



UNIVERSITY OF INFORMATION
TECHNOLOGY AND SCIENCES (UITs)
DEPARTMENT OF INFORMATION TECHNOLOGY

LAB REPORT : 6

IT-324 : DESIGN PATTERN LAB

Adapter Design Pattern

Submitted To:

Sifat Nawrin Nova
Lecturer,
Department of IT, UITs
Email:sifat.nawrin@uits.edu.bd

Submitted By:

Name:Nazmul Zaman
Student ID:2014755055
Department of IT, UITs

Contents

1	Abstraction	2
2	Theory	2
2.1	Diagram	2
3	Objective	3
4	What is the goal of the factory method pattern?	3
5	Advantage and disadvantage	3
6	Usage of Adapter Design Pattern	4
7	Working procedure	4
8	Source Code	5
8.1	(MideaPlayer Class)	5
8.2	(AdvanceMidea Class)	5
8.3	(MideaAdapter Class)	6
8.4	(Ndriver Class)	6
8.5	(Client Class)	6
8.6	Output	7
9	Conclusion	8
10	References	8

1 Abstraction

In this Lab we are going to implement Adapter design pattern. In software engineering, the adapter pattern is a software design pattern (also known as wrapper, an alternative naming shared with the decorator pattern) that allows the interface of an existing class to be used as another interface.

Adapter pattern is a software design pattern (also known as wrapper, an alternative naming shared with the decorator pattern) that allows the interface of an existing class to be used as another interface.

2 Theory

Adapter is a structural design pattern that allows objects with incompatible interfaces to collaborate.

Adapter design pattern in java is a structural design pattern. It provides solution for helping incompatible things to communicate with each other. It works as an inter-mediator who takes output from one client and gives it to other after converting in the expected format. Adapter pattern works as a bridge between two incompatible interfaces. This type of design pattern comes under structural pattern as this pattern combines the capability of two independent interfaces.

2.1 Diagram

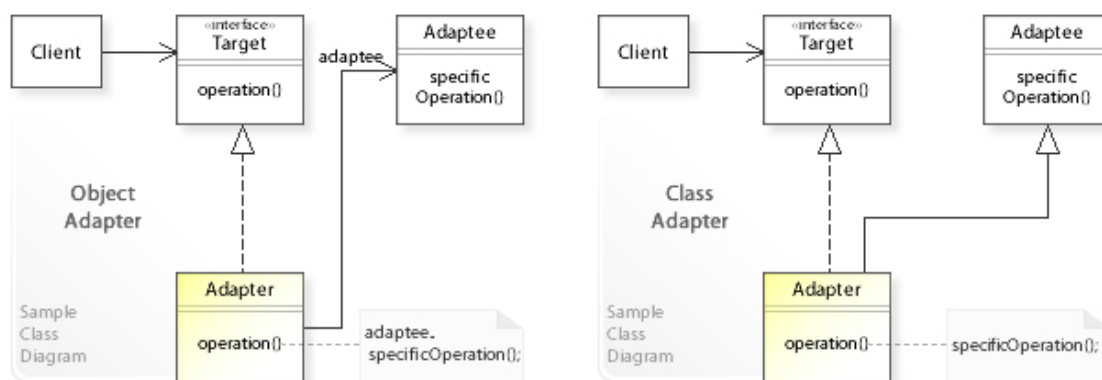


Figure 1: Output

3 Objective

Consider we want to implement a Client side web browser through Ydriver browse . Let's try to implement this with the help of adapter design pattern method. We will also learn advantages and disadvantages of adapter pattern and uses of this pattern.

4 What is the goal of the factory method pattern?

The adapter design pattern describes how to solve such problems: Define a separate adapter class that converts the (incompatible) interface of a class (adaptee) into another interface (target) clients require.

5 Advantage and disadvantage

Advantage of Adapter Design Pattern[2]:

1. allows two or more previously incompatible objects to interact.
2. It allows reusability of existing functionality.

Disadvantages of Adapter Design Pattern

Disadvantages of Adapter Design Pattern

1. All requests are forwarded, so there is a slight increase in the overhead.
2. Sometimes many adaptations are required along an adapter chain to reach the type which is required.

6 Usage of Adapter Design Pattern

1. When an object needs to utilize an existing class with an incompatible interface.
2. When you want to create a reusable class that cooperates with classes which don't have compatible interfaces.
3. When you want to create a reusable class that cooperates with classes which don't have compatible interfaces.

7 Working procedure

Step-1: Create interfaces for MideaPlayer.

Step-2: Create MideaAdapter class to communicate the MideaPlayer interface and Ndriver class.

Step-3: Use the Client to browse different types of topic.

Step-4: Verify the output.

8 Source Code

8.1 (MideaPlayer Class)

```
1 package adapter_pattern;
2
3     public interface MideaPlayer {
4         void browse ( String str ) ;
5
6     }
```

?

8.2 (AdvanceMidea Class)

```
1
2 package adapter_pattern;
3
4 /**
5  *
6  * @author A
7  */
8
9
10    public class AdvanceMidea {
11        private MideaPlayer w ;
12        public MideaPlayer getMidea ()
13        {
14            return w ;
15        }
16        public void setMidea ( MideaPlayer w )
17        {
18            this . w = w ;
19        }
20        public void browseAdvanceMidea ( String srt )
21
22        {
23            w . browse ( srt ) ;
24        }
25    }
```

?

8.3 (MideaAdapter Class)

```
1
2 package adapter_pattern;
3
4 import Com.New.Ndriver ;
5
6 public class MideaAdapter implements MideaPlayer {
7     Ndriver ww = new Ndriver () ;
8     public void browse ( String str )
9     {
10         ww . find ( str ) ;
11     }
12
13 }
```

?

8.4 (Ndriver Class)

```
1
2 package Com.New;
3
4 /**
5  *
6  * @author A
7  */
8 public class Ndriver {
9
10     public void find ( String srt )
11     {
12         System.out.println( srt ) ;
13     }
14
15 }
```

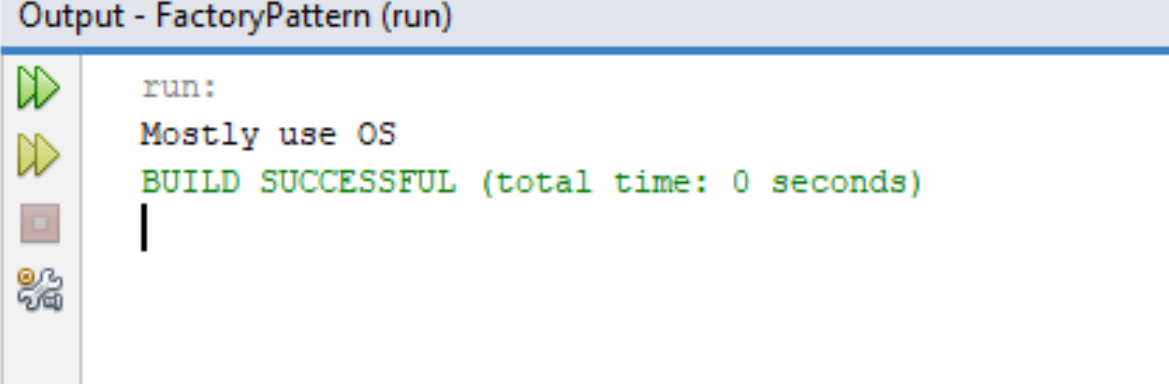
?

8.5 (Client Class)

```
1
2 package adapter_pattern;
3
4
5 public class Client {
6
7     public static void main(String [] args) {
8         MideaAdapter p = new MideaAdapter();
9         AdvanceMidea c = new AdvanceMidea();
10        c.setMidea(p);
11        c.browseAdvanceMidea ("Adapter Design pattern" ) ;
12    }
13
14 }
```

?

8.6 Output



```
run:
Mostly use OS
BUILD SUCCESSFUL (total time: 0 seconds)
|
```

Figure 2: Output

9 Conclusion

In this lab I have learnt what Adapter Design Pattern is and why and where can be used with real World example. The Adapter Pattern is an often-used pattern in object-oriented programming languages. Similar to adapters in the physical world, you implement a class that bridges the gap between an expected interface and an existing class. That enables you to reuse an existing class that doesn't implement a required interface and to use the functionality of multiple classes, that would otherwise be incompatible.

10 References

- 1.https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm
- 2.<https://www.geeksforgeeks.org/singleton-design-pattern-introduction>
- 3.<https://www.javatpoint.com/singleton-design-pattern-in-java>