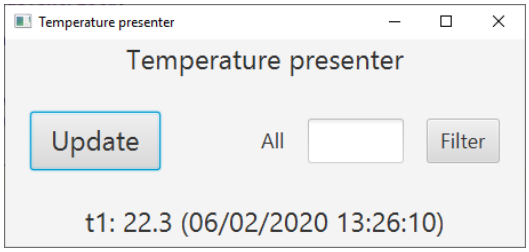
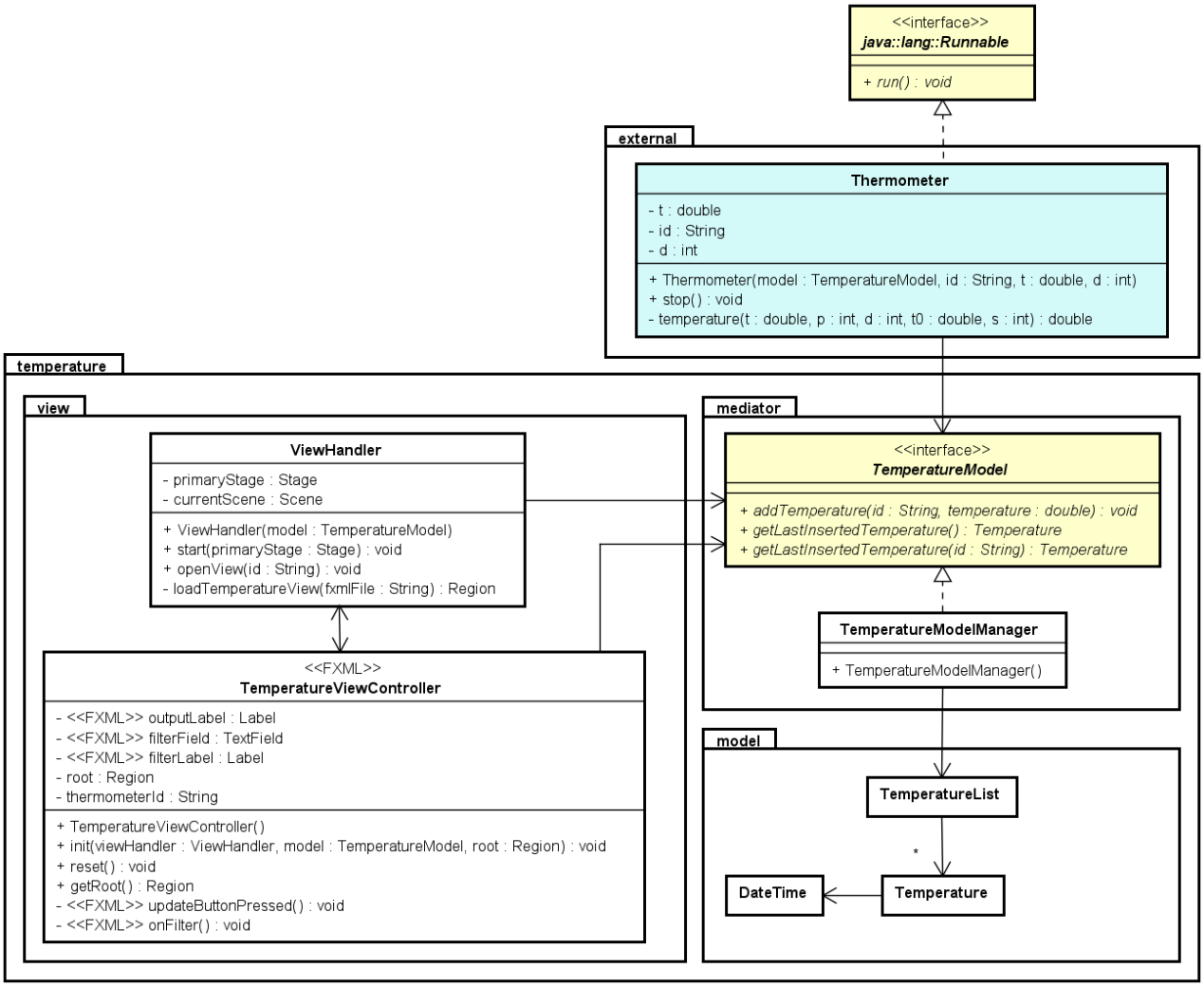
Exercise: Temperature presenter



The purpose for this exercise is to present temperature data generated by an external thread. Two applications are given: 1) A simple GUI as the one shown (without observers), and 2) an external system using threads and printing out the current temperature in given time intervals. The two applications are not combined into one system (yet)

Class diagram for the full system (Temperature presenter Model and GUI - and external system / Thermometer thread is shown below.



Step 0: Model and GUI classes

Copy classes into an IntelliJ module. Run the application to see the GUI, and note that there are no data to present (the model is never updated)

Step 1: Create the external system

Copy class Thermometer into the same module, and make the following changes (see also class diagram above)

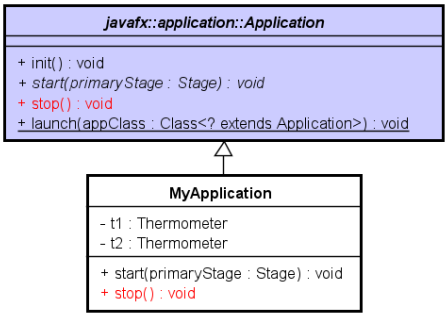
- Add a model (`TemperatureModel`) instance variable and modify the constructor to take a model reference and initialize this extra instance variable.
- Modify the `run` method to call the `addTemperature` method when a new temperature has been calculated (and check if method `addTemperature` is thread safe / synchronized)
- Update class `MyApplication` to create and start two threads with `Thermometer` objects as argument (and with id's "t1" and "t2").

Run the application again and click the Update button to verify that the data is shown in the GUI. Insert the id for one of the thermometers in the text field and click the filter button, and verify after clicking the Update button a couple of times that you only see temperatures for this thermometer. If you click the filter button with an empty field or the string "All", temperatures for all (both) thermometers are shown.

Step 1B: Stop the external threads when the GUI ends

In class `MyApplication` (see class diagram)

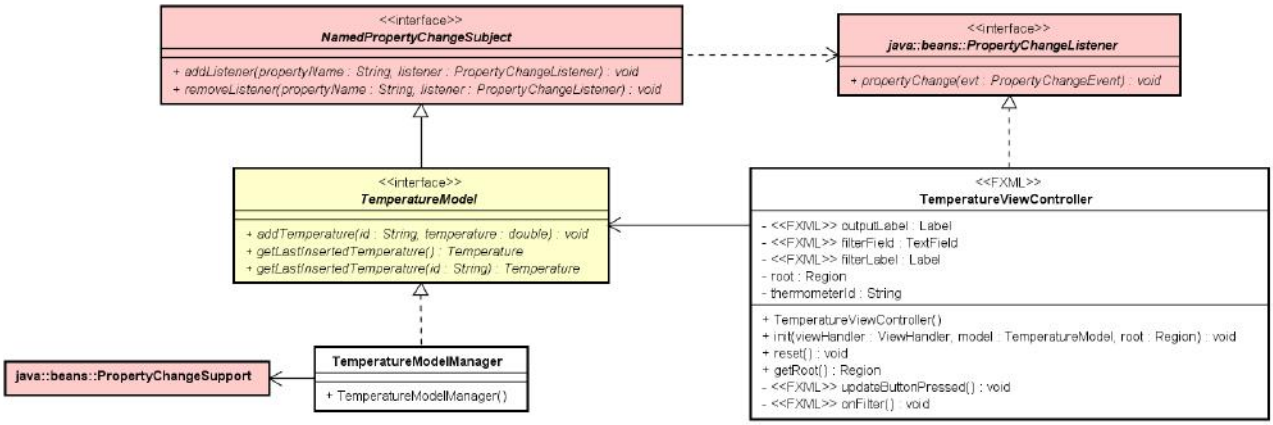
1. Declare the two Runnable objects as instance variables (instead of local variables in the `start` method)
2. Add a method `stop` (overriding the method from `Application`) in which you call `stop` for each of the two Runnable objects (which will end their `run` method and thus, terminating the threads)



Background: Class `MyApplication` (extending `Application`) overrides the abstract `start` method from `Application`. There are two empty methods in `Application`, `init` and `stop`. Method `init` will be called before the first window opens, then method `start` is called. When the last window closes, the `stop` method is called. In method `stop` we may clean up – in this case close the external threads.

Step 2: Observer

The purpose is to implement the Observer pattern in order to make automatic updates in the GUI every time the model gets a new value. Implement it exactly as shown in the class diagram below:



Notes to the class diagram:

- `TemperatureModel` (interface) is extending an interface with methods to add and remove a listener for a specific `propertyName` (e.g. `NamedPropertyChangeSubject`)
- `TemperatureModelManager` now has the two extra methods from the interface. Define a `PropertyChangeSupport` instance variable, initialise it in the constructor and delegate to this 1) when implementing the two methods, and 2) when firing an event in `addTemperature`
    - When implementing the add and remove listener methods you could define listening to all events as a `null` propertyName, i.e. like the following for add (and similar for remove)

```
@Override public void addListener(String propertyName,
    PropertyChangeListener listener)
{
  if (propertyName == null) // all events
  {
    property.addPropertyChangeListener(listener);
  }
  else // a specific event
  {
    property.addPropertyChangeListener(propertyName, listener);
  }
}
```

- `TemperatureViewController` is implementing `PropertyChangeListener` with the method `propertyChanged` – in which you update the label with the value from the event parameter variable. *Note that this statement has to be wrapped into a Platform.runLater.*
- Figure out where in the `TemperatureViewController` you are going to add and remove it as listener to the model.

External thread (without connection to the model)

```
package external;

public class Thermometer implements Runnable
{
  private double t;
  private double t0;
  private int p;
  private String id;
  private int d;
  private boolean running;
  private Thread runningThread;

  public Thermometer(String id, double t, int d)
  {
    this.id = id;
    this.t = t;
    this.d = d;
    this.p = 2;         // heaters power {0, 1, 2 or 3}
    this.t0 = 0.0;      // outdoor temperature
  }

  @Override public void run()
  {
    running = true;
    runningThread = Thread.currentThread();
    while (running)
    {
      try
      {
        int seconds = (int) (Math.random() * 4 + 4);
        Thread.sleep(seconds * 1000);
        t = temperature(t, p, d, t0, seconds);
        System.out.printf(id + " %.1f\n", t);
      }
      catch (InterruptedException e)
      {
        //
      }
    }
  }

  public void stop()
  {
    running = false;
    if (runningThread != null)
    {
      runningThread.interrupt();
    }
  }

  /**
   * Calculating the temperature measured in one of two locations.
   * This includes a term from a heater (depending on location and
   * heaters power), and a term from an outdoor heat loss.
   * Values are only valid in the outdoor temperature range [-20; 20]
   * and when s, the number of seconds between each measurements are
   * between 4 and 8 seconds.
   *
   * @param t  the last measured temperature
   * @param p  the heaters power {0, 1, 2 or 3} where 0 is turned off,
   *           1 is low, 2 is medium and 3 is high
   * @param d  the distance between heater and measurements {1 or 7}
   *           where 1 is close to the heater and 7 is in the opposite corner
   * @param t0 the outdoor temperature (valid in the range [-20; 20])
   * @param s  the number of seconds since last measurement [4; 8]
   * @return the temperature
   */
  private double temperature(double t, int p, int d, double t0, int s)
  {
```

```
    double tMax = Math.min(11 * p + 10, 11 * p + 10 + t0);
    tMax = Math.max(Math.max(t, tMax), t0);
    double heaterTerm = 0;
    if (p > 0)
    {
      double den = Math.max((tMax * (20 - 5 * p) * (d + 5)), 0.1);
      heaterTerm = 30 * s * Math.abs(tMax - t) / den;
    }
    double outdoorTerm = (t - t0) * s / 250.0;
    t = Math.min(Math.max(t - outdoorTerm + heaterTerm, t0), tMax);
    return t;
  }

}
```

Test program for the external system (to be deleted)

```
import external.Thermometer;

public class MainTemperature
{
  public static void main(String[] args)
  {
    Thermometer thermometer1 = new Thermometer("t1", 15, 1);
    Thread t1 = new Thread(thermometer1);
    t1.start();

    Thermometer thermometer2 = new Thermometer("t2", 15, 7);
    Thread t2 = new Thread(thermometer2);
    t2.start();
  }
}
```