CS 4730: Computer Game Design

Overview
For this assignment, you will implement the observer design pattern to enable your games to have the ability to easily throw custom events around. Though many languages support this functionality already (like Java), this is a good exercise in coding.

You will test your event module by making a simple game. Your final game for this course will almost certainly use custom events and this module will almost certainly make your life easier in the future.

PART 1: Create the Classes
You will be writing two classes and two interfaces (or abstract classes). As always, you don't have to follow these guidelines exactly, but I recommend you follow them closely. They are as follows:

NOTE: ALL of these classes should be placed in a new folder ("/engine/events").

- *EventDispatcher*: Equivalent to "observable". Lists the methods necessary for any object that throws events. Contains the following methods:
    - addEventListener(IEventListener listener, String eventType)
    - removeEventListener(IEventListener listener, String eventType)
    - dispatchEvent(Event event)
    - hasEventListener(IEventListener listener, String eventType).

- EventListener: Defines what all listeners must do. Has a single method:
    - virtual void handleEvent(Event event); //equivalent to notify in many textbooks.

- *Event*: Object encapsulating the information regarding a single event that occurs. Fields and methods include:
    - String eventType
    - EventDispatcher source //the object that created this event with the new keyword.
    - Getters and Setters
    - **If your event needs more information (like questId, or timestamp, then you would extend this class and add the additional information to that new event type.*

Part 2: A Simple Test Demo
Once you've written these classes. Let's test them out by making a sample game! Let's suppose we have a quest involving a character grabbing an item. Place two sprites on the screen, move one with the keyboard over to the other. Have the coin (or whatever is being picked up), throw some kind of PickedUpEvent and become invisible. Have another class called QuestManager (that keeps track of quests) listen to, hear the event, and print out something like "quest is complete". Yes, this is contrived, but it helps prove that your code works correctly. You will likely use a line of code like this:

myCoin.addEventListener(myQuestManager, PickedUpEvent.COIN_PICKED_UP);

Where PickedUpEvent.COIN_PICKED_UP is a constant static string that represents that particular event.

Turn In
As always, submit your code on Collab.