Mandatory
IKT413
2011
A heart rate monitor is a monitoring device that allows a person to measure her/his heart rate. Heart rate is measured in heartbeats per minute.

A step counter (pedometer) is a device that counts each step a person takes by detecting the motion of the person's hips (or in some other way). You are to start the development (modelling) of the software for a device monitoring heartbeats and number of steps.

Fig. 1 Watch that displays heartbeat rate and number of steps
a) The text being displayed inside the watch is handled by four different text fields; the text fields are of two different classes (types), the following simplified Java code describes the two classes:

```java
public class TextField {
    private String txt;
    public void setText(String t){}
}

public class IntTextField extends TextField {
    public int getIntTxt(){...}
    public void setIntTxt(int intTxt){...}
    public void addToIntTxt(int numberToAdd){...}
}
```

Class TextField is for displaying of alphanumeric text, e.g., texts “HB:” and “Steps:” are handled by such text fields (see Fig. 1). Class IntTextField is for displaying integer values, e.g., 132 and 4235 are handled by such text fields (see Fig. 1).

The method addToIntTxt(int numberToAdd) adds numberToAdd to the value displayed, e.g., if value displayed is 4235 and numberToAdd is 2 then 4237 will be displayed. The following simplified class is also given:

```java
public class Controller{
    private TextField txtUpperLeft, txtLowerLeft;
    private IntTextField intTxtUpperRight, intTxtLowerRight;
    ...
}
```

Propose a UML class diagram that describes the classes given above and how they relate to each others, i.e., include associations (with navigation) and multiplicity.
Task: a
b) The watch is equipped with memory and it may record the number of steps for selected time intervals. (This information may be transmitted to a PC for processing and display.) The following class diagram captures the information recorded:

![Class Diagram](image)

Fig. 2 Class diagram for step information

The following object diagram is given:

![Object Diagram](image)

Fig. 3 Object diagram that conforms to the class diagram
Proposed Java code:

```java
public class DateClass {
    int year;
    int month;
    int day;
}

public class TimeClass extends DateClass {
    int hour;
    int minute;
    Step lastStep;
    Step firstStep;
}

public class Step {
    int numberOfSteps;
    StepRegister stepRegister;
    TimeClass theEnd;
    TimeClass theStart;
}

public class StepRegister {
    Step[] step;
}
```
Representing the information of the object diagram in an XML document:

```xml
<?xml version="1.0"?>
<StepRegister name="StepRegister1">
  <step name="Step1">
    <numberofsteps="2500"/>
    <starttime year="2011" />
    <month="8" />
    <day="14" />
    <hour="15" />
    <minute="20" />
  </starttime>
  <endtime year="2011" />
  <month="8" />
  <day="14" />
  <hour="15" />
  <minute="55" />
</endtime>
</step>

<step Name="Step2">
  <numberofsteps="5700" />
  <starttime class="TimeClass" name="TimeClass3">
    <year="2011" />
    <month="9" />
    <day="2" />
    <hour="10" />
    <minute="15" />
  </starttime>
  <endtime class="TimeClass" name="TimeClass4">
    <year="2011" />
    <month="9" />
    <day="2" />
    <hour="11" />
    <minute="5" />
  </endtime>
</step>
</stepregister>
```
Representing the information of the object diagram in a relational database (use tables)

Table StepRegisterTable is not needed if we assume that there is only one step register.

<table>
<thead>
<tr>
<th>Step Register ID</th>
<th>Step ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>StepRegister1</td>
<td>Step1</td>
</tr>
<tr>
<td>StepRegister1</td>
<td>Step2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step ID</th>
<th>Step Register ID</th>
<th>Start Time</th>
<th>End Time</th>
<th>Number of Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step1</td>
<td>StepRegister1</td>
<td>TimeClass1</td>
<td>TimeClass2</td>
<td>2500</td>
</tr>
<tr>
<td>Step2</td>
<td>StepRegister1</td>
<td>TimeClass3</td>
<td>TimeClass4</td>
<td>5700</td>
</tr>
</tbody>
</table>

Time Class Table

<table>
<thead>
<tr>
<th>Time Class ID</th>
<th>Date Class ID</th>
<th>year</th>
<th>month</th>
<th>day</th>
<th>hour</th>
<th>minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimeClass1</td>
<td>DateClass1</td>
<td>2011</td>
<td>8</td>
<td>14</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>TimeClass2</td>
<td>DateClass1</td>
<td>2011</td>
<td>8</td>
<td>14</td>
<td>15</td>
<td>55</td>
</tr>
<tr>
<td>TimeClass3</td>
<td>DateClass2</td>
<td>2011</td>
<td>9</td>
<td>2</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>TimeClass4</td>
<td>DateClass2</td>
<td>2011</td>
<td>9</td>
<td>2</td>
<td>11</td>
<td>5</td>
</tr>
</tbody>
</table>

In our case Table DateClassTable is not needed since all information about times are found in table TimeClassTable.

<table>
<thead>
<tr>
<th>Date Class ID</th>
<th>year</th>
<th>month</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>DateClass1</td>
<td>2011</td>
<td>8</td>
<td>14</td>
</tr>
<tr>
<td>DateClass2</td>
<td>2011</td>
<td>9</td>
<td>2</td>
</tr>
</tbody>
</table>
This is an ER-diagram and not what we are looking for (it does not show the tables)
• Propose an OCL constraint that ensures that a TimeClass object is related to a Step object through either the endTime association or through the startTime association.

This constraint seems to be doing the job.

Context TimeClass inv:
lastStep.size() + firstStep().size() = 1
Propose an OCL constraint that ensures that a TimeClass object is related to a Step object through either the endTime association or through the startTime association.

C1: context step inv inv1:
    startTime -> size()=1 or endTime -> size()=1

C2: context Timeclass
    inv: self.endTime->NotEmpty() xor self.startTime->NotEmpty()

I am uncertain when it comes to using association name instead of role name (see C1 and C2); logically the role name can be deduced by selecting the opposite side of the association than the context.

C1: If using the interpretation described above then self.endTime->size() is the same as self.theEnd->size(). Both self.theEnd->size() and self.theStart->size() is already constrained to 1 by the multiplicity shown on the class diagram. There seems also to be other problems with this constraint…

C2: Is using xor and have an ok context. Assuming the interpretation above the this constraint seems to be doing the job.
Propose an OCL constraint that ensures that a TimeClass object is related to a Step object through either the endTime association or through the startTime association.

This constraint seems to be doing the job.

context TimeClass inv:
(firstStep->size() = 1 implies lastStep->size() = 0)
And
(firstStep->size() = 0 implies lastStep->size() = 1)
Propose an OCL constraint that ensures that the start time (which is given through the startTime association) is before the end time (which is given through the endTime association).

```ocltxt
context Step inv:
(theStart.year < theEnd.year)
or
(
  (theStart.year = theEnd.year)
  and
  (
    (theStart.month < theEnd.month)
    or
    (
      (theStart.month = theEnd.month)
      and
      (theStart.day < self.theEnd.day)
      or
      (theStart.day = theEnd.day)
      and
      (theStart.hour < theEnd.hour)
      or
      (theStart.hour = theEnd.hour)
      and
      (theStart.minute < theEnd.minute)
    )
  )
)
)
```
Task: c

- **Watch**
  - + getTime() : Time
  - + tick() : void
  - + oneMoreStep() : void
  - + oneMoreBeat() : void

- **Controller**
  - + on() : void
  - + off() : void

- **LoadSpeaker**
  - + on() : void
  - + off() : void

- **Button**
  - A
  - B
  - C
  - D

- **Clock**
  - + getTime() : Time

- **StepSensor**
  - HeartBeatSensor
Task: d
Task: e

[after some initialization]

oneMoreBeat() message received

lastHBTime = clock.getTimeInMilliSeconds();

hBInterval3 = hBInterval2;
hBInterval2 = hBInterval1;
hBInterval1 = lastHBTime – previousHBTime;
previousHBTime = lastHBTime;

tickCount = 0;
[tick() message received]

tickCount++;
[tickCount<3 AND tick() message received]

[tickCount>=3 AND tick() message received]

averageRate = (hBInterval1 + hBInterval2 + hBInterval3) / 3;
timeSinceLastHB = (clock.getTimeInMilliSeconds() – previousHBTime) / 1000;

[else]

[timeSinceLastHB<60]

heartBeatRate = 0;

[else]

If pulse is lower than 1 per min. then the system assumes there is an error!

heartBeatRate = 60000/averageRate;
intTxtLowerRight.setIntText(heartBeatRate)

page 16
Alternative solution to Task: e

The system starts by initializing variables. One interval will be the measured and used as the initial value of the three intervals (dt1, dt2, dt3) used for the average calculation. Additionally, \( t3 \) is initialized with the current time, needed to start the calculation loop in A.

This system uses fork of control to run things in parallel. Concurrency might be an issue on some platforms. To ensure safe concurrency, each calculation (\( dt1, dt2, dt3 \)) should acquire a lock on the same mutex before the calculation and unlock it at the end of the calculation. Additionally, a lock on the same mutex should be acquired before displaying and unlocked after displaying the average. Concurrent calculation and displaying is slow, but this is not a time-critical device.
Alternative solution to Task: e continues

The interval calculation A is dependent on 4 timestamps, one for each heartbeat, to make 3 time intervals. The average calculation itself is done in figure E.3.

By looping over 3 separate calculations, variables may be used in cycle to avoid moving data in memory, improving performance. This will use a little bit more memory, though worthwhile.

The intervals (dt1, dt2, dt3) are calculated from fixed pair of timestamp variables in \{t1,t2,t3\} (see Figure E.2). These variables are reused and updated. For instance;

- t1 will be the 1st, 4th, 7th and so on
- t2 will be the 2nd, 5th, 8th and so on
- t3 will be the 3rd, 6th, 9th and so on

So when t1 is the 4th, t3 is still the 3rd so that the fourth interval is t1-t3 as if it were t4-t3. The first interval t1-t3 is valid since t3 was initialized before the loop.
The display should be updated every 3 seconds with the average heartbeat rate. The calculation includes division, and because we cannot divide by zero, the conditional statement in Figure E.3 makes sure that we do not do so. The rate is initialized before the loop, and is not changed unless the divisor is more than zero.

The rate is calculated as $1 / \text{average}$ according to the task description. In this case, the average would be $(d1 + d2 + d3) / 3$ giving the rate calculation as $3 / (d1 + d2 + d3)$. Mathematical expressions should be simplified as much as possible to improve performance.

Even if the interval calculation is cyclic, the following rate calculations are equivalent:
- $3 / (d1 + d2 + d3)$
- $3 / (d2 + d3 + d1)$
- $3 / (d3 + d1 + d2)$

The system may be terminated at any time. There is no important data that needs to be properly handled before termination of the system.
Task: f

Using Structured Classifier Notation:
Parts connected via ports and Controller composed of HBIntervalUpdater and HBUpdater.

This is not part of Task f, but helps to describe it!
This system as modelled in F, have parallel execution. To get a better overview, the sequence diagram has been split into two distinct executions: figure F.1 and F.2.

The execution is actually loop, so it would be possible to use a loop construct instead.

Alternative solution to Task: f

**Figure F.1** Partial sequence diagram describing the interval calculation.
This is the display loop. The average calculation is done within the controller before put into the text field.

Alternative solution to Task: f continues…

**Figure F.5** Partial sequence diagram describing the display update loop.
Task: g

Again, the system have parallel executions. The communication diagram is split into two: figure G.1 and G.2.

**Figure G.1** Partial communication diagram describing the interval calculation.

**Figure G.2** Partial communication diagram describing the display update.
Figure H.1 Use-case diagram describing the alarm configuration.
[1] Structured Classifier

- [Ref.Man.] A classifier containing **parts** or **roles** that form its data structure and realize its behaviour.

- [Ref.Man.] A Structured class is the description of the internal implementation of a class. A structured class owns its ports, parts, and connectors. The class and its ports may have interfaces...
[2] Composite Structure Diagrams

In UML models, a composite structure diagram depicts the internal structure of structured classifiers by using parts, ports, and connectors.

The example in the figure shows how the composite structure diagram identifies the containing classifier, Car. The diagram frame shows four internal composite parts of the containing classifier, which represent the four wheels of the car and are of the type Wheel. A communication link connects the front wheels and the rear wheels with connectors named frontaxle and rearaxle.
[2] Parts & Connectors

• In composite structure diagrams, a part is a diagram element that represents a set of one or more instances that a containing structured classifier owns. A part describes the role of an instance in a classifier.

• In UML diagrams, a connector is a line that represents a relationship in a model. When you model the internal structure of a classifier, you can use a connector to indicate a link between two or more instances of a part or a port.
[2] Ports

• In composite structure diagrams, a port defines the interaction point between a classifier instance and its environment or between the behavior of the classifier and its internal parts.
In the figure, the Car class contains two internal composite parts: rear:Wheel[2], which represents the two rear wheels of a car, and e:Engine, which represents the engine of the car. The rearaxle connector links the engine of the car to the instances in the set named rear:Wheel.

In the figure, the class named Boat contains a part named Propeller, which is connected by the shaft connector to the port of the part named e:Engine. Although the part e:Engine has the same type name in both the Car and Boat classes, the parts are different instances and each one belongs to a different containing classifier.
From [3]:

Alternative notation [3]:

MyCar: Car

<table>
<thead>
<tr>
<th>E1 / e: Engine</th>
<th>powers</th>
<th>W1 / front: Wheel</th>
<th>W3 / back: Wheel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

YourBoat: Boat

<table>
<thead>
<tr>
<th>E2 / e: Engine</th>
<th>powers</th>
<th>P1 / p: Propeller</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page 30
From [4]:

**Search Port provides Search Books and Search Video interfaces and requires Inventory interface**

The **required interfaces** of a port characterize the requests that may be made from the classifier to its environment through this port. The **provided interfaces** of a port characterize requests to the classifier that other classifiers may make through this port.

Port is shown as a small square symbol. The name of the port is placed near the square symbol. The port symbol may be placed either overlapping the boundary of the rectangle symbol denoting that classifier or it may be shown inside the rectangle symbol.
Task: i

It seems more correct to have :StepSensor and :HeartbeatSensor outside the watch – since this is were they are placed (and not inside the watch as the figure shows)!

**Figure I.1** Structured classifier diagram describing the system.
Alternative with more details - see next slide for more information about the operations of the interfaces.
Figure J.1 Component interfaces.

Figure J.2 Component diagram using interfaces. Interfaces are only needed when a component is expected to be able to perform some operation. The sensor components are not expected to do any operations. However, it is expected that the components listening to the sensors can perform operations when the sensors report data. The clock is an exception because the clock should additionally provide an operation to get the current time. Even if interfaces may not be needed, it is good practice to always provide an interface in both directions if the system could grow in the future.
References


