CS1110 Fall 2010 Assignment A3
A social-networking site 2

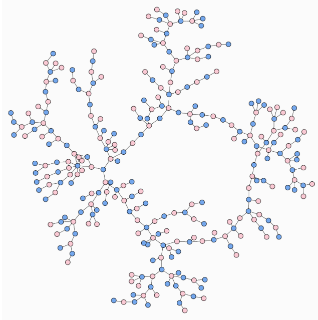
This assignment is a continuation of A1.
Keep track of the time you spend on this assignment. When you submit it, tell us how many hours you spent on it, as described in the 5th point on the last page of this assignment.

Collaboration policy You may work in groups of 2 (form your groups *immediately*; *both* partners must take action on the CMS before the group will be formed). You must do all the work together, sitting at the computer together. For example, for one person to write functions and the other person to write test cases for them, with no interaction, is dishonest.

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.

Grading. In assignment A1, if you made a mistake, we asked you to fix it and resubmit. In *this* assignment, if you make a mistake, points will be deducted. Once graded, you cannot resubmit. Your new functions must have suitable javadoc comments, there must be appropriate test cases, and all methods should be correct. *You can achieve all this most easily by writing and testing one method at a time*.

A1 learning objectives. The purpose of A1 was for you to:



- 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.).
- 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.

A3 learning objectives. The purpose of A3 is for you to:

- Learn to use functions you have already written (and tested) to save time and promote modularity in your code.
- Learn to write boolean expressions that use short-circuit evaluation (look it up in the text).
- Learn about the use of null —and testing for it.
- Learn to use static variables.

For this assignment, start with your correct A1. If you are still working on A1, you may modify it to include A3 stuff, but whatever you submit for A1 must still compile, and the A1 stuff should be correct. You will add fields and methods to your correct A1 solution, testing as you program. This may require you to change some of the testing procedures that you wrote for A1. We expect you to continue to use what you learned in doing A1, e.g. include the class invariant and javadoc specifications of all methods and program and test in an incremental fashion.

Getting help

If you don't know where to start, if you don't understand testing, if you are lost, etc., 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.

The steps to perform in doing this assignment

Step 1. Static variable. Add to class `Person` a **private static** field that contains the number of `Person` objects that have been created so far. WHENEVER A `Person` OBJECT IS CREATED, THIS

FIELD SHOULD BE INCREASED BY 1.

Add a **public static int** function `population`, with no parameters, that will return the value of the static variable. Change the constructors so that they properly maintain the value of the static variable.

Finally, add test cases to class `PersonTester` to test whether the static variable is properly maintained. Because you may not know the order in which the testing procedures are called when you click button `Test`, test the change in the static variable's value as follows: (1) save its value in a local variable, (2) write a statement that you expect will change the static variable, say, by 1, and (3) write an assertEquals statement to test whether the change in the static variable was indeed 1.

Step 2. toString function. Write a `toString` function that produces a `String` representation of a `Person` object. It is best to have a separate testing procedure for it in `PersonTester`. The representation it produces is illustrated by three examples:

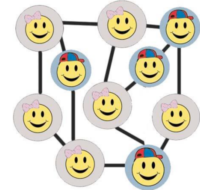
"male person Hamlet,   Prince of Denmark. Status: I   see dead people!. Born 1939. Best friend of 122 people."

"female person Florence Macbeth. Status: geez. Could use some hand soap here!!!!!. Born 1990. Best friend of 1 person."

"male person Caliban. Born 1950. Best friend of 0 people."

Here are the rules.

0. You may not use an if-statement. This function is easily written with a single return statement that consists of the catenation of several parts. For readability, put each part on a separate line. You will use conditional expressions, and we give you one of them below.
1. Exactly one blank appears between words, with the exception of the status message and display name, which can have any spacing within them.
2. Get the gender with a blank after it using this conditional expression: (isMale() ? "male " : "female ")
3. The word "person" is followed by a blank, the person's display name, and a period '.'.
4. The word “Status”, the status message, and following period appear only if the status is not null. Use a conditional expression.
5. The birth year always appears.
6. The number of people calling this person their best friend and following period appear as shown in the examples. If the number of people calling this person their best friend is 1, "person" appears; otherwise, "people" appears. Use a conditional expression.
7. Information about the male and female friends does not appear.



In testing `toString`, you need enough test cases to ensure that each different way of evaluating a conditional expression is tested. For example, to test whether the gender appears correctly, you need at least two test cases, for a female `Person` and for a male `Person`.

When making up a test case, construct it by hand, based on what you read above. Then, write an `assertEquals` statement that compares this expected value to the computed one. This is what we did.

Step 3. Comparison functions. Write the functions in the table below, each of which produces a boolean value. **Read through all of them first; then do one at a time**: (1) write the function and (2) test it thoroughly with test cases in class `PersonTester`. **Then** proceed to the next function. You may put all the test cases for these methods in one test procedure.

Here are the ground rules for writing these functions:

1. The names of your methods much match those listed above **exactly**, including capitalization. The number of parameters and their order and types must also match. The best way to ensure this is to copy and paste. Our testing will expect those method names and parameters, so any mismatch will cause our testing program to fail. Parameter names are never tested —you can change the parameter names if you want.

2. Each method **must** be preceded by an appropriate specification, as a Javadoc comment. The best way to ensure this is to copy and paste from this handout. After you have pasted, be sure to do any necessary editing.

3. You may not use if-statements or arithmetic operations like addition, subtraction, multiplication, and division. The purpose is to practice using boolean expressions, with operators && (AND), || (OR), and ! (NOT).

4. Note that function `areMutualFriendsStrict` must be static.

5. To receive full credit, your methods must call other methods you have written as much as possible. For example,

- `bestFriendsInclude` must call `bestFemaleFriendIs` and `bestMaleFriendIs`
- `isMutualFriendWith` must call `bestFriendsInclude` (twice)
- `areMutualFriendsStrict` must call `isMutualFriendWith`.

We want you to learn to call methods already written instead of doing duplicate work, saving the programmer time and avoiding redundant code. Writing modular code in this fashion is good software-engineering practice.

Method	Suggested javadoc specifications
bestMaleFriendIs(Person p)	= "p is not null, and p is this person's best male friend". *Note: the notation above means precisely this: the value of the function should be true if p is not null and p is this person's best male friend and should be false otherwise. So, p may be null (in which case, the function call's value should be false).* *The same comment holds for the remaining specifications.*
bestFemaleFriendIs(Person p)	= "p is not null, and p is this person's best female friend".
bestFriendsInclude(Person p)	= "p is not null, and p is this person's best friend (male or female)".
isMutualFriendWith(Person p)	= "p is not null, p is this person's best friend (male or female), and this person is p's best friend (male or female)". *Be careful in writing this function, for there are interesting cases you might not have first thought to consider. The existence of the next function should give you a hint about one such case.*
areMutualFriendsStrict (Person p1, Person p2)	= "p1 and p2 are different people and not null, p1 is p2's best friend (male or female), and p2 is p1's best friend (male or female)". *This function must be **static** (do not include this sentence in the specification).*

Submitting the assignment

Check these points before you submit your assignment:

1. Did you use an if-statement or conditional expression outside of `toString`, or an if-statement in `toString`? Get rid of them.
2. Is each function tested enough? For example, if an argument can be **null**, is there at least one test case that has a call on the function with that argument being **null**? If not, points will be deducted. *All* the comparison functions can have null arguments.
3. Did you check your javadoc? Click the javadoc button in the DrJava navigation bar. This will cause the specification of the classes and methods of the classes to be extracted from your program and html pages to be created that contain the specs. Look at those specs carefully and make sure that they are suitable. Can you understand precisely what a method does based on the extracted spec? If not, fix the spec, generate the javadoc, and look at it again.
4. Review the learning objectives and re-read this document to make sure your code conforms to our instructions.
5. ADD A COMMENT AT THE BEGINNING OF FILE PERSON.JAVA THAT SAYS HOW MANY HOURS YOU SPENT ON THIS ASSIGNMENT. We will report the min, mean, median, and max.
6. Upload files Person.java and PersonTester.java to A3 on the CMS by the due date, Wednesday, 29 September, 11:59PM. Be careful, because in the same directory as your .java files you may also have files that end with .class or .java~ but otherwise have the same name. It will help to set the preferences in your operating system so that extensions always appear.