out.print("Enter your choice: ");
choice = scanner.nextInt();
scanner.nextLine(); // Consume newline
switch (choice) {
case 1:
    System.out.print("Enter ID: ");
    int id = scanner.nextInt();
    scanner.nextLine();
    System.out.print("Enter Name: ");
    String name = scanner.nextLine();
    System.out.print("Enter Salary: ");
    double salary = scanner.nextDouble();
    scanner.nextLine();
    break;
case 2:
    System.out.print("Enter raise percentage: ");
    double percentage = scanner.nextDouble();
    scanner.nextLine();
    Employee.applyRaise(percentage);
    System.out.println("Raise applied.");
    break;
case 3:
    System.out.println("Total Salary Expense: $" +
Employee.calculateTotalSalaryExpense());
    break;
case 4:
    System.out.print("Enter employee ID: ");
    int empld = scanner.nextInt();
    scanner.nextLine();
    System.out.print("Enter new Salary: ");
    double newSalary = scanner.nextDouble();
    scanner.nextLine();
    for (Employee emp : employees) {
        if (emp.id == empld) {
            emp.updateSalary(newSalary);
            System.out.println("Salary updated.");

```

```

        break;
    }
    }
    break;
case 5:
    System.out.println("Total Number of Employees: " +
Employee.getTotalEmployees());
    break;
case 6:
    for (Employee emp : employees) {
        System.out.println(emp);
    }
    break;
case 7:
    System.out.println("Exiting...");
    break;
default:
    System.out.println("Invalid choice. Try again.");
    break;
}
} while (choice != 7);
scanner.close();
}
}

```

Employee [Java Application] D:\Eclipse\eclipse\plugins\org.eclipse.justj.openjdk.

Menu:

1. Add Employee
2. Apply Raise
3. Calculate Total Salary
4. Update Salary
5. Display Total Number of Employees
6. Display All Employees
7. Exit

Enter your choice: 1

Enter ID: 12

Enter Name: srm

Enter Salary: 1234

Employee added.

Menu:

1. Add Employee
2. Apply Raise
3. Calculate Total Salary
4. Update Salary
5. Display Total Number of Employees
6. Display All Employees
7. Exit

Enter your choice: 6

Employee [ID: 12, Name: srm, Salary: \$1234.0]

Menu:

1. Add Employee
2. Apply Raise
3. Calculate Total Salary
4. Update Salary
5. Display Total Number of Employees
6. Display All Employees
7. Exit

Enter your choice: 5

Total Number of Employees: 1

Menu:

1. Add Employee
2. Apply Raise
3. Calculate Total Salary
4. Update Salary
5. Display Total Number of Employees
6. Display All Employees
7. Exit

Enter your choice: 1

Enter ID: 23

Enter Name: vrm

Enter Salary: 123456

Employee added.

Menu:

1. Add Employee
2. Apply Raise
3. Calculate Total Salary
4. Update Salary
5. Display Total Number of Employees
6. Display All Employees
7. Exit

Enter your choice: 6

Employee [ID: 12, Name: srm, Salary: \$1234.0]

Employee [ID: 23, Name: vrm, Salary: \$123456.0]

Menu:

1. Add Employee
2. Apply Raise
3. Calculate Total Salary
4. Update Salary
5. Display Total Number of Employees
6. Display All Employees
7. Exit