Programming Languages and Techniques (CIS1200)

Lecture 34

Swing III: MoD

Chapters 30

Announcements (1)

- HW9: Game Project
  - TAs have given you feedback
  - Final Program Due: Tuesday, April 29th at 11:59pm
  - Grade based on demo with your TA during/after reading days
  - *NO LATE SUBMISSIONS PERMITTED*

We are almost done

- **Today** (4/21): Swing III: Abstract Classes, Timer based games
- **Wednesday** (4/23): Advanced Java Topics
- Recitation: Final exam review
- **Friday** (4/25): *Bonus Lecture: CIS and Sustainability (Dr. Benjamin Pierce)*

- **Monday** (4/29): *Bonus Lecture: OCaml at Jane Street (Dr. Richard Eisenberg)*
- **Tuesday** (4/30): Game project due at midnight
- **Wednesday** (4/31): Semester recap
- No recitation this week

- **Wednesday** (5/7): Final exam, 9-11AM

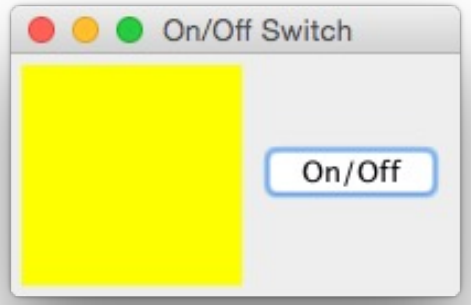
CIS 120 Final Exam

- Wednesday, May 7th 9-11AM
  - Location details coming soon
- The exam will be comprehensive, but:
  - material since Midterm 2 will be emphasized
  - "Bonus" lectures not included
  - we won't ask you to code in OCaml (but we might ask you to read it)
- Stay tuned for:
  - Recitation this week is exam review
  - Review session / Mock exam information

Recap: Anonymous Inner Classes, Lambdas

Anonymous Inner Classes

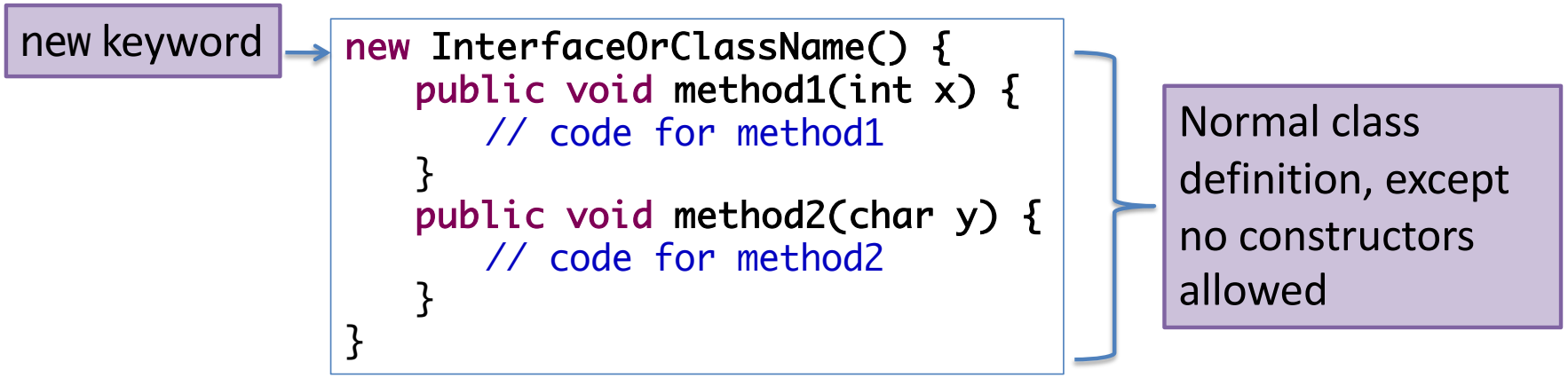
We can define a class and create an object from it *all at once* inside a method body




```
final LightBulb bulb = new LightBulb();
JButton button = new JButton("On/Off");

button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        bulb.flip();
        bulb.repaint();
    }
});
```

Anonymous Inner Classes

- New *expression* form: define a class and create an object from it all at once



Static type of the expression is the interface / superclass named after the new

Dynamic class of the created object is anonymous!
Can't refer to it.

Lambda Expressions

- Java includes *lambda expressions* that can implement classes that define only a single method


```
final LightBulb bulb = new LightBulb();
JButton button = new JButton("On/Off");

button.addActionListener((ActionEvent e) -> {
        bulb.flip();
        bulb.repaint();
    });
```

- Any interface with exactly one method is a *functional interface*
- Syntax:

```
x -> { body }                // type of x inferred
(T x) -> { body }            // arg x has type T
(T x, W y) -> { body } // multiple arguments
```

Lambdas In A Nutshell

Lambda Notation

```
x -> x + x
```

```
(x,y) -> x.m(y)
```

```
(x,y) -> {
  System.out.println(x);
  System.out.println(y);
}
```

Method names and types are inferred from the context.

"Ordinary" Java Notation

```
int method1(int x) {
  return x + x;
}
```

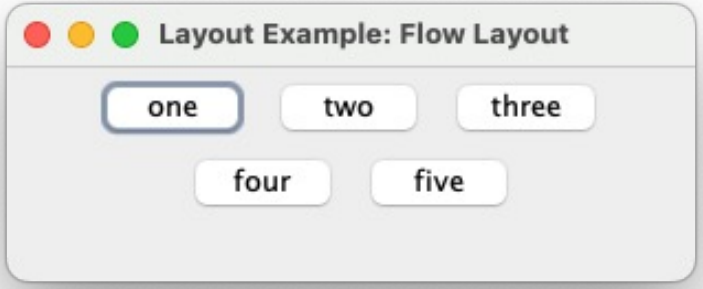
```
int method2(A x, B y) {
  return x.m(y);
}
```

```
void method3(String x,
             String y) {
  System.out.println(x);
  System.out.println(y);
}
```

Recap: Swing Layout Demo

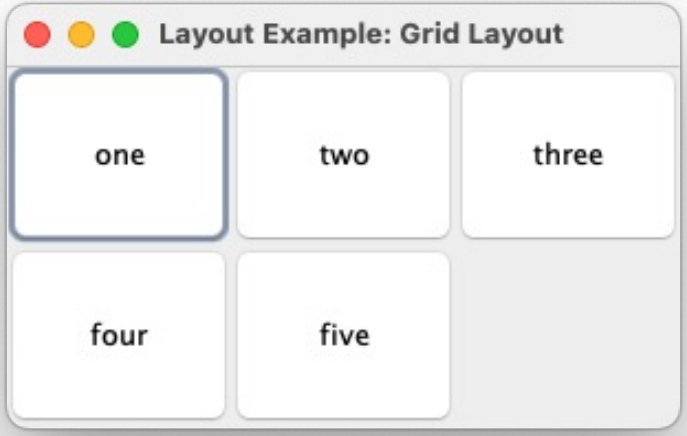
LayoutDemo.java

Layout Examples: Flow and Grid




```
JPanel panel = new JPanel();
panel.add(b1);
panel.add(b2);
panel.add(b3);
panel.add(b4);
panel.add(b5);
```

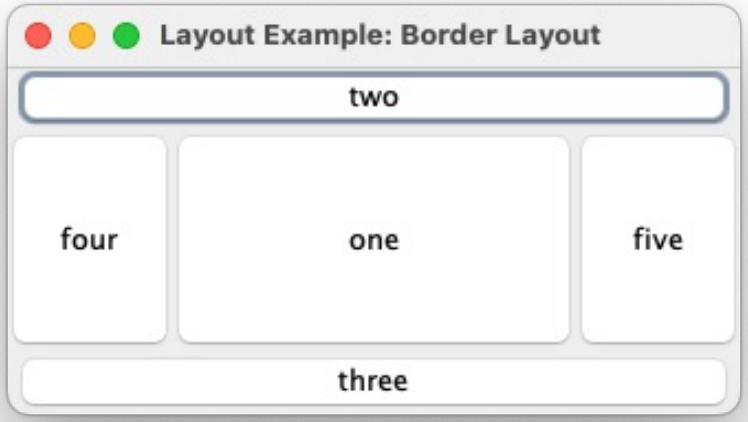
Default layout for JPanel; components "flow" like text, maintain preferred size




```
JPanel panel = new JPanel();
panel.setLayout(new GridLayout(2, 3));
panel.add(b1);
panel.add(b2);
panel.add(b3);
panel.add(b4);
panel.add(b5);
```

Constructor specifies number of rows and columns; components grow to fill grid

Layout Example: Border

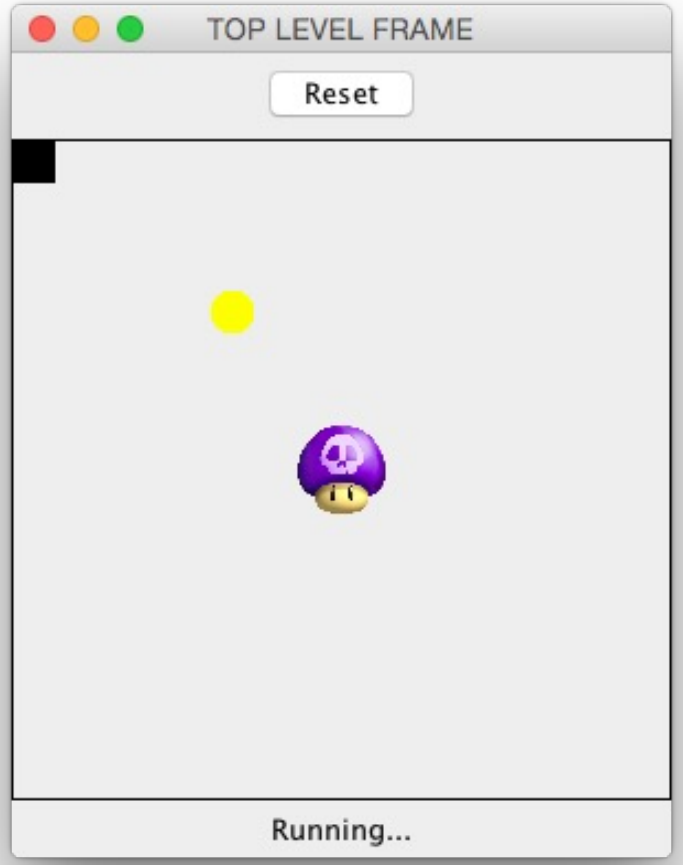


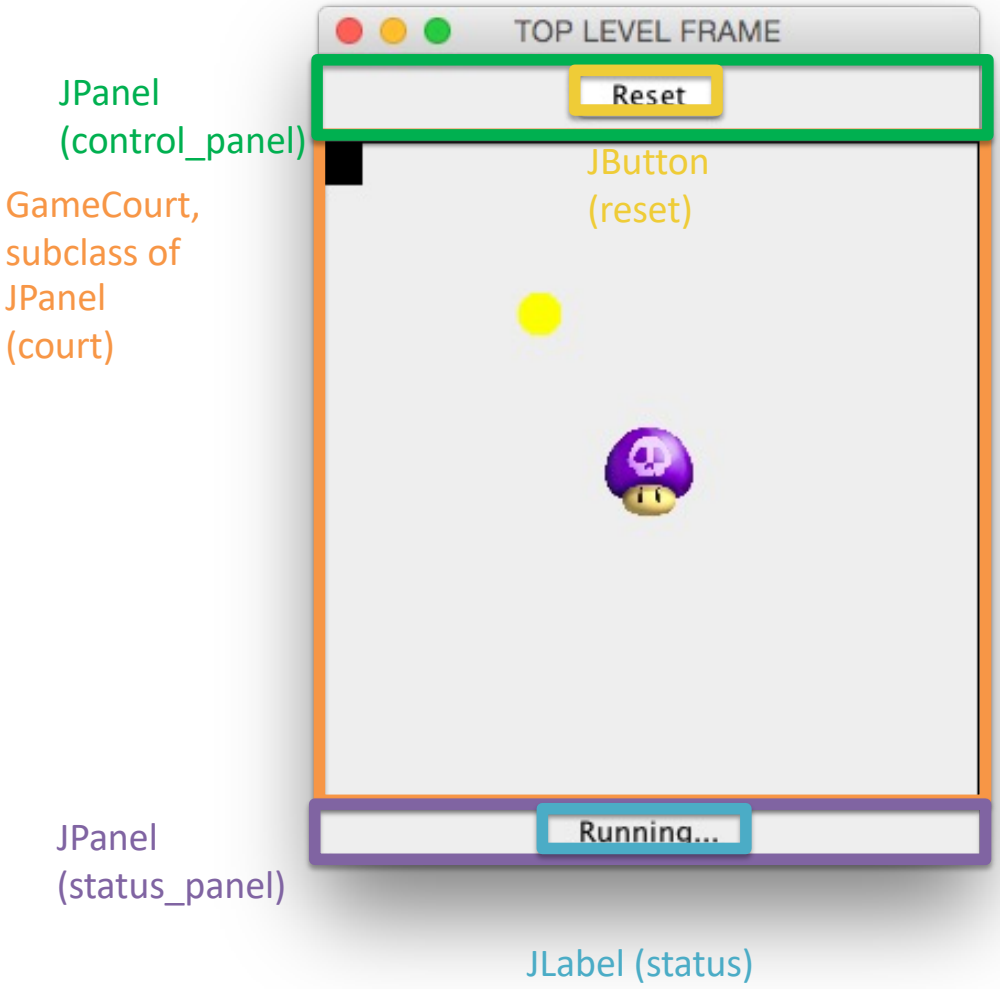

```
JPanel panel = new JPanel();
panel.setLayout(new BorderLayout());

panel.add(b1, BorderLayout.CENTER);
panel.add(b2, BorderLayout.PAGE_START); // top
panel.add(b3, BorderLayout.PAGE_END); // bottom
panel.add(b4, BorderLayout.LINE_START); // left
panel.add(b5, BorderLayout.LINE_END); // right
```

Can contain at most five components; location must be specified when added to the panel

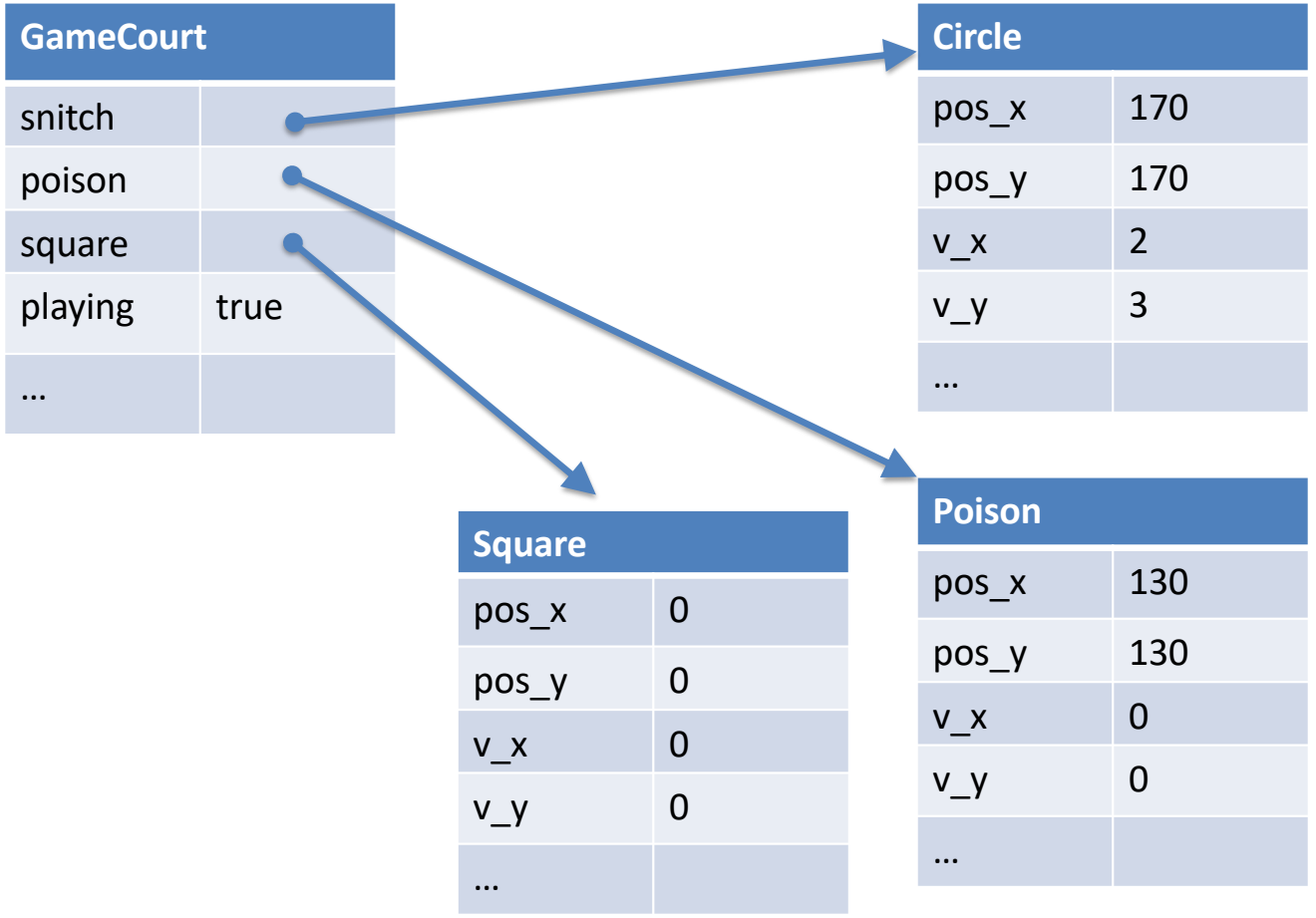
Mushroom of Doom

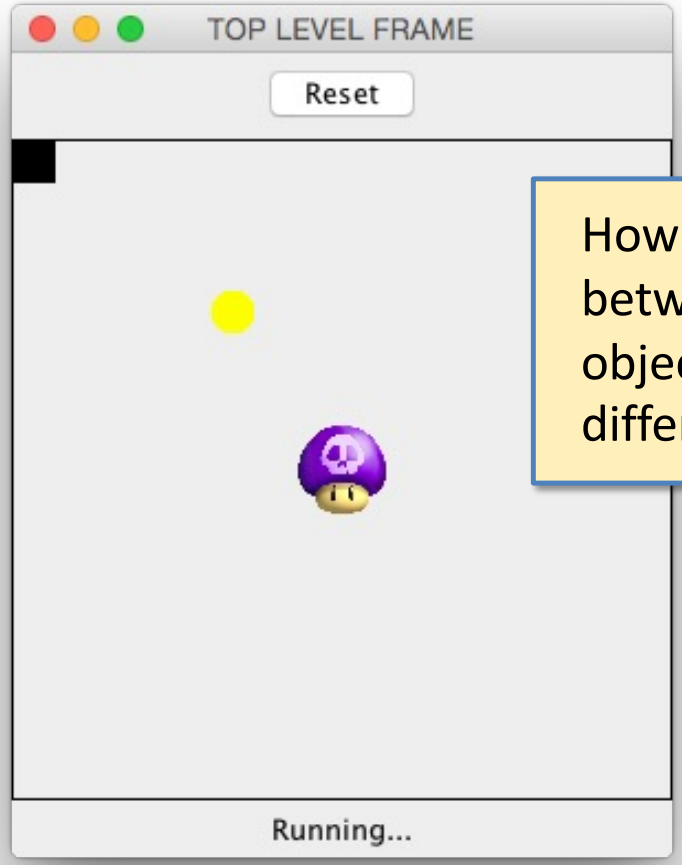
How do we put Swing components together to make a complete game?


TOP LEVEL FRAME
Reset
Running...


JPanel
(control_panel)
GameCourt,
subclass of
JPanel
(court)
JPanel
(status_panel)
TOP LEVEL FRAME
Reset
JButton
(reset)
Running...
JLabel (status)

Game Objects in the Heap





How can we share code between the game objects, but show them differently?

Abstract Classes

- An abstract class provides an *incomplete* implementation:
  - some methods are marked as abstract
  - those methods must be overridden to create instances

```
public abstract class AbstractClass {
    private int x = 0;
    public int m() {
        return frob(frob(x));
    }
    abstract int frob(int x);
}

class ConcreteClass extends AbstractClass {
    @Override
    int frob(int x) {
        return x * 120;
    }
}
```

Keyword "abstract" marks methods without implementations.

A subclass overrides the abstract method with an implementation.

36: It is possible to fill in __??__ with code so that, when run, the variable ac will contain an object of type AbstractClass.

0

True 0%

False 0%

```
public abstract class AbstractClass {
    private int x = 0;
    public int m() {
        return frob(frob(x));
    }
    abstract int frob(int x);
}

// somewhere in main:
Abstract Class ac = new AbstractClass __??__;
```

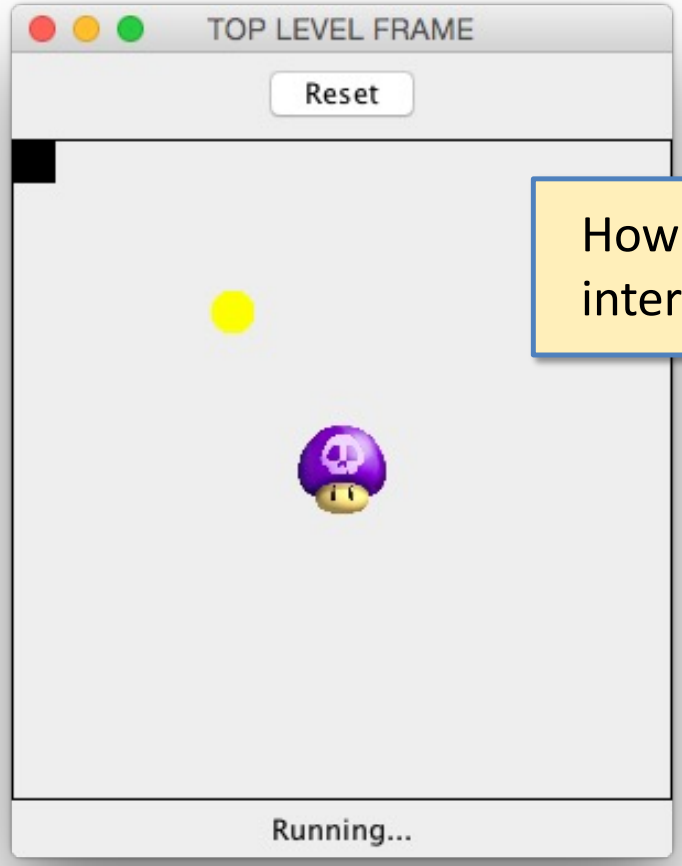
True or False: It is possible to fill in the hole marked __??__ so that, when run, the variable ac will contain a new object of type AbstractClass.

```
public abstract class AbstractClass {
    private int x = 0;
    public int m() {
        return frob(frob(x));
    }
    abstract int frob(int x);
}

// somewhere in main:
AbstractClass ac = new AbstractClass () {
    @Override
    int frob(int x) { return 0; }
};
```

Answer: True – use an anonymous inner class!

Updating the Game State: timer

```
void tick() {
  if (playing) {
    square.move();
    snitch.move();
    snitch.bounce(snitch.hitWall());      // bounce off walls...
    snitch.bounce(snitch.hitObj(poison)); // ...and the mushroom

    if (square.intersects(poison)) {
       playing = false;
       status.setText("You lose!");
    } else if (square.intersects(snitch)) {
       playing = false;
       status.setText("You win!");
    }
    repaint();
  }
}
```



How does the user interact with the game?

1. Clicking Reset button restarts the game
2. Holding arrow key makes square move
3. Releasing arrow key makes square stop

KeyListeners and Adapters

KeyAdapter

MouseAdapter

Using the Keyboard

- The "**Focus**" determines which JComponent is notified when a keyboard event occurs

- During set up

```
setFocusable(true);     // Enable key events
addKeyListener(…);      // Register reactions to events
```

- Once the component is visible

```
// Make sure that this component has the keyboard focus
requestFocusInWindow();
```

KeyListener interface

```
interface KeyListener extends EventListener {

  public void keyPressed(KeyEvent e)
  // Invoked when a key has been pressed.

  public void keyReleased(KeyEvent e)
  // Invoked when a key has been released.

  public void keyTyped(KeyEvent e)
  // Invoked when a key has been typed.
}
```

Updating the Game State: keyboard

```
setFocusable(true);
addKeyListener(new KeyListener() {
  public void keyPressed(KeyEvent e) {
     if (e.getKeyCode() == KeyEvent.VK_LEFT)
          square.v_x = -SQUARE_VELOCITY;
     else if (e.getKeyCode() == KeyEvent.VK_RIGHT)
          square.v_x = SQUARE_VELOCITY;
     else if (e.getKeyCode() == KeyEvent.VK_DOWN)
          square.v_y = SQUARE_VELOCITY;
     else if (e.getKeyCode() == KeyEvent.VK_UP)
          square.v_y = -SQUARE_VELOCITY;
     }

     public void keyReleased(KeyEvent e) {
          square.v_x = 0;
          square.v_y = 0;
     }

    public void keyTyped(KeyEvent e) { }
});
```

Make square's velocity nonzero when a key is pressed

Make square's velocity zero when a key is released

do nothing

Adapter classes

- Swing provides a collection of event *adapter* classes
- These classes implement listener interfaces with empty, do-nothing methods
- To implement a listener class, we extend an adapter class and override just the methods we need
- Another example: MouseListener and MouseMotionListener
  - Seven methods in two separate interfaces
  - Suppose we only need to override three of them

```
private class MyMouseListener extends MouseAdapter {
   public void mousePressed(MouseEvent e) { … }
   public void mouseReleased(MouseEvent e) { … }
   public void mouseDragged(MouseEvent e) { … }
}
```

KeyAdapter class

```
class KeyAdapter implements KeyListener {

  public void keyPressed(KeyEvent e) { return; }
  // Invoked when a key has been pressed.

  public void keyReleased(KeyEvent e) { return; }
  // Invoked when a key has been released.

  public void keyTyped(KeyEvent e) { return; }
  // Invoked when a key has been typed.
}
```

Updating the Game State: keyboard

```
setFocusable(true);
addKeyListener(new KeyAdapter() {
  public void keyPressed(KeyEvent e) {
    if (e.getKeyCode() == KeyEvent.VK_LEFT)
        square.v_x = -SQUARE_VELOCITY;
    else if (e.getKeyCode() == KeyEvent.VK_RIGHT)
        square.v_x = SQUARE_VELOCITY;
    else if (e.getKeyCode() == KeyEvent.VK_DOWN)
        square.v_y = SQUARE_VELOCITY;
    else if (e.getKeyCode() == KeyEvent.VK_UP)
        square.v_y = -SQUARE_VELOCITY;
    }

    public void keyReleased(KeyEvent e) {
        square.v_x = 0;
        square.v_y = 0;
    }
});
```

Make square's velocity nonzero when a key is pressed

Make square's velocity zero when a key is released