Assignment 2/6: The social network

Object-Oriented Programming and Design B2-2011

Department of Computer Science at the University of Copenhagen (DIKU)

Deadline: 2011-12-03, 18:00 CET

Preface

This is one of the six weekly assignments in the course. Every assignment gets a score of either 0, 0.5, or 1, where 1 is full credit, and 0 is no credit. You are required to acquire a sum of at least 4 points (out of the possible 6) to be admitted to the exam. Don't worry if you don't score any credit on your first attempt for the first 5 assignments. You will have a chance to revise your solution and hand in the assignment again up until a week after the initial deadline for the assignment. However, this is *not* the case for assignment 6 due to time constraints. Please check Absalon for exact deadlines for submissions and resubmissions.

Types of exercises

The assignments themselves consist of two types of exercises, those that require you to do coding only, and those that require you to respond in prose, or by taking a screenshot. Exercises that require you to do things other than coding are always postfixed with a *.

What we expect of your code

For the coding exercises we expect of you to follow the Code Conventions for the Java Programming Language[1]. Deviation from the conventions does not necessarily lead to a reduced score, but the conventions are in place to ensure that your instructor can read your source code quickly. Additionally, make sure to follow the explicit code specifications in the assignment text as your code might be subject to automated tests. Such tests usually rely on various classes, fields, methods, etc. to be named in a specific way. Also, make sure to structure your code in an intuitive way, and document your source code and interfaces where appropriate. What this last statement means exactly should become ever more clear as we advance through the course.

What to hand in

For each assignment you should hand in *one* zip archive. The actual structure of the archive, and hence the files we expect you to package into it, are specified at the end of every assignment. To allow us to easier grade the assignments, you are strictly required to follow the following naming convention:

```
<full-name>-A.<number>.zip
```

The name must additionally be lower-cased. `<full-name>` should be a dash-separated string of characters `a` to `z` representing your name as registered on Absalon. If you have special characters in your name, such as æ, ø and å, please use their `a` to `z` equivalents of `ae`, `oe` and `aa`, respectively. So for example, if your name is Kåre Søren Æbelsen, for the first assignment we'd expect you to hand in a file with the following name:

```
kaare-soeren-aebelsen-A.1.zip
```

[1] http://www.oracle.com/technetwork/java/codeconv-138413.html

Please make sure that you hand in your *source code* and *not* java bytecode. It is always a good idea to check that all of your files have been uploaded correctly by downloading them again and browsing them yourself.

As for the written exercises, your solutions should be written in plain text format. We make no explicit requirements as to whether they should be written in Danish or English, but try to stick to one. Additionally, this depends on the preferences of your instructor.

Collaboration and help

We encourage you to discuss your solutions with your peers, but your solution *must* be individual. We expect you not to share explicit code or to take credit for another student's work. All in all, we expect you to honor each other's, and your own work.

You may ask your instructor for general help with the assignment or for additional clarifications. You can also use the course discussion forum on Absalon for the exact same purposes. Please remember to read the forum rules before you post anything on the forum. You should especially remember to prefix the forum topics with e.g. A.1.2 if you're asking a question about exercise 2 in assignment 1.

Some conventions

When we say "book", we refer to the main course litterature, namely Objects First With Java 4th Edition, by David Barnes & Michael Kölling, and when we say "project library" we refer to the project library on the accompanying CD, also provided via Absalon under "Course material".

Handed out files

```
./A.2.pdf
./A.2.2.txt
./A.2.4.java
./src/
./src/Photograph.java
./test/
./test/PhotographTests.java
```

Credit assignment

The intent of this assignment is for you to get acquainted with collections, both of constant and dynamic size. You'll also try out some more complex object interaction models where you'll implement a small communication protocol. At the end of the assignment we'll also ask you to write your first unit tests.

What you'll build is a small social network, `DikuPlus`. The users of the network can either be friends or enemies with each another (unless they've never have communicated, in which case they are neither). Friends can exchange messages and enemies reject each other's messages.

A.2.1 Photograph

In this exercise you should complete the definition of the `Photograph` class provided in the handed out `zip` file, under the `./src/` directory.

We represent the actual photograph using an array of `Strings`, where each element of the array is a row of the picture. All rows have the same length – the width of the picture, and the height of the picture is hence simply the length of the actual array of `Strings`.

1. Complete the definition of the class constructor. You should store the actual photograph in some instance variable.

Note: We've already provided a partial solution. We suggest explicitly checking if the supplied argument is `null`, and if so, replace it with an empty `String` array. If we let a `null` value be stored as some variable, every time we dereference that variable, e.g. to call one of its methods, e.g. `someString.length()`, our application will crash. This is because the variable does not refer to an actual instance of the variable type, so calling a method on it is, literally speaking, pointless.

2. Complete the accessors `getWidth` and `getHeight`.

3. Complete the accessor `getRow` that gets the line at the specified index (0-indexed).

By now you should be able to pass all the tests in `test.PhotographTests`, which you'll find in the handed out `zip` file.

A.2.2 People

Create a class `Person` to represent a person on the network. The person should have a name and a photograph, which only the person can change, but anyone can view.

A.2.3 Messages

Create a class `Message` to represent messages to be sent across the network. A message has a text and some `Person` as its author. The fields should be specified when the object is constructed and cannot be changed throughout the object lifetime. However, it is a good idea to provide accessor methods to the fields. We also call an object such as an instance of `Message`, an *immutable* object.

A.2.4 Friends & Enemies

To communicate on the network, people exchange messages. You'll be implementing this in A.2.5 (page 3) , for now we'll just add some underlying architecture to the `Person` class.

Friends can communicate without limitations, and enemies reject each other's messages. However, the network starts up in a state where people are neither friends nor enemies. In this state they can try to send friend requests to each other, and may get rejected. Among people that are neither friends nor enemies, all messages other than friend requests will be rejected as spam. The friends and enemies are individual to each person, but symmetric among friends and enemies. Let's illustrate this with an example:

If A and B are neither friends nor enemies, A can request B to be friends, B can accept A, and put it into its own friend list, or reject A, and put it into it's own enemy list. A is notified whether the request was accepted or rejected, and puts B into its own corresponding list. This way we achieve symmetry between friends and enemies.

1. Add two fields to the class `Person` to represent the lists of friends and enemies. The lists should clearly be dynamically expandable.

   Hint: use a dynamically sized collection, e.g. `java.util.ArrayList<Person>`.

A.2.5 Communication

Implement the following methods in the `Person` class:

```
1 public boolean sendMessage(Person receiver, String message) { }
```

This method should simply create a message and send it to the receiver. The return value should indicate whether the message was accepted or rejected.

```
1 public boolean requestFriendship(Person otherPerson) { }
```

This method should send the special message `"/friend"` to the `otherPerson`. The return value should indicate whether the friend request was successful or not. Make sure that friendship/enemy associations are symmetric for the people in the network, i.e. if their friend request is rejected, they should add the other person to their own enemy list.

```
1 private boolean receiveMessage(Message message) { }
```

- If the author of the message is in the accepted list, the message is added to the wall.
- If the author of the message is in the enemy list, the message is rejected.
- If the author of the message is neither a friend nor an enemy, reject all messages other than those with the text `"/friend"`, which constitutes a friend request.
- We'll use the simple strategy where everyone rejects friend requests from `"Oleks"`.
- The method should return whether the message was accepted or rejected. A successful friend request should return *accepted*.

Note: the Java default for the equality operator (==) between two object references is to compare the memory address they're referring to, rather than compare the values of the fields of the instances. This might not always be what you're looking for. The answer is to test the fields of a class for equality explicitly. This should probably be an instance method of the `Person` class. *For the sake of simplicity, we'll let people with the same name be the same person.*

Note also, that since `Strings` are reference types, the following code sequence:

```
1 String a = "hello";
2 String b = "hello world";
3 String c = b.substring(0,5);
4 System.out.println("\"" + a + "\"");
5 System.out.println("\"" + c + "\"");
6 System.out.println(a == c);
7 System.out.println(a.equals(c));
```

Writes the following to `System.out`:

```
"hello"
"hello"
false
true
```

A.2.6 The network

Implement a class `DikuPlus`, that represents the actual network. The class should:

- Manage the people in the network in some sort of datastructure.
- Provide a method `void addPerson(Person person)` for adding a person to the network.
- Provide a method `void removePersonWithName(String name)` for removing a person with the specified name from the network. Removing a person that is not in the network should not crash the application.

A.2.7 A simulation of the network

Add a class `Simulation` that contains a `main` method and runs a simulation of the `DikuPlus` network. See `A.2.4.java` in the handed out zip file for an example of what this *could* look like.

Note: it is intentional that the code is incomplete.

A.2.8 * Who should be doing what?

Suppose you were to add the accessor method `humanReadableState()` to the class `DikuPlus`, that returned a generated human-readable `String` representing the state of the network. The state is constituted of the profiles of the people in the network, where a profile is the name, photograph, friend list, enemy list and wall of a person. How would you modularise the generation of the state `String`?

For instance, would you add methods to the classes `Person` and `Message` to generate each their own part, or would you let `DikuPlus` take care of everything? And so on.

A.2.9 Testing

Conduct thorough tests of the classes `Message`, `Person` and `DikuPlus`. You may use the tests we've handed out for the `Photograph` class as inspiration, but you might find them rather evolved. We do not initially expect the same quality of tests. Test whatever you think is reasonable to test.

A.2.10 Commenting and naming

Make sure that you've documented your program where appropriate and given your methods good and understandable names. Remember, that a method with a well-named signature shouldn't require much documentation, if any.

Note: The formulation of this exercise is intentionally rather vague. This is because the concept of a "good name" or a "good comment" can't seemingly be formally defined as they are both rather *subjective* concepts. Ask your instructor for help with this exercise if they don't provide it by themselves.

Files expected to be handed in

We expect for *at least* the following list of files to be handed in for full credit. `./` resembles the root of your zip archive.

```
./2.9.txt
./src/
./src/Photograph.java
./src/Message.java
./src/Person.java
./src/DikuPlus.java
./test/
./test/PhotographTests.java
./test/MessageTests.java
./test/PersonTests.java
./test/DikuPlusTests.java
```

Challenge of the week

Challenge of the week is an optional part of the weekly assignment for students that have completed the mandatory exercises above. You should feel confident enough about your solutions to the exercises above before taking up the challenge, as there is *no extra credit* for solving it. However, the most elegant solution is rewarded recognition and a free beverage from the canteen. The elegance of the solution is judged by it's exploitation of object-orientation, the Java language and it's libraries.

The deadline for completing the challenge is the same as the assignment deadline, however there are *no* resubmissions. You should upload your solution to the challenge into a separate folder, which you'll find on Absalon.

The challenge

Implement a class named `DynamicArray`, which is a simple, non-generic version of Java's `ArrayList`. It should *at least* have the following methods:

```
1 DynamicArray() { }
2 void add(int index, Object element) { }
3 void add(Object element) { }
4 void clear() { }
5 boolean contains() { }
6 Object get(int index) { }
```

For further details, see http://docs.oracle.com/javase/1.4.2/docs/api/java/util/ArrayList.html. You should implement your methods such that their interface documentation can match the one for `ArrayList` for the given methods.