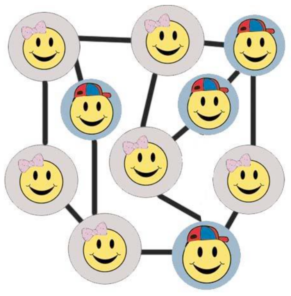
CS1110 Fall 2010 Assignment A1
A social-networking site[1]

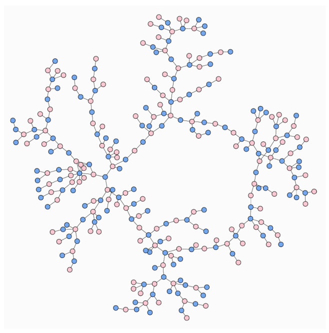
Introduction



Imagine creating your own social-networking site. It will need to keep track of various types of information about its participants.

Your task in this assignment is to develop a Java class `Person`, which will maintain information about people in this social-network site, and a JUnit class `PersonTester` to maintain a suite of testcases for `Persons`. This assignment is the first of two that, together, illustrate how Java's classes and objects can be used to maintain data about a collection of things —like entities in a large tracking system.

Learning objectives



- Gain familiarity with DrJava and the structure of a basic class within a record-keeping scenario (a common type of application)
- Work with examples of good Javadoc specifications to serve as models for your later code
- Learn to write *class invariants*
- Learn the code format conventions for this course (use of "Constructor:" and "=" in specifications, indentation, short lines, etc.), which help make your programs readable and understandable and allow us to process your assignments and give you feedback more quickly
- Learn and practice incremental coding, a sound programming methodology that interleaves method writing and testing
- Learn about and practice thorough testing of a program
- Learn about *preconditions* of a method, which need not be tested

Given these objectives, the methods we ask you to write in this assignment are quite short and simple; the emphasis in assignment A1 is on "good practices", not complicated computations.

Iterative grading feedback (revise-and-resubmit cycle)

To help ensure that the learning objectives are met, we will engage in an iterative feedback process. If an objective has not been met, we will give you feedback so you can update your code and resubmit it. This iterative process will ensure that every single student completely masters this material.

We will look at your code in several steps.

1. If the field specifications, javadoc specifications, or formatting are not appropriate, we will ask you to fix them and resubmit.

2. If the field and javadoc specs are ok, we will look at your test cases. If they are inadequate, we will ask you to fix them, test your program again yourself with the new cases, and resubmit.

[1] Research in social networks has many uses. For example, the paper “Chains of Affection: The Structure of Adolescent Romantic and Sexual Networks” by Bearman et al (www.soc.washington.edu/users/stovel/Chains.pdf) in the 1990s was interested in how diseases could spread. This paper found that sexual and romantic networks were characterized by long contact chains and few cycles. The second image on this page, of high school dating, was drawn by Mark Newman from data in this paper.

Some results from research in social networks have also entered popular culture; the phrase "six degrees of separation" comes from Stanley Milgram's foundational experiment examining the "small world" phenomenon.

The immensely popular Cornell course CS2850/Econ2040/Info2040/Soc2090 examines network structures and how they matter in everyday life, looking at networks in the web, markets, neural networks, contagion, etc. Tools of graph theory and game theory come into play. The internet and websites such like facebook and twitter has made some research in social networking more interesting and easier.

3. If the test cases are adequate, we will test your program with our own testing program. If there are errors, we will ask you to correct them and resubmit.

Until mastery is achieved, your "grade" on the CMS will be the number of revisions so far, so that we can keep track of your progress. So don't be alarmed if you see a "1" for the assignment at first! The assignment will be considered completed when it passes all three steps outlined above.

Your chances of completing the assignment in 1 or 2 rounds will be increased by reading this handout carefully and following all instructions. Many requests for resubmission are caused not by confusion about programming but simply by not following instructions.

Collaboration policy

You may do this assignment with one other person. If you are going to work together, then, as soon as possible —and certainly before you submit the assignment— get on the CMS for the course and do what is required to form a group. Both people must do something before the group is formed: one proposes, the other agrees. If you do this assignment with another person, you must *work together*. It is against the rules for one person to do some programming on this assignment without the other person sitting nearby and helping. You should take turns "driving" —using the keyboard and mouse.

With the exception of your CMS-registered partner, you may not look at anyone else's code, in any form, or show your code to anyone else, in any form.

Getting help

If you don't know where to start, if you don't understand testing, if you are lost, etc., please SEE SOMEONE IMMEDIATELY —a course instructor, a TA, a consultant. Do not wait. A little in-person help can do wonders. See the Staff page on the course homepage, http://www.cs.cornell.edu/courses/cs1110, for contact information.

How to do this assignment

Read the whole assignment before starting.

Develop class `Person` and test it using a class `PersonTester` in the following incremental, methodologically sound way. This will help you complete this (and other) programming tasks quickly and efficiently. If we detect that you did not develop it this way, we may ask you to start from scratch on a different assignment.

1. Make sure your DrJava preferences are set to indent 4 spaces (use menu item Preferences->Miscellaneous and change the entry "Indent Level" to 4).

2. Start a new directory on your hard drive to contain the files for this project. (You should create a new directory for every assignment in this course). In DrJava, write a skeleton class definition for class `Person`, with no methods or fields in it, and save it in the new directory.

3. In class `Person`, declare the following fields (you can choose the names), which are meant to hold information describing a person. Make all these fields private and properly comment them (see the "The class invariant" section below). Compile often as you proceed.

   Note: break any long lines in your code (including comments) into two or more lines so that we do not have to scroll right to read them. These steps make understanding the structure of a program easier and makes your code much easier for us (and you) to read.

   - display name (a String)
     what this `Person` wants displayed to other `Persons` as their identity It is a String of length > 0. More than one `Person` can have the same display name.
   - year of birth (an int)
     year the `Person` was born —must be > 1900.
   - status message (a String)
     status message the `Person` would like to have displayed to others —null if none, otherwise a String of length > 0. Example: “Thrilled to be taking CS1110!”

- male (a boolean)
  = "this `Person` is male"
- best female friend (a `Person`)
  and
  best male friend (a `Person`)
  the names of the object of class `Person` that are the best female friend and best male friend, respectively—null if this `Person` has no best female or best male friend, respectively.
- number of `Persons` who have indicated that this `Person` is their best friend (an int)

(Aside: why did we represent the gender as a boolean rather, than, say, a char like 'F'? While a char field would be more flexible, it could also be considered *too* flexible; we probably don't want people saying that their gender is '!'; and would we allow both 'F' and 'f'?)

The class invariant. Recall that comments should accompany the declarations of all fields to describe what each field means, what constraints hold on the fields, and what the legal values are for the fields. For example, for the display-name field, state in a comment that the field represents the `Person`'s chosen name to be displayed and that it is a String of length > 0. The collection of the comments on these fields is called the *class invariant*.

Whenever you write a method (see below), look through the class invariant and convince yourself that the class invariant still holds when the method terminates. This habit will help you catch or prevent bugs later on!

Remember to break any long comments so right-scrolling isn't necessary to read them, and indent lines properly to follow the structure of the code.

4. Start a new JUnit class and call it `PersonTester`.

5. For each of the following *groups* A, B, and C of methods, do the following.

   (1) Write the Javadoc specifications for each method in that part.

   (2) Write the methods.

   (3) Unless otherwise specified, write *one* test procedure in class `PersonTester` and add test cases to it for all the methods in the group.

   (4) Test the methods in the group thoroughly. Do not go on to the next group of methods until the group you are working on is thoroughly tested and correct.

   The descriptions below represent the level of completeness and precision we are looking for in your Javadoc comments. In fact, it is best to copy and paste these descriptions to create the first draft of your Javadoc comments. If you do not cut and paste, please adhere to the conventions we use, such as using the prefixes "Constructor: ...", or "= ..." (the equals sign), or double-quotes to enclose an English boolean assertion. Using a consistent set of good conventions in this class will help us immensely in grading.

   In our specifications, there are no references to specific field names, since the user may not know what the fields are, or even if there are fields. The fields are private. Consider class `JFrame`: you know what methods it has, but not what fields, and the method specifications do not mention fields. In the same way, a user of your class `Person` will know the methods but not the fields.

   The names of your methods must match those listed below exactly, including capitalization. The number of parameters and their order must also match: any mismatch will cause our testing programs to fail, meaning that you will have to resubmit. Parameter names will not be tested —change the parameter names if you want.

   In this assignment, you may *not* use if-statements anywhere. They are not necessary. Submissions hat contain if-statements will be returned for you to revise.

   In this assignment, do *not* write code that checks whether preconditions hold. It is the responsibility of the caller to ensure that the precondition is met. This means, for example, that you should *not* worry about (or write code that checks for) a display name that is null.

Group A: The first constructor and all the getter methods of class Person.

Constructor	Description (and suggested javadoc specification)
Person(String n, int y, boolean m)	Constructor: a new Person with display name n, birth year y, gender male if m is true and female if false, no status message to be displayed, no best female friend, and no best male friend. Precondition: n's length is > 0 and y > 1900.

Getter Method	Description (and suggested javadoc specification)	Return Type
getName()	= this Person's display name	String
getYear()	= year this Person was born	int
getStatus()	= this Person's status message (null if none)	String
isMale()	= "this Person is male"	boolean
getFBF()	= this Person's best female friend (null if none).	Person (not String!)
getMBF()	= this Person's best male friend (null if none).	Person (not String!)
getNumBFedBy()	= number of Persons who call this Person their best friend.	int

Consider testing the constructor. Based on the specification of the constructor, figure out what value it should place in each field to make the class invariant true. Then, write a procedure in `PersonTester` to test that the constructor filled in ALL the fields correctly. The procedure should: create one `Person` object using the constructor and then check, using the getter methods, that all fields have the correct values. Since there are 7 fields, there should be 7 assertEquals statements. As a by-product, all the getter methods are also tested.

Group B: the setter methods. When testing the setter methods, you will have to create one or more `Person` objects, call the setter methods, and then use the getter methods to test whether the setter methods set the fields correctly. Good thing you already tested the getters! Note that two of the setter methods may change more than one field; your testing procedure should check that *all* fields that may be changed are changed correctly.

In this assignment, we are *not* asking you to write methods that *change* a best friend from one (existing) `Person` to another. Doing so would require if-statements, which are not allowed in this assignment.

Setter Method	Description (and suggested javadoc specification)
setStatusMessage(String s)	Set this Person's status message to s. Precondition: s does not have length 0 (it can be null).
addFBF(Person p)	Add p as this Person's best female friend. Precondition: p is not null, p is female, and this Person does not have a best female friend.
addMBF(Person p)	Add p as this Person's best male friend. Precondition: p is not null, p is male, and this Person does not have a best male friend.

Note that methods `addFBF` and `addMBF` may have to change some fields of both the "befriender" and the "befriendee", in order to maintain the class invariant. But do not write code to check whether preconditions hold —for example, method `addFBF` should not check that `p` is female.

Group C: the second constructor. The test procedure for group C has to create a `Person` using the second constructor (this will require creating two `Person`'s first, using the constructor) and then check that the second constructor set *all* seven fields properly.

Constructor	Description (and suggested javadoc specification)
Person(String n, int y, boolean m, String s, Person ff, Person mf)	Constructor: a new Person with display name n, birth year y, gender male if m is true and female otherwise, status message to be s, best female friend ff, and best male friend mf. Precondition: n's length is > 0, s is null or has length > 0, y > 1900, and ff and mf are not null.

6. Click the Javadoc button in DrJava and examine the output. You should see your method and class specifications. Read through them from the perspective of someone who has not read your code, and fix them if necessary so that they are appropriate to that perspective. You *must* be able to understand everything there is to know about writing a call on each method from the specification that you see by clicking the Javadoc button — that is, without knowing anything about the private fields. Thus, the fields should not be mentioned.

   Then, and only then, add a comment to the top of your code saying that you checked the Javadoc output and it was OK.

7. Review the learning objectives and re-read this document to make sure your code conforms to our instructions; this may help reduce the number of rounds of revision that will be needed.

8. Upload files `Person.java` and `PersonTester.java` on the CMS by the due date, Saturday, 18 September, 11:59PM. Do not submit any files with the extension/suffix .java~ (with the tilde) or .class. It will help to set the preferences in your operating system so that extensions always appear.

9. **Check the CMS daily until you get feedback from a grader**. (Make sure your CMS notifications for CS1110 are set so that you are sent an email when one of your grades is changed).

   Within 24 hours, do **RRRRR**: **R**ead the feedback, **R**evise your program accordingly, **R**esubmit, and **Request a Regrade** using the CMS. The whole process should be finished within 7-10 days.