Design Patterns

宋　杰
Song Jie
东北大学 软件学院
Software College, Northeastern University

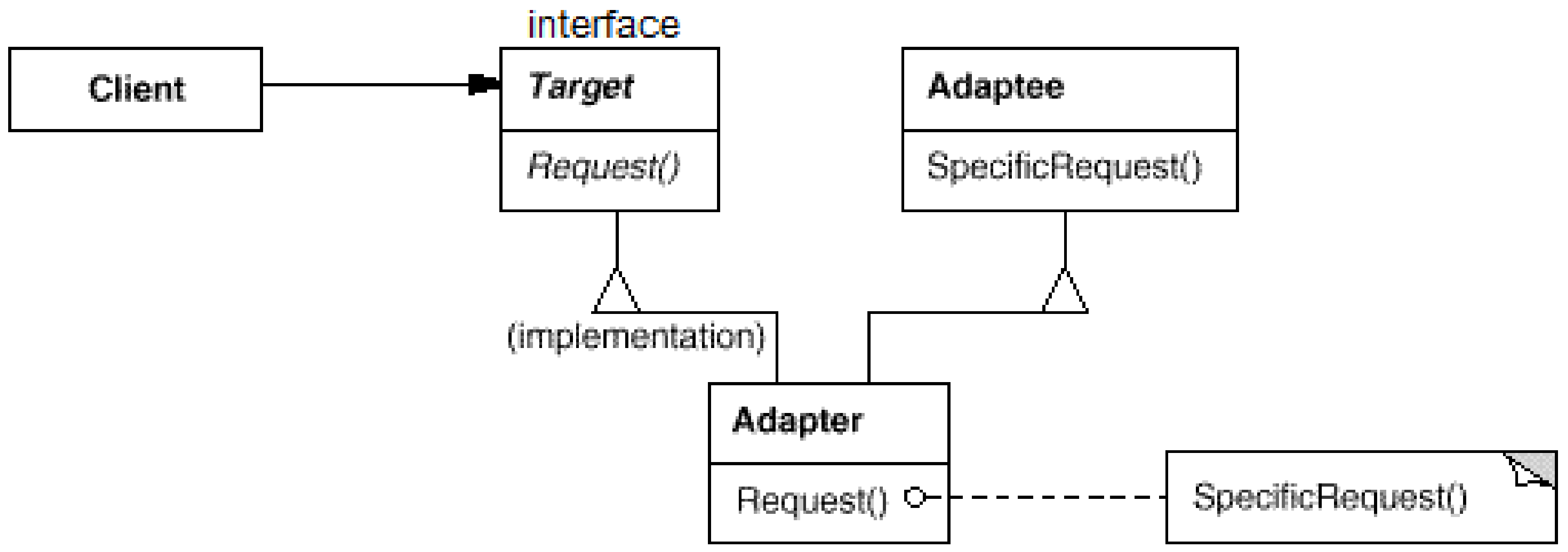
6. Adapter Pattern

Intent

- Convert the interface of a class into another interface clients expect.
- Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

Example

Structure - Class Adapter



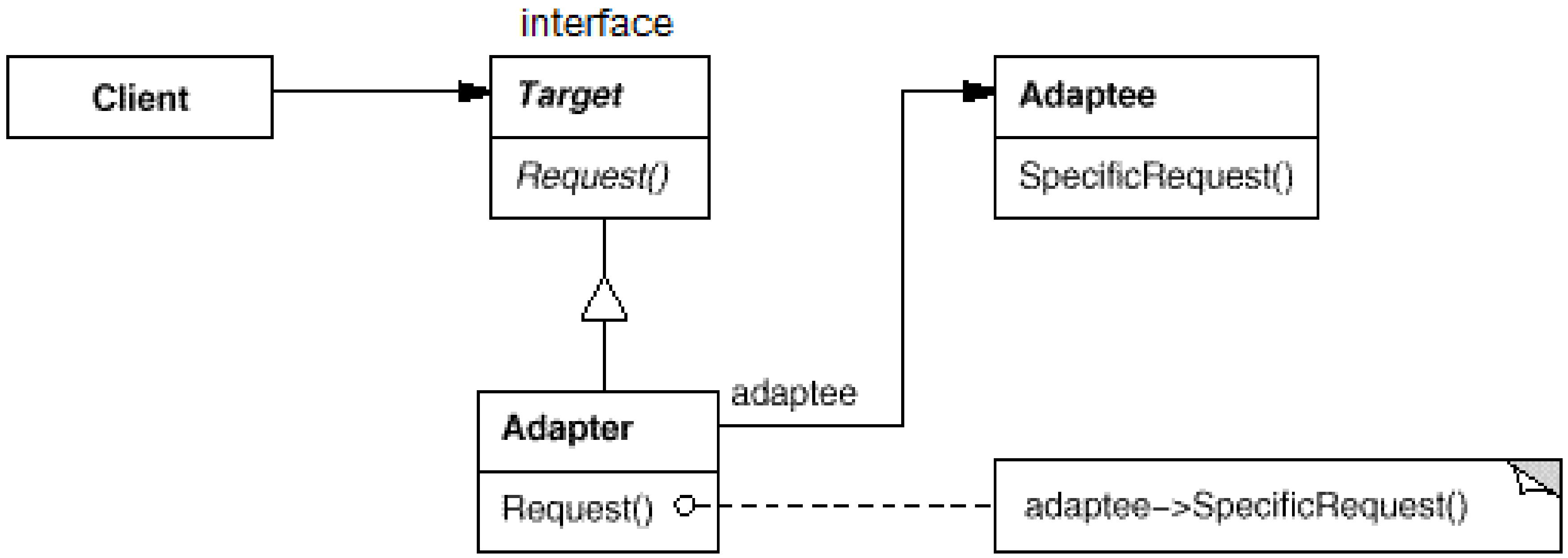
Code Example

```
interface Target {
    public void theMethod();
}
```

```
class Adaptee {
    public void anotherMethod() {

    }
}
```

```
class ClazzAdapter extends Adaptee implements Target {

    public void theMethod() {
        // do something;
        this.anotherMethod();
        // do something;
    }

//  override the super.anotherMethod() if necessary
//  public void anotherMethod() {
//
//  }
}
```

Structure - Object Adapter



Code Example

```
class ObjectAdapter implements Target {
    private Adaptee adaptee;

    ObjectAdapter(){
        adaptee = new Adaptee();
    }
    ObjectAdapter(Adaptee adaptee){
        this.adaptee = adaptee;
    }
    public void theMethod() {
        // do something;
        adaptee.anotherMethod();
        // do something;
    }
}
```

Participants

- Target: Defines the domain-specific interface that Client uses. It should be an interface;
- Client: Collaborates with objects conforming to the Target interface;
- Adaptee: Defines an existing interface that needs adapting, could be an interface, or abstract class, or class;
- Adapter: Adapts the interface of Adaptee to the Target interface.

Consequences - Class Adapter

- Adapting Adaptee to Target by committing to a concrete Adapter class;
  - Adapter could override some of Adaptee's behavior;
- A class adapter won't work when we want to adapt a class and all its subclasses;
- Introducing only one concrete Adapter class, there is only one way making client access the Adaptee.

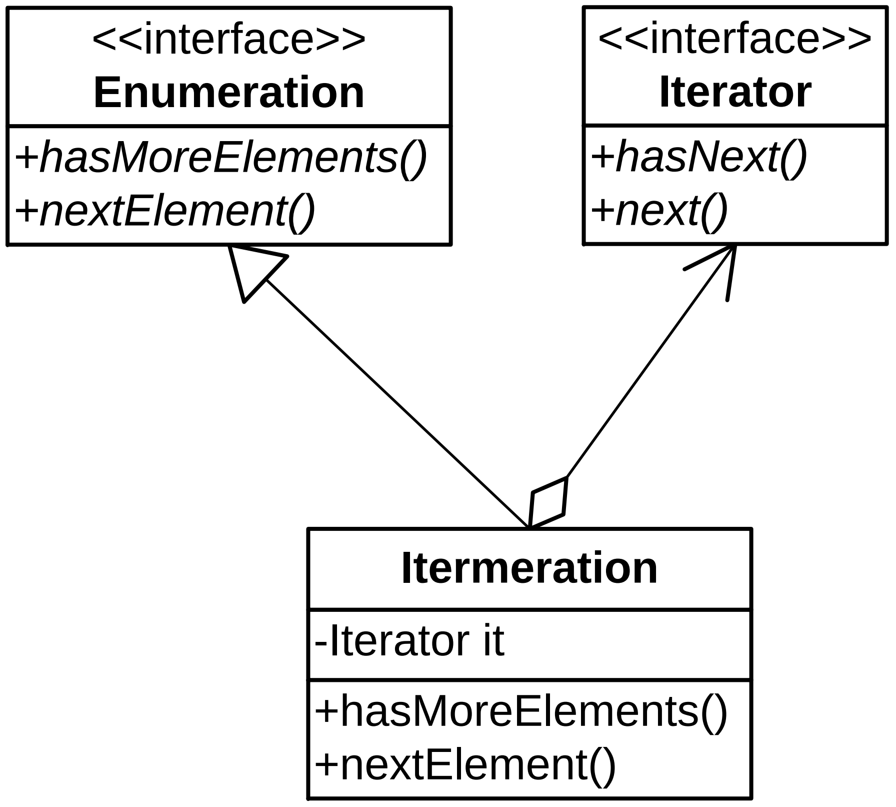
Consequences - Object Adapter

- A single Adapter work with many Adaptees—that is, the Adaptee itself and all of its subclasses (if any).
- The Adapter can also add functionality to all Adaptees at once.
- It is harder to override Adaptee behavior.
  - It will require subclassing Adaptee, then
  - Making Adapter aggregated the subclass rather than the Adaptee itself.
- It is easy to add any new methods, what's more, the added method is suitable for all Adaptee.

Applicability

- You want to use an existing class, and its interface does not match the one you need.
- You want to create a reusable class that cooperates with unrelated or unforeseen classes, that is, classes that don't necessarily have compatible interfaces.
- (object adapter only) You need to use several existing subclasses, but it's impractical to adapt their interface by subclassing every one. An object adapter can adapt the interface of its parent class.

Example: Iterator and Enumeration

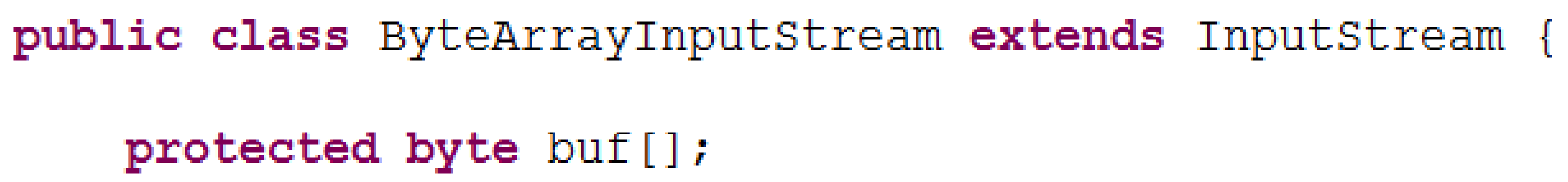


```java
class Itermeration implements Enumeration<Object> {
    Iterator<Object> it;

    public Itermeration(Iterator<Object> it) {
        this.it = it;
    }

    public boolean hasMoreElements() {
        return it.hasNext();
    }

    public Object nextElement() throws NoSuchElementException {
        return it.next();
    }
}
```

```
class ItermerationTest {
    public static void main(String args[]) {
        List<Object> list = new ArrayList<Object>();
        Iterator<Object> it = list.iterator();
        Enumeration<Object> em = new Itermeration(it);
        while (em.hasMoreElements()) {
            System.out.println(em.nextElement());
        }
    }
}
```

Example: Java I/O

- ByteArrayInputStream inherited InputStream (abstract class), and contains an byte array . It adapter an byte array to InputStream
  - ByteArrayOutputStream and byte array.
  - FileInputStream and FileDescriptor
  - FileOutputStream and FileDescriptor



```java
public class ByteArrayInputStream extends InputStream {

    protected byte buf[];
```

Example: WINE

- Wine lets you run Windows software on other operating systems. With Wine, you can install and run these applications just like you would in Windows.
- Which is Target which is Adaptee?

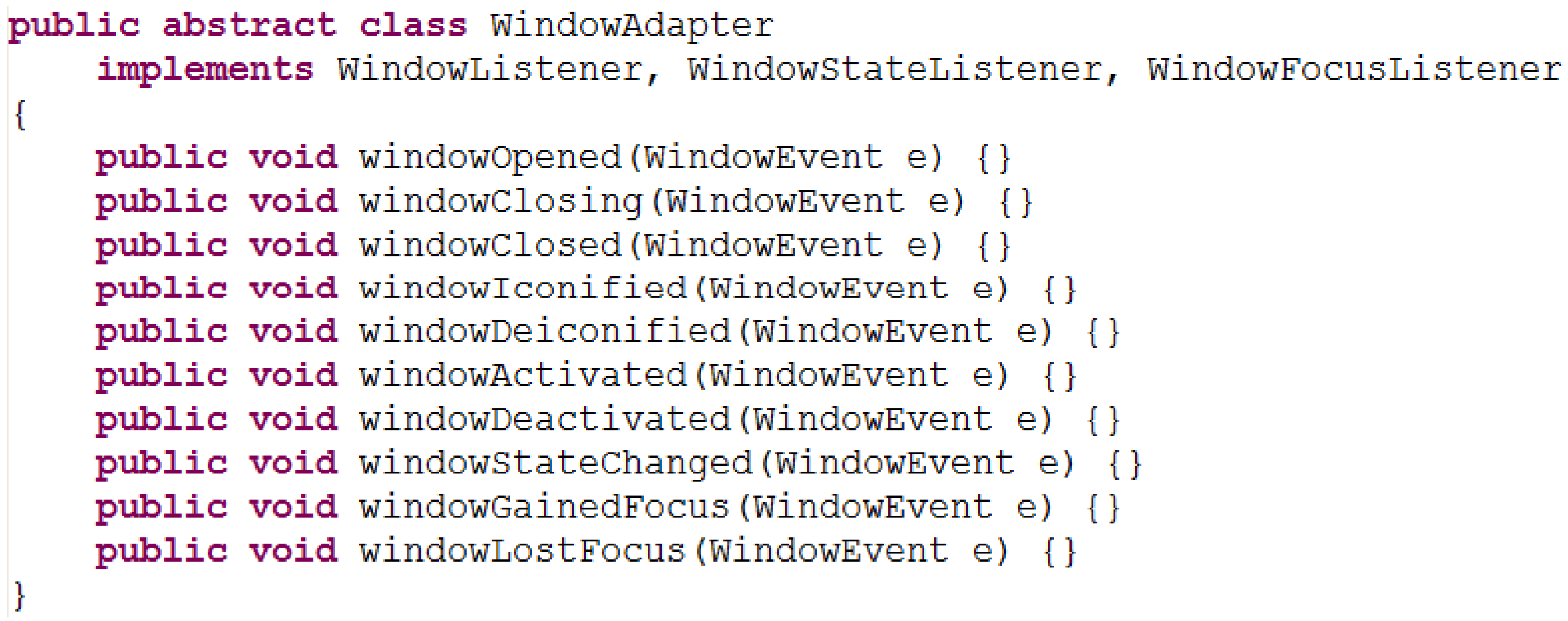
Variation 1: Default Adapter

- In some situations, a class should implement an interface but it does not want to implement every methods that are defined in the interface;
  - An solutions is let the unimplemented methods be empty;
- An default adapter implements the interface, but let all the implemented methods be empty methods, or default implementations.
- The concrete class extends default adapter for implementing the interface, overrides the special methods it wants to implement
  - Generally, the default adapter is an abstract class.

WindowListener

```
public interface WindowListener extends EventListener {
   public void windowOpened(WindowEvent e);
   public void windowClosing(WindowEvent e);
   public void windowClosed(WindowEvent e);
   public void windowIconified(WindowEvent e);
   public void windowDeiconified(WindowEvent e);
   public void windowActivated(WindowEvent e);
   public void windowDeactivated(WindowEvent e);
}
```

WindowAdapter



```java
public abstract class WindowAdapter
    implements WindowListener, WindowStateListener, WindowFocusListener
{
    public void windowOpened(WindowEvent e) {}
    public void windowClosing(WindowEvent e) {}
    public void windowClosed(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
    public void windowActivated(WindowEvent e) {}
    public void windowDeactivated(WindowEvent e) {}
    public void windowStateChanged(WindowEvent e) {}
    public void windowGainedFocus(WindowEvent e) {}
    public void windowLostFocus(WindowEvent e) {}
}
```

Let's go to next…