Commands

A *command* is something you want someone else to perform.

Instead of *directly* issuing a command, we can write it down and give it to someone to perform.

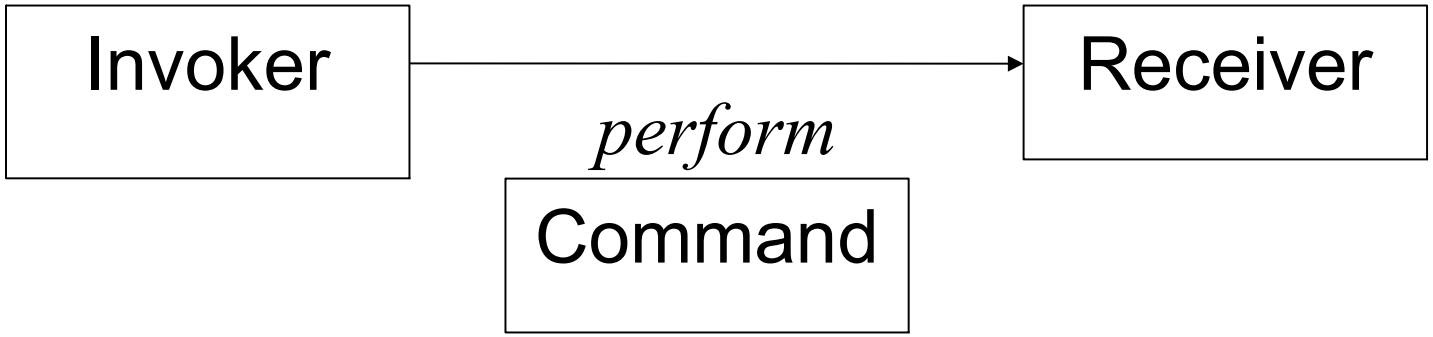
Written commands can be reused, cataloged, etc.

Sometimes they also contain "undo" commands, too.

Command Objects

In Java, we *encapsulate* a command in an object.

The command contains a method we want another object (the receiver) to perform.



Swing Command Objects

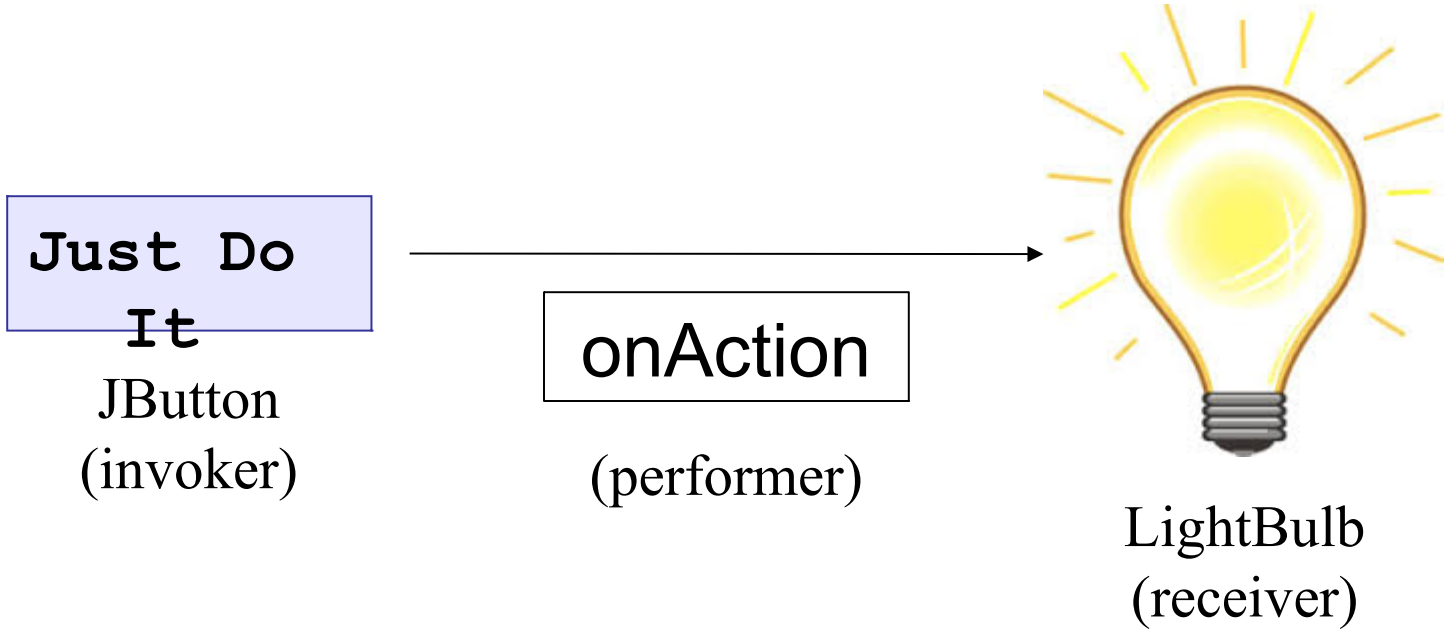
A class that implements ActionListener is a command.

```
JButton button1 = new JButton( "Just Do It" );
button1.setActionListener( new OnAction() );


// an inner class for invoking action
class OnAction implements ActionListener {
    public void actionPerformed(ActionEvent evt) {
        lightbulb.turnOn( );
    }
}
```

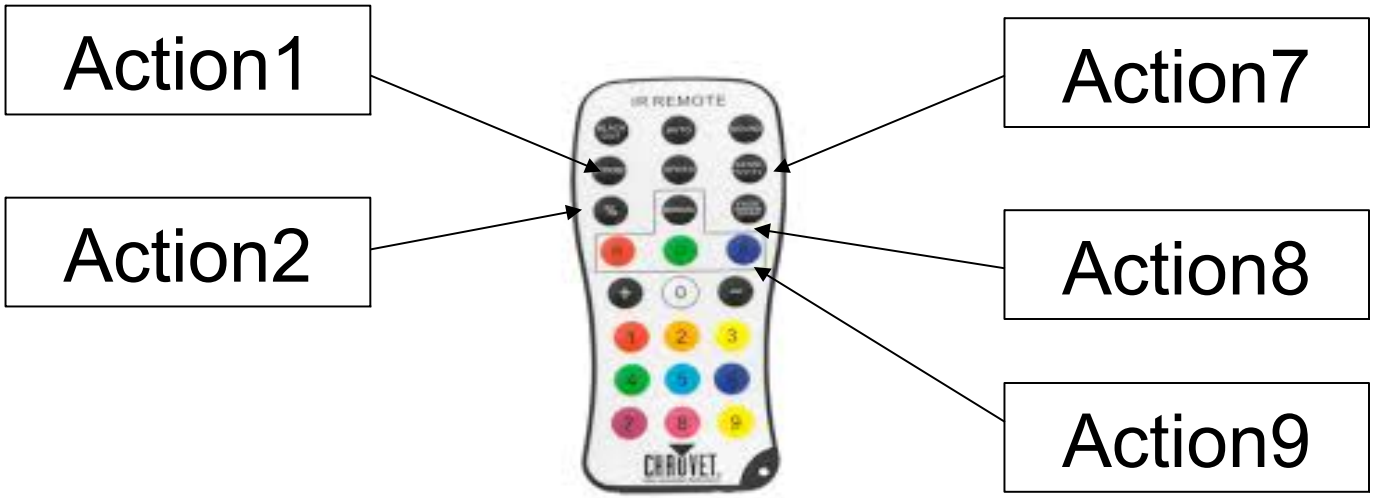
Programmable Invoker

The invoker (JButton) does not need to know what command it is invoking, or what object will perform it.



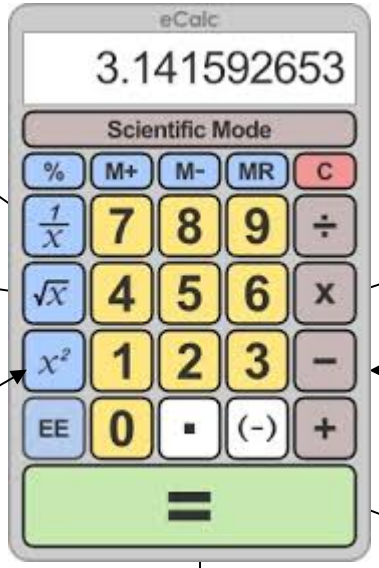
Programmable Interfaces

Use commands to "program" the user interface, instead of embedding logic haphazardly in components.



A programmable remote control

Calculator Application



UI (invoker)

InvertCommand

SqrtCommand

PowerCommand

MultiplyCommand

SubtractCommand

EqualsCommand

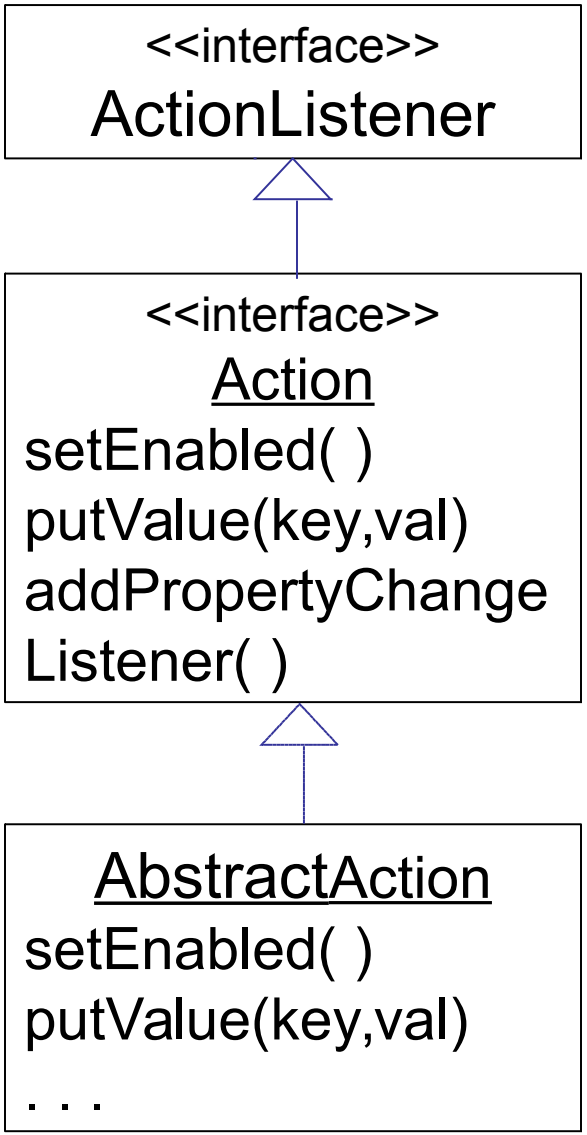
Calculator *(receiver)*

setOperand(num)
setOperator(op)
perform( )
getResult( )

Swing Action

Swing *Action* interface ...

- is ActionListener
- has name, icon, state
- contains a map for property values
- can be enabled and disabled



Using Swing Action

```
Action on = new OnAction( );
JButton button1 = new JButton( on );
panel.add( button1 );

// to disable the JButton, write:
on.setEnabled( false );

/** Define an OnAction for lightbulb. */
class OnAction extends AbstractAction {
    public OnAction( ) {
        super("Turn On");
    }
    public void actionPerformed(ActionEvent evt) {
        receiver.setOn(true);
    }
}
```

Actions

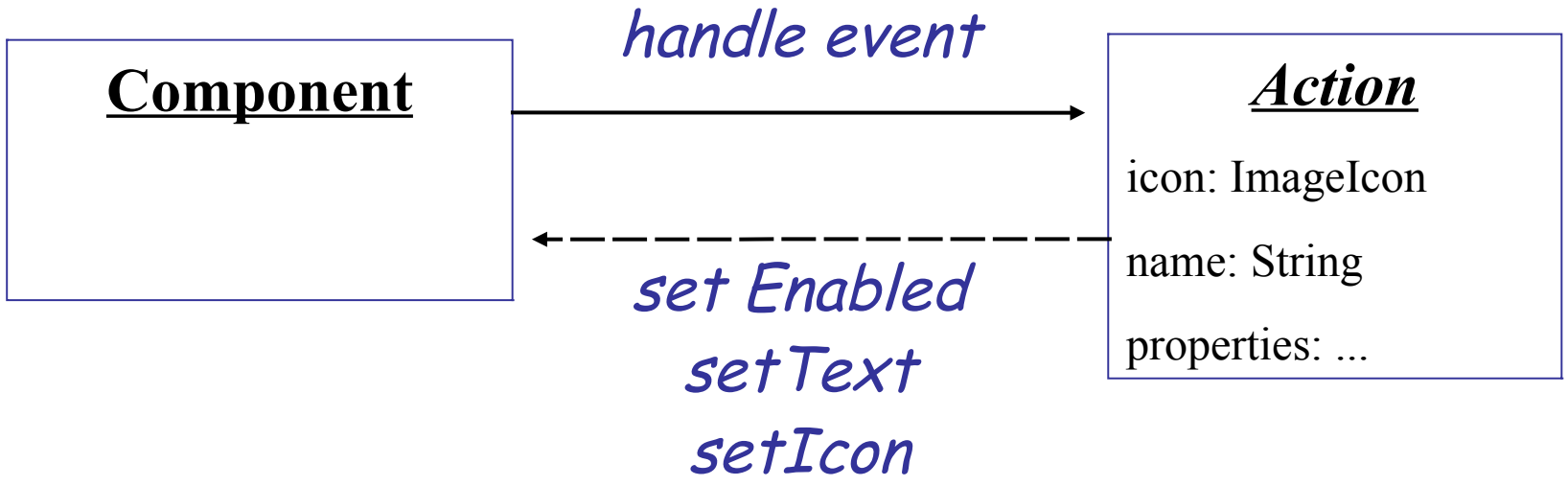
An Action is an object containing an ActionListener plus *state* information.

- Action contains a set of key-value properties.
- Action has a name.
- Action can be enabled and disabled. Useful for menus & buttons.
- Action can be assigned to a component.
  - it can also be *reused* -- assign to many components
- Action makes it easy to control components.

Actions

Encapsulate behavior in an object.

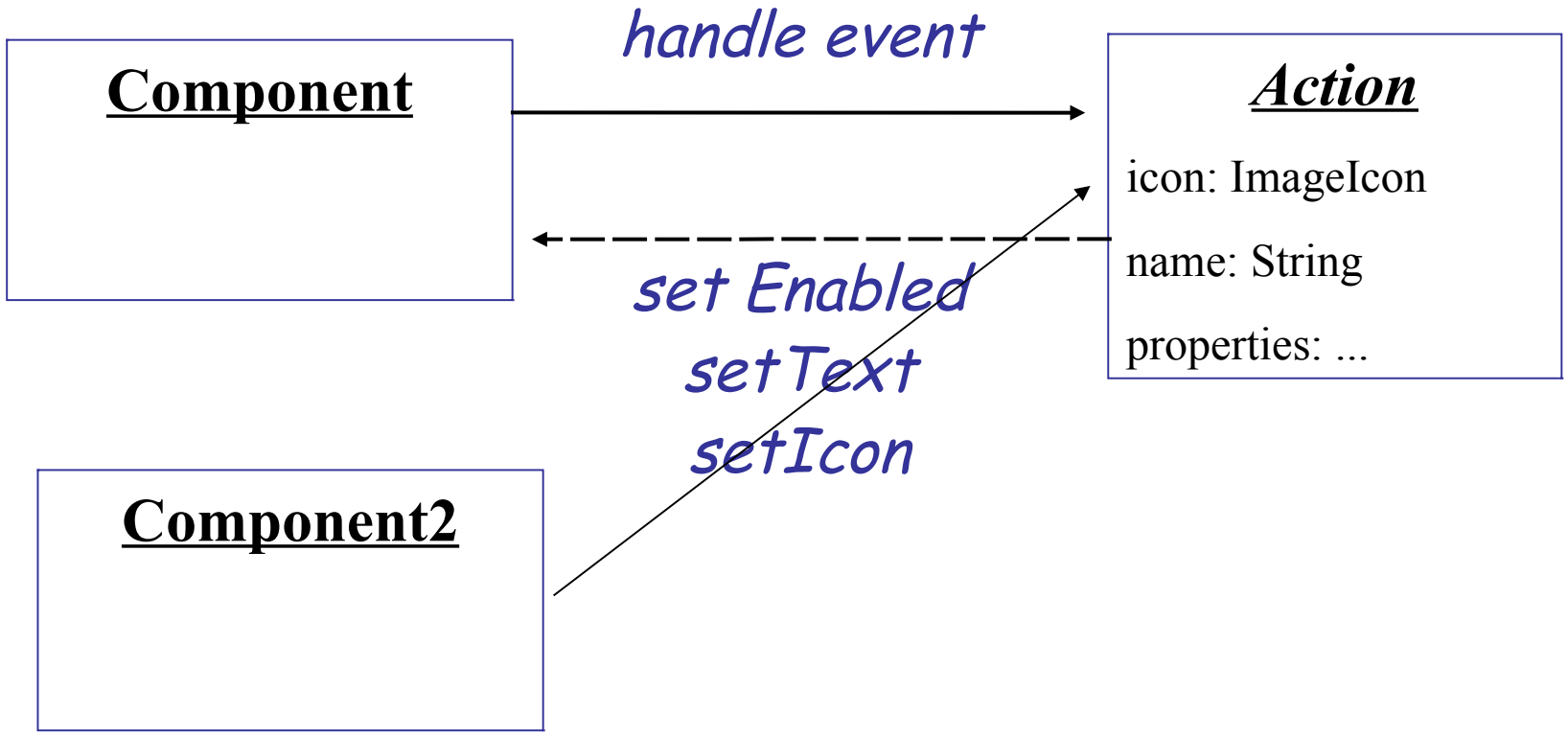
Encapsulate properties.



Action Can Control Many Component

Encapsulate behavior in an object.

Encapsulate properties.

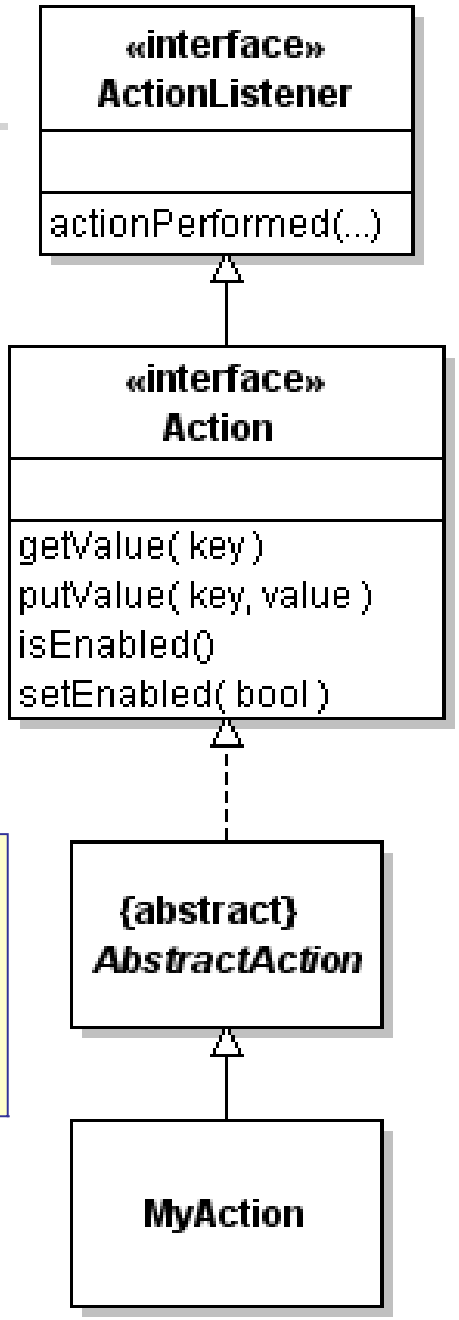


How to Use Action

Your class extends AbstractAction.

- Provide a name for the action.
- implement actionPerformed( ).
- associate Action with a component.
- if you disable the Action, the component is disabled.

```
button.setAction( myAction )
// or
button = new JButton( myAction );
```



Action Example

- Create an action for the "login" button.
- When user logs in, greet him then disable login button.

```
class LoginAction extends AbstractAction {
   public LoginAction( String name ) {
      super(name);
   }
   public void actionPerformed( ActionEvent e ) {
      String name = inputField.getText().trim();
      if (name.length() == 0) return;
      JOptionPane.showMessageDialog(frame,
                  "Hello "+name);
      setEnabled(false); // disable component
      inputField.setText("");
   }
}
```

Action Example (2)

- **`LoginAction`** must be an *inner class* so it can access the input field of GUI.
- Attach LoginAction to the JButton when button is created.
- **Delete** the "button.addActionListenter(...)"

```
public class SwingExample {
   private JFrame frame;
    private JButton button;
   ...
   private void initComponents {
      Action loginAction = new LoginAction("Login");
     button = new JButton( loginAction );
     ...
```

You don't need "addActionListener" now.

Putting Actions to Work

- Actions are useful for *encapsulating logic and behavior*
- We can *share* and *reuse* Actions.

Example: define a LoginAction and LogoutAction.

Login Action - disable input field and change button to "logout".

Logout Action - enable input field and change button to "login".

Login Action

```
class LoginAction extends AbstractAction {
   public LoginAction( String name ) {
      super(name);
   }
   public void actionPerformed( ActionEvent e ) {
      String name = inputField.getText().trim();
      if (name.length() == 0) return;
      JOptionPane.showMessageDialog(
       frame,
                    "Hello "+name);
      // change button to "logout"
  button.setAction( logoutAction );
      inputField.setText("");
  // disable typing
  inputField.setEnabled(false);
   }
}
```

Logout Action

```
class LogoutAction extends AbstractAction {
   public LoginAction( String name ) {
      super(name);
   }
   public void actionPerformed( ActionEvent e ) {
  // display a message
  JOptionPane.showMessageDialog(
       frame,
                  "You are logged out");
      // change button to "login"
  button.setAction( loginAction );
      // enable typing
  inputField.setEnabled(true);
   }
}
```

Applying the Actions

- Create attributes for loginAction and logoutAction.
- Apply loginAction to JButton (initial state is to login).

```
public class SwingExample {
   private JFrame frame;
    private JButton button;
   private Action loginAction =
          new LoginAction("Login");
   private Action logoutAction =
          new LogoutAction("Logout");
   ...
   private void initComponents {
       button = new JButton( loginAction );
        ...
```

Reusing an Action

You can attach the same action to many components.

Example:

Make the login interface easier to use:

- Enable user to type his name and press ENTER.
- Doesn't need to click "Login" button.

Solution: Apply loginAction to the JTextField.

```
private void initComponents {
      button = new JButton( loginAction );
     input = new JTextField(12);
    input.setAction( loginAction );
```