Design Patterns

- The Strategy Pattern
- The Factory Method
- Generics
- The Abstract Factory Pattern
- The State Pattern
- The Observer Pattern
- **The Adapter Pattern**
- The Composite Pattern
- The Iterator Pattern
- The Builder Pattern
- Fallen Patterns
  - The Singleton Pattern
  - The Visitor Pattern

The Adapter Pattern

- A Structural Pattern
- Allows incompatible interfaces to communicate with one another

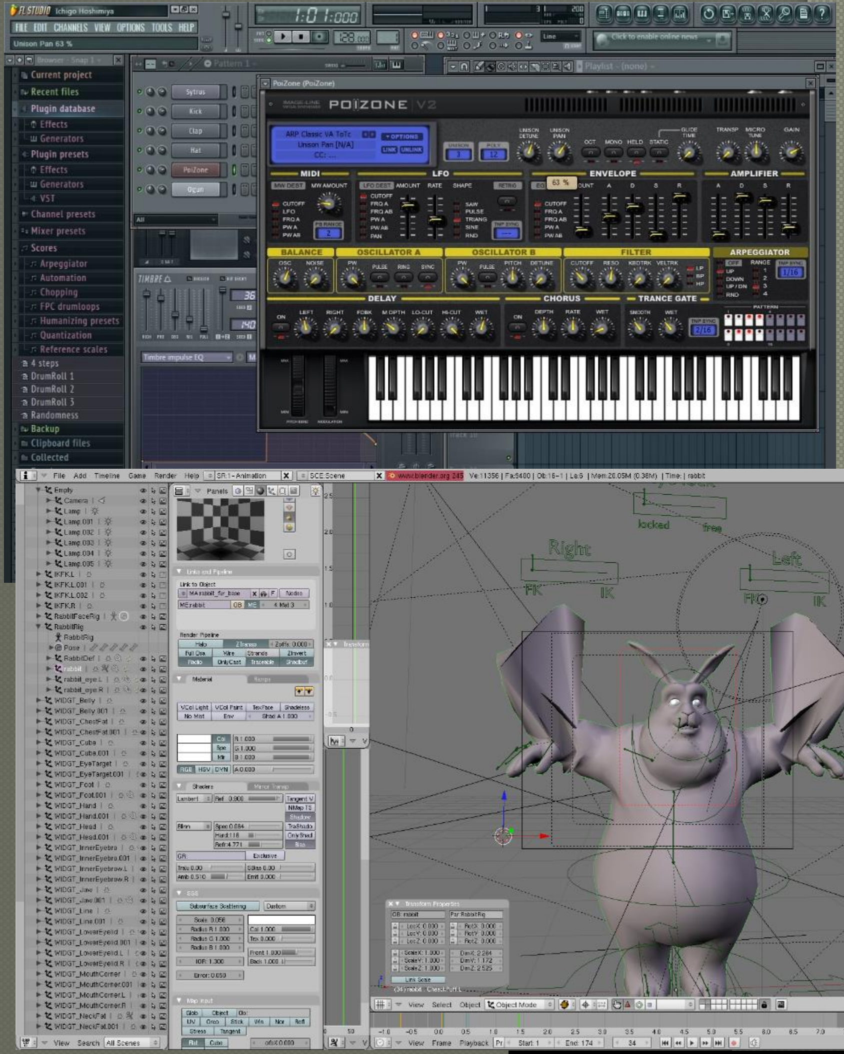
GUI Renderers vs Rendering Systems

Many third-party graphical interface systems allow the user to supply a renderer to use.

Applications like Blender, FL Studio, and Unity have special needs.

Supplying a renderer could potentially be as easy as giving your rendering system to the GUI to use.

It rarely is



The Problem

- The GUI is based on an interface called renderer:

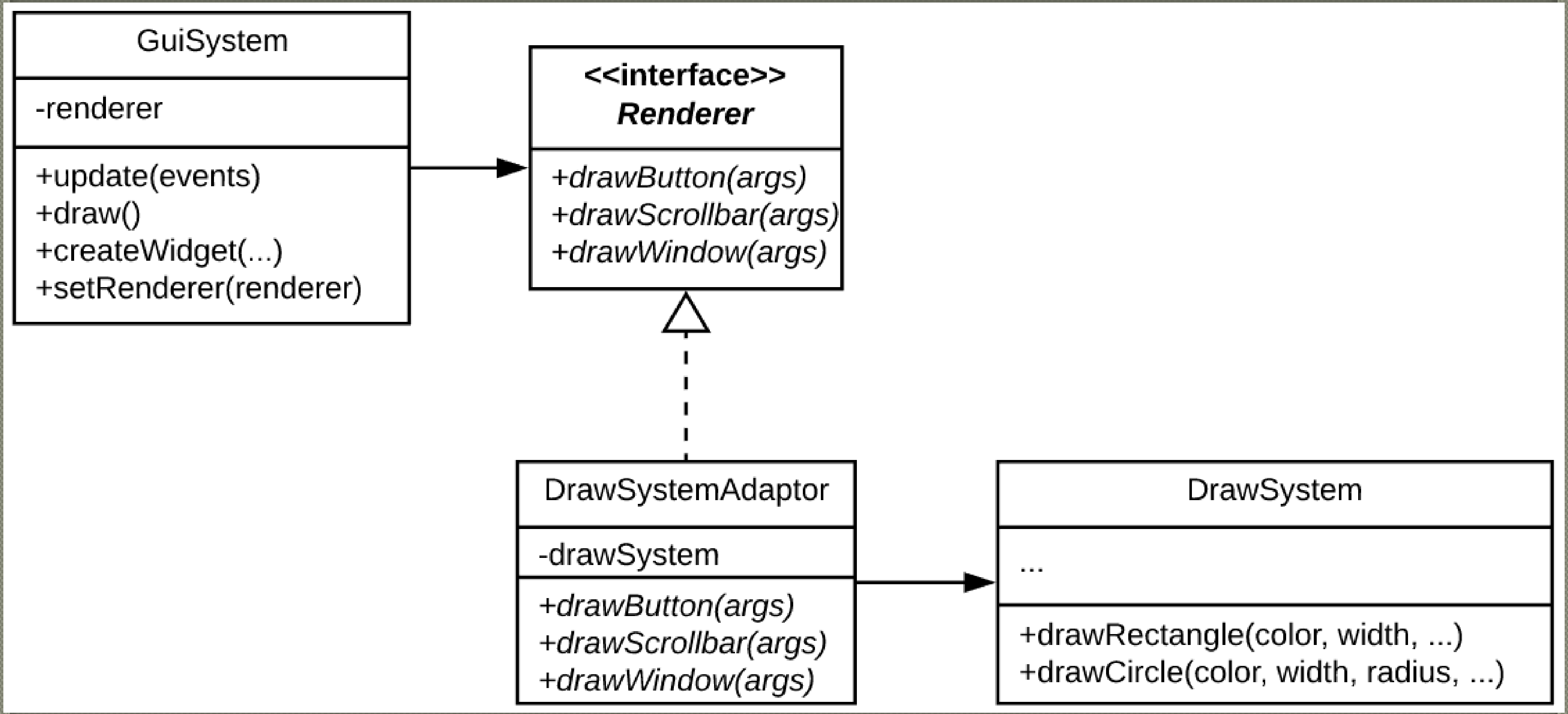
```
interface Renderer {
    void drawButton(…);
    void drawScrollbar(…);
    void drawWindow(…);
}
```

- But your render system may not support these directly:

```
class RenderSystem {
    void drawRectangle(…);
    void drawCircle(…);
}
```

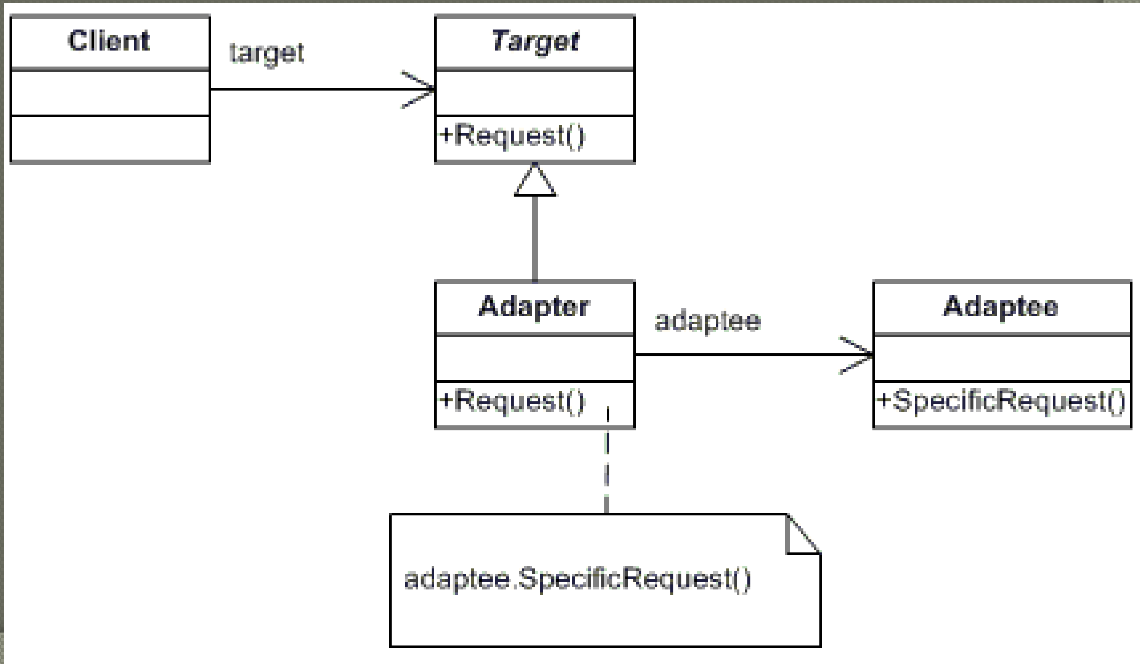
- Obviously, these can be compatible, but they currently aren’t

The Solution: We Adapt



Example Structure

- The adapter inherits from target
- Client wants to talk to a target, therefore we can substitute the adapter
- Adapter delegates method call
- Follows the Liskov Substitution Principle



How do the SOLID principles apply?

- Single responsibility
- Open-closed
- Liskov Substitution
- Interface Segregation
- Dependency Inversion

How do the SOLID principles apply?

- Single responsibility
  - Client and adaptee have specific and different roles
- Open-closed
  - We don’t extend the client to add adaptee functionality
- Liskov Substitution
  - Adaptor inherits/implements from target so client can call adaptor
- Interface Segregation
  - Target offers a minimal interface to client
- Dependency Inversion
  - Client depends on target, rather than directly on adaptee

Apply Adapter to your Project

Think about your experiences using external libraries so far.

What are some situations where you had to "adapt" an external class to work with your application.

Can you find *more than one*?