CSE1720

Week 06, **Class Meeting 18** (Lab 06)

Winter 2013 ◆
Thursday, Feb 14, 2013 & Friday, Feb 15, 2013

YORK UNIVERSITÉ UNIVERSITY

Week 06 Lab Exercise

- **Due Date:** n/a
- **Course Weight:** n/a
- **Topic**: Adding movement to your game shooter

Once you have completed the exercises, your code base will be updated and ready for the next step of game development

YORK UNIVERSITÉ UNIVERSITY

Setup

Download v.2 of the class `ShooterSprite` from the course website (lab sub-page)

It implements the positioning of the sprite in an improved way.

YORK UNIVERSITÉ UNIVERSITY

Exercise #1: Setting up a testing stub

Create a new class. Call it `Lab06Tester` and create it with a main method.

In the main method:

- declare and instantiate a `Dimension` object (in the same way that is done in `GameDriver`)
- declare and instantiate a `ShooterSprite` object (in the same way that is done in `GameCanvas`)
- mutate the `ShooterSprite`'s anchor (in the same way that is done in `GameCanvas`)
- before and after the method invocation that changes the anchor position of the shooter sprite, print out to the console the `ShooterSprite`'s X and Y coordinates (use the getter methods)

YORK UNIVERSITÉ UNIVERSITY

Exercise #1: Reflections

Even though you are not rendering the shooter sprite, it is still represented and you can still mutate it.

YORK UNIVERSITÉ UNIVERSITY

Exercise #2: Adding functionality

Now we want to modify the `ShooterSprite` class.

Create four methods: `moveLeft`, `moveRight`, `moveUp`, `moveDown`

- In the body of each method, add functionality to translate the x and y anchor point in the appropriate direction. You can choose the granularity of the move (big or small).
- for all method, the returns are `void`
- for all methods, there are no parameters
- Eventually, you will need to implement functionality to keep the sprite from going off the screen. But you can leave this until Exercise #5.

YORK UNIVERSITÉ UNIVERSITY

Exercise #2: Adding functionality

Don't forget to add method comments (the comment block at the start of the method).

Select the package and regenerate the javadoc (Project -> Generate Javadoc)

Test the methods by invoking them in the `Lab06Tester` testing stub and printing out the new position of the sprite.

YORK UNIVERSITÉ UNIVERSITY

Exercise #2: Reflections

`Lab06Tester` is serving as a (sort of) testing harness. We are using it as a sandbox for testing out the movement methods. You are examining the outcomes of the method invocations by manually checking the console outputs (e.g.,the *x,y* coordinate positions before and after the method invocations).

If we wanted to make `Lab06Tester` a fully-fledged testing harness, we would also need to implement code to **validate** the outcome of the method invocations (we should *programmatically* rather than *manually be* checking the outcomes).

YORK UNIVERSITÉ UNIVERSITY

Exercise #3: Adding visibility

Now we want to modify the `GameCanvas` class.

Add a getter method to this class so that clients of the `GameCanvas` class can obtain a reference to the `ShooterSprite` object.

9

YORK UNIVERSITÉ UNIVERSITY

Exercise #4: Connecting the pieces

Now we want to modify the `GameObserver` class.

Decide which keyboard buttons you want to correspond to the left, right, up, down movements.

Examine the API for the `KeyEvent` class. Read the class comment and pay particular attention to the part about **virtual key codes.**

Locate the static field that represents the four keyboard buttons (e.g., if you choose "u" for up, then the corresponding virtual key code would be `KeyEvent.VK_U`)

Examine the body of the `keyReleased` method. Observe that action is taken conditionally, only in the event of `KeyEvent.`*`VK_SPACE`*.

10

YORK UNIVERSITÉ UNIVERSITY

Exercise #4: Connecting the pieces

Decide which type of key event you want the movement to be tied to (key press, key release, key typed).

If you are not sure, then experiment with them all.

Modify the body of the method to also test for the directional key events. Conditionally move the shooter sprite, as per the user actions.

YORK UNIVERSITÉ UNIVERSITY

Exercise #5

Go back and refine your `moveLeft`, `moveRight`, `moveUp`, `moveDown` methods to implement checking so that the shooter sprite cannot leave the game field.

In the vertical dimension, there is a complication because the titlebar occupies screen real estate and displaces the x,y coordinate of the shooter sprite's bounding box.

To turn off the title bar, invoke the following method *immediately after the JFrame is constructed*

```
theGameWindow.setUndecorated(true);
```


YORK UNIVERSITÉ UNIVERSITY