Tutorial - 8
Design Patterns

E/15/119
E/15/202
E/15/208

1. **What is the primary objective of documenting design patterns?**

   It helps to speed up program changes as well as improve their quality

2. **Software community believes that design patterns facilitate reusable software systems. Do you agree with that? Explain your answer.**

   Yes. design patterns facilitate reusable software systems.
   Design models focus on individual objects and classes. So patterns facilitate reuse of software systems, even when other forms of reuse are infeasible (e.g., due to fundamental differences in operating system features.
   Reusing design patterns helps to prevent subtle issues that can cause major problems and improves code readability for coders and architects familiar with the patterns

3. **Assume that you have a collection of solutions to a collection of problems. Explain how you could differentiate patterns and non-patterns in these solutions.**

   If the solutions are productive and efficient and are developed by Software Engineers over the years of practice and solving problems those solutions can be identified as patterns.
   If there are known solutions which are actually bad or defective to certain kinds of problems, then they can be identified as non-patterns.

4. **State and discuss pattern organization techniques.**
   Design patterns vary in their granularity and level of abstraction. As there are many design patterns, we can organize or classify patterns to learn them faster. Here we classify the patterns along two dimensions: one is purpose which reflects what a pattern does and the second one is scope which specifies whether the pattern applies to classes or objects.

Purpose / Scope		Purpose		
		Creational (5)	Structural (7)	Behavioral (11)
Scope	Class	Factory Method	Adapter	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

The creational patterns focus on the process of object creation. The structural patterns concern is, the structure of classes and objects and the behavioral patterns focus on how the classes and objects collaborate with each other to carry out their responsibilities.

Class patterns deal with the relationships between classes which is through inheritance. So, these class relationships are static at compile time. Object patterns deal with object relationships which can be changed at run-time and these are dynamic. Almost all patterns use inheritance to some degree.

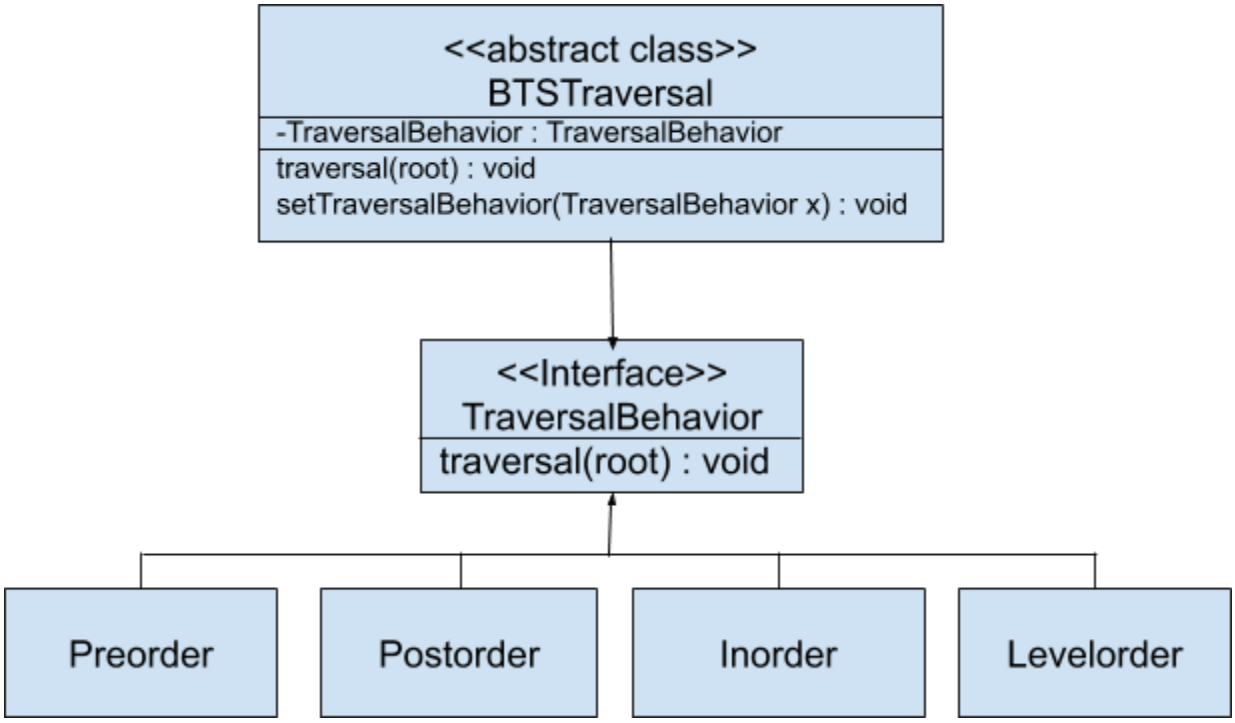
5. **What does it mean by pattern selection problem? Do you think that all organization techniques state in question 4 assist to solve this problem? Explain your answer.**
   Pattern selection problem means choosing the most suitable design pattern for a specific problem. It is a very hard task. Because there are a large number of design patterns that we can use. 23 design patterns.
   But fortunately all those 23 design patterns are categorised as shown in question 4. So when choosing a design pattern for a problem, first we have to clearly understand our problem. That means we have to specifically identify what category that our problem belongs to, from the given categories in the above table. By correctly identifying that we can reduce the number of design patterns that we have to observe to get the suitable design pattern for our problem. So that it will save our time.

6. **A binary search tree can be traversed using four methods: pre-order, in-order, post-order and level-order traversal. It is required to design an application so that users can traverse a given binary search tree using any of the four methods. explain how you could apply Strategy pattern and Factory pattern to design and implement this application. You may provide diagrams wherever appropriate.**

In computer programming, the **strategy pattern** (also known as the **policy pattern**) is a software design pattern that enables an algorithm's behavior to be selected at runtime. The strategy pattern

- defines a family of algorithms,
- encapsulates each algorithm, and
- makes the algorithms interchangeable within that family.

Factory Method **Pattern**. A **Factory Pattern** or **Factory** Method **Pattern** says that just define an interface or abstract class for creating an object but let the subclasses decide which class to instantiate. In other words, subclasses are responsible for creating the instance of the class.



As in the above diagram inside the abstract class there will be an encapsulated interface called TraversalBehavior. Inside that interface there will be a method called traversal(root) which can be implemented in different ways as pre-order, post-order, in-order or level order.

The abstract class will take a traversal behavior object as the argument. Or the setTraversalBehavior(TraversalBehavior x) method will allow the user to set the traversal method dynamically. So that user can use the algorithms interchangeably.

7. Compare and contrast creational, structural and behavioral patterns.

Various design patterns can be grouped into three categories: creational, structural and behavioral.

1. Creational patterns
   These design patterns provide ways to create objects while hiding the creation logic, instead of instantiating objects directly using the new operator. This gives the program more flexibility in deciding which objects need to be created for a given use case. And this patterns use object oriented programming efficiently during the instantiation, object-creation patterns use delegation successfully which fulfil the work. These patterns help in improving the performance to a great extent.
2. Structural patterns
   These patterns describe how to organize design components to meet certain requirements. These design patterns deal with class and object composition. The concept of inheritance is used to compose interfaces and define ways to compose objects to obtain new functionality. These are the outline designs which facilitate the structure by recognizing a straightforward approach to acknowledge relationships between elements. It is all about class and object composition. These patterns simplifies the design by finding a best method to realize relationships between entities. They use inheritance to compose interfaces.
3. Behavioral patterns
   These design patterns are specifically concerned with communication between objects.This design patterns deals with Class's objects communication or their interaction. These patterns main target of using object oriented programming is achieved by giving importance to the interaction between the objects. These designing patterns are concerned with interaction between the objects.

8. Explain how could you apply the proxy and whole-part patterns together.

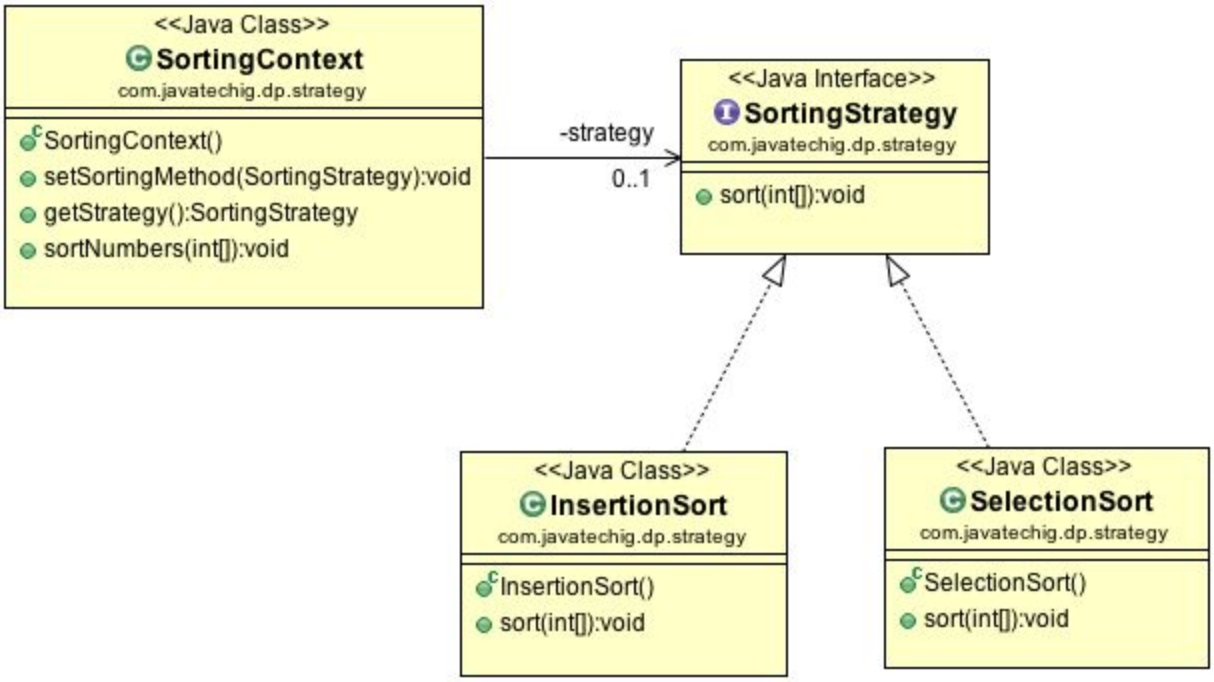
We can solve remote data access problems by using proxies with the properties of both remote and cache proxy variants. Implementing such a mix-mode proxy can be accomplished using the whole-part pattern.

9. There are two design patterns that can be applied in designing variants of an algorithm that can be interchanged independently. What are those patterns? Explain how you could use these patterns to design elementary sorting algorithms.

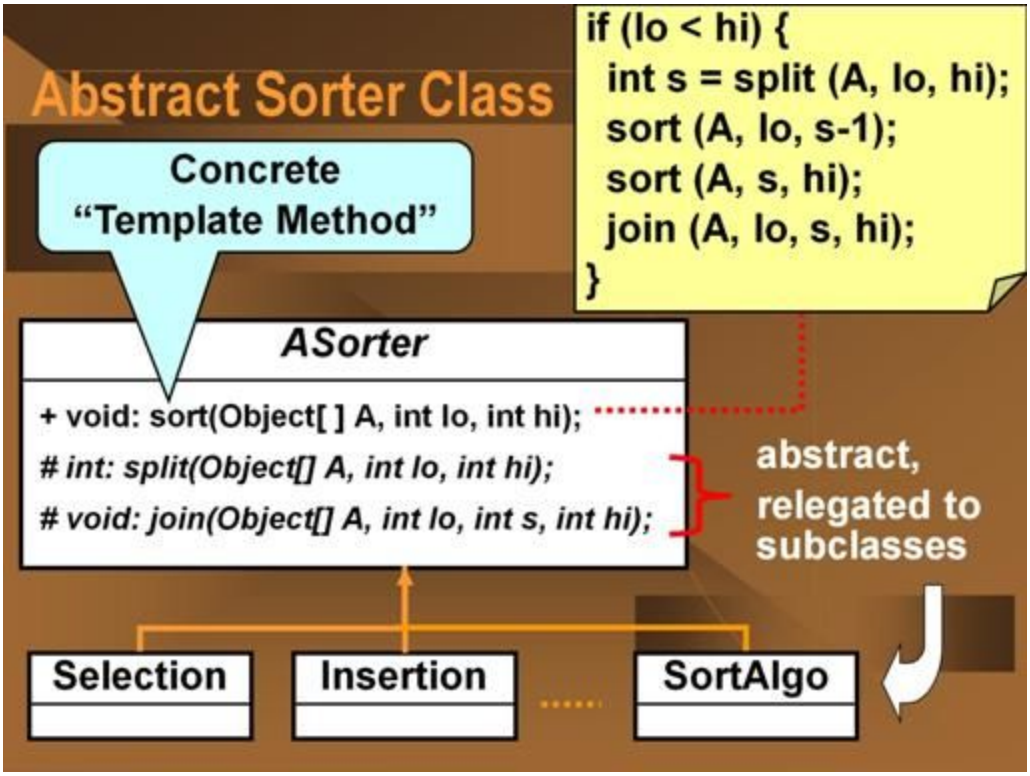
Strategy design pattern and template method design pattern.

1. Strategy design pattern

Strategy is a behavioral design pattern that lets you define a family of algorithms, put each of them into a separate class, and make their objects interchangeable. Instead of hiding the algorithm used to sort lists, lists could allow clients to inspect and modify the algorithm by providing getter and setter methods for the sorting strategy.
Use of strategy pattern in elementary sorting algorithms.



2. Template method design pattern
   Template Method is a behavioral design pattern that defines the skeleton of an algorithm in the superclass but lets subclasses override specific steps of the algorithm without changing its structure.



10. State and discuss the two Adapter patterns. You should explain the advantages and disadvantage of each pattern when applying to solve a candidate problem.

Two adapter patterns are object adapter and class adapter.

1. Object adapter

   This implementation uses the object composition principle: the adapter implements the interface of one object and wraps the other one. It can be implemented in all popular programming languages. In here adapter holds an instance of adaptee. Objects Adapters are the classical example of the adapter pattern. It uses composition, the adaptee delegates the calls to adaptee (opossed to class adapters which extends the adaptee). This behaviour gives us a few advantages over the class adapters(however the class adapters can be implemented in languages allowing multiple inheritance). The main advantage is that the Adapter adapts not only the adpatee but all its subclasses. All it's subclasses with one "small" restriction: all the subclasses which don't add new methods, because the used mechanism is delegation. So for any new method the Adapter must be changed or extended to expose the new methods as well. The main disadvantage is that it requires to write all the code for delegating all the necessary requests to the adaptee. Therefore the object adapter is based on delegation.

2. Class adapter
   This implementation uses inheritance: the adapter inherits interfaces from both objects at the same time. Note that this approach can only be implemented in programming languages that support multiple inheritance, such as C++. Class adapters can be implemented in languages supporting multiple inheritance(Java, C# or PHP does not support multiple inheritance). Thus, such adapters can not be easy implemented in Java, C# or VB.NET. Class adapter uses inheritance instead of composition. It means that instead of delegating the calls to the adaptee, it subclasses it. In conclusion it must subclass both the Target and the adaptee. There are advantages and disadvantages:It adapts the specific Adaptee class. The class it extends. If that one is subclassed it can not be adapted by the existing adapter, It doesn't require all the code required for delegation, which must be written for an Object Adapter. If the Target is represented by an interface instead of a class then we can talk about "class" adapters, because we can implement as many interfaces as we want.