CS 145: Homework 1

Figure

1 Description

Your primary goals in this homework are to learn about bundling up code into methods and using objects given to you by someone else. You will do this in the context of scripting the animation of one or more stick figures.

You will write two classes for this homework: `Utilities` and `Animation`. Place them in the `hw1` package.

2 Introduction

Keyframe animation is a method of animating in which you define the positions of objects only at certain frames. In between any two of these *key* frames, we let the computer blend the positions of the two surrounding frames. For example, I might place a ball at the left of the screen at frame 0. At frame 100 I place it at the right. When the computer draws frame 10, it'd position the ball 10% of the way to the right of the screen. At frame 50, it'd be in the middle.

In this homework, we provide a class representing a stick figure that can be positioned at keyframes of your choosing. A figure has several properties that you can set: how much its left and right arms are bent, how much its left and right legs are bent, how much it has turned around the y-axis, and how far forward and backward it has moved. You'll set these properties at certain times to make the figure do things like wave, walk, and gyrate.

3 Utilities

The Utilities class has just a single method that will be used to position the figure between keyframes. This is called *tweening*. **You should write this class first and test it before moving on.** It has:

- A `static` method `interpolate` that takes five `double` arguments and returns a `double`. Let's call the first argument valueA, the second timeA, the third valueZ, the fourth timeZ, and the fifth timeM. Given the two time-value pairs, we'd like to find what value we'd expect to see at timeM based on the two known time-value pairs and return it. If you are born with \$0 and at age 100 you have \$4, this method would report that you had \$2 at age 50.

  How do we calculate the value for timeM? First, determine what percentage of the time between timeA and timeZ has passed when we reach timeM:

$$\text{percentage} = \frac{\text{timeM} - \text{timeA}}{\text{timeZ} - \text{timeA}}$$

We then apply this percentage to find the corresponding jump between valueA and valueZ:

$$\text{value jump} = \text{percentage} \times (\text{value } Z - \text{value } A)$$

Then we jump that much from valueA:

$$\text{valueM} = \text{valueA} + \text{value jump}$$

Return valueM and the method is finished.

If this method doesn't work correctly, nothing else will. Whip up a `main` method and assert that your code does the proper thing.

4 Animation

Your `Animation` class scripts the actual animation by using some classes we provide. It must meet these requirements:

- It must have a `main` method that triggers an animation of your choosing. Here's a very simple example of a man spinning in a circle:

  ```
  Animator animator = new Animator();
  Figure man = new Figure();
  animator.register(man, 0.0);
  man.turn(360.0);
  animator.register(man, 10.0);
  animator.show();
  ```

  We register the figure in his default state at 0 seconds. We register him 10 seconds later having spun around 360 degrees. Your `Utilities.interpolate` method will be used by the `Animator` class to position him in between 0 and 10 seconds. (You need never call `interpolate` directly.)

  The `Animator` and `Figure` classes are provided to you in the SpecChecker JAR file. Just add the file to your build path and they'll be usable in your code.

- Your animation must be at least 1 minute long.

- Your code must animate all movable parts of a `Figure`—i.e., you must call each of the `bend*` methods, `turn`, and `move` for at least two different times. See its documentation at `http://www.twodee.org/teaching/cs145/2011C/homework/Figure.html` for more details on how to do so.

- Your code should break out at least two separate methods that perform some repeatable and high-level action, like waving or walking. (Pass the `Figure` and `Animator` as arguments so these methods can register keyframes.)

You are encouraged to share your animation on Piazza by hitting Export and copying the result into a response to post @152. You can also Import others' animations. (Be warned. Very little validation is done on imported animations.) Figures registered at only one frame will not appear at all. Loops will make your life easier for repetitive motion. You can animate multiple figures.

5 Other Expectations

We need to be able to read your code. Some of your grade will be based on stylistic matters—not just a functioning final result.

- Variable and method names start with a lowercase letter and have capitalized internal words, e.g., `kittenCount`.
- Code should be documented. Explain what you are doing in brief statements.
- Code should be indented properly. Hit Control-Shift-F inside your Java files to let Eclipse clean up your code.

6 Submission

This is a full homework. The SpecChecker will only see to it that you named your classes properly, placed them in the right package, and formed your methods with the right arguments and return types. It will not grade. But it will also package up your code for submission. Deadlines are not extended for botched configurations. Please follow these directions closely to test and submit your work:

1. Add JUnit to the Build Path if you haven't already done so. See `pre1` for details.
2. Download the SpecChecker JAR file.
3. Drag the JAR file onto your `hw1` package in Eclipse. Right-click on it, and select Build Path → Add to Build Path. It will migrate to the Referenced Libraries node in your project.
4. Run the JAR file by selecting it and hitting the Play button in the Eclipse toolbar. Run it as a Java application if prompted.
5. Address any errors. Any lingering errors may cost you points.
6. Choose a directory in which `hw1.zip` will be saved.
7. Drop your submission into `W:\c s\CJohnson\cs145\<YOUR-USERNAME>`. You may overwrite this file as often as you like before the deadline. Just make sure the directory contains only one *.zip file with lowercase hw1 in its name.