23.11.2015

Nils Gawlik

Lab Report for the 17th November 2015

Exercise 1:

I started with the implementation of the clock display which has an internal 12h format.

First of all I had to change the hours number display to a 12h version. To do this I first of changed it so the hours attribute is initialized with the argument 12 instead of 24. This means that the display will now roll over at 12.

This already makes for a pretty good clock. But it still is not complete without an am/pm display and a corresponding internal representation of if it's am of pm at the time.

To archieve this I created the isAm attribute of type boolean. This attribute saves if it is currently am (00.00 – 11.59). If isAm is false it is pm (12.00 – 23.59).

I then had to adapt a lot of other methods to work with the new system.

First off was setTime. I added an isAm argument on top of the hour and minute arguments. The method sets the attribute isAm to the given value and enables the user (or other parts of the program) to set the time in an am/pm format (still using 0 for 12, since the number display works this way. In retrospect you might want to also accept 12, but that feature is postponed for now).

Next up I change the constructor to require an extra isAm argument, which I pass to the setTime method used in the constructor.

Now it was the timeTick-method's turn. I had to change the method so it would also roll over the isAm attribute every 12 hours. To archieve this I nested another if-clause in the first one (the one that increments the hours). This if clause checks if hours is zero (right after hours.increment()) This means it will only be true if hours just rolled over. Inside the if clause I toggled isAm, that means I set isAm to !isAm. This way when the 12-hour display rolls over am is set to pm or pm is set to am – which is excactly the desired result.

I now took the time to do test it a little bit more thoroughly. I tested entering different times and set the time to 11.59 and then incremented it to see if isAm gets toggled correctly. I checked the results in the BlueJ object inspector and everything worked as expected.

Next up was the display. For this I only had to change the updateDisplay method. I used the ternary operator to add an " am" to the diplayString if isAm is true and an " pm" otherwise.

After some testing I realized that my current display displays 0 instead of 12 for the hours – for example 00.10 am. To fix this I used another ternary operator to replace the hours.getDisplayValue(). The operator checks if hours != 0 and if true just returns hours.getDisplayValue() (as it was before) but in the other case (that is if hours is 0) it returns 12. The getTime method now returns a sensibly formatted string.

Another bug I found while testing is that the default value for isAm is false, which means that clocks using the standard argument-less constructor will start at 12.00 pm. This is not a bug in a strict sense, but it is a different behaviour that the 24h display so I decided to fix it by just initializing isAm as true right after declaration.

The clock was now functional. I did some further testing at this point trying different times and seeing if the clock displays them properly. I also used a for loop in the code pad to increment the Clock a 100 times at a time, to test the roll-overs. In the end I found no further bugs.

<u>Exercise 2:</u>

I first of all re-downloaded the project and copied a fresh ClockDisplay.java file over into the old project (not before renaming it to ClockDisplay2). I also had to rename the class inside the file and rename the constructors, but then I was ready to go for the alternative implementation with an internal 24h format.

I realized, that I should not have to change any functions than the updateDisplay function. Since all the time-keeping is done in a 24h format internally, nothing should change at all except for the display of the clock.

I did the am/pm display part in a very similar fashion to the first implementation by using a ternary operator, only this time the condition was hours.getValue()>=12 (It was >12 at first, but I later found that bug and changed it to >=12).

I realized, that turning the 24h variable into a 12h representation will take a few lines of code, so I decided to put it into an extra method called customHourDisplayValue. This method has no arguments (because it directly uses the hours variable) and returns a string which will be displayed instead of hours.getDisplayValue().

At this point it might be worth mentioning, that I used so much custom formatting that it probably does not make sense to use the getDisplayValue method of NumberDisplay at all.

It also kind off clashes with the whole model: If the NumberDisplay object would be an actual physical display I would not be able to just modify the string to my liking – you would actually have to modify the display and replace the 0 with a 12. On the other hand: The task specifically tells me to modify the formatting, so I guess it is all right.

Anyway, in the customHoursDisplay method I used a slightly overcomplicated code:

```
        int n = ((hours.getValue()+11) % 12) + 1;
        if(n < 10)
           return "0" + n;
        else
           return "" + n;
```

This code is a little bit hard to explain. The first line does the job of replacing the 0 with a 12 and converting from 24h to 12h. Keeping the 12 is achieved by first adding 11 (which is equivalent to subtracting one because of the modulo) and then adding 1 after the modulo is done. This makes the 0 a 12.

The other lines just mimic the format of the getDisplayValue method in NumberDisplay, adding a leading zero to the number.

The second clock display was now finished and I found no bugs while testing it afterwards.

<u>Comparison: method 1 vs. method 2</u>

I think the second method where I used the internal 24h representation was overall better. A smaller number of variables seems more easy to handle and generally saves code and work. I noticed, that the internal-24h version was implemented much faster than the internal-12h version.

It should also be mentioned, that since the base code is already a 24h clock it is obviously more work to rewrite the whole class to be a 12h display, than to just adjust a few lines of formatting code.

One disadvantage of the internal 24h clock is that the setter methods and the constructor all still require a 24h input format. This is not friendly to USA/UK users, so rewriting the input to work for am/pm might be necessary. But I would also expect that this would be handled separately in a user interface class and the methods in ClockDisplay can stay the same.

On a final note, looking at my first point it would be even better if you only used a single variable for the time – the number of minutes since 00:00 maybe – and did all the formatting in the display. This would also make the NumberDisplay class mostly obsolete, but I honestly (especially in the end) felt like the NumberDisplay objects were more of a nuisance to work with and were not very helpful in implementing the clock display.

<u>Exercise 3:</u>

I implemented the alarm in NumberDisplay2 because I liked that version more as I explained.

I made the variables alarmMin and alarmHrs (in retrospect alarmHr would have been a better name, since it is only one specific hour) to represent the alarm. Both are initialized as -1. -1 means that the alarm is off (it will never be -1 o'clock).

I added the method setAlarm which sets both alarm variables to the arguments of the method, and I added the ring method which represents the clock ringing; at the moment it prints "Riiiiiiing" to the console.

To trigger the alarm I had to put if-clause at the end of the timeTick method, which checks if hours.getValue() is alarmHrs and minutes.getValue() is alarmMin. If true the ring method is executed.

I tested it and it worked fine.

23.11.2015

Nils Gawlik

<u>Execise 4:</u>

To make the clock tick I had to use Thread.sleep() and a while loop. This also requires you to catch a InterrruptException using a try/catch construction. The number of milliseconds to sleep was 60 000.

I already know about Exceptions from school, so it wasn't to difficult to get it to work.