Software Design and Development

Deliverable #2: Creational Design Patterns

Step 1. Java code Automated generation:

As we are extending the EduCertify project from the previous course (C-SW321), we have already designed a comprehensive Design Class Diagram. Also, we have transformed these class diagrams into Java class skeletons by the end of last semester. Our IDE used was visual studio 2022 in which we generated the architectural designs of the classes.

Attachments:

- *Skeleton of Codes Zip File*
- *Design Class Diagram Document*

Step 2. Creational Software Design Pattern:

Singleton Design Pattern:

1. Database Singleton Class

Intent:

- Control access to the database by allowing only a single instance shared connection throughout the application

Problem:

- The need for a singleton class came from disallowing multiple classes to open several database connections which will make performance and consistency issues.

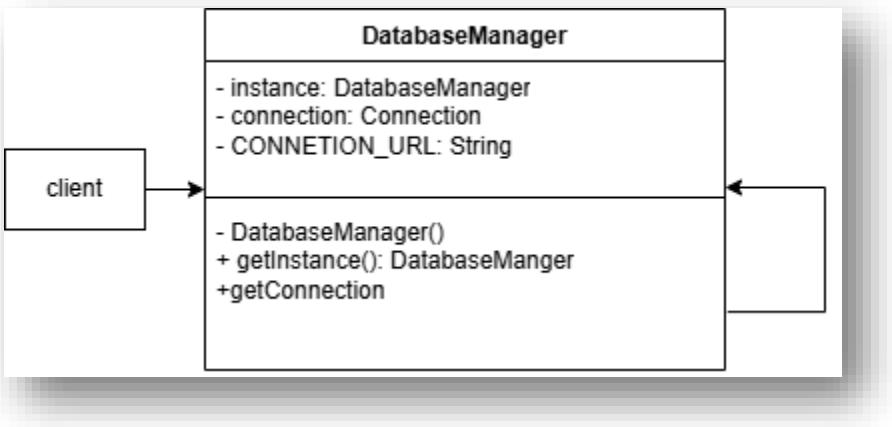
Solution:

- The solution is the implementation of a singleton class to ensure that the class is only instantiated once and has a global point of access to it.
- Implemented using a private constructor, volatile static instance, and a synchronized method with double-check locking.

Pattern Characteristics:

- Private static volatile Database instance
- Private constructor to ensure the creation of one instance
- (get Instance) public method that calls the private constructor.

Conceptual Model UML:



2. Generate Certificate Singleton Class

Intent:

- Ensures that certificates are consistently generated by a single instance.

Problem:

- Multiple instances can result in serious inconsistent formatting of certificates which may cause confusion or unprofessional output.

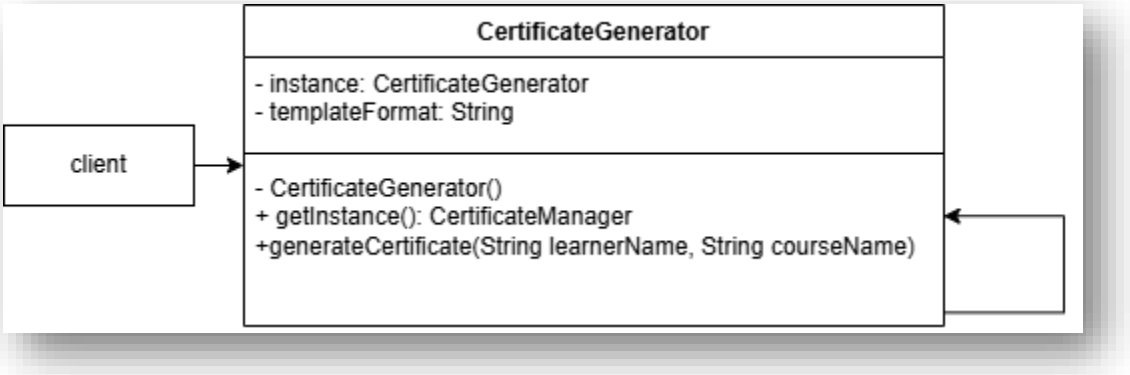
Solution:

- Applying a singleton design pattern to create a single instance of certificates format.
- This ensures that all certificates must use the same format safely in a multi-threaded environment too.

Pattern Characteristics:

- Private static volatile certificate generator instance.
- Private constructor to ensure the creation of one instance.
- (get Instance) public method that calls the private constructor.
- Generate Certificate method that returns a formatted string.

Conceptual Model UML:



Factory Method Design Pattern:

Notification Factory Method Design Pattern:

Intent:

- Create different types of notifications (Email, SMS, In-App) without hardcoding object creation by creating a common interface.

Problem:

- Adding new notification types would require changes in the main implementation which will violate the open-closed design principle.

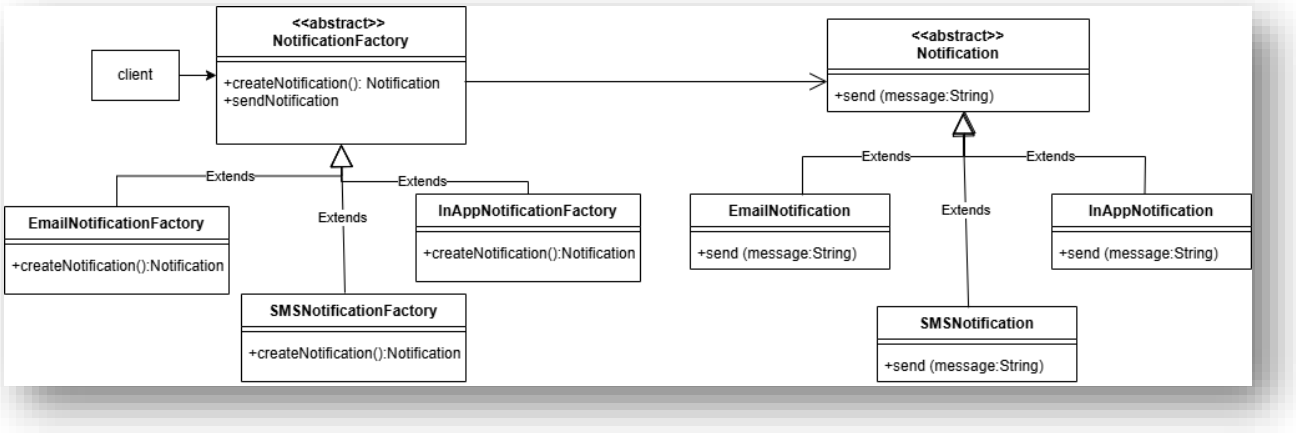
Solution:

- Using factory method pattern to create different notification objects based on the user preference.
- This pattern allows the system to decide which object to instantiate at runtime.

Pattern Characteristics:

- Notification Factory class to define createNotification() method, and concrete factories decide which type of notification to instantiate.
- sendNotification(): handles the logic for sending the correct type of notification.
- *Product Interface*: Notification (Abstract)
- *Concrete Products:* Email, SMS, In App Notifications
- *Abstract Factory*: NotificationFactory that has the factory method.
- *Concrete Factories*: Email, SMS, In App Notification Factories.

Conceptual Model UML



Abstract Factory Design Patterns:

1.Assessment Abstract Factory

Intent:

- Generate various assessment types (quiz, assignment, final exam) and each is associated with question types (MCQ, True/False, Essay)

Problem:

- Managing different assessment and question combinations without making it grow exponentially and making it more extensible to achieve the Open-Closed principle.

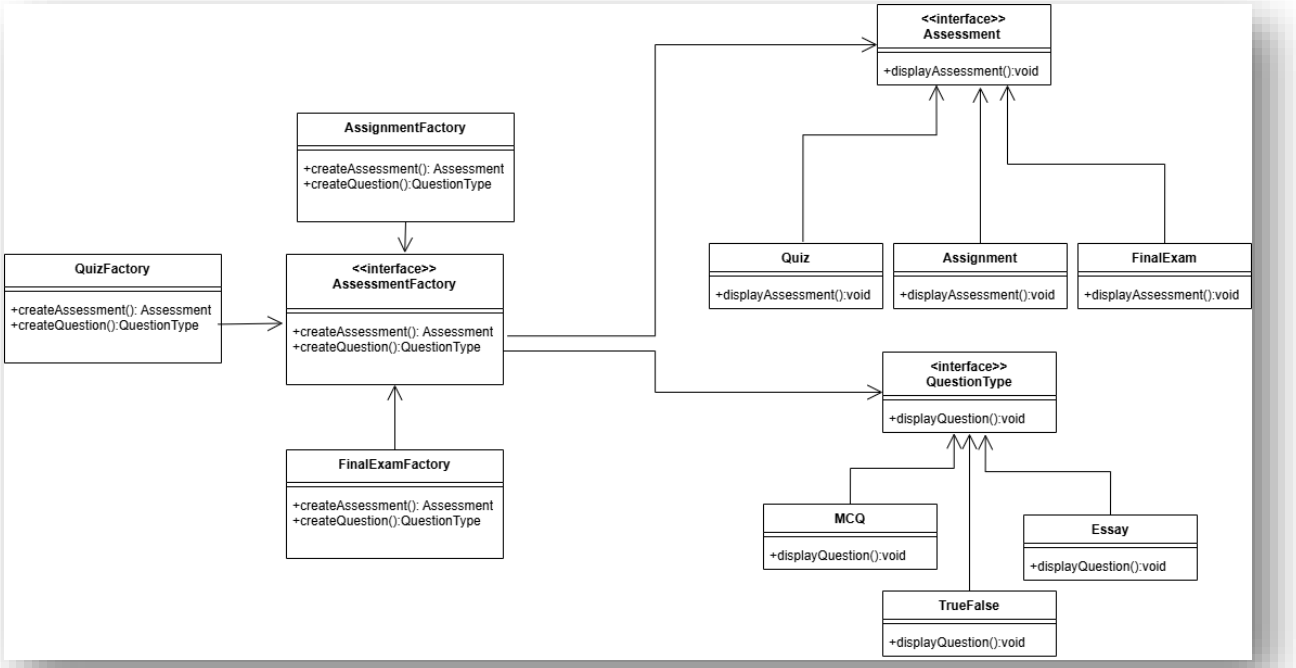
Solution:

- Create an abstract factory with product interfaces (assessment and questionType)
- Concrete classes to match pairs of combinations.

Pattern Characteristics:

- *Product 1 Interface*: Assessment
- *Product 2 Interface*: QuestionType
- *Concrete Products 1*: Quiz, Assignment, Final Exam
- *Concrete Products 2*: MCQ, T/F, Essay
- *Interface Abstract Factory*: AssessmentFactory that includes the factory method
- *Concrete Factories*: Quiz, assignment, final exam concrete factories

Conceptual Model UML



2.User Interface Theme Abstract Factory

Intent:

- Create UI components and themes depend on user setting and preferences.

Problem:

- Users with different preferences may need to require different UI styles combinations together.

Solution:

- Create an abstract factory pattern to encapsulate UI related families under a common interface. This will ensure decoupling and better creation of objects during runtime.

Pattern Characteristics:

- *Product 1 Interface*: Theme
- *Product 2 Interface*: Component
- *Concrete Products 1*: LightTheme, DarkTheme
- *Concrete Products 2*: Button, Menu, TextStyle
- *Interface Abstract Factory*: UIThemeFactory
- *Concrete Factories*: LightThemeFactory, DarkThemeFactory

Conceptual Model UML