Clocks

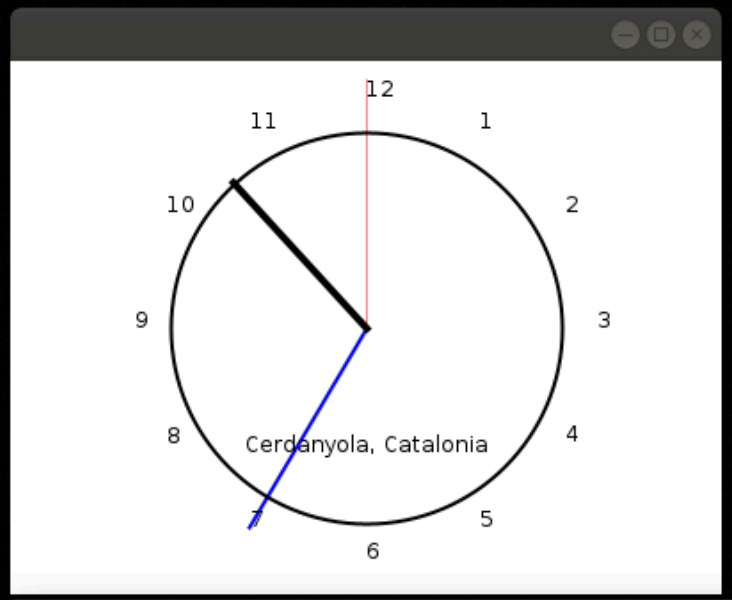
Goals

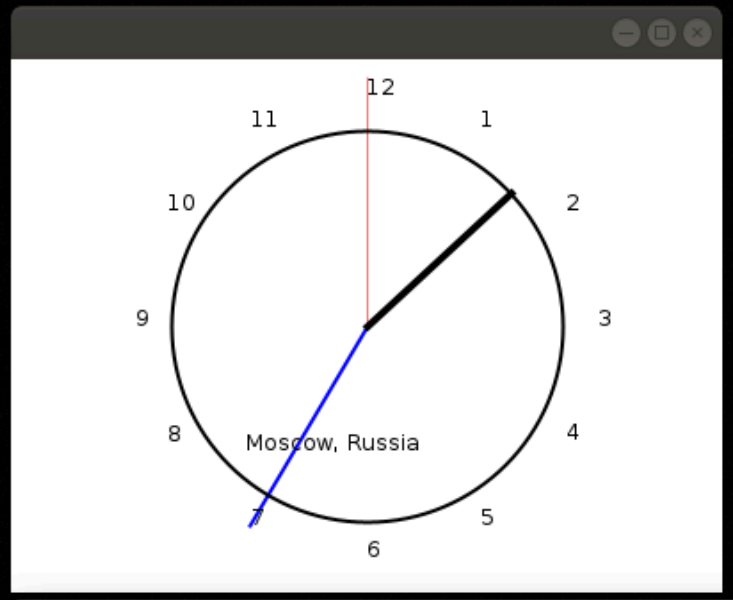
- Practice *Observer* pattern in Java
- Design and program **analog and digital clocks** separating UI from keepig time
- Extend the design to two new classes needing time notification: **stopwatch** and **countdown timer**

> Run solution

Activities solution-Main 1 2 Tue 22:35

Desktop2





15:35:00 4

Sonora, Mexico

13:35:00 4

Anchorage, Alaska

0:02:32:919

Stop

0:57:28

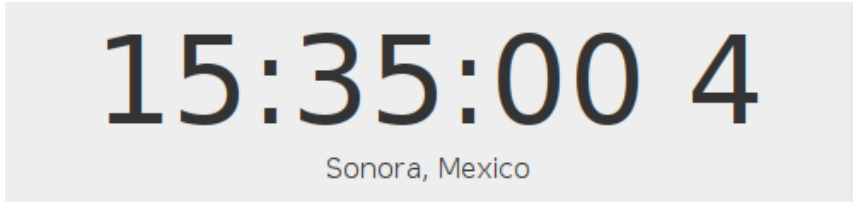
Trash

Code

We provide an initial code for an analog and a digital clock **here**.

```java
public class Main {
  public static void main(String[] args) {
    // this is how to initialize an arraylist
    List<Clock> clocks = new ArrayList<>(List.of(
        new AnalogClock(0, "Cerdanyola, Catalonia"),
        new DigitalClock(-9, "Anchorage, Alaska"),
        new AnalogClock(+3, "Moscow, Russia"),
        new DigitalClock(-7, "Sonora, Mexico"),
        new AnalogClock(-1, "Berlin, Germany"),
        new DigitalClock(-4, "Yerevan, Armenia")
    ));
    // every clock already works now, we just need to show it
    for (Clock c : clocks) {
      c.show();
    }
  }
}
```

The grahical user interface




```
public class DigitalClock extends Clock implements Runnable {
  private JLabel clockLabel;
  private final DateTimeFormatter formatter =
    DateTimeFormatter.ofPattern("H:mm:ss S");

  public DigitalClock() {
    panel = new JPanel();
    clockLabel = new JLabel(); // text of the time now, hh:mm:ss d
    panel.add(clockLabel);
    JLabel placeLabel = new JLabel();
    placeLabel.setText(worldPlace);  // "Sonora, Mexico"
    panel.add(placeLabel);
  }
  ...
  clockLabel.setText(LocalDateTime.now().format(formatter));
```

`JPanel` is a class of `java.swing`, a container of UI elements, here two texts.

```
public void show() {
  JFrame frame = new JFrame();
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frame.add(panel);
  frame.pack();
  frame.setVisible(true);
}
```

Puts the `JPanel` inside another container, `JFrame`, a window and displays it

The timer

Each individual clock contains its own **timer** to periodically ask for the system time and update the user interface.

```
public class AnalogClock extends Clock {
  ...
  private void run() {
    new Timer(repaintPeriod, e -> panel.repaint()).start();
    // calls paintComponent()
    // this Timer is *not* from java.util but from java.swing
  }
  @Override
  public void paintComponent(Graphics g) {
    ...
    LocalDateTime now = LocalDateTime.now().plus(hoursOffsetTimeZone,
      ChronoUnit.HOURS);
    // repaint the clock with the present time
    ...
  }
```

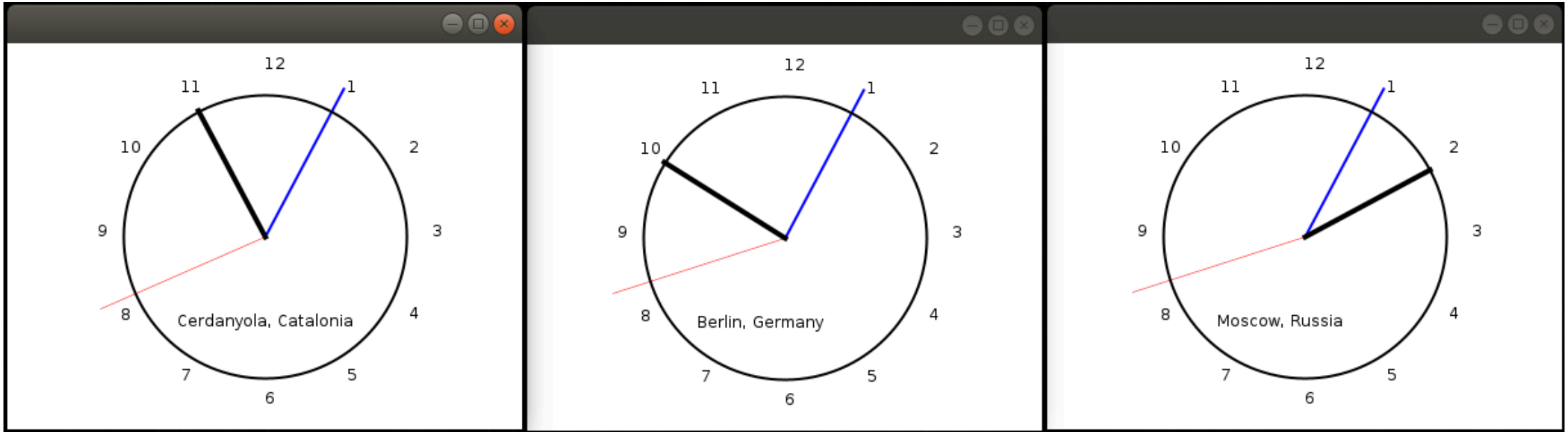
See doc. on the `LocalDateTime` class here

```
public class DigitalClock extends Clock implements Runnable {
  @Override
  public void run() {
    Timer timer = new Timer(repaintPeriod, new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent event) {
        updateClockLabel();
      }
    });
    timer.start();
  }
  public void updateClockLabel() {
    LocalDateTime now = LocalDateTime.now().plus(hoursOffsetTimeZone,
      ChronoUnit.HOURS);
    if ((lastTimeRepaint == null)
        || (now.minus(repaintPeriod, ChronoUnit.MILLIS)
            .isAfter(lastTimeRepaint))) { ...
```

`java.util.Timer` constructor accepts a a special object with an `actionPerformed()` method to be *automatically* invoked every `repaintPeriod`. This object is from an **anonymous class** that overrides `actionPerformed()`.

This is **inefficient** if the number of running clocks is large, because each timer object is a thread.

Worse, the clocks **do not seem synchronized**: they ask for the time periodically but at different time instants, since each program starts executing at a different fraction of second.



Lastly, it is **unnecessary**, with the observer pattern you can use a single timer.

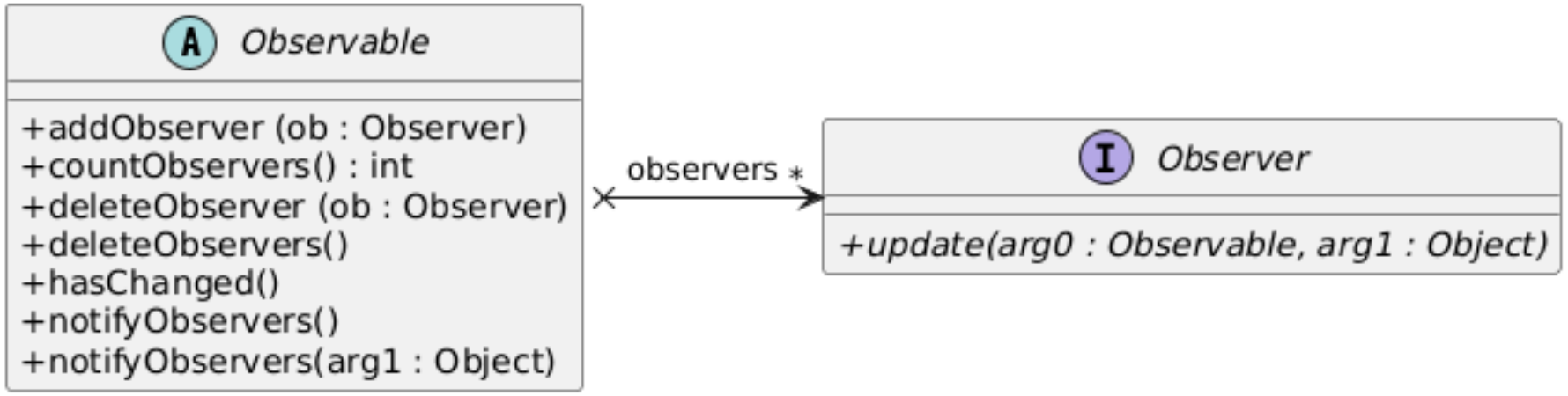
Observer pattern

The first step is to **separate time keeing from clock display**, in a `ClockTimer` class and `AnalogClock` and `DigitalClock` classes, respectively.

With the *observer pattern* :

- clocks observe a unique **clocktimer**
- the clocktimer keeps the time and **periodically notifies** them the present time, periode = 100 milliseconds for instance
- clocks know **how** (repaint the clock) and **when / if** (every second, every tenth of second... since last update) to update themselves

Pattern classes in `java.util`



- `Observable` is an abstract class with all these methods implemented
- concrete observables inherit them as they are
- `notifyObservers()` just passes the observable to `arg0` of `update()`
- `notifyObservers(arg1)` passes *also* `arg1` to `arg1` of `update()`
- `update()` is abstract that must be implemented by concrete observers

- who is observable ?
- who is observer ?
- what goes inside `update()`
- what does the observable pass to its observers ?
- draw the UML class diagram
- avoid redundancy in `DigitalClock` and `AnalogClock`, moving the common things to `Clock`

Hint: how to make a clock timer

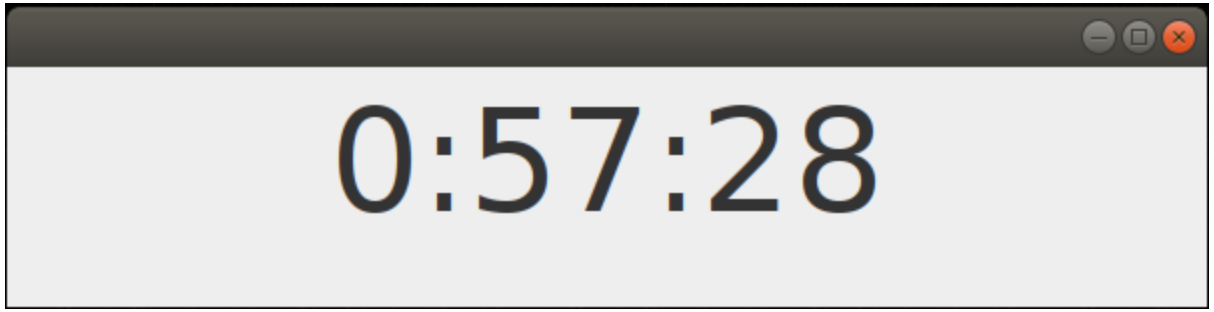
```
public class ClockTimer {
  private Timer timer;
  private int period; // milliseconds

  public ClockTimer(int period) {
    this.period = period;
    // digital and analog clocks receive the time, but stopwatches prefer the
    // period to add to the elapsed time

    TimerTask timerTask = new TimerTask() {
      @Override
      public void run() {
        setChanged();
        notifyObservers(LocalDateTime.now());
      }
    };
    // see https://www.baeldung.com/java-timer-and-timertask
    timer = new Timer();
    timer.scheduleAtFixedRate(timerTask, period, period);
    // delay is not 0 because then CountdownTimer wouldn't show the initial duration since
    // clocktimer calls update at once
  }

  public int getPeriod() {
    return period;
  }
}
```

Hint: how to check if it's time to repaint the time ?

```
private boolean isTimeToRepaint(LocalDateTime now) {
  return (this.lastTimeRepaint == null) // the first time
      || (now.minus(this.repaintPeriod, ChronoUnit.MILLIS)
         .isAfter(this.lastTimeRepaint)); // sufficient time has passed
}
```

Countdown timer

Now we can easily add a new type of observer, a countdown timer : given a time *duration*, like 1 minute, start a countdown until 0 secs. :



Hints

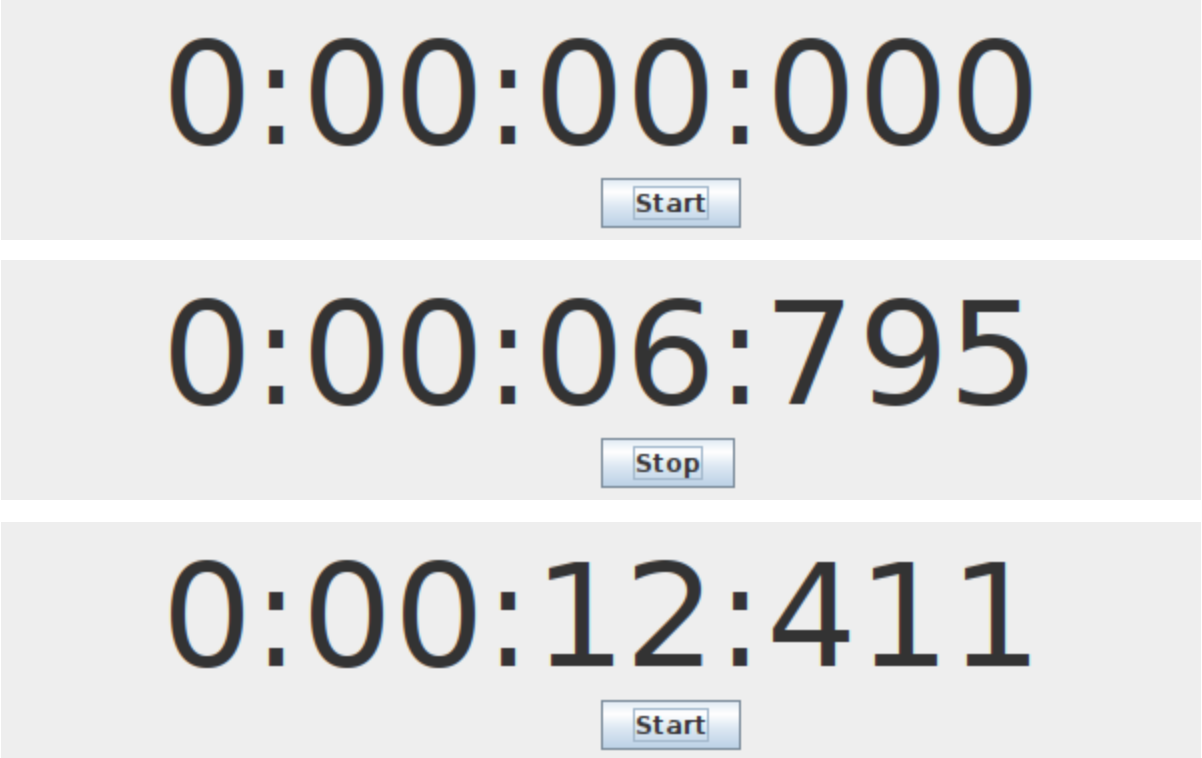
- Use class `java.util.Duration`, with static methods `Duration.ofDays()` etc., see here
- convert a `Duration` to string with

```
String formattedCountdown = String.format("%d:%02d:%02d",
  countDown.toHours(),
  countDown.toMinutesPart(),
  countDown.toSecondsPart());
```

- you just need to repaint the duration every second
- when reaching `0:00:00`, how to avoid a negative duration ?

Stopwatch

- starts at `00:00:00:000` ie, hours to milliseconds
- when you click a button it starts counting
- click again on it and will pause, click again and restarts etc.



Hints

- How to add a button and make it do something when pressed ?

```
startStopButton = new JButton("Start");
startStopButton.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    startStopButtonPressed(); // you program it
  }
});
panel.add(startStopButton);
```

- `Stopwatch` needs a reference to the clocktimer object
- What does it mean to start and stop the countdown timer ?
- Set the period of clocktimer to 9 millis and repaint countdowntimer every 10 millis for instance

Deliverables

- authors file
- source code
- detailed PlantUML class diagram with all classes + exported to .png
- small video or gif to show it works

Grading

- Analog and digital clocks, 1/2 of the exercise points
- clocks plus stopwatch and countdown timer, 1/2 of points