

CS 635 Advanced Object-Oriented Design & Programming
Fall Semester, 2022
Doc 13 Adapter, Bridge
Oct 11, 2022

Copyright ©, All rights reserved. 2022 SDSU & Roger Whitney,
5500 Campanile Drive, San Diego, CA 92182-7700 USA.
OpenContent (<http://www.opencontent.org/opl.shtml>) license
defines the copyright on this document.

Adapter



Adapter

Convert interface of a class into another interface

Use adapter when

You want to use an existing class but does not have interface on needs

You want to create a reusable class that works with unrelated or unforeseen classes

Address Book & JTable

Display an AddressBook object in a JTable

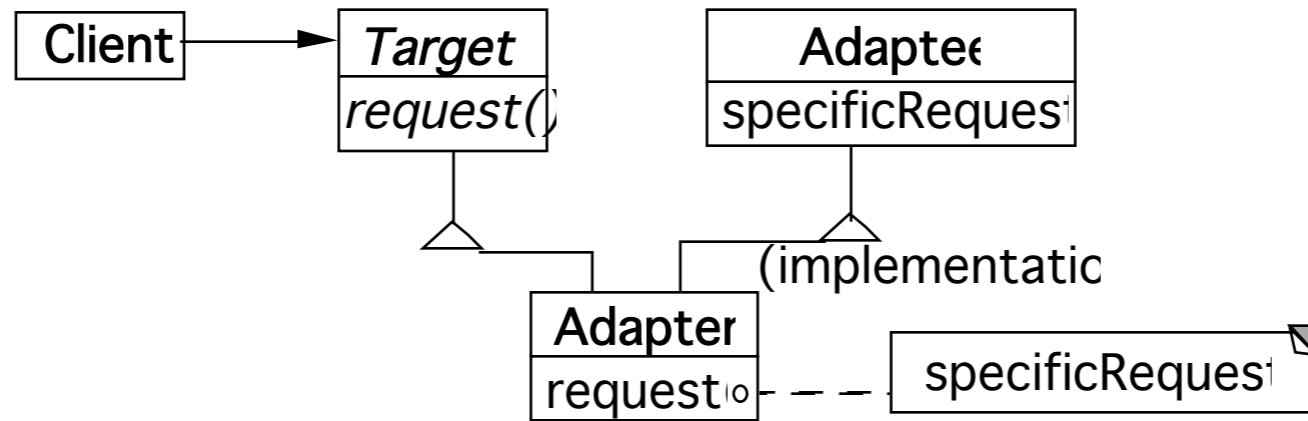
JTables require objects of type TableModel

```
public class AddressBook{
    List personList;
    public int getSize(){...}
    public int addPerson(...){...}
    public Person getPerson(...){...}
    ...
}
```

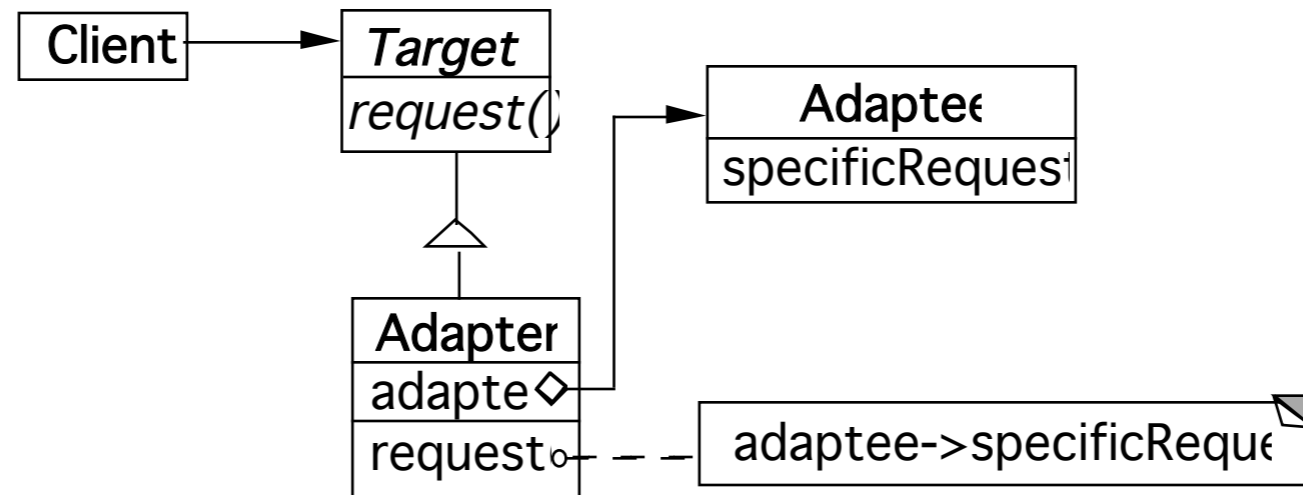
```
public class AddressBookTableAdapter implements TableModel
{
    AddressBook ab;
    public AddressBookTableAdapter( AddressBook ab ){
        this.ab = ab;
    }
    //TableModel impl
    public getRowCount(){
        ab.getSize();

    public Object getValueAt(int rowIndex, int columnIndex) {
        Person requested =
            ad.getPerson(convertRowToName(rowIndex));
        return requested.get(convert(columnIndex));
    }
}
```

Class Adapter



Object Adapter



Class Adapter Example

```
class OldSquarePeg {
    public: void squarePegOperation() { do something }
}

class RoundPeg {
    public: void virtual roundPegOperation = 0;
}

class PegAdapter: private OldSquarePeg, public RoundPeg {
    public:
        void virtual roundPegOperation() {
            add some corners;
            squarePegOperation();
        }
}

void clientMethod() {
    RoundPeg* aPeg = new PegAdapter();
    aPeg->roundPegOperation();
}
```

Object Adapter

```
class OldSquarePeg{  
    public: void squarePegOperation() { do something }  
}
```

```
class RoundPeg    {  
    public: void virtual roundPegOperation = 0;  
}
```

```
class PegAdapter: public RoundPeg    {  
    private:  
        OldSquarePeg* square;  
  
    public:  
        PegAdapter() { square = new OldSquarePeg; }  
  
        void virtual roundPegOperation()    {  
            add some corners;  
            square->squarePegOperation();  
        }  
}
```

How Much Adapting does the Adapter do?

Two-way Adapters

```
class OldSquarePeg {  
    public:  
        void virtual squarePegOperation() { blah }  
}
```

```
class RoundPeg {  
    public:  
        void virtual roundPegOperation() { blah }  
}
```

```
class PegAdapter: public OldSquarePeg, RoundPeg {  
    public:  
        void virtual roundPegOperation() {  
            add some corners;  
            squarePegOperation();  
        }  
        void virtual squarePegOperation() {  
            add some corners;  
            roundPegOperation();  
        }  
}
```

Flasher and MouseListener

```
class Flasher
  def toggle()
    @flashing = !@flashing
  end

  def pause()
    #etc
  end

  def resume()
    #etc
  end
end
```

```
class MouseListener
  def mouseClicked(event)
  end

  def mouseEntered(event)
  end

  def mouseExited(event)
  end
end
```

Actions we want

mouse click toggles flasher
mouse enter pauses
mouse exits resumes

Flasher as MouseListener

```
class Flasher
  def toggle()
    @flashing = !@flashing
  end

  def pause()
    #etc
  end

  def resume()
    #etc
  end

  def mouseClicked(event)
    toggle()
  end

  def mouseEntered(event)
    pause()
  end

  def mouseExited(event)
    resume()
  end
end
```

Simple Adapter

```
class Flasher
  def toggle()
    @flashing = !@flashing
  end

  def pause()
    #etc
  end

  def resume()
    #etc
  end
end

yellowFlasher = Flasher.new(yellow, fast)
FlasherAdapter.new(yellowFlasher)
```

```
class FlasherAdaptor
  def initialize(aFlasher)
    @flasher = aFlasher
  end

  def mouseClicked(event)
    @flasher.toggle()
  end

  def mouseEntered(event)
    @flasher.pause()
  end

  def mouseExited(event)
    @flasher.resume()
  end
end
```

A Ruby Adapter - Forwardable

```
class Flasher
  def toggle()
    @flashing = !@flashing
  end

  def pause()
    #etc
  end

  def resume()
    #etc
  end
end
```

```
require 'forwardable'

class FlasherMouseListener
  extend Forwardable

  def initialize()
    @flasher = Flasher.new()
  end

  def _delegator(:@flasher, :toggle, :mouseClick)
  def _delegator(:@flasher, :pause, :mouseenter)
  def _delegator(:@flasher, :resume, :mouseleave)

end
```

```
adaptor = FlasherMouseListener.new()
adaptor.mouseClick()
```

Parameterized Adapter

```
class MouseListenerAdapter
```

```
  def initialize(adaptee, clickMethod, enterMethod, exitMethod)
```

```
    @adaptee = adaptee
```

```
    @clickMethod = clickMethod
```

```
    @enterMethod = enterMethod
```

```
    @exitMethod = exitMethod
```

```
  end
```

```
  def mouseClicked(event)
```

```
    @adaptee.send(clickMethod)
```

```
  end
```

```
  def mouseEntered(event)
```

```
    @adaptee.send(clickMethod)
```

```
  end
```

```
  def mouseExited(event)
```

```
    @adaptee.send(clickMethod)
```

```
  end
```

```
end
```

```
yellowFlasher = Flasher.new(yellow, fast)
```

```
MouseListenerAdapter.new(
```

```
  yellowFlasher,
```

```
  :toggle,
```

```
  :pause,
```

```
  :resume)
```

Better Parameterized Adapter

```
class MouseListenerAdapter
```

```
  def initialize(adaptee, clickLambda, enterLambda, exitLambda)
```

```
    @adaptee = adaptee
```

```
    @clickLambda = clickLambda
```

```
    @enterLambda = enterLambda
```

```
    @exitLambda = exitLambda
```

```
  end
```

```
  def mouseClicked(event)
```

```
    @clickLambda.call(adaptee)
```

```
  end
```

```
  def mouseEntered(event)
```

```
    @enterLambda.call(adaptee)
```

```
  end
```

```
  def mouseExited(event)
```

```
    @exitLambda.call(adaptee)
```

```
  end
```

```
end
```

```
yellowFlasher = Flasher.new(yellow, fast)
```

```
MouseListenerAdapter.new(
```

```
  yellowFlasher,
```

```
  lambda {|flasher| flasher.toggle()},
```

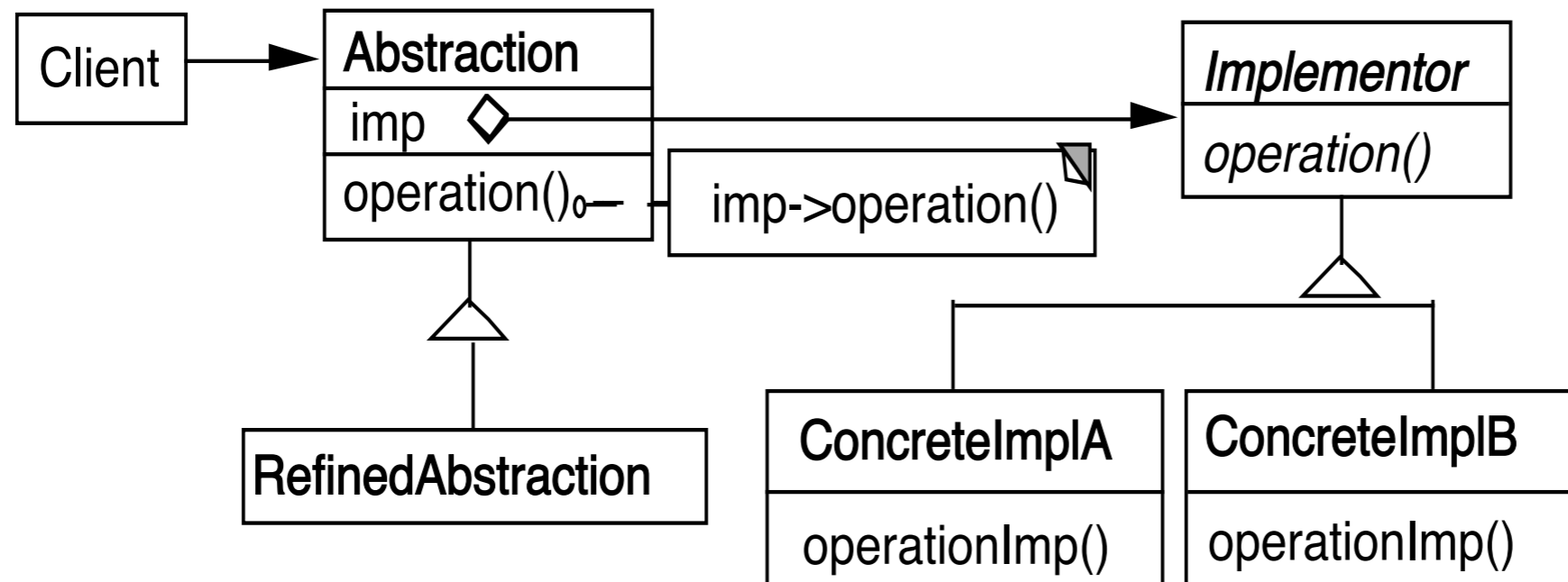
```
  lambda {|flasher| flasher.pause()},
```

```
  lambda {|flasher| flasher.resume()})
```

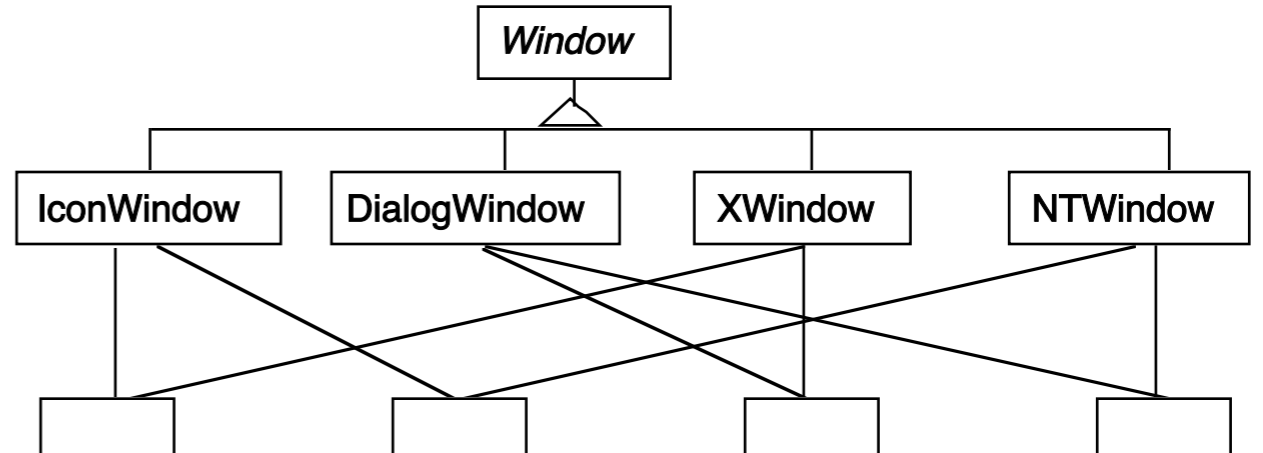
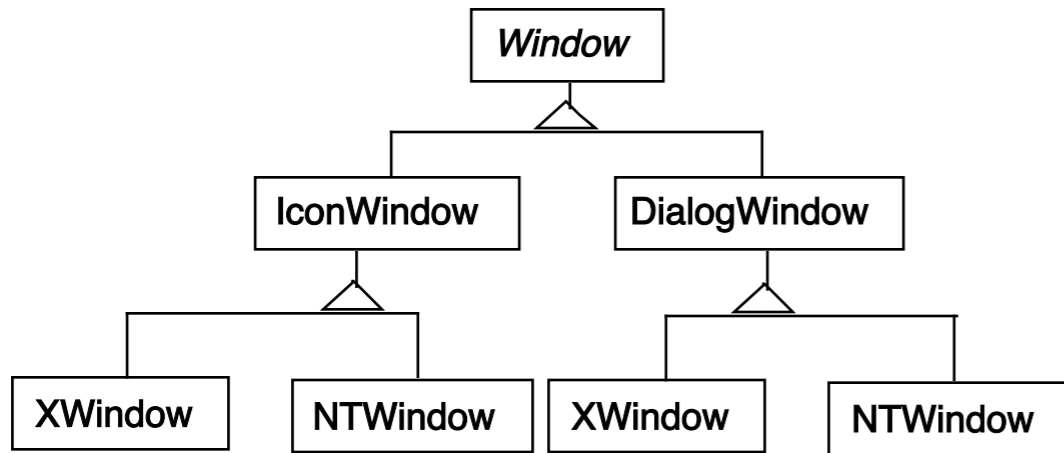
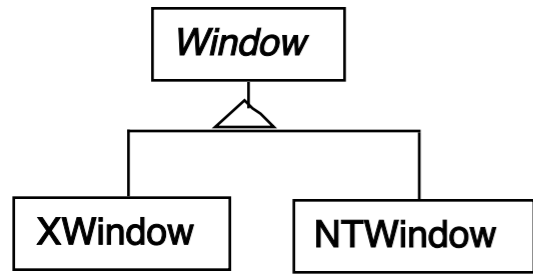
Bridge

Bridge

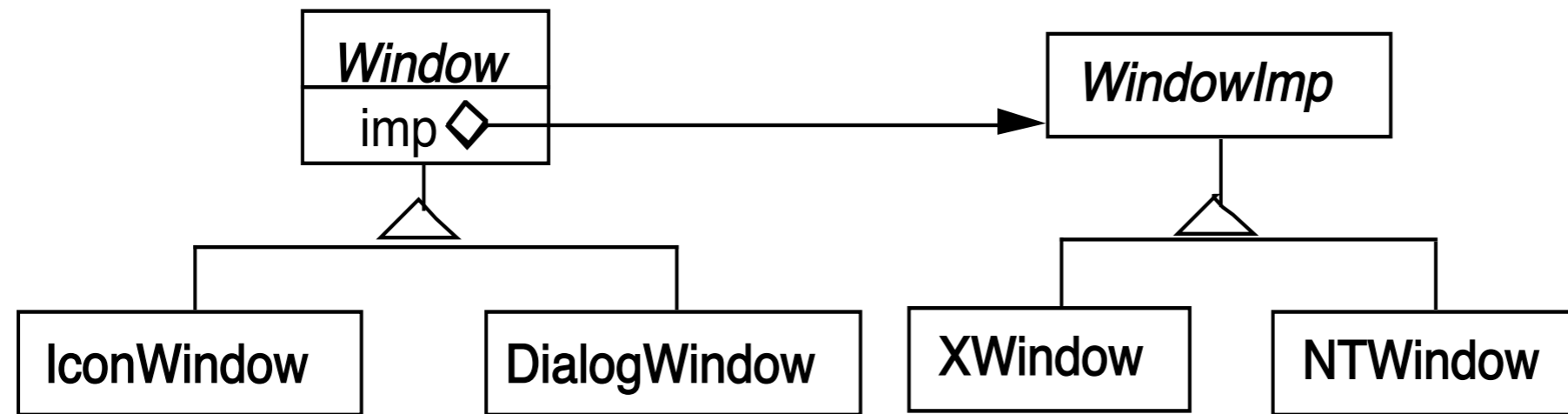
Decouple an abstraction from its implementation



Windows



Using the Bridge Pattern



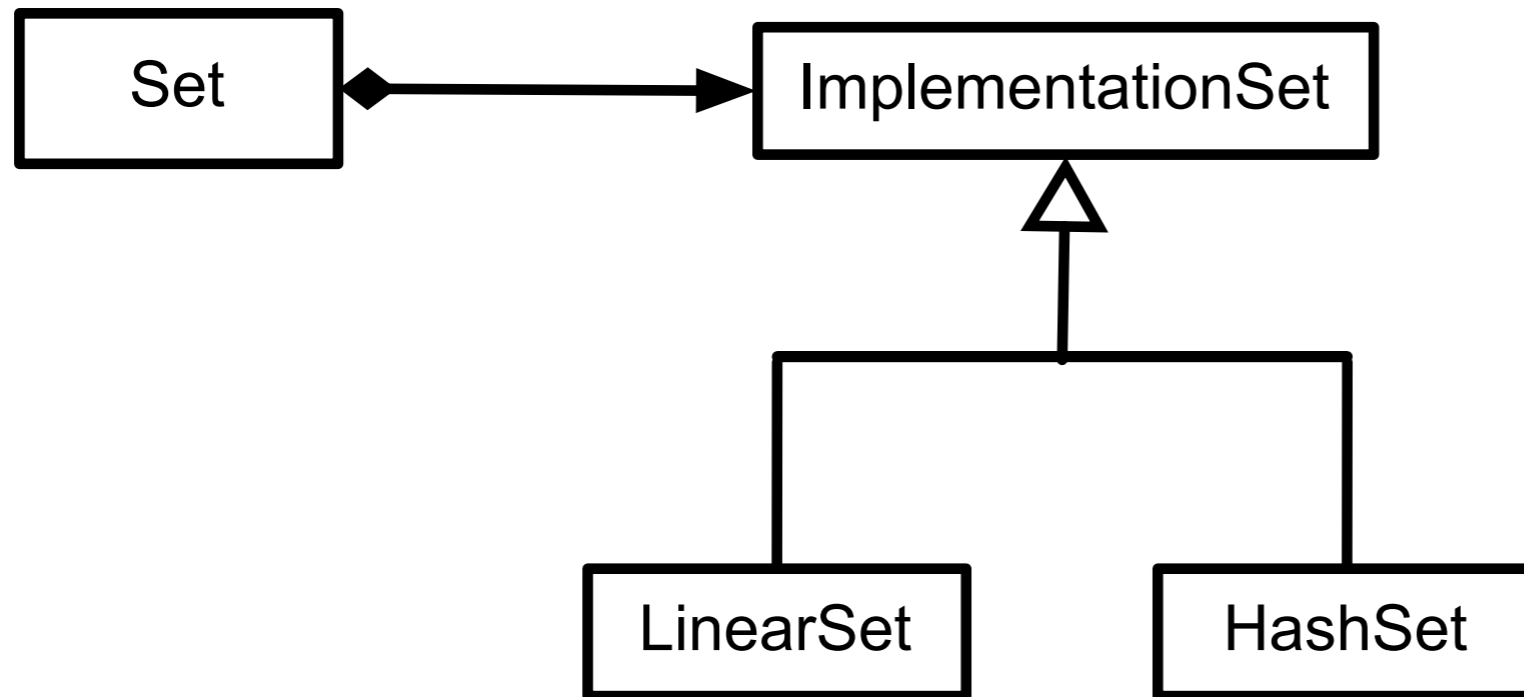
Peers in Java's AWT



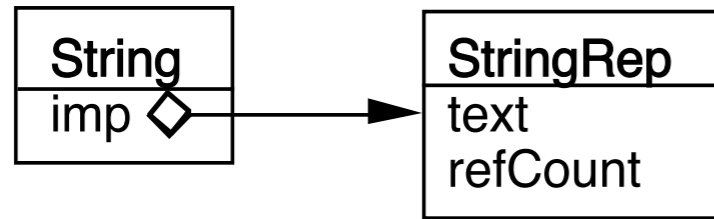
Peer = implementation

```
public synchronized void setCursor(Cursor cursor) {  
    this.cursor = cursor;  
    ComponentPeer peer = this.peer;  
    if (peer != null) {  
        peer.setCursor(cursor);  
    }  
}
```

IBM Smalltalk Collections



Smart Pointers in C++



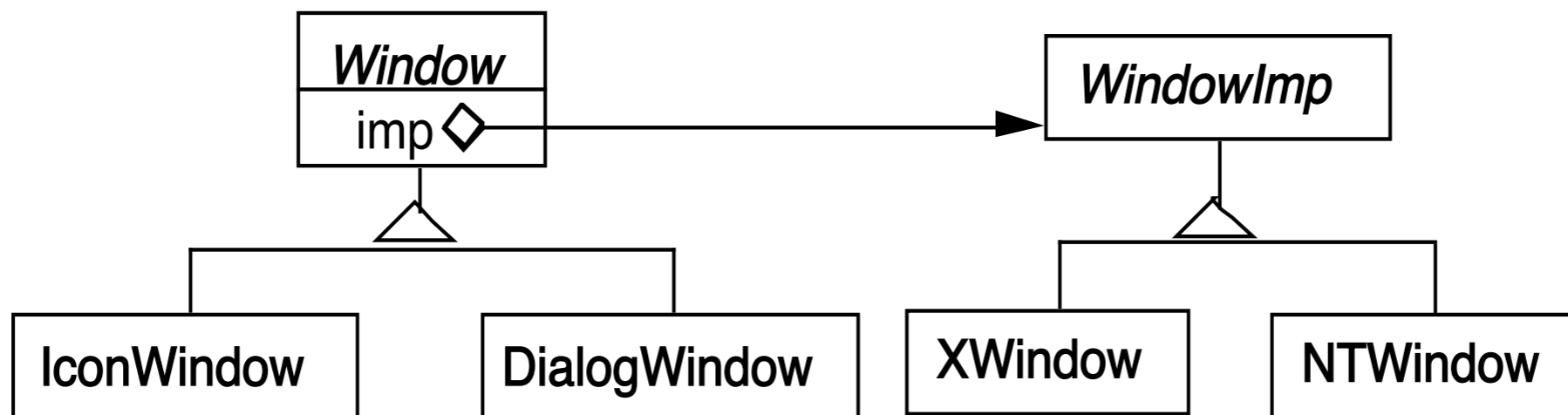
Why Use Bridge

Implementation selected at run-time

Implementation changed during run-time

Why Use Bridge

Abstraction & implementations are extensible by subclassing

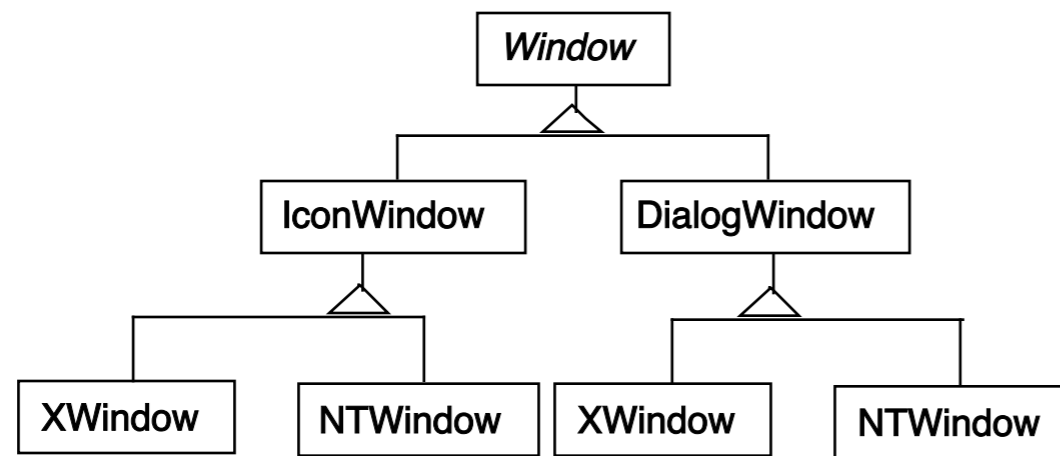


Why Use Bridge

When changes in the implementation should not require client code to be recompiled

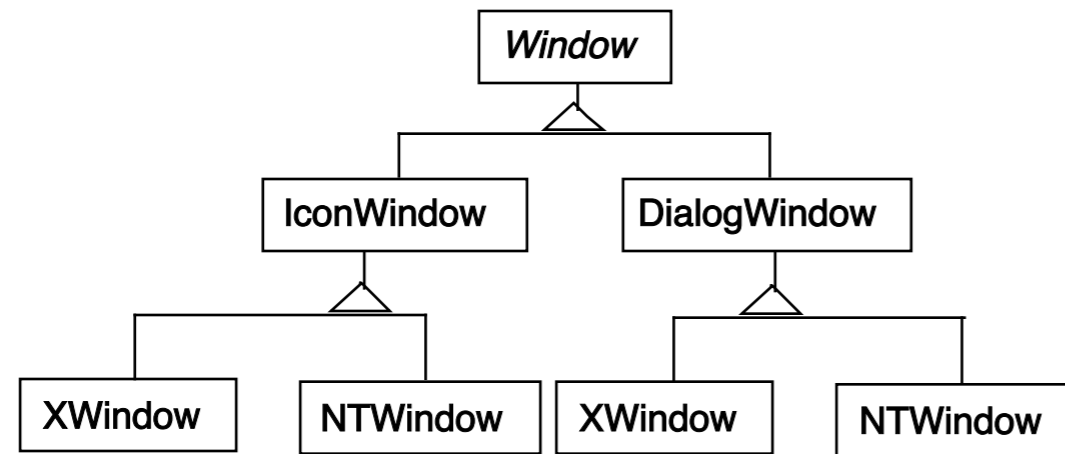
Why Use Bridge

Proliferation of classes



Why Use Bridge

Share implementation among multiple objects



Bridge verses Adapter