Observer Pattern implementation

Team up in peers.

1. Make a new project in IntelliJ and download the code for the interfaces SimpleObservable and SimpleObserver from the demo code folder on Fronter.

2. Write a class ElectronicDevice that can change between the following states: on, off, and hibernate (the state should be represented with an instance variable called "state" of type String, and there should be a getState()-method returning a String with the state). The class should implement the interface SimpleObservable.

3. Write a class DiodeLight which is observing the changes of state in the ElectronicDevice class. The DiodeLight class should implement the interface SimpleObserver. The constructor of DiodeLight should take a SimpleObservable as its parameter, and call the registerObserver method of it with "this" as an argument. The update-method of DiodeLight should print the following to the console, depending on what the new state of ElectronicDevice is: if it is "on", it should write: "The diode is now green". If it is "hibernate", it should write "The diode is now red", and if it is "off", it should write "The diode is turned off". Use the equals-method to compare the Strings.

4. Write a class PowerUsage which is also observing ElectronicDevice. Class PowerUsage should implement the interface SimpleObserver. The constructor of PowerUsage should take a SimpleObservable as its parameter, and call the registerObserver method of it with "this" as an argument. The update-method of PowerUsage should print the following to the console, depending on the new state in ElectronicDevice ("on" -> "The power usage is normal", "hibernate" -> "Power save mode", and "off" -> "No power usage").

5. Write a setState-method in ElectronicDevice that changes the state of the object. The setState-method should take a String as its parameter, indicating the new state. When the state is changed, ElectronicDevice should notify its observers. To manage the registered observers inside ElectronicDevice, use an ArrayList.

6. Write a test class, and instatiate an ElectronicDevice object, a DiodeLight object, and a PowerUsage object, adding DiodeLight and PowerUsage to the ElectronicDevice's set of observers.

7. Test to see whether your classes behave as you expect them to when calling the setState-method of ElectronicDevice.