Design Patterns in OOPS

Lesson Plan for Design Pattern in OOP, Software Development

Prepared by Piyush Upadhyay

# OVERVIEW & PURPOSE

There are some general problems faced by software developers while developing softwares. These problems can be solved using some best practices proposed by the experienced developers in OOPS.

This documentation contains a brief description about the basics of Design Pattern, their types and their uses in real world software building. Alongwith this description we would be implementing some design patterns in the real-life application like hyperlocal delivery application.

# EDUCATION STANDARDS

1. The audience is supposed to have some hands-on experience over software development using OOPS
2. Since we would be using JAVA Programming for the development, it is assumed that the reader is aware of JAVA Programming. In the case of you being not aware, please watch our previous uploaded videos on ‘Hands-on JAVA Programming’ course.

# OBJECTIVES

1. To provide best software design solutions proposed by the ‘gang of 4’ to the experienced developers.
2. To spread the best development and design practices to the budding software developers, very fast and easily.

# MATERIALS NEEDED

1. JDK on your system, JAVA\_HOME and PATH attributes set in your environment variables.
2. IDE for development (we would be using Eclipse).

# ACTIVITY

What are Design Patterns? How they came into existence?

Design Patterns are some best practices proposed by four experienced software developers in 1994. These practices are language independent as they are the solutions to some general problems while developing on Object-Oriented Programming Languages. These are not the solutions to the development problems but these are some structure to solve a problem. Learning design patterns is a good way to build the programs. Informally, we can say **OOP is like cooking** and **Design Patterns are the recipes.**

Like a Civil Engineer or an architect needs a blueprint of the plan to build the house or renovate, similarly a Software Developer needs a blueprint of the classes, interaction between them and the various actions they could perform after the software is developed.

‘Each pattern describes a problem which occurs over and over again’, these repeatedly occurring challenges were described and some practices were proposed as solutions by Eric Gamma, Richard Helm, Ralph Johnson and John Vlissides in the book **Design Patterns - Elements of Reusable Object-Oriented Software.**

Types of Design Patterns?

There are 23 Design Patterns proposed in the book **Design Patterns - Elements of Reusable Object-Oriented Software** which are divided into mainly 3 categories:

1. **Creational Design Patterns:** This Design Pattern focus on the way objects are created and are mapped to different tasks while hiding the creation logic. This Design Pattern depends more on composition (how objects are created, composed and represented) than the class inheritance. Creating objects with specific behaviors require more than simply instantiating class using **new** keyword. So the **class creational pattern** depends on inheritance while **object creational pattern** depends on instantiation to objects.
2. **Structural Design Patterns:** As the name suggests, this pattern deals with how classes and objects are composed to form larger structures. **Structural class patterns** use inheritance over the composition. Here, it uses inheritance property to compose interfaces. While the **Structural object patterns** defines the new functionalities to compose objects.
3. **Behavioral Design Patterns:** This Design Pattern focuses over the communication between objects. It is concerned over algorithms and the responsibilities assigned to the objects and between the objects. Putting in simple words, we can say it is the Design Pattern which deals with the flow of control *from* and *to* for an object. Here, **Behavioral class patterns** uses inheritance property to distribute the behaviors of objects between classes while **Behavioral object patterns** use object composition.

Real world usage of Design Patterns?

Apart from this theoretical description, let’s dive into some real world example where we can actually take help of all these 3 main design patterns and some required design patterns from 23 proposed design patterns.

**Example:** Nowadays, the e-commerce industry is regularly in news. So let’s take an example of a small e-commerce firm which works for a small geographical area. The first thing comes up is , what are the actions and features can be supported by the industry? What cases we need to consider? Which Design Pattern shall be used for the particular case?

So let’s first jot down the cases which are must and need to be implemented for an ideal e-commerce application. Some of the use cases possible are mentioned as following :

1. User can search different stores nearby
2. User can select a store
3. User can see items in a store
4. User can add an item from a store
5. User can order the items
6. User can cancel the order
7. User can pay for the order

For all these 7 cases we need different Design Patterns according to the requirements and actions. The table below shows the design pattern that can be used against the mentioned Use Cases. So now we will discuss what these specific design pattern does. Also there will be a question in your mind like ‘Can there be multiple design pattern or How will we choose a particular design for the use case’? So we will try to find the answer one by one to each.

|  |  |
| --- | --- |
| **Event / Use-Case** | **Design Pattern** |
| User can search different stores nearby | Interpreter Design Pattern |
| User can select a store | Observer/ Select from option |
| User can see items in a store | Iterator Pattern |
| User can add an item from a store | Builder Design Pattern |
| User can order the items | Command Design Pattern |
| User can cancel the order | Command Design Pattern |
| User can pay for the order | Strategy Design Pattern |

What these design patterns do in specific?

1. **Interpreter Design Pattern:** Just like the name , it does the same. So for an e-commerce application if we make it console based then we need to give inputs like “Find me a laptop near some Location” or “lenovo laptop in Guwahati”. So here we have a string and we have to figure out the brand, electronic device and location of the store such that prepositions or conjunctions are avoided. So for that purpose we need an **interpreter design pattern**. That is something in really simple words.

Putting it in real definition we can say that, it helps us to evaluate a language grammar or an expression.

**Where can we use in our example?** When the user will send us the expression of device and location, we’ll need it. So the initial phase of running the application we will require this design pattern.

1. **Observer Design Pattern:** When on taking a decision the state changes we shall use this design pattern.

Putting it in real definition we can say that, when there is one to many dependency/relationship in objects, **observer design pattern** is used.

**Where can we use in our example?** So now we have location, device and brand and we have to select the store from the list of store which changes the state of all the stores (one being used and others being rejected). We will use it in selecting a store from a list of stores.

1. **Iterator Design Pattern:** When there are a lot of product options and we need to iterate through all of them we use this design pattern.

So the real definition goes like this, it is useful when we need to iterate through or access elements of a collection object in sequential manner as they were added.

**Where can we use in our example?** Since we have stored laptops and cameras in ArrayList collection while the Phone device in HashTable collection, now we need to show the available devices to the user on a particular store. So we need to iterate through a particular collection and that is why we used this design pattern.

1. **Builder Design Pattern:** Whenever we are confused about the attributes of a device and can not fill all the details, **builder design pattern** comes to the rescue. Putting it simply, like there is a popular meme where they say girls choose their phone on the basis of color(No offence!) while boys go for the specifications, so at the time of **putting filters** we require this design pattern. So girls can go for selecting only color while boys can set **OS, RAM, Processor, Memory and others**. So the world wide accepted definition goes like this, when we need to create a complex objects using some smaller and simpler objects in a step by step approach we use this pattern.

**Where can we use in our example?** So we have selected the shop and seen all the items and now we want to add an item of particular specification, we’ll build that portion using this design pattern. So we build break a laptop object into smaller objects like Operating System, Display Type, RAM, Hard Disk, Battery, Processor and others. So at the time of adding the user can simply set a particular object if he/she is not focussed about other objects. **For example, ‘**A**’** wants a 6 GB RAM, i7 processor and 8000mAh battery then he would set only these features and will skip other features.

1. **Command Design Pattern:** So now we need to order/cancel the added item. For that we need some concrete command to work and for that purpose we use **command design pattern** which has an invoker, concrete commands and receiver which notifies the seller about the product being bought or cancelled. Obviously we do have a client in this.

So the definition goes like this, a data driven design pattern which wraps a request under object and sends it to an invoker which forwards it to the receiver.

**Where can we use in our example?** As the added item can be ordered or cancelled there must be an object which takes the placing/cancelling request to the seller and notifies him about the event. So we use this in our application when the client/user cancels/places the order to notify the seller about the product being bought and the receiver generates the bill.

1. **Strategy Design Pattern:** So far we have added the product and ordered it but the main event has to occur now which is payment. For payment we use **strategy design pattern** as there are multiple strategies of paying like Net Banking, Card Payment, VISA Payment, Mobile Transfer and others. The payment method can be changed every time the user buys a product so this design pattern helps in creating a context object that uses different algorithm for a different transaction chosen every time.

So the definition goes like this, a class behavior can be changed at run time and for that **strategy design pattern** represents various strategies. The strategy algorithm changes for the context.

**Where can we use in our example?** As mentioned above when we are providing different payment options we use **strategy design pattern**. So in our application for the payment section we use **strategy design pattern** where we create a context object which changes the algorithm (Card Holder’s name,expiry date, Card number for Card Payment strategy while Mobile Number, Mobile Operator for Mobile Transfer strategy) for different transaction methods.

References

1. Gamma, Erich, et al. "Design patterns: elements of." *1994* (1994).
2. <https://www.tutorialspoint.com/design_pattern/index.htm>
3. Derek Banas Videos on YouTube for Interpreter, Iterator Design Patterns.
4. Telusko Learning Videos on YouTube for Builder Design Pattern.
5. A Beginner’s Guide to Design Pattern by CodeTuts.
6. Design Pattern concepts by oodesign.