Object-Oriented Programming
Adapter Pattern

CSIE Department, NTUT
Woei-Kae Chen
Adapter: Intent

- Convert the interface of a class into another interface clients expect.
  - Adapter lets classes work together that could not otherwise because of incompatible interfaces.
- Also known as Wrapper
The adapter is responsible for functionality the adapted class does not provide.
Adapter: Applicability

- You want to use an existing class, and its interface does not match the one you need.
- You want to create a class that cooperates with unrelated or unforeseen classes
  - incompatible interfaces.
- Object adapter only
  - You need to use several existing subclasses, but it’s impractical to adapt their interface by subclassing every one.
  - adapt the interface of its parent class
Adapter: Structure (1)

Class adapter: use multiple inheritance

Client

Target
Request()

Adaptee
SpecificRequest()

Adapter
Request()

SpecificRequest()
Adapter: Structure (2)

Object adapter: use object composition

Client → Target
  Request()

Adaptee
  SpecificRequest()

Adapter
  Request()

adaptee

adaptee->SpecificRequest()
Adapter: Participants

- **Client (Drawing Editor)**
  - collaborates with objects conforming to the Target interface.
- **Target (Shape)**
  - defines the domain-specific interface that Client uses.
- **Adapter (TextShape)**
  - adapts the interface of Adaptee to the Target interface
- **Adaptee (TextView)**
  - defines an existing interface that needs adapting.
Clients call operations on an Adapter instance. In turn, the adapter calls Adaptee operations that carry out the request.
Adapter: Consequences (1)

Class adapter ⇔ Object adapter

- **Class adapter**
  - adapts Adaptee to Target by committing to a concrete Adaptee class → doesn't work when adapting a class and all its subclasses.
  - allow Adapter to override some of Adaptee’s behavior
  - introduces only one object → no additional pointer indirection is needed to get to the adaptee.

- **Object adapter**
  - a single Adapter works with many Adaptees (Adaptees and all of its subclasses; add functionality to all Adaptees at once.
  - harder to override Adaptee behavior.
Adapter: Consequences (2)

- **How much adapting does Adapter do?**
  - Adapters **vary** in the amount of work they do to adapt Adaptee to the Target interface.

- **Pluggable adapters**
  - classes with build-in interface adaptation.

- **Two-way adapters (provide transparency)**
  - an adapted object no longer confirms to the Adaptee interface → can’t be used as an Adaptee object.
  - useful when two different clients need to view an object differently.
Adapter: Consequences (3)

Two-way adapter using multiple inheritance

(to QOCA hierarchy) (to Unidraw class hierarchy)

ConstraintVariable

StateVariable

ConstraintStateVariable
Adapter: Implementation (1)

- **Class adapter in C++**
  - public from Target and private from Adaptee.

- **Pluggable adapters**
  - classes with build-in interface adaptation.
  - narrow interface: the smallest subset of operations for adaptation.
  - three implementation approaches
    - Using abstract operations
    - Using delegate objects
    - Parameterized adapters
      - supports adaptation without subclassing
      - Smalltalk: block construct; Java: reflection
Adapter: Implementation (2)

Pluggable adapter: using abstract operations to simply **XXXTreeDisplay**

```java
TreeDisplay
GetChildren(Node)
CreateGraphicNode(Node)
Display()
BuildTree(Node n)

DirectoryTreeDisplay
GetChildren(Node)
for each child {
    AddGraphicNode(
        CreateGraphicNode(child)
    )
    BuildTree(child)
}

Client, Target

Adapter

Narrow interface

Template Method

GetChildren(n)

FileSystemEntity

Adaptee
```
Pluggable adapter: using delegate objects to simply XXXBrowser

delegate->GetChildren(this,n)
for each child {
    AddGraphicNode(
        delegatep->
            CreateGraphicNode(child)
    )
    BuildTree(child)
}
Adapter: Related patterns

- **Bridge**
  - Bridge has a structure similar to an object adaptor and different intents.

- **Decorator**
  - Decorator enhances another object without changing its interface, which is more transparent than an adapter.

- **Proxy**
  - Proxy defines a surrogate for another object without changing its interface.

- **Template Method**
  - Can be used to implement pluggable adapter.

- **Strategy**
  - Can be used to implement pluggable adapter.