CSE1720

Week 04, **Class Meeting 10** (Lecture 07)

Winter 2013 ◆ Tuesday, January 29, 2013

YORK UNIVERSITÉ UNIVERSITY

This lecture will be using code from the following package to illustrate concepts:

```
game_Lect07Version
```

Objectives for this class meeting

- Consider the following question “How do I get the game shooter to shoot?”
  - We will discuss the answer to this
    - in terms of design specification
    - in terms of implementation
- Pose and answer conceptual questions about the interactive aspects of the game

YORK UNIVERSITÉ UNIVERSITY

Big picture recap…

- So far:
  - our app asked the window manager for a window
    - via the services of the Frame class
  - we created and placed a component inside this window
    - this is the GameCanvas object
  - we used the services of this component to implement drawing
    - via the component’s services that encapsulate the Graphics2D object
- This component and the window manager **coordinate** in order to do the drawing

YORK UNIVERSITÉ UNIVERSITY

About shooting…

- Shooting is a basic behaviour that is a defining characteristic of shooter games
- We can employ encapsulation:
  - encapsulate the shooter
  - encapsulate the projectile
- Shooting entails:
  - waiting for user input
  - rendering the trajectory over a sequence of frames

5

YORK UNIVERSITÉ UNIVERSITY

About the trajectory…

- We need functionality to implement repeated frame drawing
- the speed at which frames are draw is called the **frame rate**
  - **TV: 60 fps**
  - **Movies: 24 fps**
  - **The Hobbit 3D experiment: 48 fps**
  - **Threshold of human perception: 10-12 fps**
- Side note about a key concept: **suspension of disbelief**
  - humans will suspend judgment about the implausibility of a narrative in certain conditions, but not others
  - characteristics of the medium can trigger this

6

YORK UNIVERSITÉ UNIVERSITY

How we implement frames

- We need functionality to implement repeated frame drawing
  - we instantiate a Timer object
  - this launches a new thread
  - the thread fires events at the specified interval
  - what does this mean?
    - we need to talk about the observer pattern
    - we'll come back to this

```
FrameAdvancer frameAdvancer = new FrameAdvancer(theCanvas);
Timer frameAdvancerTimer = new Timer(msecPerFrame, frameAdvancer);
frameAdvancerTimer.start();
```


YORK UNIVERSITÉ UNIVERSITY

Shooting requires interactivity

- This component and the window manager **coordinate** in order to do the drawing
- This component and the window manager also **coordinate** in order to **handle user input**
- This component is also an on-screen object with which the user can interact
  - click on the canvas with the mouse
  - press keyboard keys
- These happening are first handled by the window manager…

YORK UNIVERSITÉ UNIVERSITY

The WM and user input

- When something like a mouse click happens, the WM causes the following to happen:
  - instantiates an event object
  - the object encapsulates the source of the event, some info about when the event happened, and some other details
  - "dispatches" this event

What is meant by *dispatching* the event?

YORK UNIVERSITÉ UNIVERSITY

The Radio Analogy…

- Think of it this way…
  - The WM is like a broadcast radio conglomerate
  - It has a whole bunch of radio stations
  - It is broadcasting content over all of its stations
  - It is organized
    - certain content goes over certain stations
    - instead of a continuous radio signal, the content is packaged up in discrete objects called events

YORK UNIVERSITÉ UNIVERSITY

What does an app do?

- As a default, NOTHING.
- Your app is like a radio
  - By default, it is turned on, but not tuned to any station
  - Since it is not tuned to anything, your radio is silent
  - To listen to a station, you need to tune your radio to a station
- The radio analogy has some complications:
  - your radio can also broadcast content
  - your radio can be tuned to several channels at once!

11

YORK UNIVERSITÉ UNIVERSITY

How to “tune your radio”…

1. Identify the **observee** component
   - this is the component that is dispatching events that you care about
2. Create an **observer** component
   - this will be a component that is capable of “listening” to those types of events
   - this is like “tuning” to the station
3. Use the services of the **observee** to tell it that it has an **observer**

12

YORK UNIVERSITÉ UNIVERSITY

Concrete example

1. The `GameCanvas` is the component that we want to observe
   - it is the **observee** component; it dispatches events

2. Create an **observer** component
   - this is the `GameObserver` component
   - it encapsulates a `KeyListener`

3. Use the services of the **observee** to tell it that it has an **observer:**

```
theCanvas.addKeyListener(observer);
```

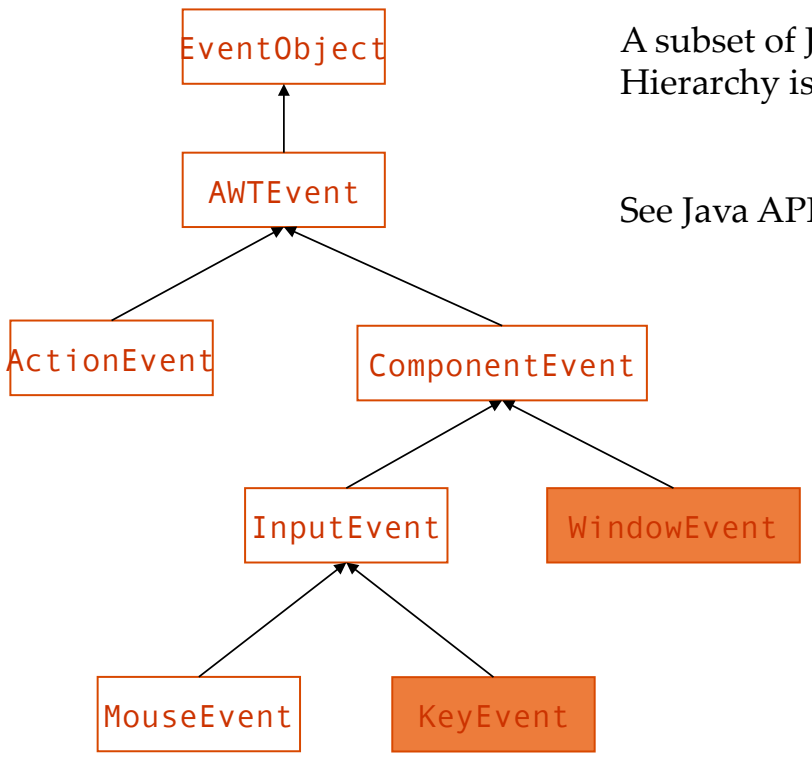
13

About Events in General…

- Events are objects that encapsulate some sort of external "happening"
  - the user did something
    - e.g., performed a mouse or keyboard action
  - the window manager did something
    - e.g., opened a window, shifted focus

14

Java's Event Class Hierarchy



A subset of Java's Event Class Hierarchy is shown here

See Java API for full hierarchy

YORK UNIVERSITÉ UNIVERSITY

Back to the Timer object…

- Can you identify the observer and the observee?

YORK UNIVERSITÉ UNIVERSITY

Tasks

- slow down the projectile to have a slower trajectory
- make the projectile expire before it reaches the edge of the screen
- change the interface so that the ‘f’ key fires the shooter instead of the space bar

YORK UNIVERSITÉ UNIVERSITY