Design Patterns

What are three reasons for studying design patterns?

Answer: Patterns make it possible to reuse solutions. Patterns help with communication between analysts, giving a shorthand terminology. Patterns give your perspective on the problem, freeing you from committing to a solution too early

Define pattern.

Answer: - A pattern is a solution to a problem that occurs in a given context.

What is a design pattern?

Answer: - In software engineering, design pattern or simply pattern is a universal reusable solution for a commonly occurring problem in software design. It is not a finished task that can be converted into codes directly. It is a template or description for how to solve a problem that can be used in different circumstances/situations. In Object Oriented Programming (OOP) it describes the relationship and interaction between classes or objects.

Why should we use design patterns?

Answer: - Design Patterns provides us lot of advantages. That's why we should use them. Some common advantages are given bellow:

- It provides solution to common problems which occur in software design
- It provides common platform for developers means developer can implement these in any language
- It provides a standard terminology and is specific to particular scenario

What are three reasons for studying design patterns?

Answer: -

- Patterns make it possible to reuse solutions
- Patterns help with communication between analysts, giving a shorthand terminology.
- Patterns give your perspective on the problem, freeing you from committing to a solution too early.

What about anti-patterns?

Answer: - The selection of wrong design pattern can perform a negative effect on codes. That's why it is required to choose appropriate design pattern. The selections of wrong desing pattern is defined as "anti-pattern".

Are design patterns the same thing as frameworks?

Answer: - No, both are different. Design pattern provides only the guideline, not codes. But Framework deals with codes.

What are the key elements in the description of a design pattern?

Answer: - To be complete, a pattern description must have the following eight elements-

- Name: a label that identifies it
- Intent: a description of the purpose of the pattern
- Problem: a description of the problem being solved
- Solution: what the solution is in the given context
- Participants / Collaborators: the entities involved in the solution
- Consequences: what happens as a result of using the pattern? What forces are at work.
- Implementation: how to implement the pattern in one or more concrete ways.
- GoF Reference: where to look in the Gang of Four book for more information.

What are the types of design patterns?

Answer: - Design Patterns can be organized in four pattern groups depending on the nature of the design problem they intend to solve

- Gang of Four (GoF) Patterns
- Enterprise Patterns
- SOA and Messaging Patterns
- Model-View Patterns

What is gang of four (gof) design patterns?

Answer: - In 1994, four authors Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides published a book named Elements of Reusable Object-Oriented Software that explains the concept of design pattern in software development. These four authors are known as Gang of Four (GOF).

Which pattern is the foundation of design patterns?

Answer: - Gang of Four (GoF) design patterns is the foundation for all other design patterns.

How patterns are in GoF design patterns?

Answer: - 23

What are the main categories of GoF pattern?

Answer: - GoF design pattern can be categorized into 3 groups.

- Creational Patterns deals mainly with creation of Classes & Objects.
- Behavioral Patterns deals with Class & Object communication.
- Structural Patterns deals with Class & Object Composition.

Give the list of GoF design patterns?

Answer: - The list is given bellow:

- Creational Design Patterns
  - Factory Method
  - Abstract Factory
  - Builder
  - Prototype
  - Singleton
- Structural Design Patterns
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Façade
  - Flyweight
  - Proxy
- Behavioral Design Patterns
  - Chain of Responsibility
  - Command
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Visitor
  - Template Method

The Singleton Pattern

What is Singleton Pattern?

Answer:-The one of the simplest design patterns is Singleton pattern. This pattern ensures that a class has only one instance in the whole application and provides a global point of access to it.

When to use Singleton Pattern?

Answer:-In a project when we want only one instance of a particular class will be created and all the modules of the entire project will use this instance. For example consider the class country which will loads the entire country list. We would like to share a single instance of this class by not hitting the database again and again. This will increase the performance of the application.

How to implement singleton pattern in .NET?

Answer:-We can implement the singleton pattern in .net by following the three steps:

- Create a class with static members.
- Define a private constructor to the class.
- Provide a static method to get access

Provide a static method to get access to the singleton object.
For example consider the following C# codes:

```
public class mySingleton

{

   private static mySingleton instance;

   private mySingleton() { }

   public static mySingleton Instance

   {

      get

      {

         if (instance == null)
```

```
        {
            instance = new mySingleton();
        }
        return instance;
    }
  }
}
```

What is the intent of the Singleton pattern?

Answer: Ensure a class only has one instance, and provide a global point of access to it.

The Singleton uses a special method to instantiate objects. What is special about this method?

Answer: When this method is called, it checks to see if the object has already been instantiated. If it has, the method simply returns a reference to the object. If not, the method instantiates it and returns a reference to the new instance.

To ensure that this is the only way to instantiate an object of this type, I define the constructor of this class to be protected or private.

What do I say is the difference in when to use the Singleton and Double-Checked Locking patterns?

Answer: The distinction between the patterns is that the Singleton pattern is used in single-threaded applications while the Double-Checked Locking pattern is used in multithreaded applications. Double-Checked must focus on synchronization in creations in case two objects try to create an object at exactly the same moment. This avoids unnecessary locking. It does this by wrapping the call to new with another conditional test. Singleton does not have to worry about this.

Can the singleton class be sub classed?

Answer: Singleton is just a design pattern and it can be sub classed. However it is worth to understand the logic or requirement behind sub classing a singleton class as the child class might not inherit the singleton pattern objective by extending the Singleton class. However the sub classing can be prevented by using the final keyword in the class declaration.

Protected Constructor

It is possible to use a protected constructor to in order to permit the sub classing of the singleton. This technique has 2 drawbacks that makes singleton inheritance impractical:

- First of all, if the constructor is protected, it means that the class can be instantiated by calling the constructor from another class in the same package. A possible solution to avoid it is to create a separate package for the singleton.

- Second of all, in order to use the derived class all the getInstance calls should be changed in the existing code from Singleton.getInstance() to NewSingleton.getInstance()

Why can't we use a static class instead of singleton?

Answer:-Intention of a singleton pattern is to ensure that a single instance of a class is instantiated. So why not static class? Reasons are below:-

- One of the key advantage of singleton over static class is that it can implement interfaces and extend classes while the static class cannot (it can extend classes, but it does not inherit their instance members). If we consider a static class it can only be a nested static class as top level class cannot be a static class. . Static means that it belongs to a class it is in and not to any instance. So it cannot be a top level class.
- Static class will have all its member as static only unlike Singleton
- It can be lazily loaded whereas static will be initialized whenever it is first loaded
- Singleton object stores in Heap but, static object stores in stack
- We can clone the object of Singleton but, we cannot clone the static class object
- Singleton can use the Object Oriented feature of polymorphism but static class cannot.

What type of pattern is the Singleton? What general category of pattern does it belong to?
Answer: - It is a type of Factory pattern

The Factory Method Pattern

What are factories responsible for?
Answer: - Factories are responsible for creating objects and ensuring objects are available to be used.

What is the essential reason to use a Factory Method?
Answer: - You want to defer the decision for instantiating a derivation of another class to a derived class

What is Factory method pattern?

Answer: - The factory method pattern creates objects at run-time based on demand. This pattern is the most used design pattern in modern programming languages like Java and C#. For example consider an ice-cream factory which produces different types of ice-creams. If we went to the ice-cram factory and order for chocolate ice-cream, factory method will produce chocolate ice-cream and will provide us.

When to use Factory method pattern?

Answer: - The factory method pattern is used when:-

- The Creation of object is done when it is required.
- Flexibility is important.
- Subclasses decide what objects should be created.

The Abstract Factory Pattern

What is Abstract factory pattern?

Answer:-The abstract Factory patterns acts as like super-factory which creates other factories. That's why this pattern is also called Factory of factories. This pattern is used to create a set of related objects or dependent objects. For creating objects this pattern internally use Factory method design pattern. Also this pattern may use Builder design pattern or prototype design pattern. But this completely depends on choice.

When to use Abstract factory pattern?

Answer:- The Abstract factory pattern is used when the need to create a set of related objects or dependent objects

What is the difference between factory and Abstract factory pattern?

Answer:-The factory method is a single method and an abstract factory is an object. Factory method produces one product. Abstract factory produce families of products. For example consider Mr. XX needs a door for his house. He approaches a carpenter. He gives the measurement of the door. After some days the carpenter provides the door. In this case, the carpenter is a factory of doors. Here specifications are the inputs for the factory and the door is the output or product of the factory.
Consider the same situation. Mr. XX goes to a carpenter or door shops. All of them are door factories. Based on the situation he can approach what kind of factory he needs. This is like abstract factory.

What are the three key strategies in the Abstract Factory?

Answer:-

- Find what varies and encapsulate it.
- Favor aggregation over inheritance.
- Design to interfaces, not to implementations.

What are the consequences of the Abstract Factory pattern?

Answer:-The Abstract Factory isolates the rules about which objects to use from the logic about how to use these objects.

In this pattern, there are two kinds of factories. What does the "Abstract Factory" class do? What do the "concrete factory" classes do?

Answer:-The "Abstract Factory" class specifies which objects can be instantiated by defining a method for each type of object.
The "concrete factory" classes specify which objects are to be instantiated

What is prototype pattern?

Answer:-The prototype pattern creates a new object from the existing instance of the object. This pattern is used to create a duplicate object or clone of the current object to enhance performance.

When to use prototype pattern?

Answer:-The prototype pattern is used when the creation of object is costly or complex, a limited number of state combinations exist in an object.

What is MVC pattern?

Answer:-The Model View Controller (MVC) is a design pattern used in software engineering. This pattern separates the structure and behavior of the applications into three component/parts: model, view and controller.

1. Model
   Model is the data access layer. This can be direct data access, web services, etc
2. View
   View is the presentation layer of the application. This the users interface.
3. Controller
   Controller is the business logic layer for the application. It controls Models & View.

Which pattern is widely used in programming?

Answer:-The MVC pattern is widely used in software development with many programming language like: java, C#, C, C++, PHP etc.

The Façade pattern

Define Façade?

Answer: - A Façade is "The face of a building, especially the principal face". It is the front that separates the street from the inside.

What is the intent of the Façade pattern?

Answer: - Provide a unified interface to a set of interfaces in a sub-system.

What are the consequences of the Façade pattern? Give an example.

Answer: - The Façade simplifies the use of the required subsystem. However, since the Façade is not complete, certain functionality may be unavailable to the client.

Example is a reporting application that needs a routine way to access on certain portions of a database system: The Façade would provide an interface to those portions and not the entire API of the database.

In the Façade pattern, how do clients work with subsystems?

Answer: - Clients work with sub-systems through the Façade's interfaces. They do not interact with the underlying methods directly.

Does the Façade pattern usually give you access to the entire system?

Answer: - Not usually. In general, Façade give access to a portion of the system, one that is customized to our needs

The Adapter Pattern

Define Adapter.

Answer: - Adapter" is something that allows one thing to modify itself to conform to the needs of another thing.

What is the intent of the Adapter pattern?

Answer: - The intent of the Adapter is to match an existing object that is beyond your control to a particular interface.

What are the consequences of the Adapter pattern? Give an example.

Answer: - A consequence is that the pattern allows for preexisting objects to fit into new class structures without being limited by their interfaces (p. 102). The example in the book is the drawing program that wants to use an existing Circle object but the existing object doesn't provide exactly the same methods as the rest of the system. The Adapter provides a translation to these methods

Which object-oriented concept is being used to define the relationship between Shape and Points, Lines, and Squares?

Answer: - Polymorphism

What is the most common use for the Adapter pattern?

Answer: - To allow for continued use of polymorphism. It is often used in conjunction with other design patterns.

What does the Adapter pattern free you from worrying about?

Answer: - Adapter frees me from worrying about the interfaces of existing classes when doing a design. If the class doesn't do what I need, I can create an Adapter to give it the correct interface.

What are the two variations of the Adapter pattern?

Answer: -

- Object Adapter: relies on one object to contain the other object.
- Class Adapter: uses multiple inheritance to provide the interface.

The Façade pattern and the Adapter pattern may seem similar. What is the essential difference between the two?

Answer: - In both cases, there is a preexisting class or classes that have functionality I need. In both cases, I create an intermediary object with interfaces that my system wants to use and that has responsibility for mapping that to the preexisting class. Both Façade and Adapter are wrappers.

The Adapter is used when the client already has predefined interfaces that it expects to use and when I need to use polymorphism.

The Façade is used when I need a simpler interface to the existing object.

What is another example from real life that illustrates an Adapter?

Answer: - Another example could be a travel agent, seen as the common interface between a passenger making arrangements and an airline with its own systems. Each has competing systems, speaking different languages.

The Bridge Pattern

Define decouple and abstraction with respect to Bridge pattern

Answer: - Decouple means to separate or detach one thing from another. In our context, it means to have one thing behave independently from another (or at least to state explicitly what that relationship is).

Abstraction means to generalize or conceptualize: to step back from the more concrete to the more conceptual or abstract.

How is implementation defined in the context of the Bridge pattern?

Answer: - Implementation refers to the objects that the abstract class and its derivations use to put themselves into operation or into service.

What is the basic problem being solved by the Bridge pattern?

Answer: - The derivations of an abstract class must use multiple implementations without causing an explosion in the number of classes

What is Experts view of how to use patterns? Does he advocate starting with the solution first or the problem to be solved first? (For Bridge pattern)

Answer: - Experts says that a pattern describes a problem which occurs over and over again in the environment and then describes the core of the solution to that problem. This means that it is most important to understand the problem first and then tackle the solution. It is a mistake to try finding the solution first.

Define the "one rule, one place" strategy.

Answer: - "one rule, one place" says you should implement a rule in only one place. (p. 144). Note that this results in a greater number of smaller methods.

What are the consequences of the Bridge pattern?

Answer: - Decoupling of the implementations from the objects that use them increases extensibility. Client objects are freed from being aware of implementation issues.

Why can tight coupling lead to an explosion in the number of classes?

Answer: - Tight coupling means that as you get more variations in implementation, each class has to be responsible for its own implementation.

The Decorator Pattern

What does each Decorator object wrap?

Answer: - Decorators wrap their trailing objects. Each Decorator object wraps its new function around its trailing object.

What are two classic examples of decorators?

Answer: -

- Heading and footers
- Stream I/O

How does the Decorator pattern help to decompose the problem?

Answer: - The Decorator pattern helps to decompose the problem into two parts: How to implement the objects that give the new functionality; and how to organize the objects for each special case.

Why are the Bridge and Decorator patterns more correctly classified as structural rather than behavioral patterns?

Answer: - They both are tying together functionality, which is what structural patterns do. In the Bridge pattern, we typically start with abstractions and implementations and then bind them together with the bridge. In the Decorator pattern, we have an original functional class, and want to decorate it with additional functions

The Observer Pattern

What is the intent of the Observer pattern?

Answer: - The Gang of Four says that the intent of the Observer pattern is to "define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically."

The Template Method Pattern

The Template Method pattern makes the method call in a special way. What is that?

Answer: - The method itself is general. It makes the method call via a reference pointing to one of the derived classes to handle the special details.

What is the difference between the Strategy pattern (chapter 9) and the Template Method pattern?

Answer: - The Template Method pattern is applicable when there are different, but conceptually similar processes.

The Strategy pattern controls a family of algorithms. They do not have to be conceptually similar. You choose the algorithm to employ just in time.

How Do Experts Design?

Experts uses the term, "alive" to characterize good designs. What terms do I suggest using when it comes to software?

Answer: - "When you read ‘alive', think ‘robust' and ‘flexible' systems.

Good design requires keeping what in mind?

Answer: - Keeping the big picture in mind. Being able to consider the forest first and then the trees.

Experts suggests that the best approach to design involves "complexification." What does this mean?

Answer: - Complexification is the approach to design to starts by looking at the problem in its simplest terms and then adds additional features (distinctions), making the design more complex as we go because we are adding information

To Experts, what relationships does a pattern define?

Answer: - A pattern defines relationships between the entities in his problem. This is why define a pattern as a solution to a problem in a context. The entities describe the context in which the pattern exists.

What are Experts five steps to design?

Answer: -

- Identify patterns that are present in your problem.
- Start with context patterns (those that create context for other patterns)
- Work inward from the context
- Refine the design
- Implement.