CS100J Fall 2005 Assignment A2
Due (submitted on the CMS) on Wednesday, 21 September

Introduction

Read the WHOLE handout before you begin to write the assignment. You may do this assignment with one other person. If you are going to work together, then, as soon as possible, get on the CMS for the course and do what is required to form a group. We will give some hints on doing this in class.

CBS is planning on launching a gritty new drama series, called *The Greeces*, kind of like *As the World Turns*. The producers of this show have a problem, however: the Greece clan is so large that they have trouble keeping track of who is related to whom, what the relation is between any two characters, and which couples can't get married because they're cousins.

That's where you come in, as the CBS's resident computer expert. They've asked you to devise a system that will keep track of all the kinfolk in this show and allow them to add relatives and keep track of relatives as the show progresses. Make sure that the Java class you implement to store information on the Greece clan is a good one --or they may not renew your job for next season.

Requirements

For this assignment, you are required to design two classes. Class `KinPerson` is the class that is used by CBS to keep track of people. It has lots of fields and methods, but each method is simple. If you do one thing at a time, not worrying about the overall picture, you should have little trouble.

Class `KinPersonTester` is used to test class `KinPerson`. We will describe how to do this on 15 September. Do not think too much about it when first reading this handout.

Class KinPerson

An instance of class `KinPerson` represents a single person in this big family (in this case, one of the Greece relatives). It has several fields that one might use to describe a relative, as well as methods that operate on these fields.

KinPerson Variables

Class `KinPerson` will have the following fields, all of which should be **private**. You can choose the names of these fields.

- name (a `String`)
- gender (a `String`)
- day of birth (an `int`)
- month of birth (an `int`)
- year of birth (an `int`)
- criminal record (a `boolean`)
- father (a `KinPerson` object)
- mother (a `KinPerson` object)
- number of children (an `int`)
- family size (a static `int`)

Here are a few specifications about the fields of class `KinPerson`:

- The name is always the last name followed by a comma followed one or more blanks followed by the first name (e.g. "Gries, David").
- The gender field is one of two `String` values: `"male"` and `"female"`.
- The day of birth is in the range 1..31, representing the day within the month. The month of birth is in the range

1..12, representing a month from January to December. The year of birth is something like 1857 or 2005. Do not worry about invalid dates; do **not** write code that checks whether dates are valid: assume they are valid. *Error-checking may be penalized.*

- The criminal-record field contains `true` if the person has a criminal record and `false` otherwise.
- The father and mother fields are references to the `KinPerson` objects that correspond to this person's parents. They are `null` if not known.
- The family-size field contains the number of family members that have been recorded (created) so far. They do not have to be directly related to one another, since non-relatives are sometimes included in this type of 'family'. WHENEVER A `KinPerson` OBJECT IS CREATED, THIS FIELD SHOULD BE INCREASED BY 1.

KinPerson Methods

Class `KinPerson` supports the following methods. Pay close attention to the parameters and return values of each method. The descriptions, while informal, are complete.

Constructor	Description
`KinPerson(String name, String gender, int day, int month, int year)`	Constructor: a new `KinPerson`. Parameters are, in order, the name of the person, their gender, the day, month, and year of birth. The new person is not a criminal and its parents are not known. Precondition: gender is one of "male" and "female".
`KinPerson(String name, String gender, KinPerson father, KinPerson mother, int day, int month, int year)`	Constructor: a new `KinPerson`. Parameters are, in order, the name of the person, their gender, their father and mother (not `null`), and the day, month, and year of birth. The new person is not a criminal. Precondition: gender is one of "male" and "female".

Method	Description
`getName()`	= the full name of this person (a `String`) in th form "last-name, first-name"
`getLastName()`	= the last name of this person
`getGender()`	= the gender of this person (a `String`).
`getDOB()`	= the day of the month this person was born (an `int`).
`getMOB()`	= the month in which this person was born, in the range 1..12 (an `int`).
`getYOB()`	= the year in which this person was born (an `int`).
`getFather()`	= (the name of the object representing) the father of this person (a `KinPerson`).
`getMother()`	= (the name of the object representing) the mother of this person (a `KinPerson`).
`getNumberChildren()`	= the number of children of this person (an `int`).
`getFamilySize()`	Static method. = the number of `KinPerson` objects created thus far (an `int`).
`isCriminal()`	= "this person is a criminal" (a `boolean`)
`setName(String n)`	Set the name of this person to `n`.
`setGender(String g)`	Set the gender of the person to `g`. Precondition: g is one of "male" and "female".
`setDOB(int i)`	Set the day of the month this person was born to `i`.
`setMOB(int i)`	Set the month of birth for this person to `i`.
`setYOB(int i)`	Set the year of birth for this person to `i`.
`setGuilt(boolean b)`	Set whether this person is guilty of a crime to `b` (`true` or `false`).
`setFather(KinPerson fm)`	Set this person's father to `fm` (and increment fm's number of children). Precondition: This person's father is `null`, `fm` is not `null`, and `fm` is male.
`setMother(KinPerson fm)`	Set this person's mother to `fm`. Precondition: This person's mother is `null`, `fm` is not `null`, and `fm` is female.
`isOlder(KinPerson fm)`	= "this person is older than fm" (a `boolean`). Precondition: `fm` is not null.
`areSameAge(KinPerson fm1, KinPerson fm2)`	Static function. = "`fm1` and `fm2` are not null and are the same age --i.e. have the same birth date " (a `boolean`).
`haveSameLastName(KinPerson fm1, KinPerson fm2)`	Static function. = "`fm1` and `fm2` are not null and have the same last name" (a `boolean`). Capitalization is not important --e.g. "Greece" and

	"greece" are the same last name.
`isBrother(KinPerson fm)`	= "`fm` is this person's brother" (a `boolean`). Note: a guy is called your brother if you and he (has to be a "he") are different and have at least one parent in common. Precondition: `fm` is not `null`.
`isSister(KinPerson fm)`	= "`fm` is this person's sister" (a `boolean`). Note: a gal is called your sister if you and she (has to be a "she") are different and have at least one parent in common. Precondition: `fm` is not `null`.
`areSiblings(KinPerson fm1, KinPerson fm2)`	Static method. = "`fm1` and `fm2` are not null and `fm1` and `fm2` are siblings (brothers or sisters)" (a `boolean`).
`isMotherOf(KinPerson fm)`	= "this person is `fm`'s mother" (a `boolean`). Precondition: `fm` is not `null`.
`isFatherOf(KinPerson fm)`	= "this person is `fm`'s father" (a `boolean`). Precondition: `fm` is not `null`.
`isParentOf(KinPerson fm)`	= "this person is `fm`'s parent" (a `boolean`). Precondition: `fm` is not `null`.
`areTwins(KinPerson fm1, KinPerson fm2)`	Static method. = "`fm1` and `fm2` are not null and `fm1` and `fm2` are siblings and have the same birth date" (a `boolean`).
`isEvilTwin(KinPerson fm)`	= "this person is an evil twin of `fm` --'evil' means having a criminal record. Precondition: `fm1` is not `null`.

Make sure that the names of your methods match those listed above **exactly**, including capitalization. The number of parameters and their order must also match. The best way to ensure this is to copy and paste. Our testing will be expecting those method name names and parameters, so any mismatch will fail during our testing. Parameter names will not be tested --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. Note that a precondition should not be tested by the method; it is the responsibility of the caller to ensure that the precondition is met. As an example, in method `isMotherOf`, the method body should not test whether `fm` is null. However, in function `areSiblings`, the tests for `fm1` and `fm2` not `null` MUST be made.

The number of children of a newly created person is 0. Whenever person P is made the mother or father of another person, P's number of children should increase by 1.

It is possible for person P1 to be P2's mother, and visa versa, at the same time. We do not check for such strange occurrences.

Your method bodies should have no if statements. Your method bodies should contain only assignments and return statements. Points will be deducted if **if** statements are used.

Class KinPersonTester

How do you know whether class `KinPerson` that you are designing is correct? The only way you can be sure is to test it, to see if it does what it is supposed to do. It is not enough simply to try out your class `KinPerson` in the interactions pane. Every time you write a method for your class `KinPerson`, you should also write a couple of tests for it. And run your collection of tests frequently to make sure that everything works correctly.

Class `KinPersonTester` will contain your JUnit test suite; it will perform these testing tasks for you. Make sure that your test suite adheres to the following principles:

- For each method in your class `KinPerson`, your test suite should have **at least** one test case that tests that method.
- The more interesting or complex a method is, the more test cases you should have for it. What makes a method 'interesting' or complex can be the number of interesting combinations of inputs that method can have, the number of different results that the method can have when run several times, the different results that can arise when other methods are called before and after this method, and so on.
- Test very basic methods early in your test suite; then move on to more complex ones. (Make sure that car parts work on their own before you take the whole car for a test drive.)
- Don't try to test too many things in a single test case. Each test case should test only a couple of conditions.

Remember that if you change static variables in an early test, they will retain their values in later tests. Also, the tests are not necessarily run in the order in which you list them in your test suite. So when testing static variables, record their initial value at the beginning of the test, and test that the *change* in the value is what you expect.

Hints and Tips

- We suggest that you proceed as follows.

    - First, declare the fields in class `KinPerson`, compiling often as you proceed.
    - Second,
        - (1) Write the first constructor and all the getter methods of class `KinPerson`.
        - (2) Put a method in class `KinPersonTester` that tests whether the first constructor and all the getter methods work.
        - (3) Check that the first constructor and all the getter methods work as required. Don't go on to the next step until this is done.
    - Third, for the second constructor, perform the same three steps you did for the first constructor.
    - Fourth, write each of the setter methods, add a method in `KinPersonTester` to test them, and test them.
    - Fifth, write each of the comparison methods, add a method in `KinPersonTester` to test them, and test them.

  At each step, make sure all methods are correct before proceeding to the next step. When adding a new method, cut and paste the comment and the header from the assignment handout.

- Submit only files that end with the `.java`. Be careful about this, because in the same place as your `.java` files you may also have files that end with `.class` or `.java~`. but otherwise have the same name. In particular:
    - Microsoft operating systems hide file extensions by default. **Change this so that the extensions ALWAYS appear!** Don't let Microsoft tell you what to do.
    - DrJava creates "backup files", which means that you will see file names like "`KinPerson.java~`". The `~` indicates that the file is a backup file, which happens to be an older version of your program.

  Since `.class` files cannot be read by TAs or run with our testing programs, submitting the wrong files creates havoc with grading your assignment. Every year, a half-dozen students submit the wrong file. Don't do it!

- **Do not** use `if` statements when completing this assignment. For boolean expressions the `&&` (AND), `||` (OR), and ! (NOT) operators are sufficient to implement all the methods shown above. You will lose points for using `if` statements.
- Some of the `KinPerson` methods can be implemented easily by using other `KinPerson` methods that you have already created. Look for these cases, and take advantage of them as much as possible.
- Methods `substring`, `toUpperCase`, and `toLowerCase` in class `String` may be useful.
- Remember that a `String` literal is enclosed in double quotation marks and a `char` literal is enclosed in single quotation marks.
- Use method `.equals` to compare objects (including String objects) for equality and `==` to compare primitive values for equality.
- Only object variables can have the value `null`. So comparisons between primitive types and `null` are not legal.
- To create a JUnit test suite, select menu item `File ->?New JUnit Test Case`, and then replace the `testX` method with many methods that test your `KinPerson` functionality.