CPS122 - OBJECT-ORIENTED SOFTWARE DEVELOPMENT

Excerpts from an example of the Observer Pattern

```java
/** Display a GUI and allow user to change it.  This is an abstract class -
 *  concrete subclasses must specify conversion to/from celsius
 */
abstract class TemperatureDisplay extends Frame implements Observer
{
     private Label label;
     private TextField data;
     private Temperature temperature;

     /** Constructor
      *  @param description descriptive label to show
      *  @param temperature Temperature object whose value is to be shown
      */
     protected TemperatureDisplay(String description, Temperature temperature)
     {
          ...
          temperature.addObserver(this);// THIS DISPLAY WILL BE UPDATED WHENEVER
          ...                           // ANY DISPLAY CHANGES THE TEMPERATURE
     }

     /** Convert value in system used by this object to celsius
      *   @param value value to convert
      *   @return equivalent celsius value
      */
     protected abstract double toCelsius(double value);

     /** Convert a celsius value to system used by this object
      *   @param celsius celsius value to convert
      *   @return equivalent value in system used by this object
      */
     protected abstract double fromCelsius(double celsius);

     /** Method required by Observer interface - update display when model is
      *  changed
      *   @param o the Observable object that changed
      *   @param arg the argument to the notifyObservers() method of object
      */
     public void update(Observable o, Object arg)
     {
          double newValue = ((Temperature) o).get();  // THIS METHOD IS CALLED
          data.setText("" + fromCelsius(newValue));   // WHENEVER THE TEMPERATURE
     }                                                // CALLS notifyObservers()

     /** Change the temperature recorded by the model as a result of a change
      *   made by the user to the value shown in this window.
      */
     private void changeTemperature()
     {
          try
          {
               double newValue = Double.valueOf(data.getText()).doubleValue();
               temperature.set(toCelsius(newValue));
          }
          catch(NumberFormatException exception)
          {
               data.setText("Malformed number");
               data.selectAll();
          }
     }

}
```

```
/** Concrete implementation of TemperatureDisplay for Fahrenheit system
 */
class FahrenheitDisplay extends TemperatureDisplay
{
     public FahrenheitDisplay(Temperature temperature)
     {
          super("Fahrenheit", temperature);
     }

     protected double toCelsius(double value)
     {
          return (value - 32) * 5 / 9;
     }

     protected double fromCelsius(double celsius)
     {
          return 32 + 1.8 * celsius;
     }
}

/** Concrete implementation of TemperatureDisplay for Celsius system
 */
class CelsiusDisplay extends TemperatureDisplay
{
     ...
}

/** Concrete implementation of TemperatureDisplay for Kelvin system
 */
class KelvinDisplay extends TemperatureDisplay
{
     ...
}
```

```
/** Store a temperature and make it available for access/update by GUI
 */
class Temperature extends Observable
{
     private double value;              // recorded internally in Celsius
     ...
     /** Accessor for temperature
      *   @return temperature (in celsius)
      */
     public double get()
     {
          return value;
     }
     /** Mutator - set temperature to new value
      *   @param celsius new temperature (in celsius)
      */
     public void set(double celsius)
     {
          value = celsius;
          setChanged();                 // THESE CAUSE ALL OBSERVING DISPLAYS
          notifyObservers();            // TO BE UPDATED
   }
}
```