Adapter Pattern Tutorial

Written Date : October 7, 2009

What is the Adapter Design Pattern?

The Adapter Design Pattern is a structural design pattern employed to enable collaboration between two otherwise incompatible interfaces. It is frequently utilized when an existing system or class needs to be reused, but its interface does not align with the rest of the system. Instead of altering the existing system, which may not be feasible or desirable, an adapter can be created to serve as a bridge between the existing component and the new system.

The adapter comprises two primary elements: an interface that matches the system it is adapting to, and an implementation that translates calls from the adapted system to the existing system. This translation facilitates seamless communication and cooperation between the two systems.

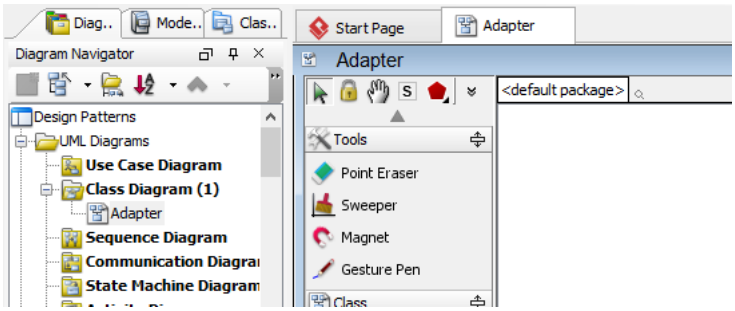
The Adapter Design Pattern can be implemented using two main approaches: class adapters and object adapters. Class adapters leverage multiple inheritance to adapt one interface to another, whereas object adapters achieve the same outcome through composition. Both methods have their respective advantages and disadvantages, and the choice between them depends on the specific requirements of the system.

Overall, the Adapter Design Pattern is an invaluable tool for integrating existing systems and classes into new software systems without requiring modifications to the original components. It is a powerful technique that promotes code reuse and helps in building more robust software architectures.

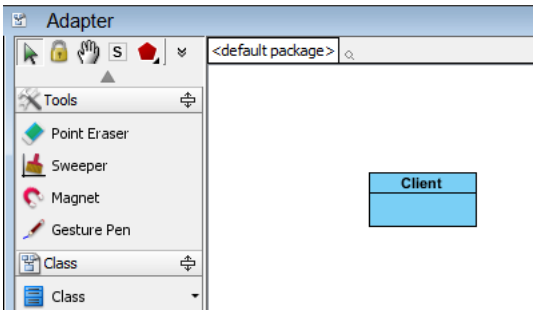
Modeling the Design Pattern with a Class Diagram

1. Create a new project named *Design Patterns*.

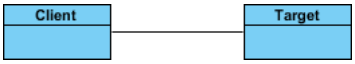
2. Create a class diagram named *Adapter*.



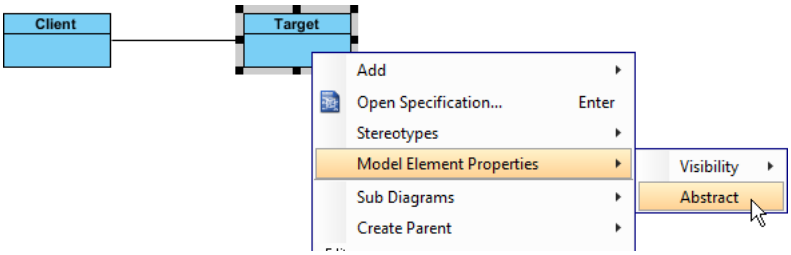
3. Select **Class** from the diagram toolbar. Click on the diagram to create a class and name it *Client*.



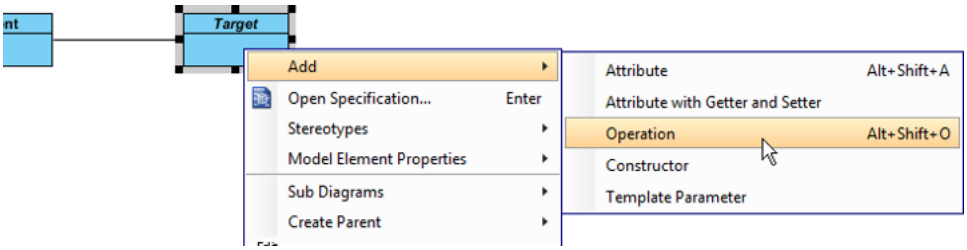
4. Move the mouse cursor over the *Client* class, and drag out **Association** > **Class** to create an associated class named *Target*.



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

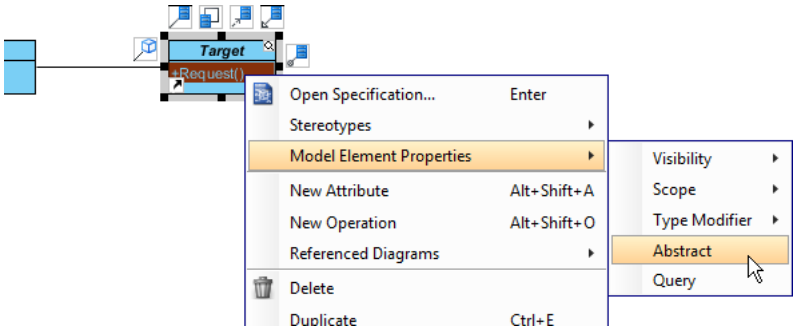


6. Right-click on the *Target* class, and select **Add** > **Operation** from the popup menu.

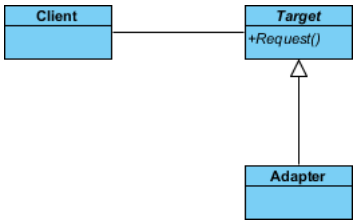


7. Name the operation *Request()*.

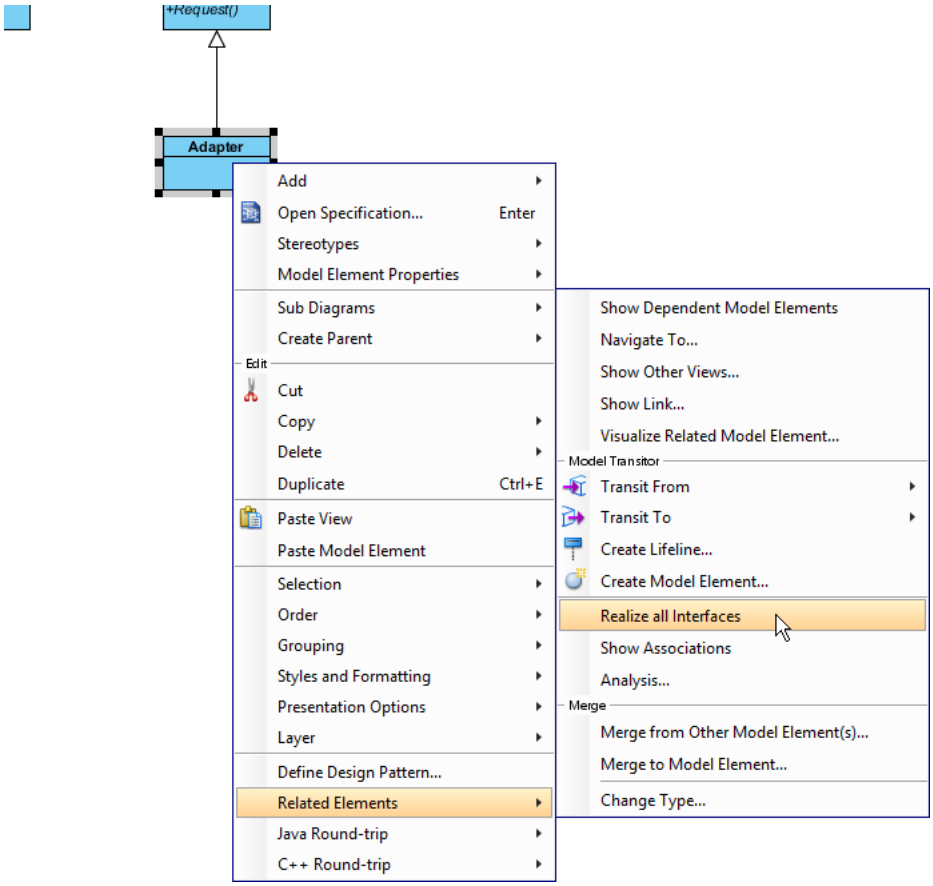
8. Right-click on *Request*, and select **Model Element Properties** > **Abstract** to set it as abstract.



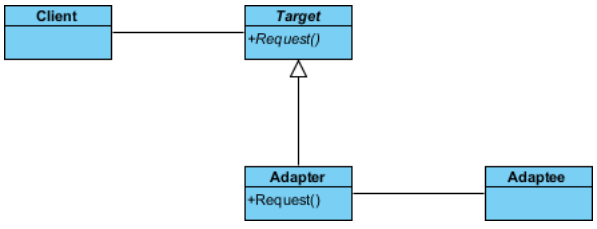
9. Move the mouse cursor over the *Target* class, and drag out **Generalization** > **Class** to create a subclass named *Adapter*.



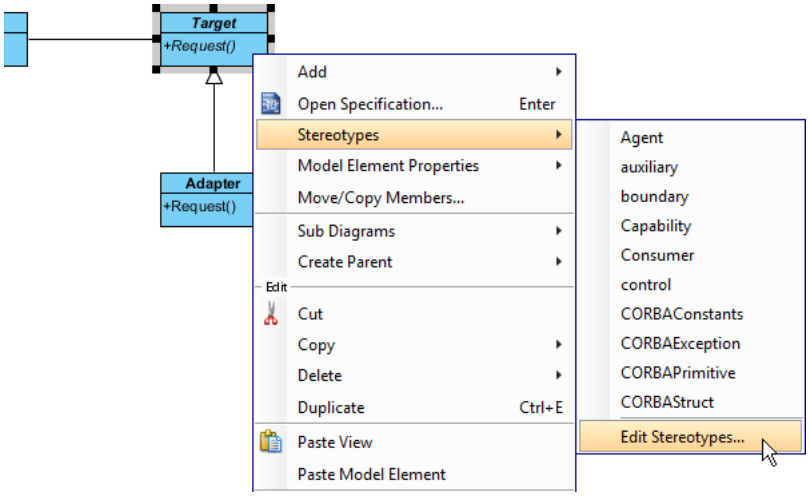
10. The *Adapter* will inherit operations from *Target*. Right-click on *Adapter* and select **Related Elements** > **Realize all Interfaces** from the popup menu.



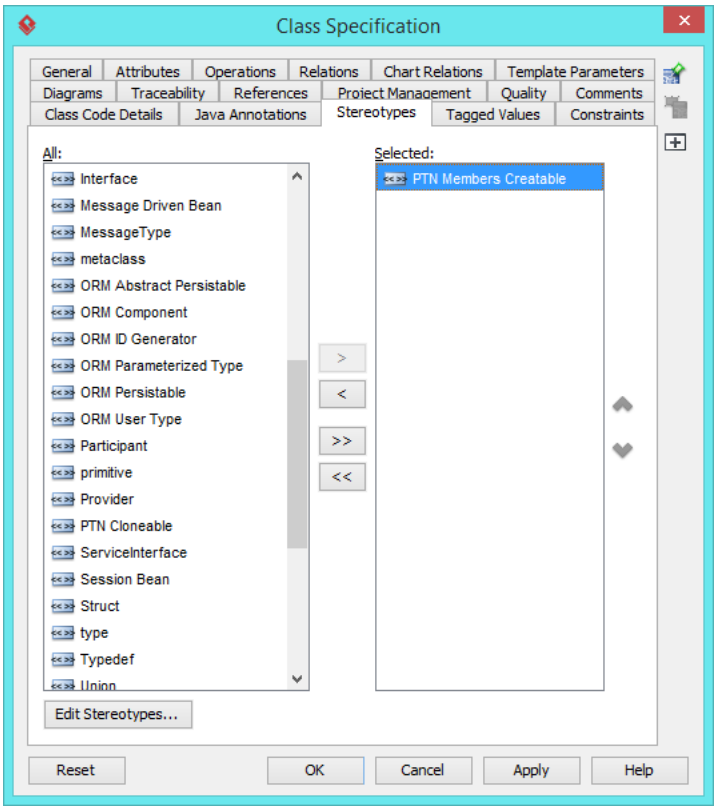
11. Move the mouse cursor over the *Adapter* class, and drag out **Association** > **Class** to create an associated class named *Adaptee*.



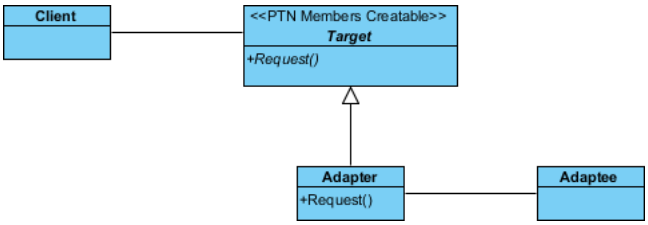
12. In practice, there may be multiple requests. To represent this, stereotype the class *Target* as **PTN Members Creatable**. Right-click on *Target* and select **Stereotypes** > **Stereotypes...** from the popup menu.



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

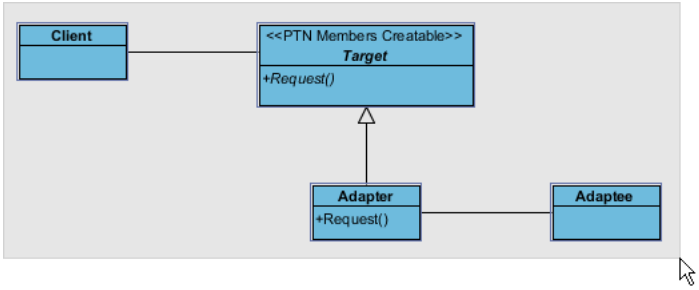


At this point, the diagram should look like this:

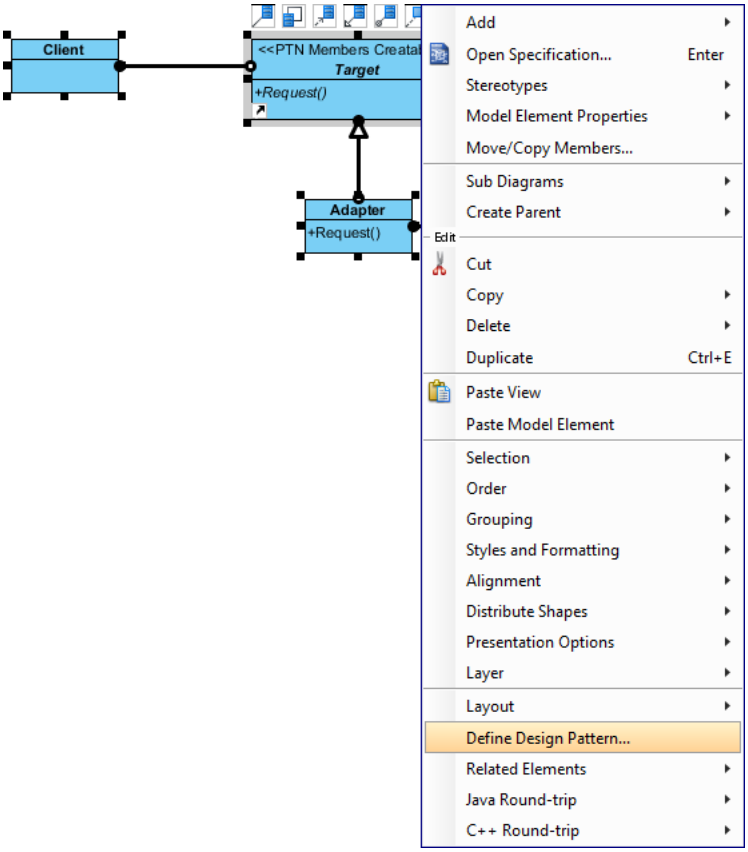


Defining the Pattern

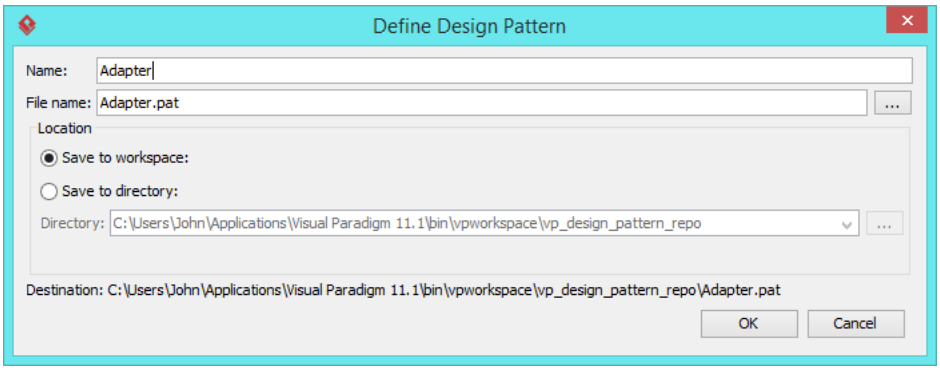
1. Select all classes on the class diagram.



2. Right-click on the selection and select **Define Design Pattern...** from the popup menu.



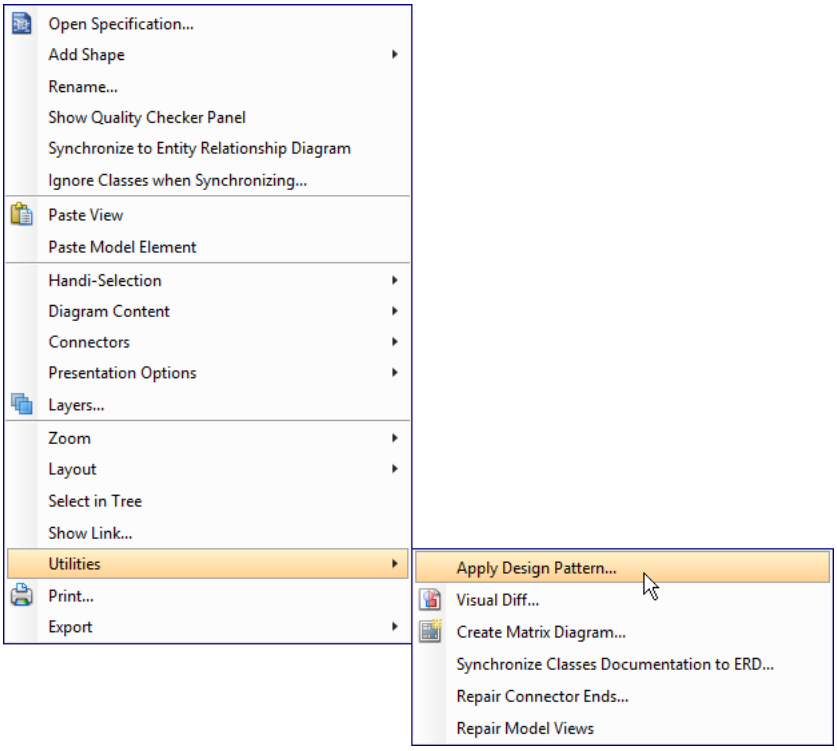
3. In the **Define Design Pattern** dialog box, specify the pattern name as *Adapter*. Keep the file name as is. Click **OK** to proceed.



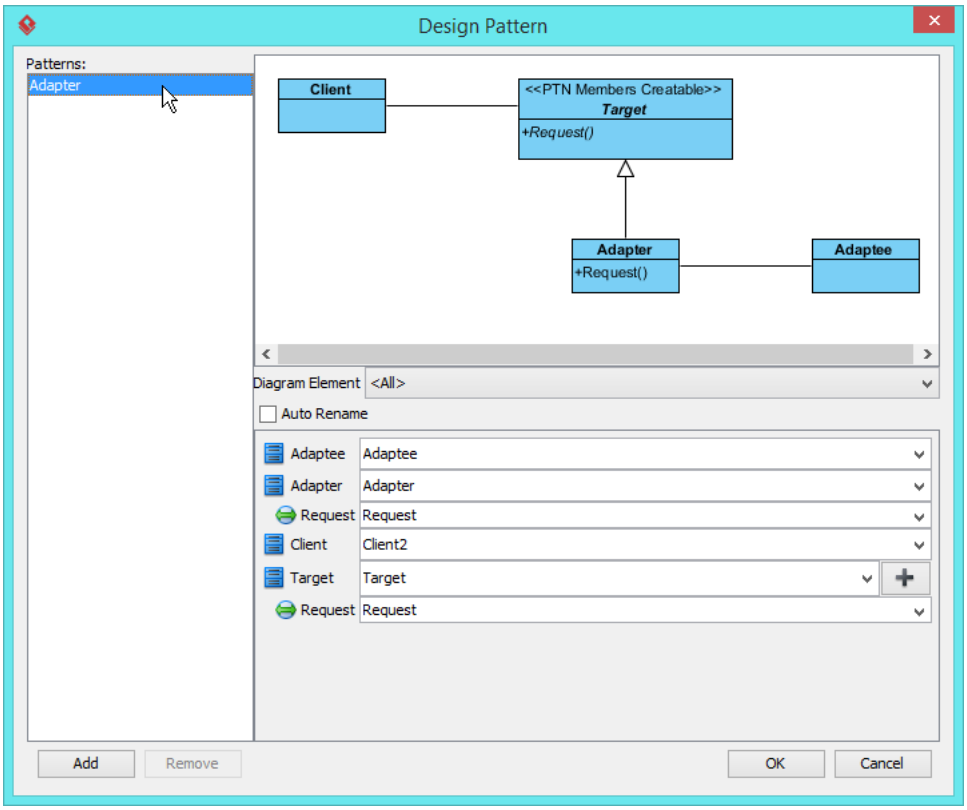
Applying the Design Pattern to a Class Diagram

In this section, we will apply the Adapter pattern to wrap a legacy *Shape* class.

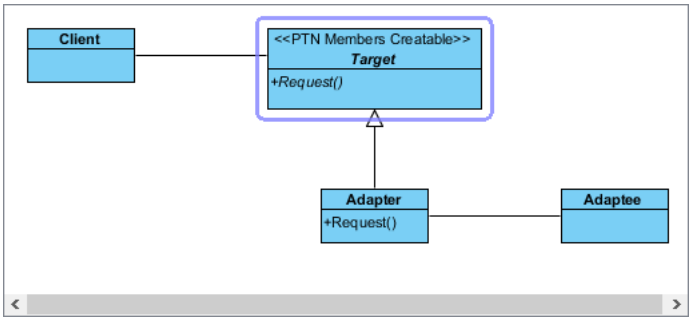
1. Create a new project named *Diagram Editor*.
2. Create a class diagram named *Domain Model*.
3. Right-click on the class diagram and select **Utilities** > **Apply Design Pattern...** from the popup menu.



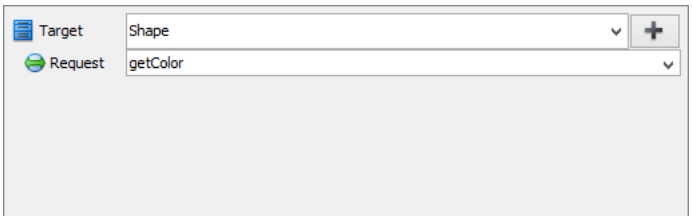
4. In the **Design Pattern** dialog box, select *Adapter* from the list of patterns.



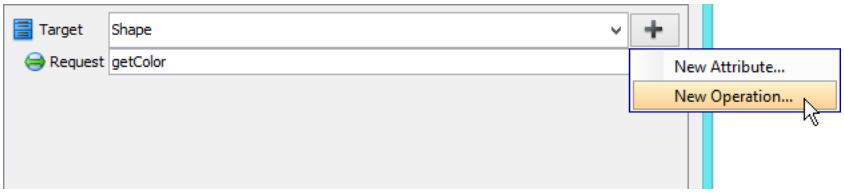
5. Click on *Target* in the overview pane.



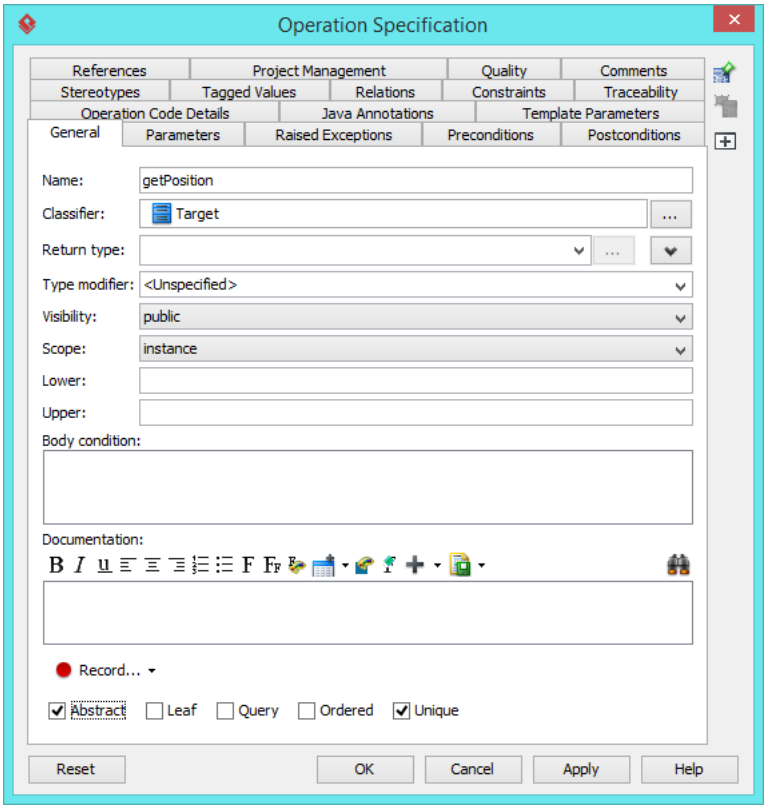
6. Rename *Target* to *Shape*, and its operation *Request* to *getColor* in the bottom pane.



7. Besides the operation *getColor*, we also require two more operations: *getPosition* and *draw*. With *Target* still selected, click the **+** button at the bottom pane, and select **New Operation...** from the popup menu.

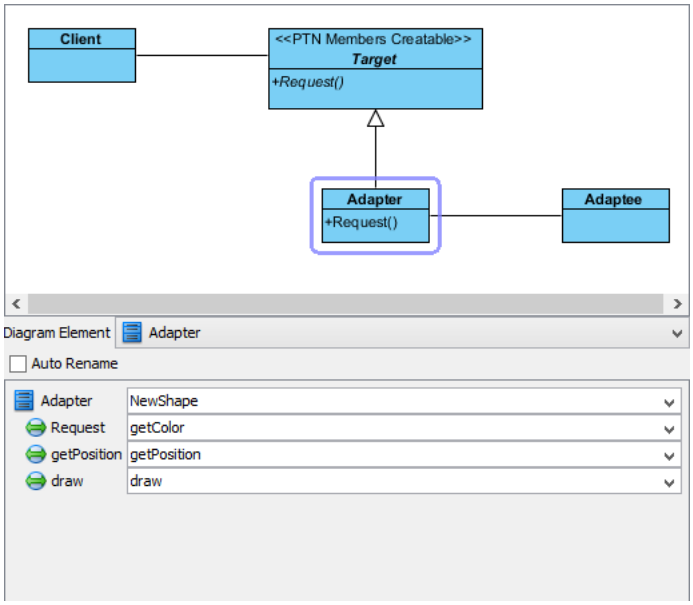


8. In the **Operation Specification** dialog box, name the operation *getPosition*. Check the **Abstract** option at the bottom of the dialog box.

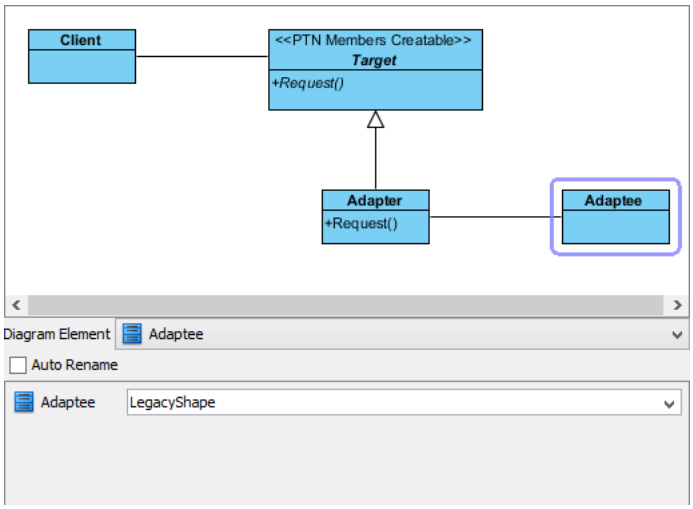


9. Repeat steps 7 and 8 to create the operation *draw*.

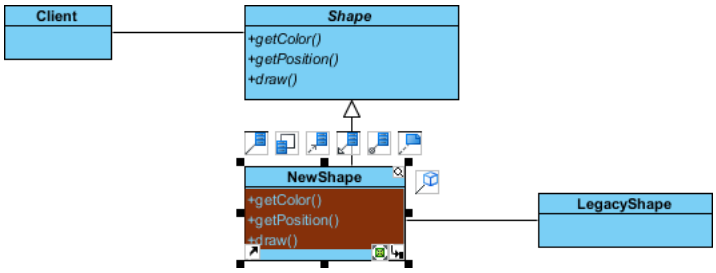
10. Select *Adapter* in the overview, and rename it as *NewShape* in the bottom pane. Also rename the operation *Request* to *getColor*. Note that if the **Auto Rename** option is enabled, renaming the operation is not necessary as it will be handled automatically.



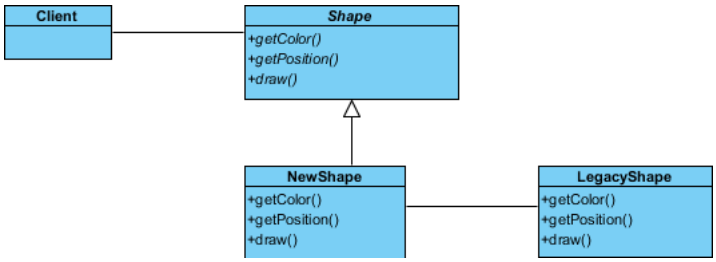
11. Select *Adaptee* in the overview, and rename it as *LegacyShape* in the bottom pane. Click **OK** to apply the pattern to the diagram.



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



13. Press and hold the *Ctrl* key, then drag the selected operations 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

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

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