Assignment 4

Is due today, at 11:55. However ...
Due date: March 19th
Team size == 1
In this assignment, you will implement the Command Pattern for the physical simulation system you worked on in assignment 1.
This is probably one of the shortest assignment I've ever given, the solution being less than 150 lines of code.
But beware assignment 6, it's going to be a long one.
jME (jMonkey Engine) is a high performance scene graph based graphics API.
- The scenegraph allows for organization of the game data in a tree structure.
- Typically, these nodes are organized spatially to allow the quick discarding of whole branches for processing.

jME was built to fulfill the lack of full featured graphics engines written in Java.
Mouse Handling in jMonkey

- Mouse handling in jMonkey is slightly different.
- For keyboard input, it uses an action handler system similar to the Command Pattern.
  - The user can specify the behavior of his application by implementing an InputHandler.
  - He then gives behavior to the InputHandler by registering InputAction objects.
- For mouse input, we just need to register a different kind of listener.
private class MouseListener implements MouseInputListener{

    public void onButton(int button, boolean pressed, int x, int y) {
        if(pressed)
            dispplay.fireMousePressedEvent(x, y, getMouseButton(button), 1);
        else
            dispplay.fireMouseReleasedEvent(x, y, getMouseButton(button), 1);
    }

    public void onMove(int xDelta, ...
    public void onWheel(int wheelDelta, ...
}

MouseInputListener
Another example of the Adapter pattern is Java RMI.

To better understand this example, we need to take a look at Java RMI.
- RMI stands for Remote Method Invocation.
- In simple terms, RMI allows you to execute methods on objects found on a different machine.
- It is similar to other method invocation system, such as CORBA (Common Object Request Broker Architecture).
- RMI makes invoking methods transparent:

```java
LinkedList remoteList = RMI.getObject("nameoflist");
remoteList.add(new Integer(1));
```
Components of a RMI System

- Client
- Stub
- Network
- Skeleton
- Server
Components of a RMI System

- Client
- Stub
- Network
- Skeleton
- Server

Adapter
Object hosted on the server must implement the java.rmi.Remote interface.

However, you might not want to modify your existing object.

Thus you need an adapter.
A class is more reusable when you minimize the assumptions other classes must make to use it.

By building interface adaption into a class, you eliminate the assumption that the other classes see the same interface.

In other words, interface adaptation lets us incorporate our class into existing systems that might expect different interfaces to the class.
In our example

- Our example showed three example of tree widgets in Eclipse.
- Each of these widget showed a different tree structure:
  - Files
  - Packages
  - Type Hierarchy
- However, each of these tree will have their own data structure.
- So how do I build a tree widgets that works with any type of tree class.
First Step

- First, you must find a “narrow” interface for the Adaptee.
  - The smallest subset of operations that let us do the adaptation.
- The “narrower” the interface, the easier the adaptation.
- But what do I need to know about a tree to display it?
To draw a tree

- I need two functions:
  - `GetChildren(node: Node) : Set<Node>`
  - `CreateGraphicsForNode(node: Node) : void`
- But how do I implement this?
Using Abstract Operations

```
TreeWidget
+ GetChildren(node: Node) : Set<Node>
+ CreateGraphicsForNode(node: Node) : Graphic
+ BuildTree(node: Node)
+ DisplayTree()

FileTreeWidget
- FileSystem: FileSystem
+ GetChildren(node: Node) : Set<Node>
+ CreateGraphicsForNode(node: Node) : void

PackageTreeWidget
- Package: PackageHierarchy
+ GetChildren(node: Node) : Set<Node>
+ CreateGraphicsForNode(node: Node) : void
```
children = GetChildren(N)
for each child in children
    graphic.add(CreateGraphicsForNode(child))
BuildTree(child)
Using Delegate Objects

TreeWidget
+BuildTree(node: Node)
+DisplayTree()
+SetDelegate(delegate: TreeWidgetDelegate)

TreeWidgetDelegate
+GetChildren(node: Node): Set<Node>
+CreateGraphicsForNode(node: Node): Graphic

FileTreeWidget
-FileSystem: FileSystem
+GetChildren(node: Node): Set<Node>
+CreateGraphicsForNode(node: Node): void

PackageTreeWidget
-package: PackageHierarchy
+GetChildren(node: Node): Set<Node>
+CreateGraphicsForNode(node: Node): void
BuildTree(node: Node)

children = delegate.GetChildren(N)
for each child in children
    graphic.add(delegate.CreateGraphicsForNode(child))
    BuildTree(child)