The University of Melbourne

SWEN90010: High Integrity Software Engineering

Assignment 1

Due Date: 11:59pm, Sunday 20 March, 2016

1 Introduction

This assignment is worth 10% of your total mark.

The aim of this assignment is to provide you with an opportunity to implement a simplified safety- and security-critical system in Ada. Doing so will help to explore the properties of Ada, and how they relate to safe programming.

Tip: Perhaps the most difficult part of this assignment is getting your head around the system itself, and the packages already supplied. Be sure to read this carefully before you start, and to understand the example scenario and how its uses the interfaces of the supplied packages.

2 System overview

The system to be produced as part of the assignment is part of a fictional fitness monitoring device called the *FatBat*[1].

A FatBat looks like a wristwatch and includes functions for monitoring the wearer's health and uploading this information into a cloud-based system. It monitors heart rate and uses GPS locations and footsteps to measure a person's activity and fitness.

The FatBat has four intended features:

1. to record a person's vital signs
2. to inform emergency services when a person suffers from a cardiac arrest (as judged by irregularities in heart beat),
3. to share some limited information with an insurance company, in order to reduce premiums, and
4. to share information with friends for social reasons; *e.g.* competitions to see who can take the most steps.

Each FatBat's data is stored in the cloud. Each user can set access permissions to their own data, described below.

You are tasked with designing and implementing part of the FatBat cloud user administration sub-system and its cloud storage that communicates just the right amount of information to just the right people at the right times.

[1]This device is fictitious and any similarity to existing products is entirely coincidental

3 Requirements

3.1 Users and roles

Every person who uses the system must have a user identity, represented as a non-negative integer.

Each user can play zero of more of the four following roles in the system:

1. *Wearer*: A user wearing a FatBat.
2. *Emergency*: The Emergency services. There is only one user ID assoicated with the emergency services role at any time. This user ID is always 0.
3. *Insurance*: An insurance company providing services to zero or more users.
4. *Friend*: Users part of a social circle for a given user. For simplicity in this assignment, each user has at most one friend.

3.2 Data

Each FatBat has a user id and three types of personal data that can be sent to and stored on the cloud system:

1. location: provided by an internal `GPS` locator;
2. footsteps: a daily `footsteps` count;
3. vitals: information provided about the users vital signs, including heart rate. For the purposes of this assignment, the specific details of the vitals are not relevant.

For simplicity in this assignment, the FatBat system will record only the latest updates for the three types of personal data.

Each user has two types of contacts:

1. insurance: the user ID of their insurance company;
2. friend: the user ID who their nominated friend.

Each user has a set of permissions determining who is able to view their data. Another user can read vitals, footsteps, and location of a user if and only if that wearer has provided access to that data for that type of user (e.g. friend, insurere).

3.3 Functional requirements

The sub-system to be implemented has the following requirements.

3.3.1 User management

R1.1 The system must provide new user accounts, which return a unique user ID.

3.3.2 Communication with FatBat Device

R2.1 The system must receive updates specifying the location of the user and store this.

The timing of the update is via a setting on the FatBat device, and is pushed to the cloud system; i.e. the update is passive.

R2.2 The system must receive updates specifying the number of steps taken by the user during the current day and store this.

The upload occurs at 11:59pm every day, but is pushed by the FatBat device; i.e. the update is passive.

R2.3 The system must receive events specifying that the user has suffered a cardiac arrest, with accompanying location and vitals.

If the system receives such an event, it must:

(a) Send a message to the emergency services user ID (0) specifying the location and vitals of the user.

(b) Store the location and vitals for future use.

R2.4 In the event of a cardiac arrest, the system must send a message and store location and vitals the above operators if and only if the emergency services have permission to read their vitals. That is, the system must share the user's location even if this permission is NOT granted to 0 otherwise.

3.3.3 Permissions

R3.1 The system must allow the user to read their own data at all times.

R3.2 The system must allow the user to change their nominated insurance provider.

R3.3 The system must allow the user to remove their nominated insurance provider.

R3.4 The system must allow the user to change their nominated friend.

R3.5 The system must allow the user to remove their nominated friend.

R3.6 The system must allow a user with permission as a user's insurance provider to read that user's footsteps.

R3.7 The system must allow a user to change the read permissions of their vitals, footsteps, and location that they give to their insurance provider, friend, and emergency services.

R3.8 The system must allow a user to give different permissions to their insurance provider, friend, and emergency services.

R3.9 The system must restrict read permissions of any user's data to only their insurance provider, friend, and the emergency service.

4 Your tasks

1. Implement an Ada package called `AccountManagementSystem`, which implements the functionality described above. It *must* adhere to the package specification in `accountmanagementsystem.ads`, provided as part of the assignment. The functions should use the `Emergency` package provided to contact emergency.

   Note: You do *not* have to implement a user interface.

   You can modify the provided code for the purpose of testing etc., however, we will use our own implementations of `Emergency`.

2. Implement a package called `FatBatExample`, containing a procedure called `Run`, which implements a scenario showing the following:

   (a) Creating at least one wearer, one friend, and one insurance provider.

   (b) Setting and changing a wearer's friend and insurance provider.

   (c) Updating vitals and footsteps for a wearer.

   (d) Changing permissions to at least one of the types of data for a wearer.

   (e) Contacting emergency for a wearer.

5 Criteria

Criterion	Description	Marks
Design	The design of the system is of high quality. The correct components have been included, the design is loosely coupled, and suitable information hiding strategies have been used.	2 marks
Correctness	The implementation behaves correctly with respect to the user requirements.	2 marks
Completeness	The implementation is complete. All components have been implemented and all requirements have been addressed.	1 marks
Clarity	The design and implementation are clear and succinct.	1 marks
Code formatting	The implementation adheres to the code format rules (Appendix A).	2 marks
Tests	The implementation passes our tests.	2 marks
Total		10 marks

6 Submission

Create a zip file called *your_username*`.zip` or *your_username*`.tgz`. The file should contain your code for the *complete* FatBat system, including code provided by subject staff.

Submit the zip file to the LMS.

7 Academic Misconduct

The University misconduct policy applies to this assignment. Students are encouraged to discuss the assignment topic, but all submitted work must represent the individual's understanding.

The subject staff take plagiarism very seriously. In the past, we have successfully prosecuted several students that have breached the university policy. Often this results in receiving 0 marks for the assessment, and in some cases, has resulted in failure of the subject.

A Code format rules

The layout of code has a strong influence on its readability. Readability is an important characteristic of high integrity software. As such, you are expected to have well-formatted code.

A code formatting style guide is available at `http://en.wikibooks.org/wiki/Ada_Style_Guide/Source_Code_Presentation`. You are free to adopt any guide you wish, or to use your own. However, the following your implementation must adhere to at least the following simple code format rules:

- Every Ada package must contain a comment at the top of the specification file indicating its purpose.
- Every function or procedure must contain a comment at the beginning explaining its behaviour. In particular, any assumptions should be clearly stated.
- Constants and variables must be documented.
- Variable names must be meaningful.
- Significant blocks of code must be commented.

  However, not every statement in a program needs to be commented. Just as you can write too few comments, it is possible to write too many comments.
- Program blocks appearing in if-statements, while-loops, etc. must be indented consistently. They can be indented using tabs or spaces, and can be indented any reasonable number of spaces (three is default for Ada), as long as it is done consistently.
- Lines must be no longer than 80 characters. You can use the Unix command "`wc -L *.ad*`" to check the maximum length line in your Ada source files.