Moto of Design Patterns

Code should be reliable,reusable,scalable

Clean Object Generation and Integration

Types of Design Pattern

1) creational -deals with object creation and intalization.this pattern gives program more flexiblity in deciding which objects needs to be created for a given case. eg: singleton, factory,abstract factory

2) structural - deals with class and object composition. this pattern focuses on decoupling interface and implementation of classes and its objects. eg: adapter,bridge etc.

3) behavioural - communication between classes and objects. eg; chain of responsiblity,command, intetrperator.

1. a) Single Ton Pattern

when one object of class is created. all reference to the objects are refered to the same underlying instance created

Lazy Initialization : The lazy initialization of an object improves the performance and avoids unnecessary computation till the point the object is accessed. Further, it reduces the memory footprint during the startup of the program. Reducing the memory print will help faster loading of the application.

Singleton with Lazy keyword (.NET 4.0) : Lazy keyword provides support for lazy initialization. In order to make a property as lazy, we need to pass the type of object to the lazy keyword which is being lazily initialized.

By default, Lazy<T> objects are thread-safe.  In multi-threaded scenarios, the first thread which tries to access the Value property of the lazy object will take care of thread safety when multiple threads are trying to access the Get Instance at the same time.

Therefore, it does not matter which thread initializes the object or if there are any thread race conditions that are trying to access this property.

1. Factory Pattern

Define an interface for creating an object but let subclasses decide which class to instantiate. The factory method lets a class defer instating it uses to subclasses.

Factory pattern creates objects without exposing the creation logic to the client and refer newly created object using a common interface

We will use factory - when objects need to extend to subclass, product impelemtnatio tend to change and change remain unchange,classes dont know what exact subclass it has create

Eg: usage

if

else

else if

benefits:-

tightly coupled business logic and more mainteance effort