Com S 430
Spring 2015
Homework 3

Start soon. There are likely to be questions, ambiguities, errors, or other issues. Post questions on Piazza or talk with the staff.

1. The **`function_plotter`** package (see **`examples/week5`**) is a program for graphing functions. The main logic is in the class **`FunctionPlotter`**, and there is a text-based user interface you can run in **`FunctionPlotterMain`**. The **`FunctionPlotter`** starts up a Swing application, **`GraphWindow`**, that displays a graph of the function.
Read the code and try running it. It may work fine, or not, but it has some potentially serious violations of the GUI confinement rules. Modify the **`GraphWindow`** code to fix these problems. Add a comment at the top of **`GraphWindow`** explaining what you did and why.

2. The **`message_passing`** package (see **`examples/week5`**) contains the basic ingredients for an actor-style component framework based on one-way message passing, as discussed in class last week. There is a base **`Component`** type and a few examples of messages that you can use. Create a program similar to our original Client.java from homework 1, using the model of single-threaded actors that pass messages. The basic rules are that each component executes its code in a single thread and that sending a message buffers the message without blocking. (Not all components have to do this exactly the same way.)

Start with the following:

- An Input component that reads from the console and sends instance of TextMessage to the Client
- A Client component that incorporates most of the code from **`hw1.Client.java`**. It handles TextMessage from the Input component and ResultMessage from the Proxy component
- A Proxy component that handles RequestMessage from instances of Client and passes requests over the network to the server. (You should assume that there may be multiple Client instances.)

a) The latter is tricky since we are using blocking I/O. For the first iteration, try modeling it after a typical solution to homework 1; that is, each request to the server is handled by a different thread. In this programming model, that means defining a new Handler component type for this purpose and creating them as needed in the Proxy.

b) Design a mechanism that works more like a thread pool with a fixed number of Handlers.

c) Add a Timeout component that allows the Client to set a timeout for each request and receive a notification if a given result was not received within the timeout period.

Notes:

- Be sure you remove any unnecessary synchronization from your previous solution to homework 1.

- The request/result messages do not include the "key". This is intentional. The "key" represents part of the Client's local context that must be saved and later restored when the result message arrives.

- If you are confused by the double dispatch mechanism, I promise that you are not alone. The file **`examples/week5/DoubleDispatchExample.java`** is a skeleton of how it works with a detailed explanation at the top of the file.

3. Write the basic code for a "Yahtzee Flash" cube, using the framework above as a foundation. Since we do not have physical cubes with radio transmitters, assume we have two static methods available,

`Universe.broadcastLeft(Component, IMessage)` and
`Universe.broadcastRight(Component, IMessage),`

where the Universe knows the "physical" locations of the cubes and will invoke the **`send()`** method of the cube in range, if any. You don't have to implement the Universe, I'll do that. Implement enough (including whatever message types you need) to get the cubes to display the initial menu (1, 2, 3, 4, 5) starting from the leftmost cube when all of them are lined up together and behave appropriately when they aren't all lined up.

By "display" we mean that if the Universe sends a GetDisplay message, the cube responds with a corresponding DisplayMenu message containing a value 0 through 5 (where 0 corresponds to "unknown", i.e., the cube is isolated).

The cubes should recheck their locations periodically and should use a uniform timeout value if a response message is not received. Make sure these values are easily configurable. Start with (say) a 50 ms polling period and a 500 ms timeout.

If you missed class the day we demoed the flash cubes, you can find various demonstrations on YouTube, e.g. https://www.youtube.com/watch?v=xhUiY9b2X78