UML for Software Engineering

Labor Exercise

Clock, Stopwatch and Timer

While the popularity of smartphones increases more and more, there is rising up a new hype in the world of mobile gadgets. The company Smar™ will join this trend and wants to develop a digital stop watch. The CEO Dr. Chr. Onometer sums its ideas to this - admittedly simple – software with the following requirements, which first have to be realized as a prototype using UML and Java.

The clock has three modes: Clock, stopwatch and timer. One button is used to switch between the modes from Clock -> Stopwatch -> Timer -> Clock… Switching the mode resets the recently active mode. The initial active mode when turning the system on is the clock mode.

The following sections describe the requirements to the modes in detail.

Clock

The clock mode will always show the current time of the day on a display in the following format: hh:mm:ss (i.e.: 16:04:02). The clock will count seconds itself and every second the displayed time is refreshed.

When the whole application is initialized (not every time when entering clock mode), the current time is retrieved from a radio unit, which receives a DCF-77 signal. However, when in clock mode, it is also possible for the user to set the time to another value using an UP-Button and a Down-Button. If the UP-Button and DOWN-Buttons are pressed more than two seconds, the time increases/decreases for one second every 100ms.

Switching modes will not result in stopping the clock from counting seconds and increasing the current time, but the current time is not shown on the display anymore.

Stopwatch

The stopwatch shows the expired time in hours, minutes and seconds. Therefore, a display shows the minutes and seconds and a colon between these values. While the stopwatch runs, the colon flashes every second with a frequency of 1Hz. The stopwatch is activated and deactivated by a STOP-Button. The time begins to run, if the STOP-Button is pressed. If the STOP-Button is pressed again, the time stops. If the STOP-Button is pressed again, the time continues to run. To reset the time, the STOP-Button has to be pressed two times within two seconds.

In addition to the graphical display, a status LED flashes with a frequency of 1Hz during the time runs. If the stop watch is stopped, the LED is on. After a reset or in the initial state, the LED is off. When switching modes, the stopwatch is stopped (but the current stopwatch time value is not resetted when leaving the mode or entering the mode next time).

Timer

The timer allows to count down seconds from a configured start time. If the timer mode is active, the time will also be displayed with hours, minutes and seconds and the colon between these values. The colon flashes as described in the stopwatch mode. The start time of the timer can be configured using the UP-Button and DOWN-Button. Each press in one of these direction will increase or decrease the time adequately. If the UP-Button and DOWN-Buttons are pressed more than two seconds, the time increases/decreases for one second every 100ms. The time can only have a value between 00:00:00 and 10:00:00.

The STOP-Button starts and stops the timer. When pressing the STOP-Button two times within two seconds, the time will be resetted to the last configured start time. While the time runs, the colon flashes as well as the status LED. While the timer is paused, the LED is on. After the time expired, the LED flashes with a frequency of 4Hz. After a reset or in the initial state, the LED is off. The behavior when switching modes is like the stopwatch mode.

Notes for the prototype realization:

- Display:
  - The display can show the time values in the different modes on the console using an operation like “System.out.print(…)”. There is no need for implementing a graphical user interface.
    - When using “System.out.print(..)”, you can use “\r” to go to the beginning of the line and overwriting something previously written on the console, for example like this: System.out.print("\rOn ");

- Radio Unit:
  - Of course, in the Java prototype implementation no real DCZ-77 signal is received. The radio unit will get the current time by calling an operation like “System.currentTimeInMillis()”

- Sending Events between objects:
  - An object can send an event *event1* to another object, which it references with an association (role name=itsObject1) like this: itsObject1.gen(new event1());

Labor report

- Your labor report has to contain all your diagrams
- Further, you have to describe the solution as well as the solution approach comprehensible
- The final labor report has to be finished at the end of the semester – before the date of the exam. However, it is highly recommended to begin or complement the report for each milestone to make clear the changes from one milestone to another during the realization.

Rating

Exam 60 %	**Labor exercise** 40 %		
	M1 10 %	**M2** 20 %	**Report** 10 %

Naming conventions

For a better readability, it is recommend to use the following naming conventions:

ev… Event
op… Triggered Operation

uc… Use Case diagram
sd… Sequence diagram
sc… State chart diagram
cd… Class diagram
od… Object diagram

Common hints

- Always keep your models consistent! Update formerly finished diagrams if necessary!
- Model as extensive as necessary, but also as simple as possible!
- Model or realize no functionality, which is not required or needed!
- Read the tasks and additional hints carefully before you start the realization!
- Save you project for each milestone separately. This allows you to continue with the next milestone even if the last one is not yet rated without the risk to break your current solution.
- **Ask questions *in time*, if something of the tasks is not clear!**

Milestone 1: Requirements analysis and design

Use-Case diagram

Identify all use cases from the view of the watch wearer in a Use-Case diagram.

For each use case, give a description, which describes the use case in detail. Enter the description in the comment field of a use case.

Use the following schema for the description

Preconditions:
Process:
Exceptions:
Postconditions:

Take care, *not* to describe a possible implementation at this moment.

Add also useful dependencies between the use cases and give them correct stereotypes.

Object diagram

Identify the objects of the software and model them in an object diagram. Use links to connect objects, which communicates with others.

Class diagram

Classify the objects to classes and model them in a class diagram. Use associations to connect classes which communicates with others. Then, specify the objects and links by the created classifiers and associations.

Sequence diagram / extension of the class diagram

Sketch three sequence diagrams which shows the communication between the objects. One describes the clock mode, one the stop watch mode and the other one describes the timer mode. Each sequence starts with the event which causes the activation for the corresponding mode.

Then, extend the classes in your class diagram with the appropriate events, triggered operations and (simple) operations, which you have identified in the sequence diagram before.

Milestone 2: Modeling Behavior

Statechart diagram

Identify the states for the classes of your system. Model a statechart diagram for each class.

Model the state transitions. If applicable, use the events and triggered operations, which you already identified by the sequence diagram.

When naming the states, care for expressive names.

Hint:
When identifying the states, a look at your sequence diagrams can be very helpful. Consider, which state of an object will be active, after it receives an incoming message.

Panel Diagram

Create a panel diagram with control elements for all the mentioned buttons. In addition, create an LED in the panel diagram for the led mentioned in the requirements. Bind all the elements in the panel diagram to variables, states or events to behave correctly.

Bonus: Create three digital displays in the panel diagram, each for showing the current time from the different modes.

System test / Simulation

Test your system. Therefore, first generate code from your model and make sure, it can be compiled without errors. Then, execute the system and use the simulation/animation capabilities of Rhapsody to examine its behavior. Make sure all the requirements are fulfilled.

Compare the execution behavior with the sequence diagrams you created in mile stone 1.