COMP 303
Software Development

Design Patterns 4

Readings

- Textbook: Chapter 10

Final Exam

- Tuesday, April 24, 2:00-5:00 PM
- Covers all of the course
  - Lectures
  - Assignments
  - Class tests
- Five questions
  - UML, Identify DP, Propose DP, Well designed techniques
  - Generics, Object type, abstract, inheritance, interfaces, contracts, etc.
  - Case study analysis
  - No java docs, no junit, not much programming
- Tutorials & new office hours during exam TBD

Outline

- Important:
  - Visitor
  - Factor Method Pattern
- If time permits:
  - Adaptor Pattern
  - Command Pattern

Design Patterns 4

THE VISITOR PATTERN

The Visitor Pattern

Example:

You would like to select these three shapes and with one command change their fill colour.

Easy → They all extend from Shape. ArrayList them.
foreach(x in ArrayList) x.setColor(RED);

Cool, but… this only works because setColor(color)
is a method in the parent class

WHAT IF… you want to do the above, but, with a method
not defined in the parent class?

The Visitor Pattern

How can we create an **extensible** class?

Examples:
- You want to add methods to a Java library class
- Adding a new method to the root class of a large inheritance tree
- You would like to create something like a plugin

The Visitor Pattern
Via A Trick

The parent class contains this method:

void accept(Visitor v)

object that implements the interface

defined in
parent class

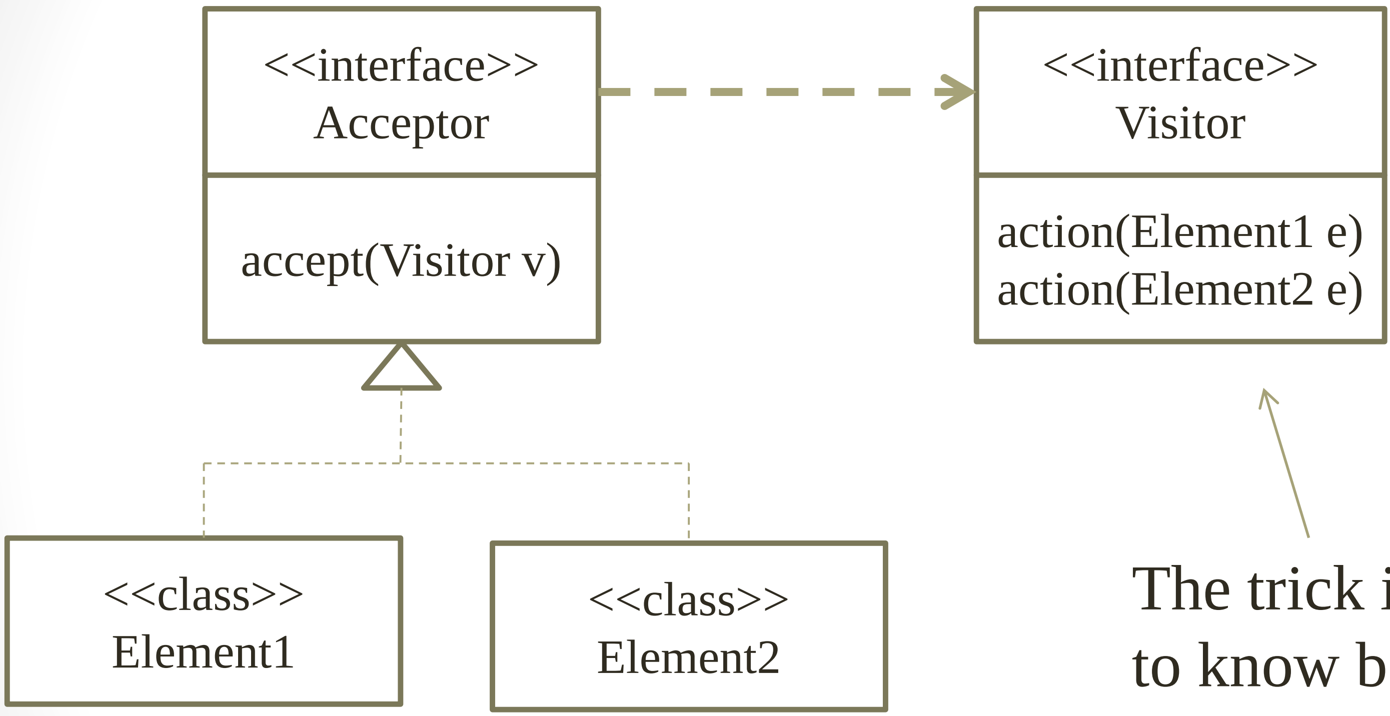
interface

```
public void accept(Visitor v)
{
   v.visit(this);
}
```

the parent class is passed to v… permitting v to use the public methods in some new way

The Visitor Pattern

The benefit is action() calls <<class>> public methods in a new combination.



The trick is we need to know beforehand all the <<class>> names.

If a finite number then okay.

The Visitor Pattern
Example

Let us say we have this root class from a complex inheritance tree:

```
class Size implements Acceptor {
   private int x1, y1, x2, y2;
   Size(int x1,int y1, int x2, int y2) {…}
   public int length() {…}
   public int slope() {…}
   public int get…() {…} // for each private
   public accept(Visitor v) {v.action(this);}
}
```

And we would like to add the `areaOfSquare()` method to it…

The Visitor Pattern

Example

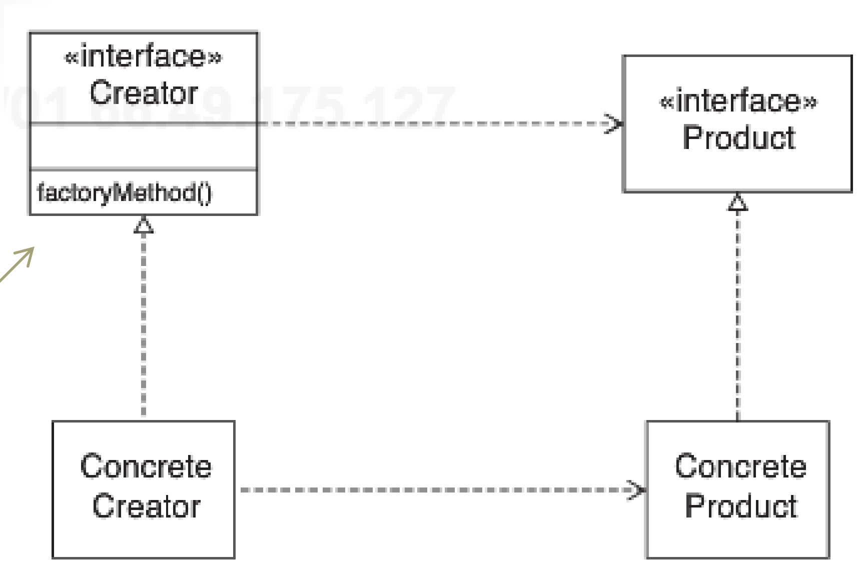
We need to create the class to house the action() method:

```
class MyAction implements Visitor {
   // Returns the area of the square
   public void action(Size p) {
     int length = p.length();
     int height = calcH(p.getX()…);
     System.out.print(length*height);
   }

   private int calcH(int x1,y1,x2,y2) {…}
}
```

Design Patterns 4

FACTORY METHOD PATTERN

Factory Method Pattern



We don't know it's type because we return an interface reference

We want to call an interface method…
But we don't know the type of the object to instantiate…

So, use a method to instantiate the unknown object (that implements the desired interface)

Factory Method Pattern
Date Case Study

We don’t know the kind of date being used…

```
DateFormat formatter = DateFormat.getDateInstance();
Date now = new Date();
String formattedDate = formatter.format(now);
```

Persian, Hebrew, European, Asian dating forms?

Design Patterns 4

ADAPTOR PATTERN

Adaptor Pattern

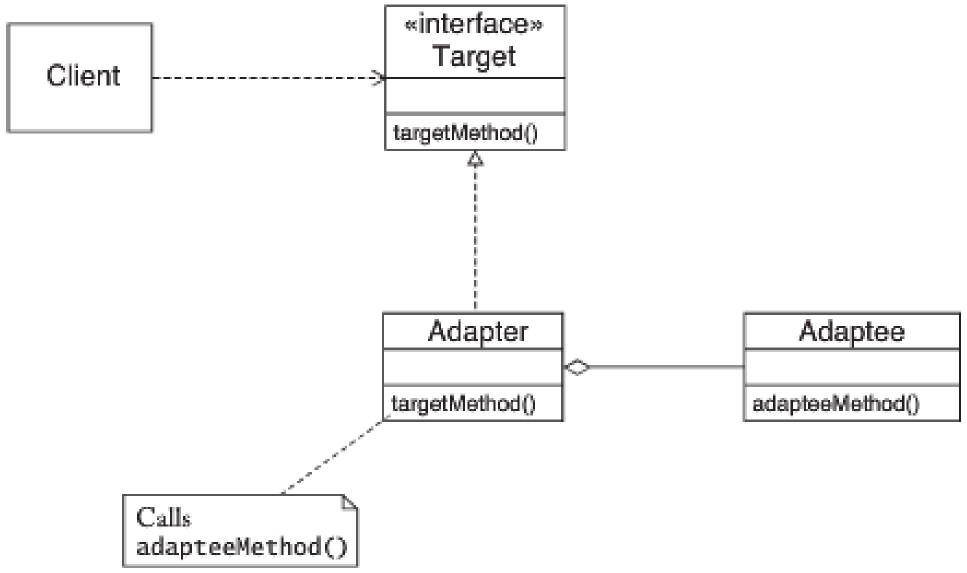
Target

Adaptor

Adaptee

A device that interfaces an adaptee with a target appliance.

Adaptor Pattern

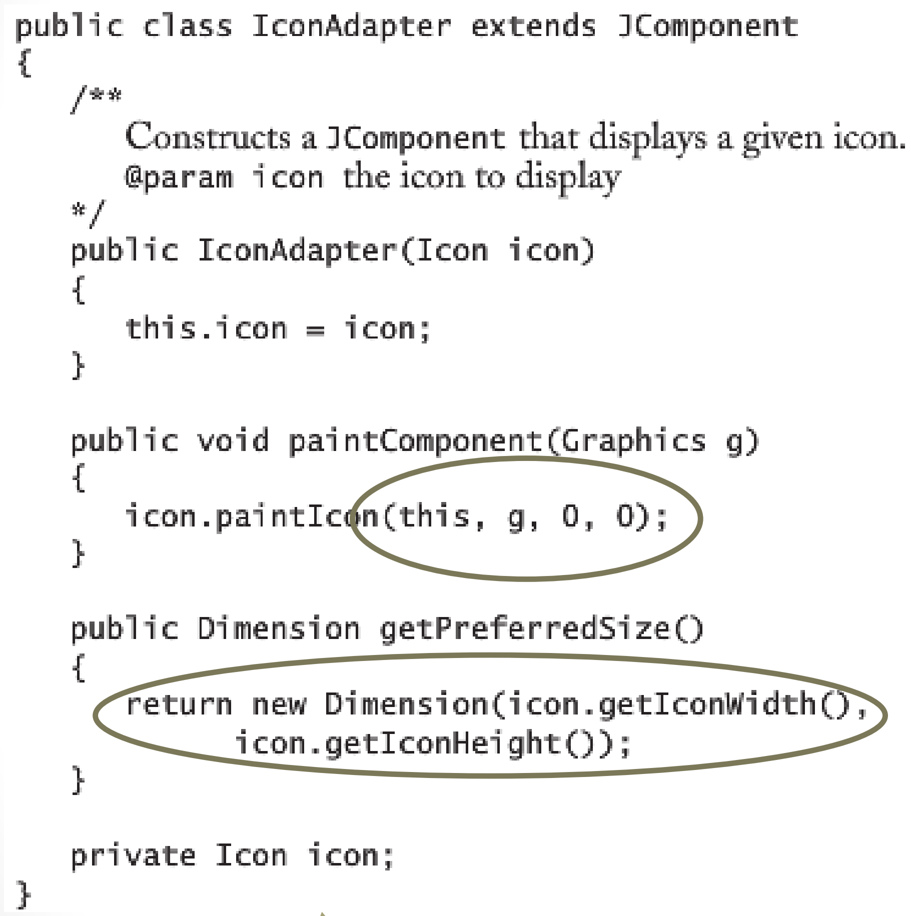


EXAMPLE

Name in Design Pattern	Actual Name
Adaptee	Icon
Target	JComponent
Adapter	IconAdapter
Client	The class that wants to add icons into a container
targetMethod()	paintComponent(), getPreferredSize()
adapteeMethod()	paintIcon(), getIconWidth(), getIconHeight()

The adapted methods →

Adaptor Pattern



```
public class IconAdapter extends JComponent
{
    /**
       Constructs a JComponent that displays a given icon.
       @param icon the icon to display
    */
    public IconAdapter(Icon icon)
    {
       this.icon = icon;
    }

    public void paintComponent(Graphics g)
    {
       icon.paintIcon(this, g, 0, 0);
    }

    public Dimension getPreferredSize()
    {
       return new Dimension(icon.getIconWidth(),
             icon.getIconHeight());
    }

    private Icon icon;
}
```

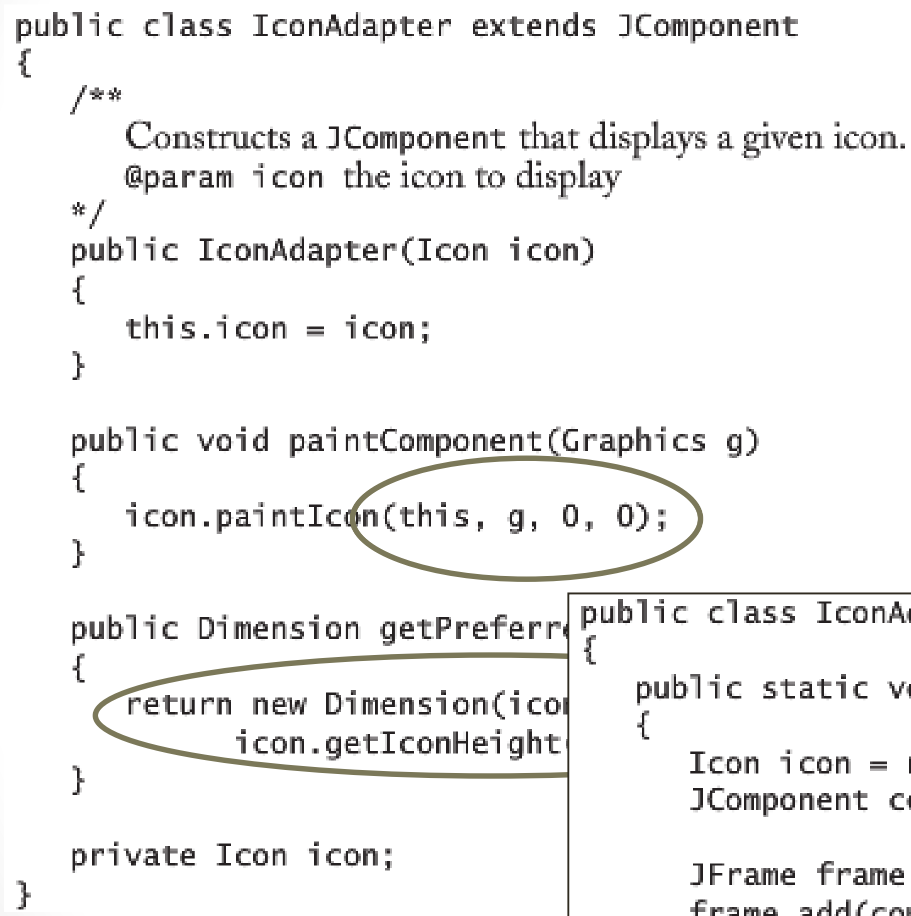
Converting an ICON for use as a Component.

void paintComponent(Graphics g)

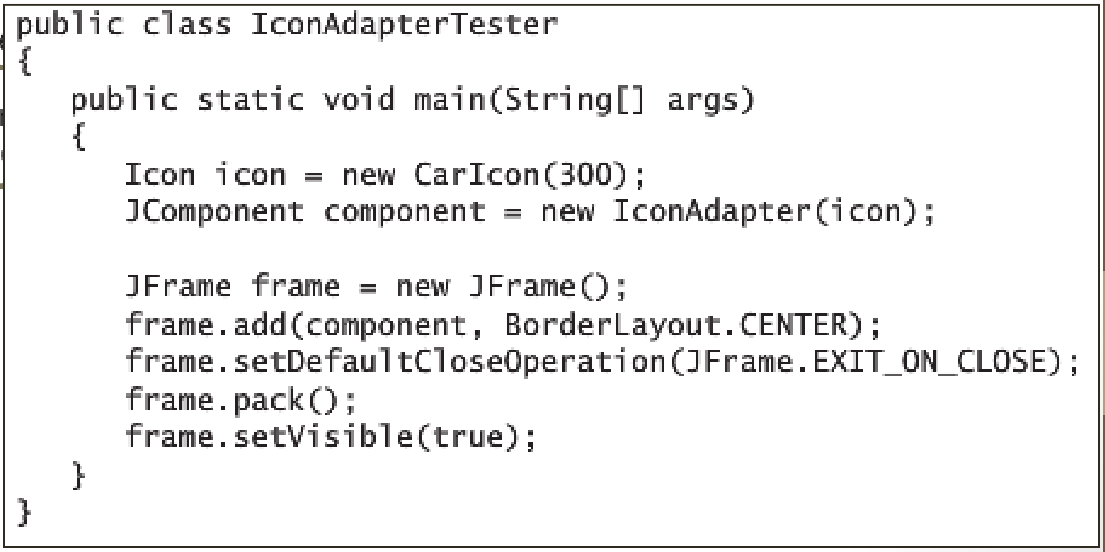
----

void paintIcon(Component c,
Graphics g, int x, int y);

Adaptor Pattern



```
public class IconAdapter extends JComponent
{
   /**
      Constructs a JComponent that displays a given icon.
      @param icon the icon to display
   */
   public IconAdapter(Icon icon)
   {
      this.icon = icon;
   }

   public void paintComponent(Graphics g)
   {
      icon.paintIcon(this, g, 0, 0);
   }

   public Dimension getPreferr
   {
      return new Dimension(icon
            icon.getIconHeight(
   }

   private Icon icon;
}
```



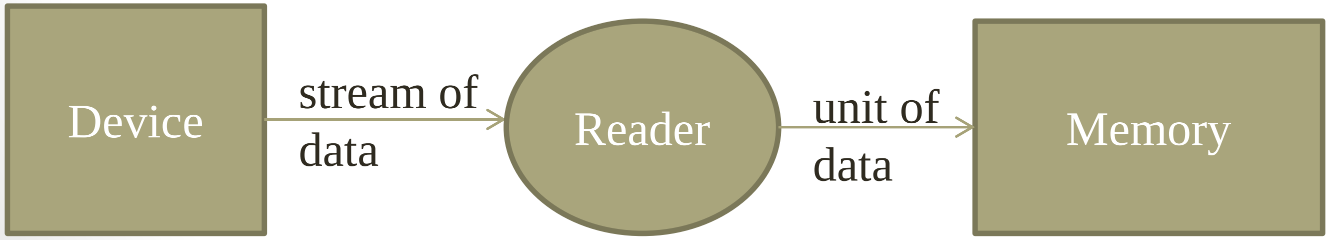
```
public class IconAdapterTester
{
   public static void main(String[] args)
   {
      Icon icon = new CarIcon(300);
      JComponent component = new IconAdapter(icon);

      JFrame frame = new JFrame();
      frame.add(component, BorderLayout.CENTER);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.pack();
      frame.setVisible(true);
   }
}
```

Adaptor Pattern

Stream Reader Case Study

```
Reader reader = new InputStreamReader(System.in);
   // Uses the default character encoding
```

```
Reader reader = new InputStreamReader(System.in, "UTF-8");
   // Uses the specified character encoding
```



Δt

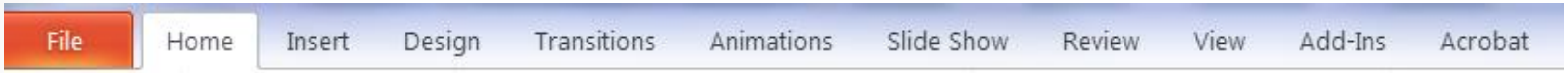
Does the stream type agree with the memory type?

Design Patterns 4

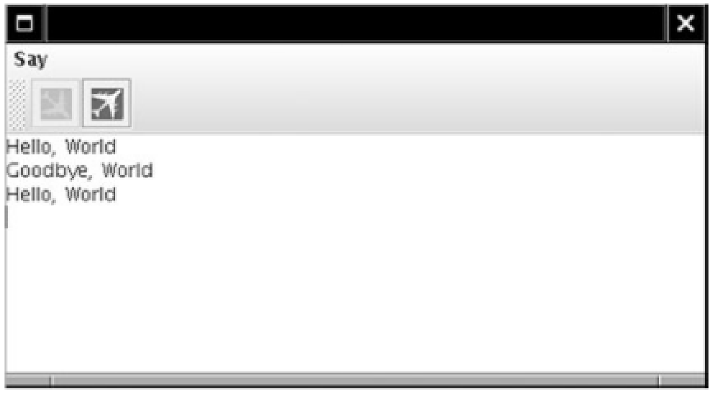
COMMAND PATTERN

Command Pattern

Traditional Commands



Click once and it happens… not a pattern.

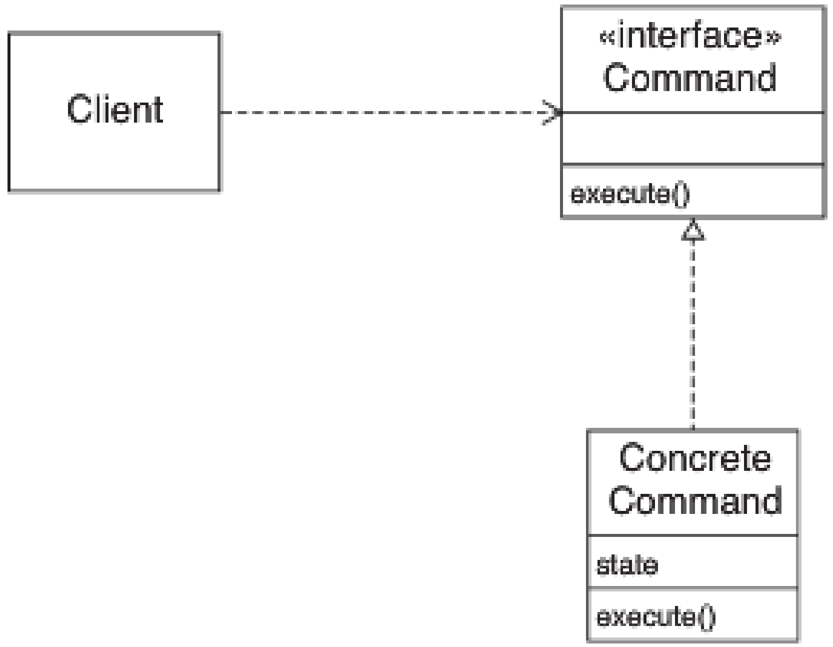


```
Hello, World
Goodbye, World
Hello, World
```

Command with sate.

Collect multiple commands and then execute as a group.

Command Pattern

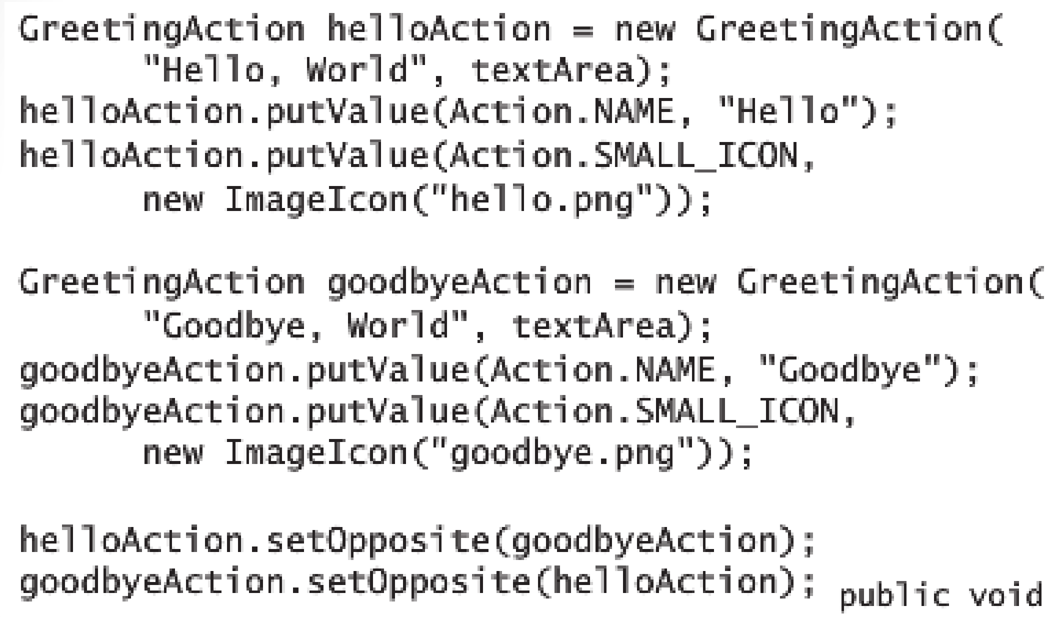


Eg: remembering that a button was pressed

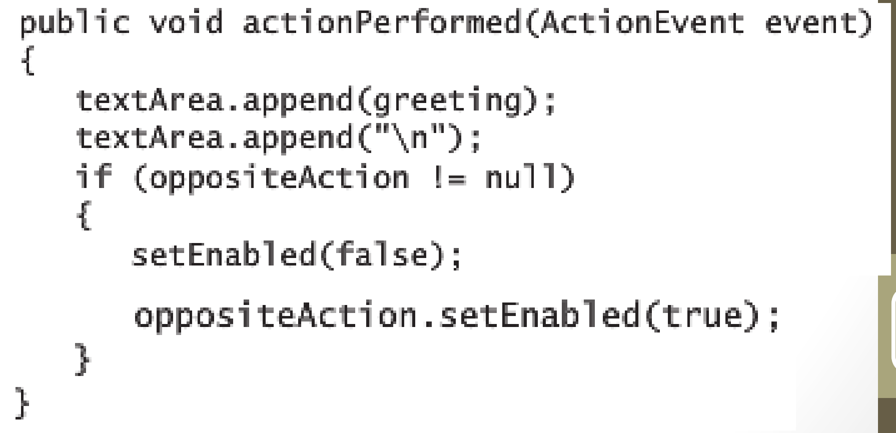
You want to implement commands that behave like objects, either because you need to store additional information with commands, or because you want to collect commands.

Command Pattern

ICON Collector Case Study



```
GreetingAction helloAction = new GreetingAction(
      "Hello, World", textArea);
helloAction.putValue(Action.NAME, "Hello");
helloAction.putValue(Action.SMALL_ICON,
      new ImageIcon("hello.png"));

GreetingAction goodbyeAction = new GreetingAction(
      "Goodbye, World", textArea);
goodbyeAction.putValue(Action.NAME, "Goodbye");
goodbyeAction.putValue(Action.SMALL_ICON,
      new ImageIcon("goodbye.png"));

helloAction.setOpposite(goodbyeAction);
goodbyeAction.setOpposite(helloAction);
```



```
public void actionPerformed(ActionEvent event)
{
   textArea.append(greeting);
   textArea.append("\n");
   if (oppositeAction != null)
   {
      setEnabled(false);

      oppositeAction.setEnabled(true);
   }
}
```