CSC 133
Object-Oriented Computer Graphics Programming

Event II

Dr. Kin Chung Kwan

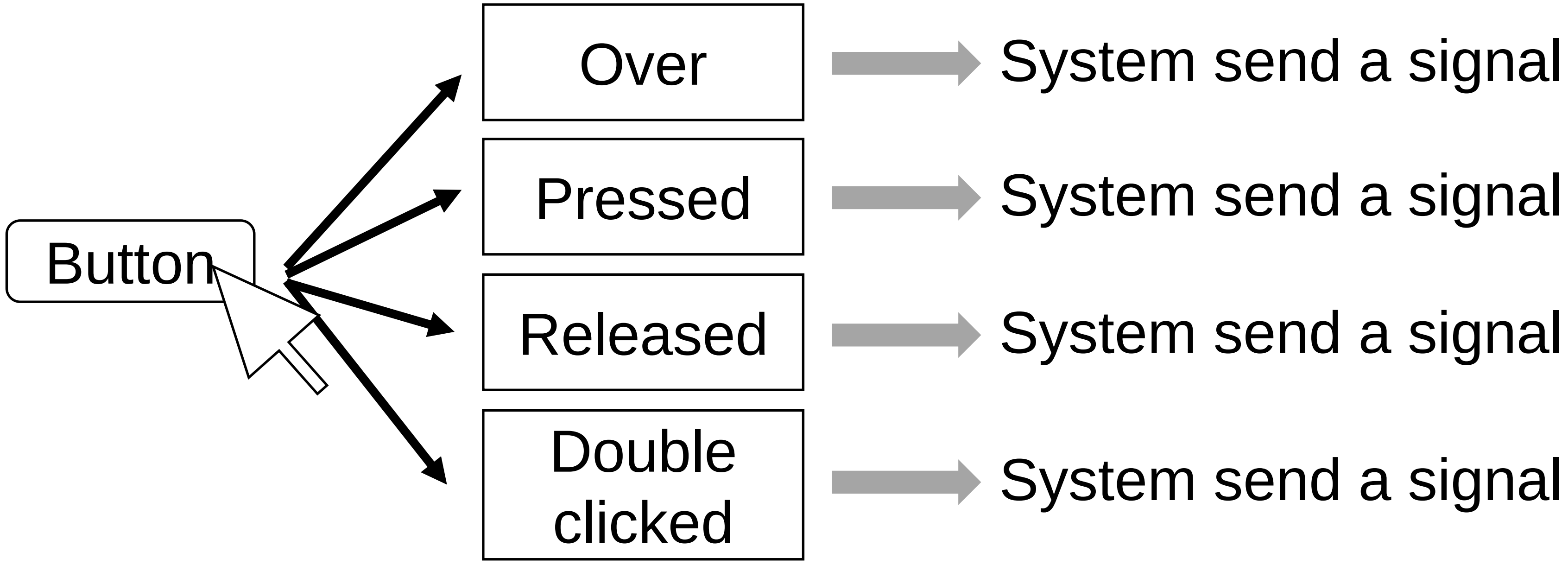
Spring 2023

Computer Science Department
California State University, Sacramento

SACRAMENTO STATE

Event

- When you did sometime on the UI, event happen.



ActionListener Interface

- Listeners must implement interface `ActionListener` (built-in in CN1)

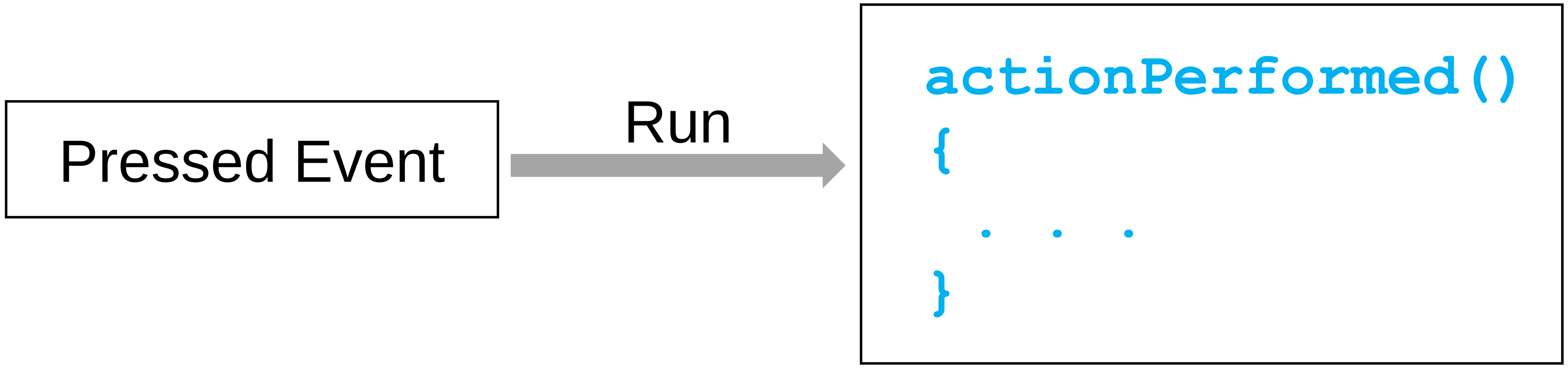
Implements this

```
interface ActionListener
{
   public void actionPerformed (ActionEvent e);
}
```

Provide a method

Event-Driven

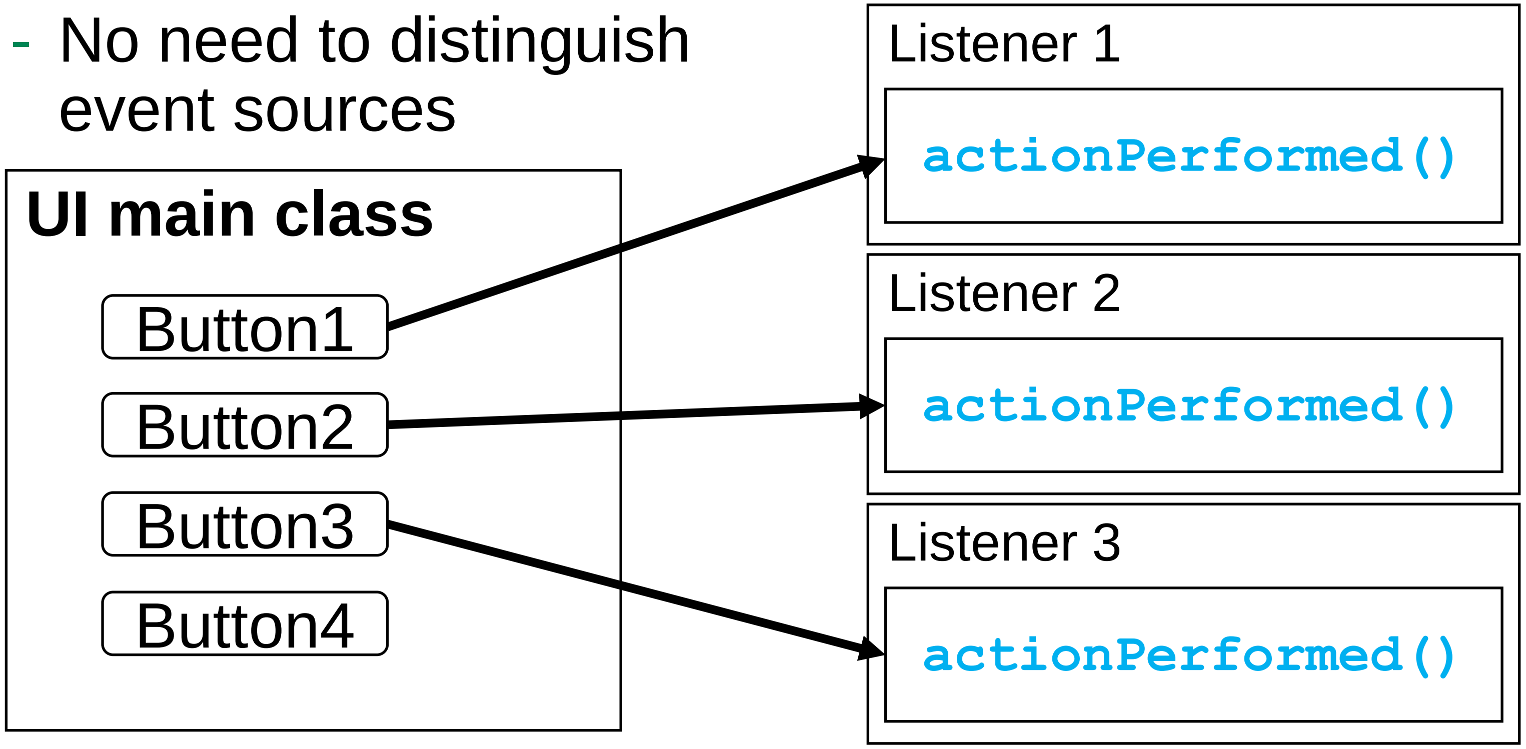
- Implement interface and provide
  - ActionPerformed() Callback function
- Run when the event happened



```
actionPerformed()
{
   . . .
}
```

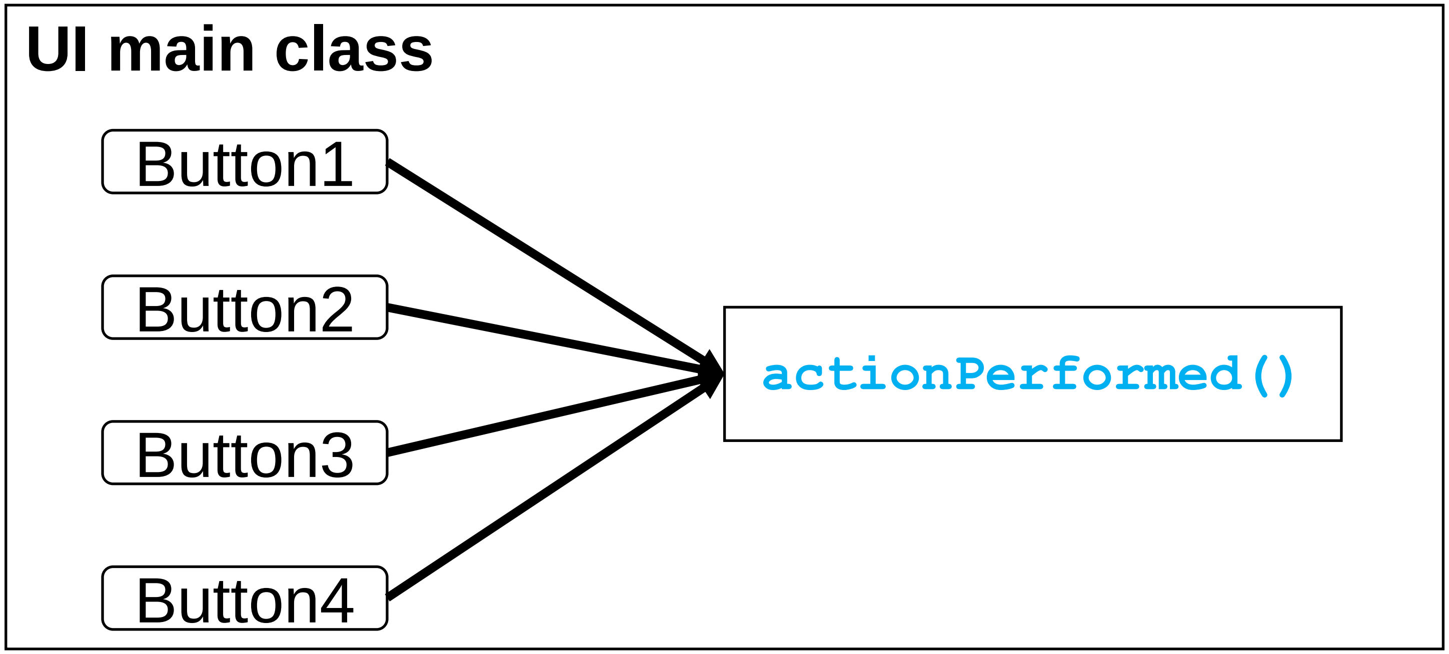
Approach (1a)

- Separated listener creates the components
  - One listener for one button
  - No need to distinguish event sources



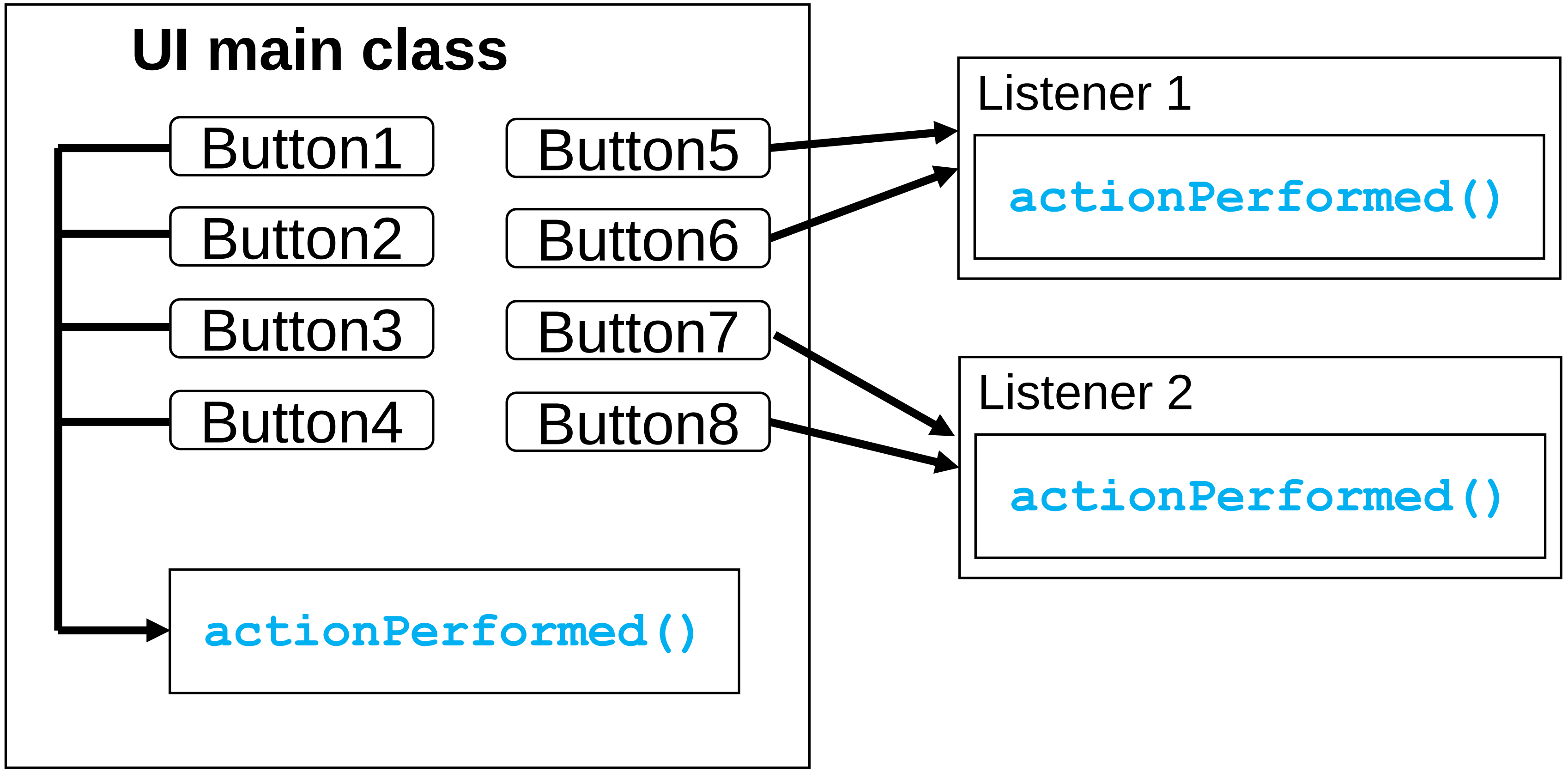
Approach (1b)

- Listener and Component in the same class
  - Need to distinguish event sources
  - Becomes bigger and bigger



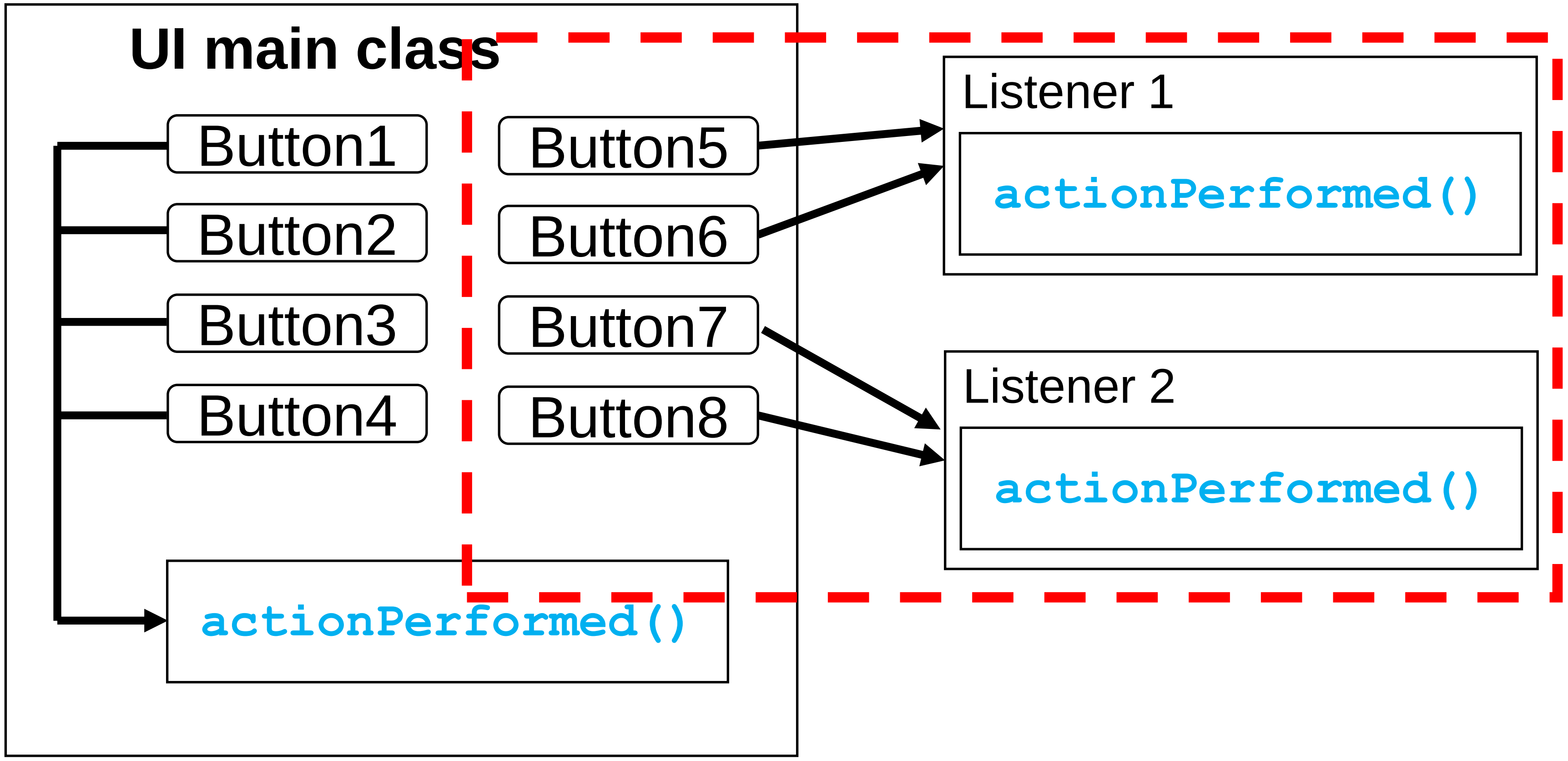
Better Approach

- Combined them together!



Problem

- How to distinguish the event source?



Approach (2)

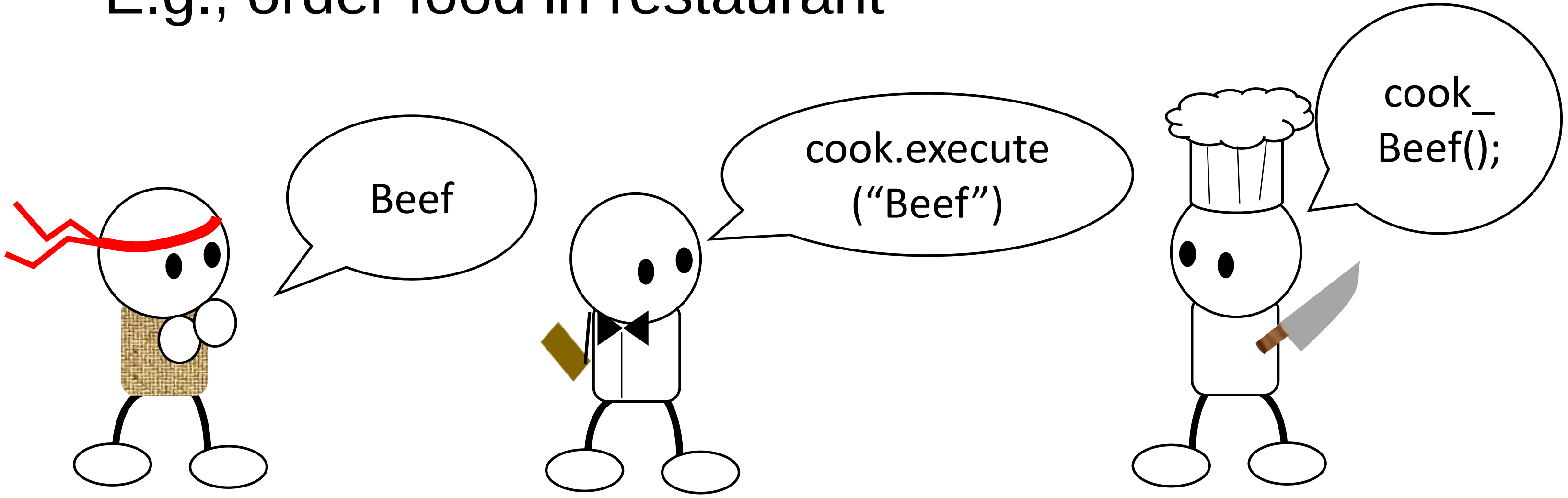
- Use single listener
  - for all related components

- Multiple listeners
  - for different groups of components

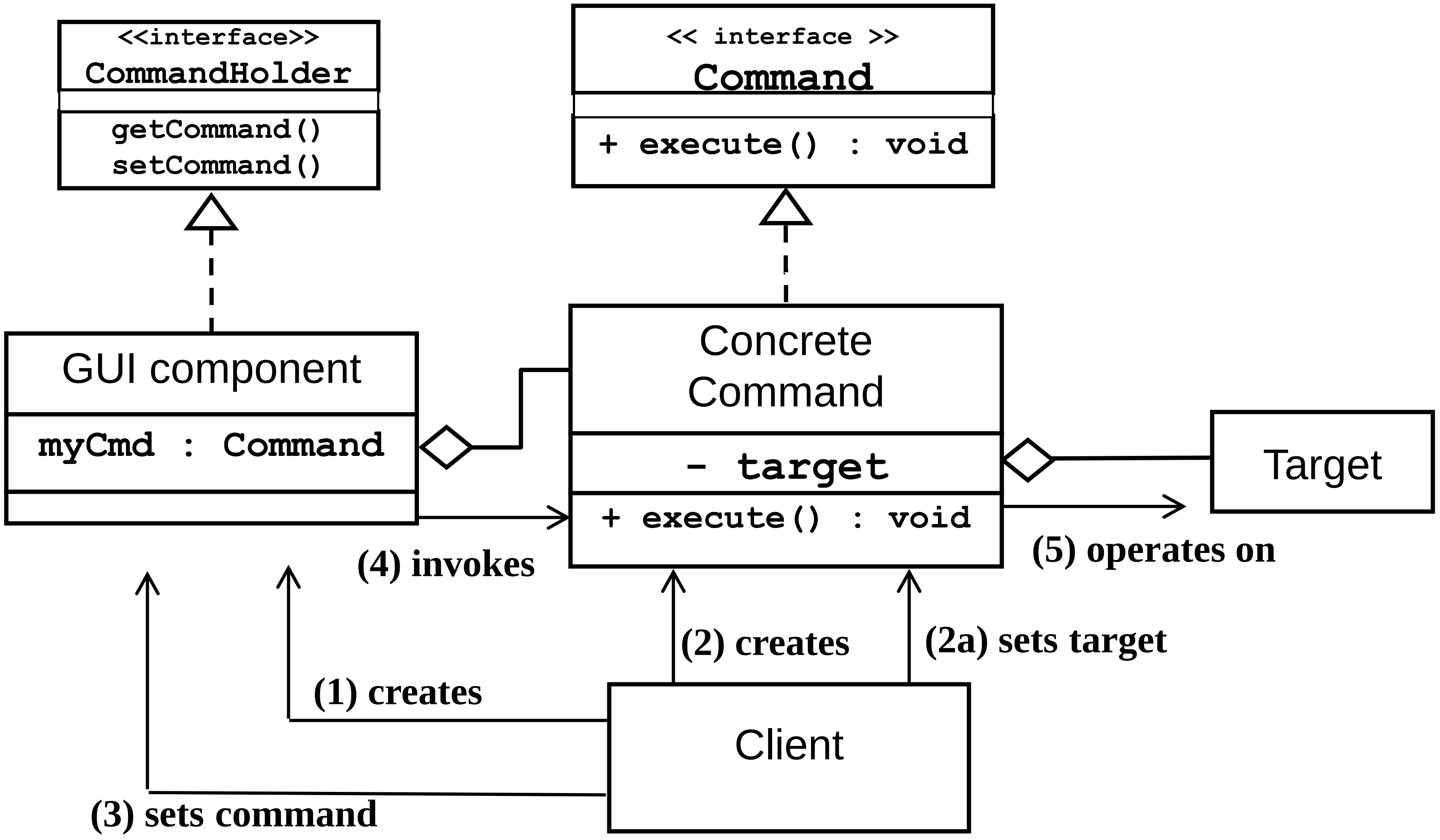
- But how separate listener distinguish event sources?
  - Command Pattern

Command Pattern

- Behavioral
- Set up a list of command for `execute()`, only receiver know how to do it.
    - E.g., order food in restaurant

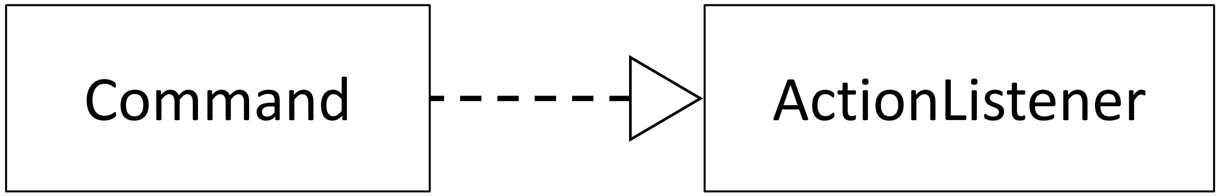


Command Pattern Organization



CN1 `Command` Class

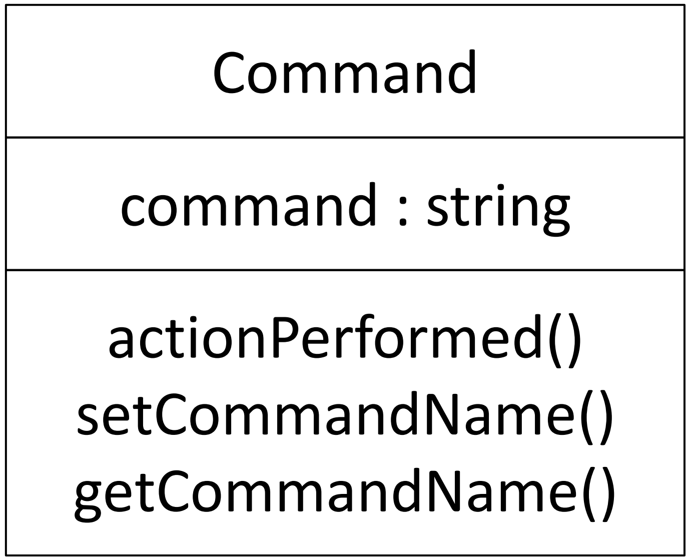
- A build-in class implements `ActionListener`
  - Provides empty body implementation for: `actionPerformed() == "execute()"`



- An actionListener with a string variable
  - Command: "save," "load," "play," etc.

How to Use?

- Build a new class
- Extend from `Command`
- Override `actionPerformed()`
  - To perform operations that we like to execute.
- In the constructor, do not forget to call `super("command name")`



Example (Listener 1)

```
import com.codename1.ui.Command;
import com.codename1.ui.events.ActionEvent;

public class SoundCommand extends Command {
  public SoundCommand(String command) {
    super(command);
  }
  public void actionPerformed(ActionEvent ev) {
    switch(getCommandName()){
      case "Cow": System.out.println("Moooooooo~"); break;
      case "Pig": System.out.println("Oinking~"); break;
      case "Chicken": System.out.println("Cluck~"); break;
      default: System.out.println("...");
    }
  }
}
```

CN1 `Button` Class

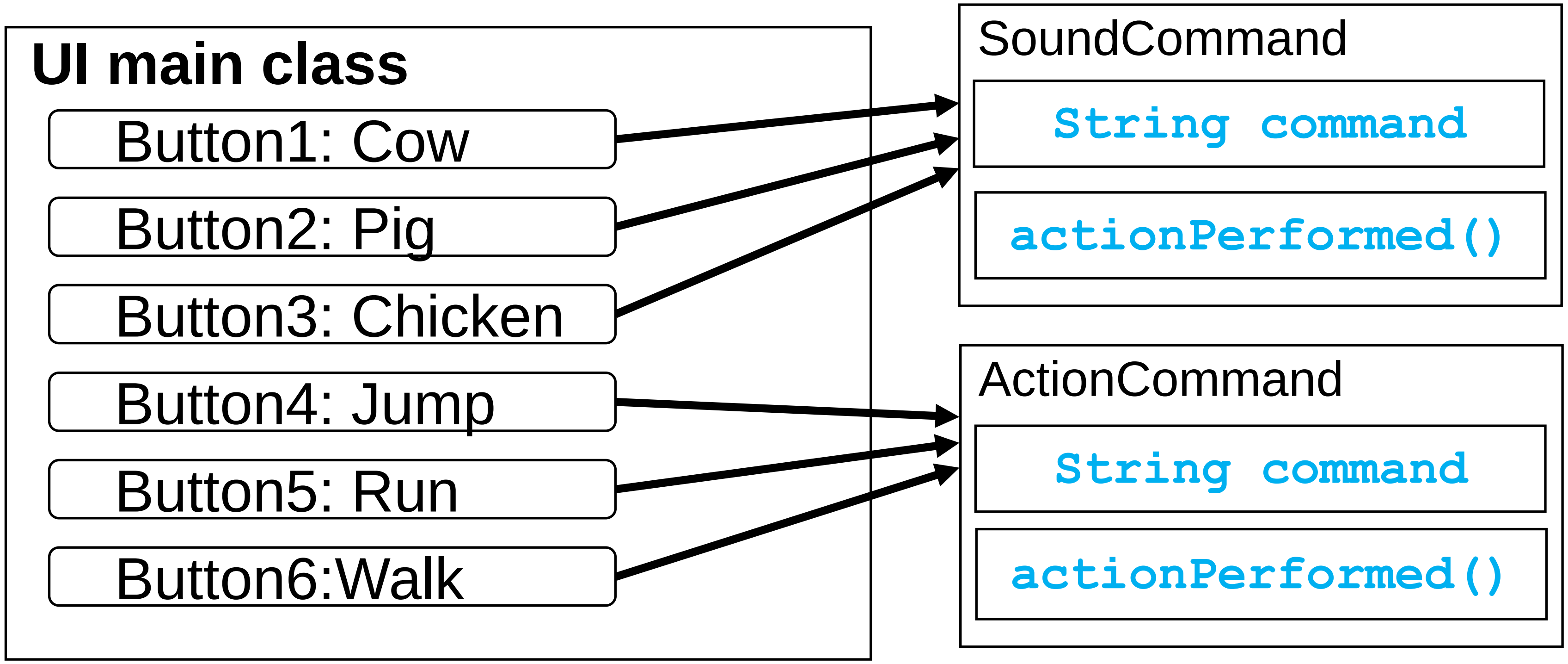
- **`Button`** is a "command holder"
  - with `setCommand()`, `getCommand()`

- When called `setCommand()`, `addActionListener()` is automatically added

- `setCommand()` changes the label of the button to the "command name" too.

Approach 2

- Multiple buttons using same listener w
  - with a string command to distinguish event source.



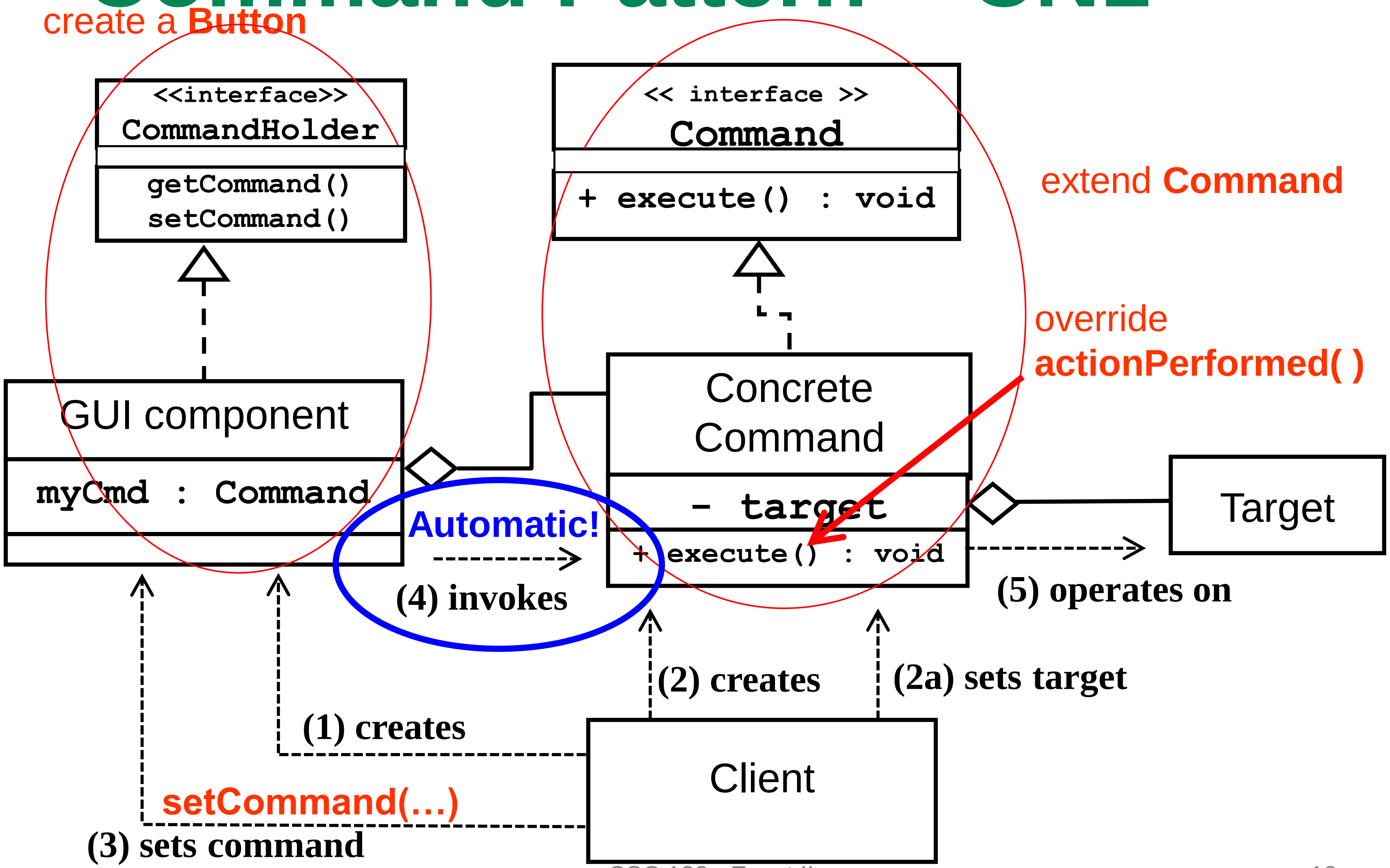
Example (Listener 2)

```
public class ActionCommand extends Command {
  public ActionCommand(String command) {
    super(command);
  }
  public void actionPerformed(ActionEvent ev) {
    switch(getCommandName()){
      case "Jump": System.out.println("Jump"); break;
      case "Run": System.out.println("Running~"); break;
      case "Walk": System.out.println("Walking"); break;
      default: System.out.println("...");
    }
  }
}
```

Example (Main)

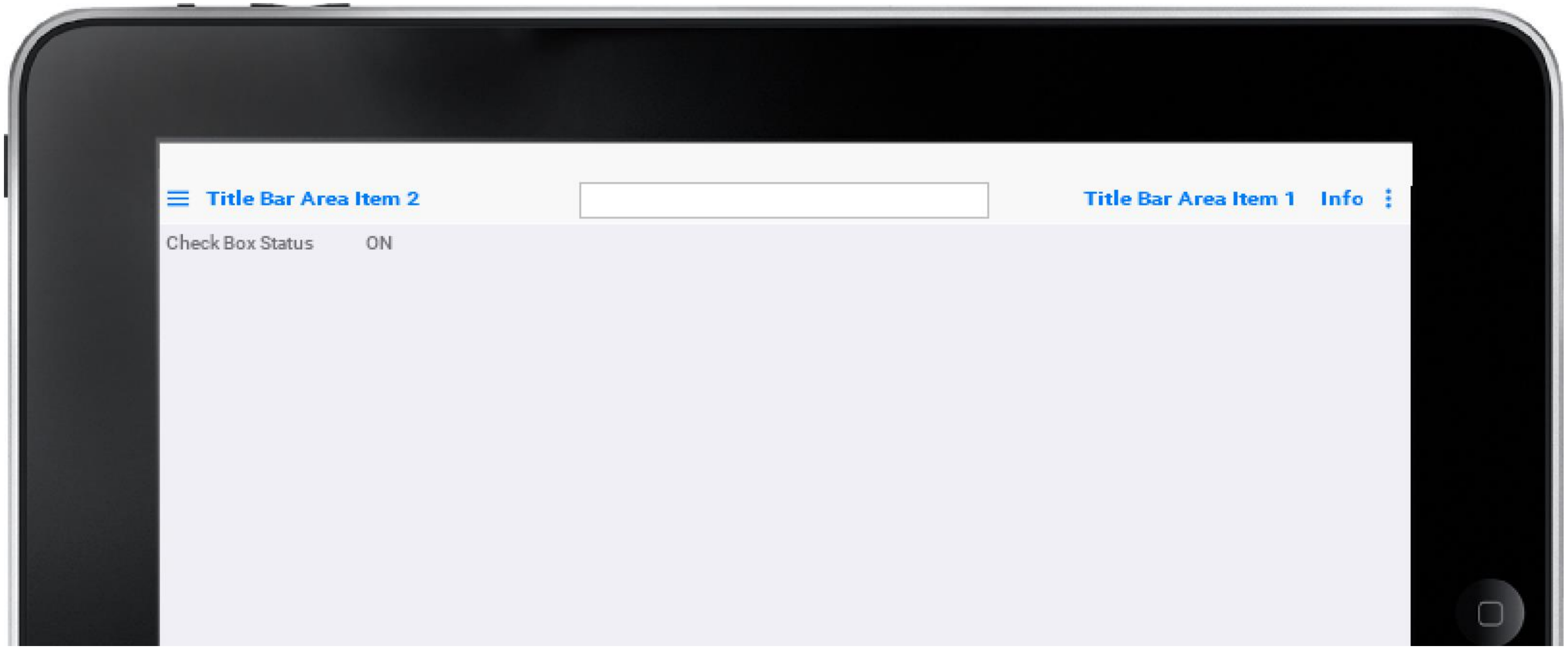
```
public MyForm() {
   Button b1 = new Button("Cow");
   Button b2 = new Button("Pig");
   Button b3 = new Button("Chicken");
   Button b21 = new Button("Jump");
   Button b22 = new Button("Run");
   Button b23 = new Button("Walk");
   b1.setCommand(new SoundCommand("Cow"));
   b2.setCommand(new SoundCommand("Pig"));
   b3.setCommand(new SoundCommand("Chicken"));
   b21.setCommand(new ActionCommand("Jump"));
   b22.setCommand(new ActionCommand("Run"));
   b23.setCommand(new ActionCommand("Walk"));
   add(b1).add(b2).add(b3).add(b21).add(b22).add(b23);
   show();  }
```

Command Pattern – CN1



How about Title Bar?

- We still have button there



Adding to Title Bar

- Using `Toolbar`'s `addCommandTo...()`

```
addCommandToSideMenu()
addCommandToOverflowMenu()
addCommandToRightBar()
addCommandToLeftBar()
```

- Automatically generated and added items to the title bar

- The command is the listener of that item

```
Command sideMenuItem1 = new Command("Side Menu Item 1");
myToolbar.addCommandToSideMenu(sideMenuItem1);
```

Problem

```
Command sideMenuItem1 = new Command("Side Menu Item 1");
myToolbar.addCommandToSideMenu(sideMenuItem1);
```

- Command is the listener

- Default `actionPerformed()` is **empty**!

Creating `Command` SubClass

Two options

1. Shortcut

2. Create separate class for different command group
   - Recommended!

Shortcut

- A temporary class implements the interface
  - With auto-generated class name
  - “new” directly

```
A a = new A(){
    public void go(){
        System.out.print(“A”);
    }
};
```

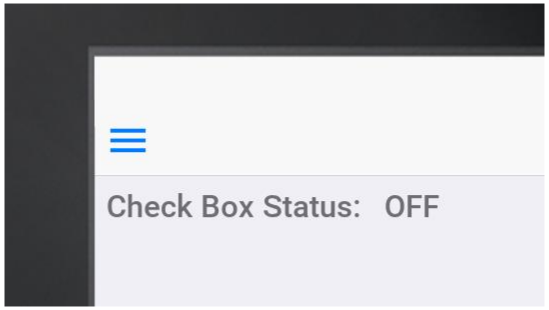
Shortcut for Command

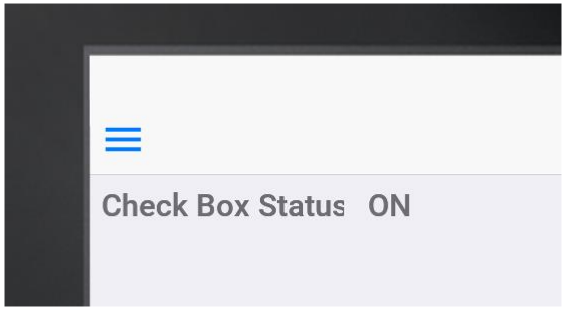
```
/* Code for a form that creates an object of anonymous sub-class
of the Command */
//create a Toolbar called myToolBar and add it to the form
//create the object (called inforTitleBarAreaItem) of anonymous
sub-class of Command
Command infoTitleBarAreaItem = new Command("Info") {
  public void actionPerformed(ActionEvent ev) {
    String Message = "I provide information.";
    Dialog.show("Info", Message, "Ok", null);
  }
};
myToolbar.addCommandToRightBar(infoTitleBarAreaItem);
```

Do **NOT** use shortcut in assignments!

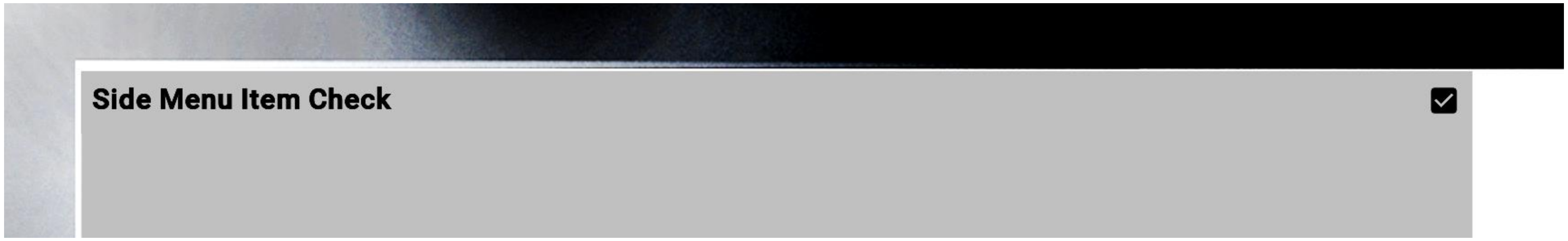
Example

- Side Menu


Check Box Status: OFF


Check Box Status ON

By


Side Menu Item Check ✓

Command for Side Menu

```
public class SideMenuItemCheck extends Command {
  private SideMenuItemCheckForm myForm;
  public SideMenuItemCheck (SideMenuItemCheckForm
  fForm){
    super("Side Menu Item Check");
    //do not forget to set the “command name”
    myForm = fForm;
  }
  public void actionPerformed(ActionEvent evt){
    myForm.setCheckStatusVal(
        ((CheckBox)evt.getComponent()).isSelected());
    myForm.closeSideMenu();
  }
}
```

Adding to Side Menu

```
public class SideMenuItemCheckForm extends Form {
  private Label checkStatusVal = new Label("OFF");
  private Toolbar myToolbar = new Toolbar();
  public SideMenuItemCheckForm() {
      setToolbar(myToolbar);
      CheckBox checkSideMenuComponent = new CheckBox("Side Menu
      Item Check");
      Command mySideMenuItemCheck = new SideMenuItemCheck(this);
      checkSideMenuComponent.setCommand(mySideMenuItemCheck);
      myToolbar.addComponentToSideMenu(checkSideMenuComponent);
      Label checkStatusText = new Label("Check Box Status:");
      add(checkStatusText).add(checkStatusVal);    show();
  }
  public void setCheckStatusVal(boolean bVal){
     checkStatusVal.setText( bVal? "ON" : "OFF");
  }
  public void closeSideMenu() { myToolbar.closeSideMenu(); }
}
```

Pointer Handling

- Components also generate an `ActionEvent` when there is pointer (mouse / touch) event.

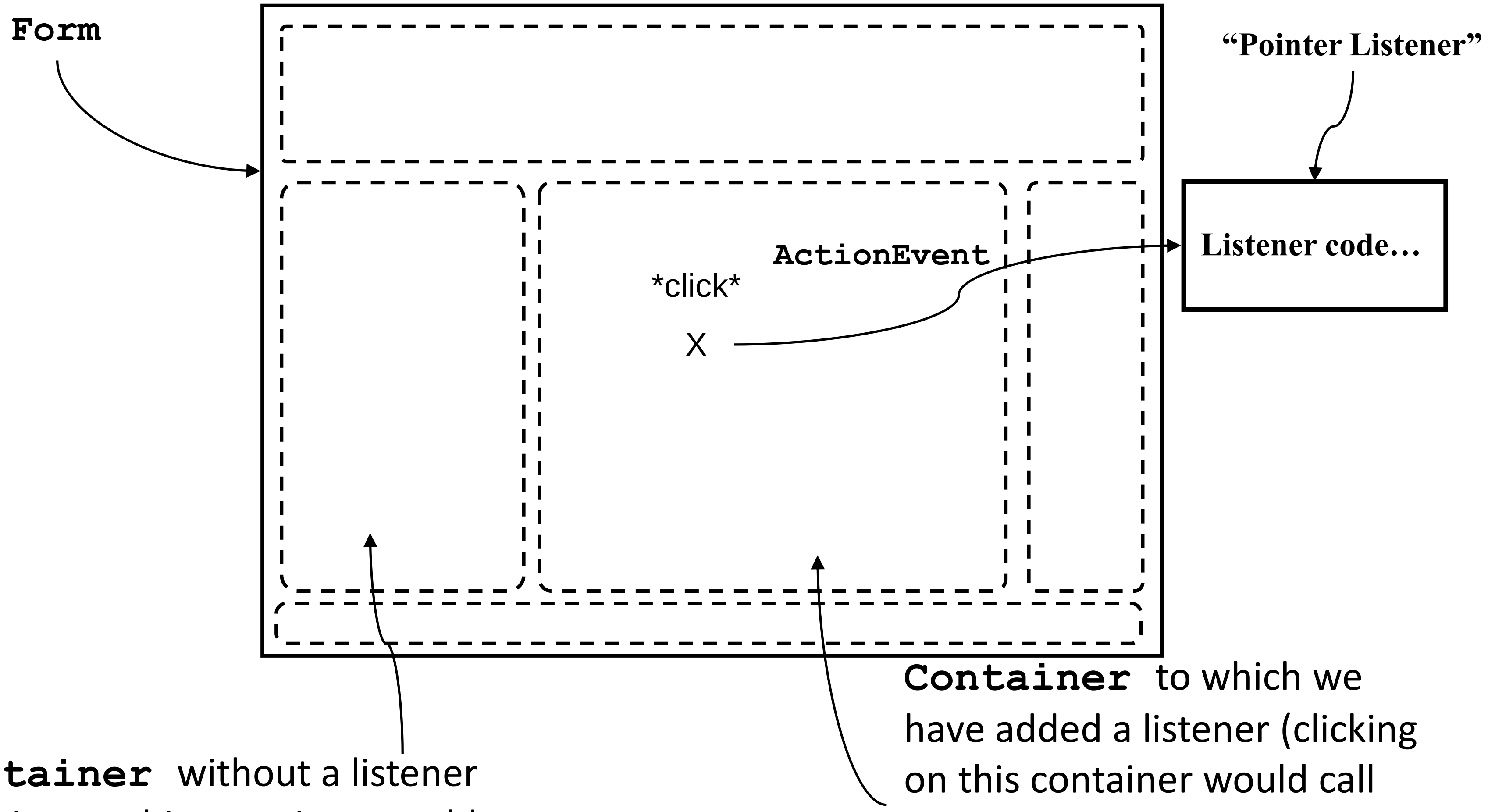
- `Component` class provides:

```
addPointerPressedListener( )
addPointerReleasedListener( )
addPointerDraggedListener( )
```

which take a parameter of `ActionListener`.

- Or attach a **`Command`** for Command Design Pattern!

Pointer Handling Area



`Container` without a listener (clicking on this container would not invoke any action/method)

`Container` to which we have added a listener (clicking on this container would call **`actionPerformed()`** of the attached listener)

ActionListener

- It creates actionEvent
  - Need to implement **`ActionListener`** interface:

```
interface ActionListener{
   public void actionPerformed (ActionEvent e);
}
```

- **`ActionEvent`** in **`actionPerformed()`** method has **`getX()`** and **`getY()`** methods
  - returns “screen coordinate” the pointer location.

Pointer Listener Example

```
public class PointerListenerForm extends Form{
  public PointerListenerForm() {
    Container myContainer = new Container();
    PointerListener myPointerListener = new PointerListener ();
    myContainer.addPointerPressedListener(myPointerListener);
    ...
  }
}
public class PointerListener implements ActionListener {
  public void actionPerformed(ActionEvent evt) {
      System.out.println("evt.getX() + " " + evt.getY());
  }
}
```

Question

What happens if I add the listener to the form instead of the container in the form?

```
public class PointerListenerForm extends Form{
  public PointerListenerForm() {
    PointerListener myPointerListener = new PointerListener();
    this.addPointerPressedListener(myPointerListener);
   //…[add containers and components to the form]
 }
}
```

Answer:

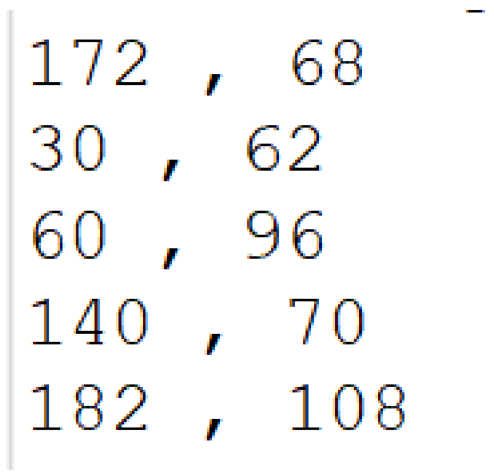
Clicking anywhere on the form (including the title bar area) would print out the values…

Question 2

- `ActionEvent` has `getX()` and `getY()`
- What will they return if the `actionEvent` is generated by a `button`?

Answer

- Return the pointer position



```
172 , 68
30 , 62
60 , 96
140 , 70
182 , 108
```

Listeners for Different Pointer Actions

- There are three approaches:
  1. Add a separate listener for them

```
myContainer.addPointerPressedListener(myPressedListener)
myContainer.addPointerReleasedListener(myReleasedListener)
myContainer.addPointerDraggedListener(myDraggedListener)
```

     This approach requires us to have three separate listener classes.

  2. Single listener for all and distinguish between different actions by using `ActionEvent`'s `getEventType()`.
     Need to have if-then-else structure

Overriding Pointer Methods

- `Component` class also has following methods:

    `pointerPressed()`

    `pointerReleased()`

    `pointerDragged()`

- If you are extending from a `Component`
    - override these functions.
    - Recommended approach: easier than adding a listener for each separate pointer action.

Overriding Pointer Methods

```
/* Center container of the form is a PointerContainer which
extends from Container */
public class PointerListenerForm extends Form{
  public PointerListenerForm() {
     PointerContainer myPointerContainer = new
        PointerContainer();
    this.add(BorderLayout.CENTER,myPointerContainer);
    ... }
}
------
/* We can override the pointer methods in the Container */
public class PointerContainer extends Container{
  public void pointerPressed(int x,int y){...}
  public void pointerReleased(int x,int y){...}
  public void pointerDragged(int x,int y){...}
}
```

ASSIGNMENT 2
DESIGN PATTERN & GUI

In Canvas

Mar 02, 2023

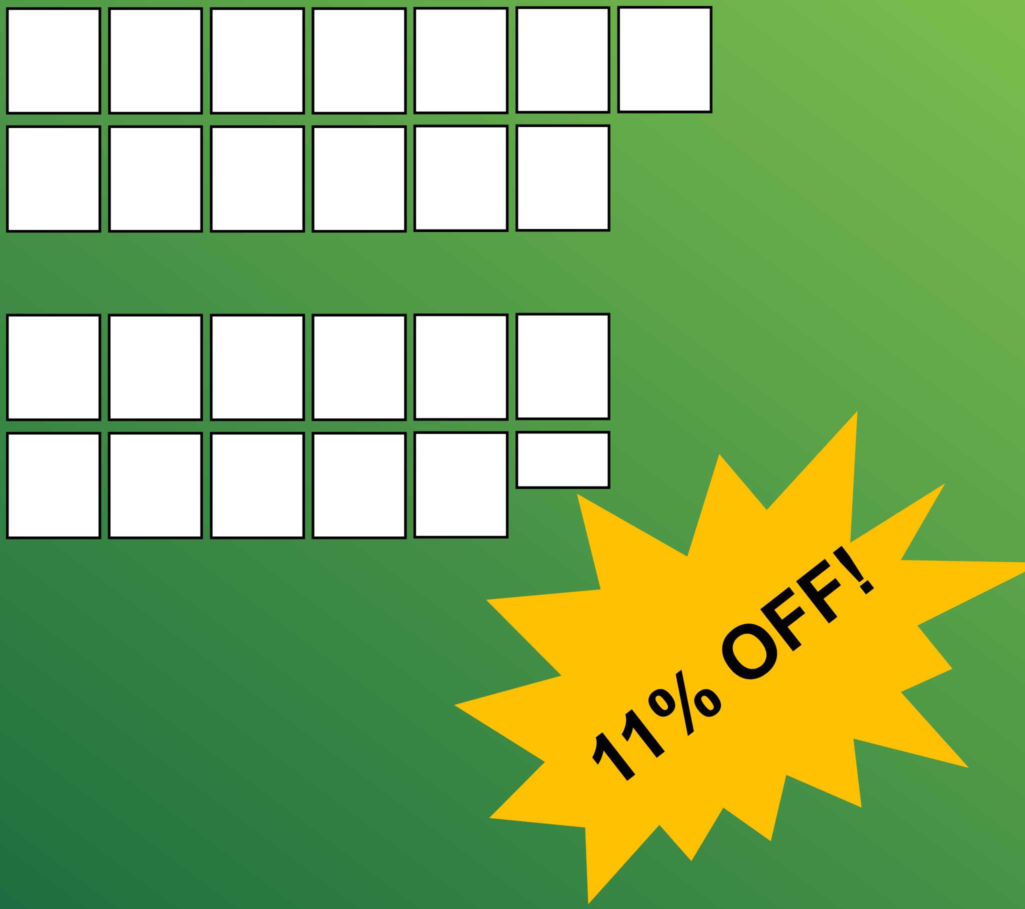
Available on the
Canvas

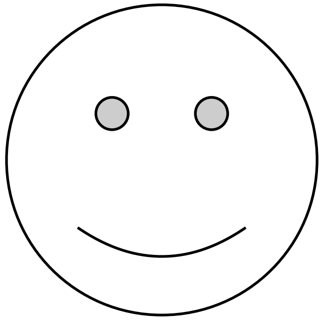
Assignment 1

13 pages

Assignment 2

11.5 pages





Happy?

Any Questions?