Adapter Pattern Tutorial

Written Date : October 7, 2009

This tutorial is aimed to guide the definition and application of Gang of Four (GoF) adapter design pattern. By reading this tutorial, you will know how to develop a model for the adapter pattern, and how to apply it in practice.

What is Adapter Design Pattern?

The Adapter Design Pattern is a structural design pattern that is used to make two incompatible interfaces work together. It's often used when we have an existing system or class that we want to reuse, but its interface is incompatible with the rest of the system. Rather than modifying the existing system, which may not be possible or desirable, we can create an adapter that acts as a bridge between the existing system and the rest of the system.

The adapter has two main components: an interface that matches the interface of the system it's being adapted to, and an implementation that translates the calls from the adapted system to the existing system. This translation allows the two systems to communicate and work together seamlessly.

The Adapter Design Pattern can be implemented in two ways: class adapters and object adapters. Class adapters use multiple inheritance to adapt one interface to another, while object adapters use composition to achieve the same result. Both approaches have their advantages and disadvantages, and the choice between them will depend on the specific requirements of the system.

Overall, the Adapter Design Pattern is a useful tool for integrating existing systems and classes into new systems without having to modify the existing systems. It's a powerful technique that can make it easier to reuse code and build more robust software systems.

Modeling Design Pattern with Class Diagram

1. Create a new project *Design Patterns*.
2. Create a class diagram *Adapter*.
3. Select **Class** from diagram toolbar. Click on the diagram to create a class. Name it as *Client*.

4. Move the mouse cursor over the *Client* class, and drag out **Association** > **Class** to create an associated class *Target*.

5. Right-click on *Target*, and select **Model Element Properties** > **Abstract** to set it as abstract.

6. Right-click on the *Target* class, and select **Add** > **Operation** from the popup menu.

7. Name the operation *Request()*.
8. Right-click on *Request*, and select **Model Element Properties** > **Abstract** to set it as abstract.

9. Move the mouse cursor over the *Target* class, and drag out **Generalization** > **Class** to create a subclass *Adapter*.

10. Adapter will inherits the operations from Target. Right-click on *Adapter* and select **Related Elements** > **Realize all Interfaces** from the popup menu.

11. Move the mouse cursor over the *Adapter* class, and drag out **Association** > **Class** to create an associated class *Adaptee*.

12. In practice, there may be multiple requests. To represent this, stereotype the class *Target* as **PTN Members Creatable**. Right-click on *Target* and select **Stereotypes** > **Stereotypes...** from the popup menu.

13. In the **Stereotype** tab of the **Class Specification** dialog box, select **PTN Members Creatable** and click > to assign it to *Target* class. Click **OK** to confirm.

    Up to now, the diagram should look like this:

Defining Pattern

1. Select all classes on the class diagram.
2. Right-click on the selection and select **Define Design Pattern...** from the popup menu.
3. In the **Define Design Pattern** dialog box, specify the pattern name *Adapter*. Keep the file name as is. Click **OK** to proceed.

Applying Design Pattern on Class Diagram

In this section, we are going to apply the adapter pattern to wrap a legacy Shape class.

1. Create a new project *Diagram Editor*.
2. Create a class diagram *Domain Model*.
3. Right-click on the class diagram and select **Utilities** > **Apply Design Pattern...** from the popup menu.
4. In the **Design Pattern** dialog box, select *Adapter* from the list of patterns.
5. Click on *Target* in the overview.
6. Rename *Target* to *Shape*, and operation *Request* to *getColor* at the bottom pane.

7. Besides the operation *getColor*, we also need two more operations for *getPosition* and *draw*. Keep *Target* selected, click on the + button at the bottom pane, and select **New Operation...** from the popup menu.

8. In the **Operation Specification** dialog box, name the operation *getPosition*. Check **Abstract** at the bottom of dialog box.

9. Repeat steps 7 and 8 to create operation *draw*.

10. Select *Adapter* in overview, and rename it as *NewShape* at the bottom pane. Rename also the operation *Request* to *getColor*. Note that if the option **Auto Rename** is on, rename of operation is not needed as this will be done automatically.

11. Select *Adaptee* in overview, and rename it as *LegacyShape* at the bottom pane. Click **OK** to apply the pattern to diagram.

12. We need to create the specific requests in *LegacyShape*. Select the operations in *NewShape*.

13. Press on the *Ctrl* key, and drag to *LegacyShape* to copy them.
    This is the result:

Resources
1. Adapter.pat
2. Design Patterns.vpp

Related Links
- Full set of UML tools and UML diagrams

Visual Paradigm home page
(https://www.visual-paradigm.com/)

Visual Paradigm tutorials
(https://www.visual-paradigm.com/tutorials/)