# **Assignment: Exploring Java Static Keyword and Real-World Scenarios**

# **Question 1: Singleton Design Pattern**

# Implement a Singleton class called **ConfigurationManager**, which stores application configuration settings. The class should have a private static instance variable and a static method **getInstance()** to retrieve the instance. Demonstrate how the Singleton pattern ensures that only one instance of **ConfigurationManager** can be created.

# **Question 2: Utility Class with Static Methods**

# Create a utility class named **MathUtil** that contains static methods for performing basic mathematical operations: addition, subtraction, multiplication, and division. Utilize the class to perform calculations without creating instances. Provide an example of how to use these static methods.

# **Question 3: Object Instance Counter**

# Design a class named **Product** that represents products in an inventory. Implement a static variable to keep track of the number of **Product** instances created. Include a constructor to initialize product details and demonstrate how the instance counter changes with each object creation.

# **Question 4: Managing Shared Resources**

# Develop a class named **DatabaseConnection** that simulates a database connection. Implement a private static variable to hold the connection status. Design static methods to open and close the connection. Ensure thread safety when accessing the shared connection resource.

# **Question 5: Global Configuration Management**

# Create a global configuration manager class called **AppConfig** with static methods to set and retrieve application-specific settings such as logging level, timeout values, and API endpoints. Implement a method to load initial configuration from a properties file.

# **Question 6: Static Factory Methods**

# Design a static factory class named **ShapeFactory** that provides static methods to create instances of different geometric shapes: **Circle**, **Rectangle**, and **Triangle**. Each method should accept necessary parameters to create the shape. Demonstrate the usage of these factory methods.

# **Question 7: Class Constants and Enums**

# Define an enum named **Color** with static methods to return color codes for common colors (e.g., RED, GREEN, BLUE). Additionally, create a class **Constants** with static final variables for common mathematical constants like PI and Euler's number.

# **Question 8: Caching Mechanism**

# Implement a simple caching mechanism using a static **HashMap**. Create a class named **CacheManager** with static methods to add data to the cache, retrieve data from the cache, and clear the cache. Demonstrate the caching process with sample data.

# **Question 9: Inheritance and Static Methods**

# Create a superclass named **Vehicle** with a static method **showInfo()** that prints "This is a vehicle." Create a subclass **Car** that extends **Vehicle** and overrides the **showInfo()** method to print "This is a car." Demonstrate how the behavior of static methods changes when accessed through the subclass.

# **Question 10: Using Static Initialization Blocks**

# Design a class named **StaticInitializer** with a static variable **count** and a static initialization block that increments **count** by 1. Create multiple instances of the class and observe the value of **count** for each instance.