Creational pattern

Singleton pattern

Context:

A class admin should exit from it one instance only.

Problem:

Ensure that it is never possible to create more than one instance of a singleton class (Admin).

Solution:

Make the constructor private (Admin) to ensure that no other class will be able to create an instance of the class singleton (Admin).

Define a public static method (Admin), The first time this method is called , it creates the single instance of the class “singleton” and stores a reference to that object in a static private variable.

BEHAVIOURAL Patterns

Observer (Publish-Subscribe) Pattern

Context:

We have two class teacher & student and we want to couple between them to be in touch with each other.

Problem:

We need a mechanism to ensure that when the state of an object from teacher class changes related objects from student class are updated to keep them instep.

Forces:

We want different parts of system to kept in step with one another without being too tightly coupled.

Solution:

teacher has the role of the subject (publisher) and student has the role of the observer (subscribers) the observers (student) register themselves with the subject (teacher) and if the state of the teacher changes the user are notified & can update them selves

Structural pattern

Delegation Pattern

Context:

When we design a method (view schedule) in a class teacher. We realize that student class

has a method (view schedule) which provides the required service.

Problem:

How can we make use of a method that already exists in the other class?

Forces:

We want to minimize development cost by reusing methods.

Solution:

The delegating method (teacher. View schedule) in the delegator class (teacher) calls a method in

the delegate class(student) to perform the required task(the method of view schedule in teacher class). An association must exist between the delegator and delegate classes.

Abstraction occurrence

Context:

In a domain model you find a set of related objects “occurrences”(course id); the members of such a set share common information but also differ from each other in important ways.

Problem:

What is the best way to represent such sets of occurrences?

Forces:

You want to represent the members of each set of occurrences without duplicating the common information.

Solution:

Create an “abstraction” class (quiz ,exam)that contains the common data.

Then create an “occurrence”(course id)class representing the occurrences of this abstraction.

Connect these classes with a one-to-many association.