EXTENDING ACORN.COM

CS 1110 SPRING 2012: ASSIGNMENT A3

Due to CMS by Tuesday, February 28.

This assignment is a continuation of Assignment A1, the first Acorn.com assignment. In fact, you will be submitting the same files (`AcornProfile.java`, and `AcornProfileTester.java`) once again. However, these files will be modified to include new features, so you will probably want to save them in a different folder (and hence leave your old versions for Assignment A1 unchanged). This is doubly important if you are still revising Assignment A1, as we do not want you to submit Assignment A3 modifications as part of your revision.

The previous assignment was intended to get you introduced to the basic elements of Java, writing JUnit tests, and JavaDoc specifications. This assignment will give you practice with some of the features that we have introduced in class since you started Assignment A1:

- How to use static variables.
- How to use `null` and test for it.
- How to use conditionals to enforce invariants.
- How to use functions you have already written (and tested) as helper functions in order to save time and promote modularity
- How to write boolean expressions that use short-circuit evaluation (look up that phrase in the text).

1. Before You Get Started

There is an online version of these instructions at

http://www.cs.cornell.edu/courses/cs1110/2012sp/assignments/assignment3/index.php

1.1. **Grading Policy.** The revise-and-resubmit policy was for the first assignment only; we do not have the time to do this for each assignment. Therefore, *you will not be allowed to resubmit.* Your new functions must have suitable JavaDoc comments, there must be appropriate test cases, and all methods should be correct in order to get full credit. You can achieve all this most easily by writing and testing one method at a time.

We will be taking a few minor points off for formatting and style issues. Please see the course style guidelines, which are available at

http://www.cs.cornell.edu/courses/cs1110/2012sp/materials/style.php

Important style issues to keep in mind are as follows:

- Classes are capitalized, variables are lowercase.
- Classes and variables that are compound words have the interior words capitalized.
- JavaDoc specifications go before the class/method, not after.
- The file should be indented properly; DrJava automatically indents a line if you type ”tab” while the cursor is on the line.

1.2. **Collaboration Policy.** You may do this assignment with one other person. If you are going to work together, then form your group on CMS as soon as possible. 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.

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

1.3. **Assignment Scope.** Everything that you need to complete this assignment should have been covered by Lecture 8 in class. You are now allowed to use if-statements, but *only where we tell you to.* Please read the instructions below carefully.

1.4. **Getting Help.** If you do not know where to start, if you do not understand testing, or if you are completely lost, please see someone immediately. This can be the course instructor, a TA, or a consultant. Do not wait until the last minute, particularly since this is due on a weekend. A little in-person help can do wonders. See the Staff page on the course homepage:

`www.cs.cornell.edu/courses/cs1110/2012sp/about/staff.php`

---

2. Enforcing Invariants with Conditionals

One of the invariants in Assignment A1 is that the name of a profile cannot be empty and it cannot be null. Indeed, this was a precondition in our constructor. However, with if-statements, we do not always need to to rely on preconditions; we can enforce invariants directly.

Write a new setter method `setName` with the following specification:

Method	**Description (JavaDoc Specification)**
setName(String n)	Sets the name of this profile as n if n is neither null nor empty. Otherwise, it sets the name as "<unknown>".

Note the lack of preconditions in this setter. You should implement this method exactly as it says in the specification. Futhermore, you should not need to modify the constructors. The preconditions for the constructors still hold.

When you are done implementing this method, you should add new tests to the test procedure `partB` in `AcornProfileTester`, as that is the test procedure that you used to test setters. The tests should verify that this setter works correctly.

Important: This is the only part of the assignment where you will use if-conditionals.

---

3. Adding a Static Variable

Add a private static field to the class `AcornProfile` that contains the number of profiles that have been created so far. Whenever an `AcornProfile` **object is created, this field should be increased by one**; this will require that you change the constructors accordingly. You never delete profiles, so there is no need to be concerned with decrementing this value.

In addition, you should add a public static int function `population()`, with no parameters, that will return the value of the static variable. Finally, add test cases to class `AcornProfileTester` 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**, you cannot count on the population starting at 0. Therefore, you should test the population as follows:

- Write a statement to save the population in a local variable which you might call `pop`.
- Write a statement you expect will change the static variable and subtract `pop` from this new value.
- Write an assertEquals call to test whether the change in the static variable was what you expected.

Your test cases should appear in a test procedure (e.g. not `partA()`, `partB()`, or `partC()`). Remember that you should not declare fields in the class `AcornProfileTeseer`. The only variables declared in the class should be local variables of the procedures.

---

4. Adding `toString()`

By default, when you look at an `AcornProfile` object in the interactions pane, you can see the "folder name" of the object. We want to change that, and have a more descriptive representation of `AcornProfile` instances. To do this, we write a function `toString()` with no parameters that produces a String representation of the `AcornProfile` object.

We are not going to tell you exactly how you should design the `toString()` method. However, the examples below give you some idea of what we are looking for in this assignment:

- "Male profile Doug. Born 5/1963. 1 child."
- "Female profile Ellen. Born 6/1965. 1 child."
- "Female profile Ammie. ID 19. Born 6/2006. Mother Ellen. Father Doug. 0 children."

There are several rules that you need to follow when making `toString()`:

- You may not use an if-statement. However, you will use conditional expressions, and we give you one of them below.
- This function is best written with a single return statement that consists of the catenation of several parts. For readability, you should each part on a separate line. In developing the return statement, it is best to work with one part at a time, making sure it is right before proceeding to the next part.
- Exactly one blank appears between words (with the exception of the profile name, which can have any spacing within it, depending on what is in the field). Periods '.' are as indicated in the examples.
- You can get the gender with a blank after it using this conditional expression:
  `(isMale() ?  "Male " :  "Female ")`
- The word "profile" is followed by a blank, the name in the profile, a period '.', and a blank.
- The profile identifier is given in the form " ID <integer>. " It appears only if the profile has an identifier.
- The birth month and year always appear, as in the examples.
- Suppose the mother of a profile is known and has the name "xxx ". Then "Mother xxx. " appears should appear in the child profile, as in the examples. Similarly for the father. If the mother (father) is not known, nothing should be said about it. The mother appears first.
- The number of children appears as shown in the examples, with a period after it. When it is 1 child, use "child"; otherwise, use "children".

You should write a separate test procedure (e.g. outside of `partA()`, `partB()`, and `partC()`) to test that `toString()` works properly. 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 `AcornProfile` and for a male `AcornProfile`.

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 will do in our test program.

---

5. Comparison Functions

Write the functions in the table below. Each produces a boolean value. You should do one at a time. For each function, write it, and then test it with test cases in class `AcornProfileTester` before proceeding to the next function. You should put all the test cases for these methods in one single (new) test procedure.

The comparison functions all obey the following rules:

- 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, so you can change the parameter names if you want.
- 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.
- You may not use if-statements, conditional expressions, or arithmetic operations (e.g. addition, multiplication, division). The purpose is to practice using boolean expressions, with operators && (AND), || (OR), and ! (NOT).
- The function `isParent` must be written using `isMother` and `isFather` as helper methods. We want you to learn to call methods already written instead of doing duplicate work, saving you time and avoiding redundant code. Similarly, the static function `isSister` (the second one in the table) should be written using a call on the non-static version of `isSister`.

The methods for this part of the assignment are as follows:

Method	**Description (JavaDoc Specification)**
isMother(AcornProfile p)	Yields: "p is not null and p is this profile's mother". **Note**: The notation above means that the value of the function is true if p is not null and p is this profile's mother; it is false otherwise. So, e may be null (in which case, the function call's value should be false). The same comment holds for the remaining specifications.
isFather(AcornProfile p)	Yields: "p is not null and p is this profile's father".
isParent(AcornProfile p)	Yields: "p is not null and p is this profile's parent".
isSister(AcornProfile p)	Yields: "p is not null and p is this profile's sister". Precondition: p is not null **Note**: B is A's sister if (1) B and A are different profiles, (2) B is female, and (3) B and A have at least one parent in common (though not necessarily both).
isSister(AcornProfile p, AcornProfile q)	Yields: "p is q's sister". Precondition: p and q are not null. **Make this function static and write it using the previous `isSister(AcornProfile)` as a helper method.**
isYounger(AcornProfile p, AcornProfile q)	Yields: "p is younger than q". Precondition: p and q are not null. **Make this function static.**

6. Finishing the Assignment

We do not require that you check the JavaDoc this time, but we will be grading that your specifications (e.g. JavaDoc comments) are correct. A good way to see if your JavaDoc comments are correct is to press the JavaDoc button. Look at the list of methods in the web page that it generates. Are your specifications there? If not, then you need to go back and fix your specifications.

In addition, please make sure that you have ample test cases. 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.

6.1. **Turning it In.** When done, you should upload new versions of your files `AcornProfile.java` and `AcornProfileTester.java` on the CMS by the due date: **Tuesday, February 28th at 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.