PES University, Bengaluru

Department of Computer Science and Engineering

UE22CS352A: OBJECT ORIENTED ANALYSIS AND DESIGN

LAB 9 : Creational Design Patterns

1) Title : Implementing a Thread-Safe Logger System using Singleton Pattern using Java/C++

Description:

You are required to design and implement a Logger System in Java/C++ using the Singleton design pattern. The system should allow multiple threads to write log messages to a shared Linked List, where each log entry is represented as a node containing:

1. A timestamp indicating when the request was received by the logger class.
2. A log message as a string which starts with your srn.

Each thread should write exactly 10 log messages, with a random delay between 0 to 1 second between consecutive writes. Once all threads finish execution, the log messages stored in the linked list should be displayed in the order they were written.

Requirements:

- Singleton Implementation: Implement a `Logger` class following the Singleton pattern to ensure only one instance manages log writing.
- Thread Safety: Use appropriate synchronization to prevent race conditions when multiple threads log messages concurrently.
- Timestamped Log Entries: Each log node should store a timestamp marking when the request was received.
- Random Delay Simulation: Introduce a random timeout between 0 to 1 second between log writes to mimic real-world logging behavior.

- Multi-Threaded Execution: Create multiple threads, each writing 10 messages to the logger.
- Linked List Log Storage: Instead of writing to a file, maintain a singly linked list where each node represents a log entry (timestamp + srn + message).
- Final Log Display: Once all threads have finished, print the contents of the linked list in order.
- Singleton Verification: Print the hash code of the log object for each log entry generated by each thread to verify the singleton instance.

2) Title : Implementing an E-commerce Payment Processing System using Abstract Factory Pattern using Java/C++

Description:

You are required to design and implement an E-commerce Payment Processing System in Java/C++ using the Abstract Factory design pattern. The system should support multiple payment methods (Credit Card, UPI, and Payment on Delivery) while ensuring flexibility in adding new payment types in the future.

Requirements:

- Abstract Factory Design: Implement an abstract factory interface or class that declares methods for creating different types of payments, such as `CreditCardFactory`, `UPIFactory`, and `PaymentOnDeliveryFactory`.
- Concrete Factories: Create concrete implementations of the abstract factory for each payment type. Each factory should generate specific payment objects:
  - `CreditCardFactory`: Creates `VisaPayment`, `MasterCardPayment`.
  - `UPIFactory`: Creates `GooglePayPayment`, `PhonePePayment`.
  - `PaymentOnDeliveryFactory`: Creates `CashPayment`, `WalletPayment`.
- Payment Interface: Define a common `Payment` interface with methods like `processPayment(double amount)` and `getDiscount(double amount)`.
- Concrete Payment Implementations: Implement different payment options, ensuring each follows the `Payment` interface and has its own processing logic.
- Payment Gateway: Create a payment gateway class that utilizes the abstract factory pattern to process transactions dynamically based on user selection.
- Ensure that your SRN is included in every payment confirmation output generated by your code.