CSE1720

Week 10, Lecture 19

Winter 2014 ◆ Thursday, Mar 26, 2015

YORK UNIVERSITÉ UNIVERSITY

Goals & Objectives

- Become acquainted with MVC architecture
- understand the notify mechanism
- augment our controller to respond to events other than ActionEvents, such as KeyEvents

YORK UNIVERSITÉ UNIVERSITY

Model View Controller

- is a software architecture (a configuration of several design patterns)
- separates the aspects of *program logic* from the aspects of *presentation* and *input handling*
- was first devised in 1979 as part of Smalltalk
  - Smalltalk was an early object-oriented language developed at Xerox PARC
- conists of three components
  - MODEL
  - VIEW
  - CONTROLLER

YORK UNIVERSITÉ UNIVERSITY

The Model

- encapsulates information about the state of the game world
- manages the data of the application domain
- is NOT responsible for any graphics, only data
- is NOT responsible for handling any user inputs
- provides services for updating its state (mutators)
- provides services for providing information about its state (accessors)
- becomes part of an Observer design pattern (it becomes an observee, the view becomes an observer)
  - when its state changes, it notifies the view of the change
- the invocation of the mutators is (usually) done by the controller
- the invocation of the accessors is (usually) done by the view

YORK UNIVERSITÉ UNIVERSITY

The View

- encapsulates information about appearance
- is responsible for rendering the model into a form suitable for interaction (e.g., as a graphical user interface)
- is NOT responsible for handling any user inputs
- is NOT responsible for keeping track of any information about the game world

- multiple views can exist for a single model.
  - different views are possible for any given model (e.g., may have different appearances for different purposes or different users)
- the view provides the graphical display, which is the context for many user actions (mouse events, keyboard events);
  - we pass the view to the controller's constructor so that the controller may install listeners on the view (to be able to track user input actions)

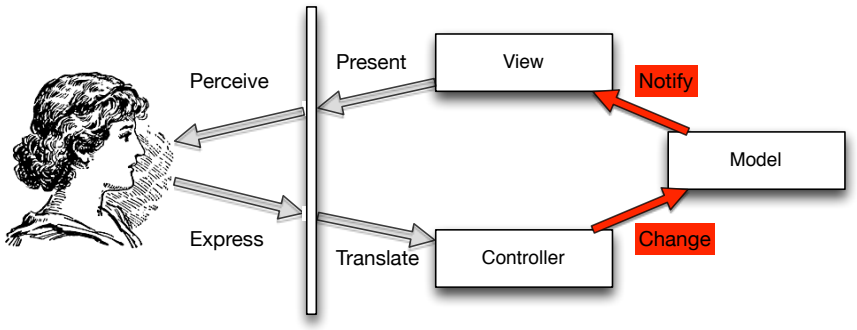
The Controller

- encapsulates functionality about which user inputs have what type of impact in the game
- is NOT responsible for keeping track of any information about the game world
- is NOT responsible for any graphics

- receives user input
- initiates a response by modifying the the model

- for animations, the controller receives `ActionEvents`, which triggers frame advancement

Schematic of MVC



7

YORK UNIVERSITÉ UNIVERSITY

Walkthrough of the flow

- The user interacts with the user interface, triggers input events
- The controller handles the input events
- The controller translates the input events into impacts on the game world model
- The state of the game world model changes
- The game world model notifies all of its listeners it has changed
- The view is listening to the model; upon notification of a change to the model, the view regenerates itself (update to reflect new state of the game world).
- The user interface stands waiting for subsequent user input actions

8

YORK UNIVERSITÉ UNIVERSITY

Consider `Example00`

- `AnimationModel`:
  - sets up a world of 5 sprites, anchored in the center
  - has service `drawNewFrame(graphics2D)`
    - iterate over sprites, reposition according to game world logic, then redraw sprites
- `AnimationController_v0`
  - is an `ActionListener` which gets installed on a `Timer`
  - sets up `RasterImage` (as drawing surface)
  - listens for `ActionEvents`, when they occur the `actionPerformed(ActionEvent)` method is invoked
  - body of `actionPerformed(ActionEvent)` method is set up to invoke `drawNewFrame(graphics2D)`

9

YORK UNIVERSITÉ UNIVERSITY

Consider our `AnimationController_v0`

- Notice that the information about the View is embedded within the Controller
- We will abstract all of that away and instead encapsulate in a View class

10

YORK UNIVERSITÉ UNIVERSITY

AnimationController_v0 vs AnimationController_v1

```
import java.awt.event.ActionListener;
import java.lang.reflect.InvocationTargetException;

import javax.swing.SwingUtilities;

import simulation_L19Version.AnimationController_v0;
import simulation_L19Version.AnimationModel;
import simulation_L19Version.Model;
import simulation_L19Version.SimulationRunnable;

public class Example00 {

        public static void main(String[] args) throws InvocationTarg
                        InterruptedException {

                Model theModel = new AnimationModel();
                ActionListener dude2 = new AnimationController_v0(th
                SimulationRunnable theSimulation = new SimulationRun
                SwingUtilities.invokeAndWait(theSimulation);

        }
}
```

```
import java.awt.event.ActionListener;
import java.lang.reflect.InvocationTargetException;

import javax.swing.SwingUtilities;

import simulation_L19Version.AnimationController_v1;
import simulation_L19Version.AnimationModel;
import simulation_L19Version.Model;
import simulation_L19Version.SimulationRunnable;

public class Example01 {

        public static void main(String[] args) throws InvocationTarg
                        InterruptedException {

                Model theModel = new AnimationModel();
                ActionListener dude2 = new AnimationController_v1(th
                SimulationRunnable theSimulation = new SimulationRun
                SwingUtilities.invokeAndWait(theSimulation);

        }
}
```

11

YORK UNIVERSITÉ UNIVERSITY

```
public class AnimationController_v0 implements ActionListener

        // attributes that concern the drawing environment
        private RasterImage theFrameImage;
        private Graphics2D graphics2D;

        // attribute for what is drawn on-screen (encapsulate
        private Model animationModel;
```

```
public class AnimationController_v1 implements ActionListener

        // attributes that concern the drawing environment
        private View_RasterImageVersion theView;

        // attribute for what is drawn on-screen (encapsulate
        private Model animationModel;
```

12

YORK UNIVERSITÉ UNIVERSITY

```
public AnimationController_v0(Model model) {
        // initialize the attribute(s) that concern t
        // frame
        animationModel = model;

        // assign the value of the attribute by invok
        // and invoking required methods
        this.theFrameImage = new RasterImage(model.ge
                        model.getWorldHeight());
        theFrameImage.show();
        this.theFrameImage.setTitle(this.getClass().g
                        + UtilityClass.getTimestamp()

        // assign the value of the attribute by invok
        graphics2D = theFrameImage.getGraphics2D();
}
```

```
public AnimationController_v1(Model model, View_Raste
        // initialize the attribute(s) that concern t
        // frame
        animationModel = model;

        // assign the view
        theView = view;

}
```

13

```
public void actionPerformed(ActionEvent ae) {
        theFrameImage.setAllPixelsToAColor(Color.WHIT

        // *****
        // now we draw the new frame; delegate to the
        // to draw the next frame on the passed graph
        // encapsulates the operations to update the
        // the next frame)
        animationModel.drawNewFrame(graphics2D);

        // *****
        // the graphics2D object has been updated, so
        // manager needs to repaint it.
        theFrameImage.repaint();
}
```

```
public void actionPerformed(ActionEvent ae) {
        System.out.println("advancing next state");
        // delegate to the model to determine the nex
        animationModel.advanceNextState();

}
```

14

```
@Override
public void advanceNextState() {
    for (Sprite s : myCollectionLeftwardUpward) {
        s.moveLeftwardUpward();
    }
    //System.out.println("notify model changed");
    notifyModelHasChanged();
}


public void notifyModelHasChanged() {
    for (ModelListener listener : listeners) {
        listener.changed();
    }
}
```

15

YORK UNIVERSITÉ UNIVERSITY

```
@Override
public void changed() {
        System.out.println("changed");
        this.drawOnScreenElements(rasterImageGraphics2D);
        theFrameImage.repaint();
}

/**
 * This method draws all of the required on-screen elements on the passed
 * graphics context.
 *
 * Implementation note: the client should not assume that the graphics
 * context corresponds to what is shown on-screen. In fact, it may be a
 * graphics buffer that will be rendered at some other point.
 *
 * @param g
 *            as specified above
 */
public void drawOnScreenElements(Graphics2D g) {
        theModel.drawSceneBackground(g);
        theModel.drawCurrentState(g);
}
```

16

Abstracting away the View

- and so the preceding example demonstrates how we can abstract away the view
- can take this a step further and pull out the view even in the main app

```
public class Example01b_SeparatedView {

    public static void main(String[] args) throws InvocationTargetException,
            InterruptedException {

        Model theModel = new AnimationModel();
        View_RasterImageVersion theView = new View_RasterImageVersion(theModel);
        // demonstrates use of controller (improved version, with view
        // abstracted away and passed as constructor argument)
        ActionListener dude2 = new AnimationController_v1(theModel, theView);
        SimulationRunnable theSimulation = new SimulationRunnable(dude2);
        SwingUtilities.invokeAndWait(theSimulation);

    }
}
```

The motivation for doing this won't become apparent until v3

We need a better View

- `AnimationController_v1` makes use of the services from the class `View_RasterImage`
- `RasterImage` is ok for a first step, but we need to use something with better graphic performance
- We can use double buffering with a `Canvas`, which is part of java.awt
- `AnimationController_v2` makes use of the services from the class `View`

18

YORK UNIVERSITÉ UNIVERSITY

How our improved View gets redrawn

- About the class `View`
  - note!!! with a `Canvas` component, we also call `repaint()` directly
  - this alerts the WM and the WM, in turn, calls the component's `paint(Graphics)` method
  - the WM passes the `Graphics` parameter to the paint method
  - Notice that this is a key difference between the two View classes – in the first class, we kept track of the Graphics2D object, whereas in the second class, we do not need to do this

YORK UNIVERSITÉ UNIVERSITY

Consider `Example02`

- This app makes use of our improved view

YORK UNIVERSITÉ UNIVERSITY

Evolving the controller

- now that we have a better view, we can proceed with improving our controller
- have a look at `AnimationController_v3`

YORK UNIVERSITÉ UNIVERSITY

Responding to Key Events

- `AnimationController_v2` only responds to `ActionEvents`
- We need an controller that will also respond to `KeyEvents`

```
public class AnimationController_v2 implements ActionListener
public class AnimationController_v3 implements ActionListener, KeyListener
```

- When a class says it will implement the `KeyListener` interface, the compiler enforces the condition that the class provides implementation for all of the `KeyListener` methods:
  - `public void keyTyped(KeyEvent e)`
  - `public void keyPressed(KeyEvent e)`
  - `public void keyReleased(KeyEvent e)`

YORK UNIVERSITÉ UNIVERSITY

Responding to Key Events

- `public void keyTyped(KeyEvent e)`
  - fires when a key is pressed that can be converted into a unicode character
  - happens when key is pressed down and then is released back up
- `public void keyPressed(KeyEvent e)`
  - fires when key is pressed down; obtain raw key presses
- `public void keyReleased(KeyEvent e)`
  - fires when key lifts back up (after being pressed down); ; obtain raw key presses

YORK UNIVERSITÉ UNIVERSITY

```
@Override
public void keyReleased(KeyEvent e) {
    if (e.getKeyCode() == KeyEvent.VK_SPACE) {
        animationModel.moveAllSpritesToCentre();
    }
}
```

- `animationModel` gets mutated
- when sprites are repositioned, model notifies listeners that it has changed
- the view learns that model has changed, signals that it needs to be repainted
- the WM then launches repainting of view, it invokes `paint(Graphics)` method, triggering sequences that redraws view

YORK UNIVERSITÉ UNIVERSITY

Exercises To Complete

- Exercise 1:
  - Modify the controller so that a different key action causes the sprites to move to the centre
- Exercise 2:
  - modify the model so that the game consists of a single sprite only that moves only with user control
  - modify the controller so that a keypress (you choose which one) from the user makes the sprite move along the diagonal (RightwardDownward)
  - modify the controller so that a keypress (you choose which one) from the user makes the sprite move along the diagonal (LeftwardUpward)
  - BONUS: figure out how to do continuous movement with sustained key press (start/stop the movement with keyPress and keyRelease)
- Exercise 3:
  - modify the model to start with no sprites at all
  - modify the controller so that the user adds sprites to the world
  - BONUS: implement different keypresses for different sprites

25

YORK UNIVERSITÉ UNIVERSITY