ACORN.COM

CS 1110 SPRING 2012: ASSIGNMENT A1

Due to CMS by Tuesday, February 14.

Social networking has caused a return of the dot-com madness. You want in on the easy money, so you have decided to make your online social networking company. Your idea, Acorn.com, is a genealogy site designed to help people determine if and how they are related to one another. For that reason, the site does not keep track of friendships, but only handles family relationships. But unlike Facebook, it has profiles for people both living and dead, so that people can track family histories fairly far back.

There is a lot of work to setting up a social networking website, but you have decided to get started at the very beginning: profiles. In this assignment, you will make two classes. The first, `AcornProfile`, stores the data found in a profile on Acorn.com. You need to fully test this class before sending it to the other developers in your company; for that reason, you will also make a JUnit testing class called `AcornProfileTester`. Full instructions are included below.

1. Before You Get Started

There is an online version of these instructions at

http://www.cs.cornell.edu/courses/cs1110/2012sp/assignments/assignment1.php

1.1. **Read Carefully.** In order to give you as much guidance as possible, these instructions are fairly long. Your chances of completing the assignment quickly be increased by reading carefully and following all instructions. Many requests for resubmission are caused not by issues with programming but simply by not following instructions.

We have tried to lay out the instructions in sections to make it clear as possible. You should pay particular attention to Section 4: Iterative Development, as it contains important instructions for the remaining sections, and we will not repeat these instructions for each section.

1.2. **Grading Policy (Revise-and-Resubmit Cycle).** To ensure that everyone masters this assignment, we will use an iterative feedback process. If one of the objectives below is not properly met, we will give you feedback and expect you to revise and resubmit. This process will continue until you are done. This process should be finished by 18th of February; once you finish you will receive a perfect score of 10. In our experience, almost everyone is able to achieve a perfect score within two submissions.

In grading your code, we will focus on the following issues:

(1) If the field specifications, method specifications, or formatting are not appropriate, we will ask that you fix them.
(2) Once the specifications and formatting are okay, 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.
(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.

Formatting is graded according to the course style guidelines, which are available at

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

Until we have decided that you have mastered (e.g. 10/10) the assignment, your "grade" on CMS will be the number of revisions so far. This allows us to we can keep track of your progress. Do not 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.

1.3. **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. **This must be completed before you submit the assignment. Both people must do something to form the group**. The first person proposes, and then the other accepts.

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. Take turns "driving"; alternate using the keyboard and mouse.

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.

1.4. **Assignment Scope.** Everything that you need to complete this assignment should have been covered by Lecture 5 in class. In particular, **you may not use if-statements anywhere in this assignment**, as they are not necessary. Submissions containing if-statements will be returned for you to revise.

1.5. **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. Working with DrJava

To do this assignment, DrJava must be set up properly. If you have not already done this, follow the instructions on the course web page:

www.cs.cornell.edu/courses/cs1110/2012sp/materials/drjava.php

Next, you should adjust your DrJava preferences to make programming a littler easier. Open up the Preferences window, found in the Edit Menu. First, change the indent level; choose "Miscellaneous" in the Preferences window and change the entry "Indent Level" to 4. In addition, you should choose "Display Options" and check the box "Show All Line Numbers".

Once you do that, you will find that you can properly format the code in your entire file simply by hitting the tab key once; DrJava will format everything automatically. Given the size of these tabs, however, you may find yourself with long lines that require you to scroll to the right to read them. We do not want to have to scroll to the right, so you should **break up long lines (including both code and comments) into two or more smaller lines**.

Finally, you will want to create a folder on your hard drive that is dedicated to this assignment and this assignment only. Every time that you work on a new assignment, we want you to make a new folder, as otherwise it is easy to mix up your assignments.

---

3. Initial Draft of AcornProfile

The first thing to get start is to create a skeleton class definition for class `AcornProfile`. This is a class declaration with nothing – no fields, constructors, or methods – in between the two outer braces. You should also add a class comment with two bits of information:

- The statement "Instances of this class are profiles for the web site Acorn.com"
- Your name and the date on which you completed this assignment

Save this file into the new directory and compile it. You will want to compile often as you proceed. In general **it is a good idea to compile every time you complete an objective**.

3.1. **Class Fields.** The first thing to do when making a class is to create the fields. All of these fields should be private and properly commented. We have more instructions on commenting below. The fields for this class are as follows:

- **name (a String)**. This is the name (first, last, and perhaps middle) for this profile. This should not be the empty string. It is possible for different profiles to have the same name.
- **year of birth (an int)**. To cut down on the size of your family trees, you are currently restricting them to profiles of people born after (>) the year 1700. In addition, no profiles can be born later than 2012.
- **month of birth (an int)**. This is month in which the person in this profile was born. It should be a number between 1..12, with 1 meaning the month of January.
- **profile ID (an int)**. This is an integer greater than or equal to 0, and is the ID for this person on Acorn.com. If this person is not registered with Acorn.com (e.g. it is a profile of a non-living ancestor), then it is -1.
- **gender (a char)**. This must either be 'M' for male or 'F' for female.
- **mother (an AcornProfile)**. This is a (name of) an object of class AcornProfile that is the mother of this profile. It is `null` if this mother is unknown, or is not currently have a profile on Acorn.com.
- **father (an AcornProfile)**. This is a (name of) an object of class AcornProfile that is the father of this profile. It is `null` if this father is unknown, or is not currently have a profile on Acorn.com.
- **number of children (an int)**. This is a number that depends on the mother and father fields of other AcornProfile objects.

3.2. **Class Invariant.** Every field declaration should be accompanied by a comment. This comment should describe what the field means, the constraints on the field, and the legal values for the field. This type of comment is called an *invariant*. The collection of invariants for all fields is called the *class invariant*.

For example, for the year-of-birth field, state in a comment that the field contains the year of birth and that it must be greater than 1700, but less or equal to 2012. Here is an example of a declaration with a suitable comment:

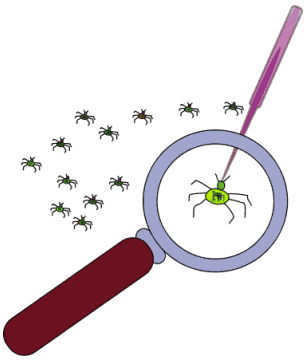
```
private char gender; // The gender of this profile;
                     // 'M' for male and 'F' for female
```

Note the we do *not* JavaDoc style comments for fields in this class.

Do not include things like "(an int)" in a comment for a field. We only included that above so that you know what type the field should be. It is not needed in the comment because the type is obvious from the declaration.

---

4. Iterative Development



You are not quite done yet with AcornProfile, but no one ever writes a complete class without testing parts of it first. For the rest of the assignment, you will go through an *iterative development cycle*. This means that you will write a little bit of the class, and then fully test it before you write any more. This process makes it easier to find bugs; you know that any bugs must have been part of the work you did since the last test. This section is an overview of what you need to do for the rest of this assignment.

4.1. **Skeleton of AcornProfileTester.** Iterative development hinges on proper JUnit testing. Create a new JUnit class in DrJava and call this class `AcornProfileTester`. DrJava might provide you with a skeleton of a test procedure (e.g. a procedure with nothing inside the inner braces) called `testX()`. You are to remove this test procedure, and replace it with three test methods: `testPartA`, `testPartB`, and `testPartC`. Each test procedure should be empty for now; you will fill in the tests later.

4.2. **Instructions for the Remainder of the Assignment.** The rest of the assignment is broken into three parts (listed Part A, B, and C). In each part, you will work on a group of methods in `AcornProfile`. In each part, you will do the following:

(1) Write the Javadoc specifications for each method given. Make sure they are complete and correct; look at the specifications we give you. You can even copy and paste.
(2) Write the methods themselves.
(3) Use *one* of the test procedures in `AcornProfileTester` for all of the methods in this part. Add test cases to this procedure for *all* of the methods given in this part
(4) Run the JUnit test and make sure everything is correct.

Remember: To run a JUnit test, have the class you are testing `AcornProfile` selected in DrJava and hit the "Test" button. DrJava will ask you for the class to test with and you will select `AcornProfileTester`.

4.3. **Writing Method Specifications.** The descriptions that we provide in each part 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: ...", for constructors and "Yields: ..." for functions. You should also use double-quotes to enclose an English boolean assertion. Using a consistent set of good conventions in this class will help us all.

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. Remember the 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 `AcornProfile` will know about the methods but not the fields.

---

5. Part A: Constructor and Getters

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; you can change the parameter names if you want.

5.1. **Constructor.** For this part, you will write a single constructor. It will have the following form:

```
AcornProfile(String n, int y, int m, char g)
```

The specification for this method is as follows:

> Constructor: a new profile with name n, birth year y, birth month m, and gender g. Its parents are unknown, and it has no children.
> Precondition: n's length is > 0. $y > 1700$. m is in the range $1 \ldots 12$. g is either 'M' (for male) or 'F' (for female)

Your constructor should initialize all of the fields in the profile object to satisfy the specification above. You should **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 name that is null.

5.2. **Getters.** In addition to the constructor, you should write several getter methods. The complete list of methods to write for this part is as follows:

Getter Method	Description (JavaDoc Specification)	Return Type
getName()	Yields: the name for this profile	String
getYear()	Yields: the year the person in this profile was born	int
getMonth()	Yields: the month the person in this profile was born. 1 means Jan, 2 means Feb, etc.	int
isMale()	Yields: "this profile is a male"	boolean
getID()	Yields: this profile's ID (>= 0) if defined; otherwise -1	int
getMother()	Yields: the mother of this profile (null if unknown)	AcornProfile (not String!)
getFather()	Yields: the father of this profile (null if unknown)	AcornProfile (not String!)
getNumChildren()	Yields: the number of children for this profile	int

5.3. **Testing.** Once you have created the constructor and getters, it is time to create your JUnit test. Your first test will be in the procedure `testPartA` in the `AcornProfileTester`. This procedure should first create an AcornProfile object (with the constructor). It should then use the getter methods to test that the fields all have the correct values. A by-product of writing the test this way is that you will test both the constructor and the getters at the same time.

There are eight fields, so you should be able to do this with eight `assertEquals` statements, which we showed in class. Note from the constructor specification that the eight fields should all have an exact value after the constructor; nothing should be unknown. Hence you do not need to use `assertTrue`.

---

6. Part B: Setters

Three of the fields in `AcornProfile` were missing from the constructor. That means we need setters for these three fields so that they can be assigned later. The setters that you need to write are as follows:

Setter Method	Description (JavaDoc Specification)
setID(int i)	Set the identifier for this profile to i. Precondition: i >= 0.
addMother(AcornProfile p)	Add p as the mother of this profile. Precondition: p is not null, p is a female, and this profile does not currently have a mother.
addFather(AcornProfile p)	Add p as the father of this profile. Precondition: p is not null, p is a male, and this profile does not currently have a father.

Again, you should not write code to check whether the preconditions holds. For example, `addMother` should not check that p is female. In addition, we are not asking you to write methods that change an existing mother or father to a different `AcornProfile`. Both of these require if-statements and **if-statements are not allowed in this assignment**.

Hint: The methods `addMother` and `addFather` are actually a bit tricky. Not only do you need to change the fields of this profile, but you also need to change a field of the parent profile p. Look at the fields again and see why this is the case.

6.1. **Testing.** You will test these setter methods in the test procedure `testPartB`. When testing the setter methods, you will have to create one or more AcornProfile objects, call the setter methods, and then use the getter methods to test whether the setter methods set the fields correctly. Hence, it is a good thing you already tested the getters!

As we said, the methods `addMother` and `addFather` change more than one field. Your testing procedure should check that all fields that may be changed are changed correctly.

---

7. Part C: Another Constructor

Your last goal in this assignment to write a second constructor for the class. This constructor is a thorough constructor that initializes all of the fields, and not just a select few. It will have the following form:

```
AcornProfile(String n, int y, int m, char g, int id,
             AcornProfile ma, AcornProfile pa)
```

The specification for this constructor is as follows:

> Constructor: a new profile with name n, birth year y, birth month m, gender g, identifier id, mother ma, and father pa.
> Precondition: n's length is > 0. $y > 1700$. m is in the range 1...12. g is either 'M' (for male) or 'F' (for female). id >= 0 (or -1 if not defined). ma and pa may not be null.

7.1. **Testing.** The final round of testing should occur in the test procedure `testPartC`. As part of this test, you will need to create an `AcornProfile` using the second constructor. Note that this requires that you first create two `AcornProfile` objects using the first constructor, to serve as the mother and father. Once you have done this, you should again use the getters to test that all of the fields have been set properly.

As before, you do not need to check that the preconditions are satisfied. That is the responsibility of the person using this class, not the person writing it.

---

8. Finishing the Assignment

Once you have everything working you should go back and make sure that your program is properly formatted so that we can easily read it. In particular, you should check that the following are all true.

(1) All lines short enough that horizontal scrolling is not necessary (about 80 chars is long enough).
(2) There is a blank line before the specification of each method and no blank line after the specification.
(3) The class invariant is appropriate (e.g. you have listed constraints for all the fields).
(4) The specifications for all of the methods are complete.
(5) You have created a JavaDoc version and put a comment at the top of the class.

The last step requires some additional explanation.

8.1. **JavaDoc Creation.** Recall that JavaDoc is how you create the web pages that you see in the Java API. In this class, we use it mainly to check that you have filled in all of your specifications correctly. 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 (e.g. without knowing anything about the private fields). Thus, the fields should not be mentioned.

Once you have done this, add a comment at the very top of your code saying that you checked the Javadoc output and it was OK.

8.2. **Turning it In.** Upload files `AcornProfile.java` and `AcornProfileTester.java` on the CMS by the due date: **Tuesday, February 14th 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.

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**: **Read** the feedback, **Revise** your program accordingly, **Resubmit**, and **Request** a **Regrade** using the CMS. If you do not request a regrade, we have no simple way of knowing that you have resubmitted. The whole process should be finished within one week.