Project assignment 2

Quality development and testing

Overview
You are given an interface to a simple genealogical database system. The interface is not object-oriented, and similar to interfaces used by C libraries or simple communication protocols like RPC.

Your job is to provide an implementation for the interface. You should experiment with different techniques for quality assurance to create the best possible implementation. We provide you with the CDD version of EiffelStudio, which allows you to create, extract and manage test cases.

For this and the next assignment in this course you must use CDD EiffelStudio, even if you are not going to write test cases. Of course, test cases will help you to write better software.

Genealogical database
The genealogical database can be accessed through the interface defined in the class GENEALOGY. This interface is not object-oriented: parameters are restricted to INTEGER, BOOLEAN and STRING.

Different operations are provided to add information to the database (create a new person, add a marriage, adopt a person, etc.) and to query the database. Some queries implement complex algorithms to find relations between people.

There are only few preconditions provided for very basic checks. All other checks are implemented using defensive programming; if an operation would make the database inconsistent, the operation is completely ignored and an error message is stored in the query last_error.

The classes ROOT_CLASS and SIMPLE_PARSER provide you with a command line interface to your implementation that you can use for testing. A short help text is available using the command "help".

Dates are provided to the database as strings made out of ten characters of the form "DD.MM.YYYY"; "19.03.2008" would be such a string. Any of the three parts of the date can be replaced by "??" or "????" to mark an unknown value, but if the year is unknown, then the month needs to be unknown, and if the month is unknown, the day needs to be unknown. For example, the strings "???.???.2004" or "???.03.1920" are acceptable, but not the strings "01.???.2003" or "12.12.????".

Requirements and assumptions
You are to provide an implementation for the interface that satisfies two major soundness constraints:

- The database should at any time be structurally sound. Examples: person cannot be some other persons ancestor and descendant at the same time, a person cannot marry or adopt him- or herself, a dead person cannot marry.

- The database should at any time be chronologically sound. Examples: a father cannot be younger than his son, the death has to happen after the birth, divorces after marriages. Chronological soundness has to take unknown dates into account: if the birth date of the father is not known, then the son needs to be at least as young as the grandfather (and so forth).

Natural age limits should not be required or enforced: a person may have children or marry as soon as he/she is born, and there is no upper age limit.
Having multiple living spouses at the same time is not allowed. People remain married if the partner dies, but the living partner (the widow) can marry again. Gay marriages are allowed, but not marriages between direct relatives (siblings, direct ancestors). It is not allowed to be adopted by a descendant.

You can assume that new data is added to the database in the right chronological order. For example, marriages are added to a person in the right order, and before that person dies. You are allowed to reject the addition of new data that is not in the right order (for example, adding a marriage for a dead person, even if the marriage date lies before the death date). At any time, the database should be structurally and chronologically consistent.

**CDD EiffelStudio**

CDD EiffelStudio (contract-driven development) is a research branch of EiffelStudio that integrates an advanced testing infrastructure. It allows to create, extract, manage and run test cases as part of the software project. They are automatically checked after recompilation, offering a fast round-trip between testing and implementing. CDD EiffelStudio automatically extracts test cases from exceptions, allowing you to build up a test suite much more quickly than previously possible. You will have to use CDD EiffelStudio (and not the regular EiffelStudio). Download the tool from [http://dev.eiffel.com/CddBranch](http://dev.eiffel.com/CddBranch) and check back frequently for updates. The following will explain how to use the testing extension provided with CDD EiffelStudio:

**Manually Written Test Cases.** CDD EiffelStudio allows you to create a new “empty” test case. These test cases are similar to jUnit test cases. A manually written test class must start with the word “**TEST**” and all test routines also have to start with the word “test”. It also has to inherit from class **CDD_TEST_CASE**.

**Extracted Test Cases.** CDD EiffelStudio automatically extracts test cases whenever you run your program and an exception is triggered. This feature is novel and not yet part of any other testing environment. You will be the first to try it out.

**Test outcomes.** A test case checks whether your program contains a particular bug. A test cases can **fail** indicating that the bug is present in your program, or **pass** indicating your program does not contain this bug. Sometimes test cases will be **unresolved**, in which case the testing framework was unable to find out whether the test case **passed** or **failed**. A test case can be **unresolved** for several reasons. Please see the CddBranch web page for details.

![Testing](image)

**Debug Test Case.** Select a test case and press this button to run a test case in the debugger

**Enable/Disable Execution of Test Cases in Background.** If enabled, all test cases are retested every time you press compile. If disabled not test cases are executed.

**Enable/Disable automatic extraction of Test Cases.** If enabled every time an exception is
triggered a set of test cases that try to reproduce this exception is extracted. If disabled no test cases are extracted.

**Clean up/Delete.** You can use the “Clean Up/Delete” button in two different ways. By simply pressing it you will delete all unresolved test cases. By pick and dropping a test case to the “Clean up/Delete” button (right click on test case, move mouse to button and right click again) you can delete a test case. By the way, test cases are just regular classes. So you can use all existing tools that apply to classes in EiffelStudio too.

**Create new manual test class.** Press this button to create an empty test class. You can then edit the class to add manually written test cases.

### When to enable automatic extraction

If you are supervised by:

**Hermann or Wolfgang:** You must not have automatic extraction enabled for this assignment. (You will be allowed to use it for the next assignment). Make sure the testing panel looks like this for you:

![Testing Panel](image)

If the panel does not look like this, press the “Enable/Disable automatic extraction of Test Cases” button to change its state. You are allowed to add as many manually written test cases as you like. By the way, we can see through the logs whether you had extraction enabled. Don’t try to have it enabled and then delete the test cases. It will jeopardize our user study.

**Arsenii or Joseph + Roman:** You can use automatically extracted tests and manual tests for this assignment (In the next assignment you will only be allowed to use manual tests). Make sure the testing panel looks like this for you:

![Testing Panel](image)

If the panel does not look like this, press the “Enable/Disable automatic extraction of Test Cases” button to change its state. You can add/extract as many manual test cases/extracted test cases as you like.

### Teaching Goals

The goal of this exercise is to learn how to increase the quality of your own code. While the interface looks primitive at first, it has many intrinsic challenges to maintaining structural and chronological consistency. Also, some of the algorithms are difficult to implement.

Try to apply the techniques taught in the course, for example developing test cases or group-internal code reviews. The use of test suites and sample data is very important to verify the progress of the development and to prevent regression. The internal test suite represents the deployment of code into a real-world context.
**Deliveries**

1. The implementation of the GENEALOGY class and all other classes that you have developed for the implementation.

2. Test cases or sample data that you have developed to improve and assert the quality of your implementation.

3. A statement summarizing why you think your code is of good quality and how you have achieved this goal.

4. Log files generated by the CDD branch. Logs are stored together with the test cases in the "cdd_tests" folder. For every EiffelStudio session a new log file is created. Feel free to zip old logs to save some hard disk space.

**Grading**

Grading for this exercise will be based on three criteria:

1. Bugs contained in implementation, based on the internal test suite and inspection by the assistants.

2. Overall quality of the code, including design, readability, contracts, comments.

3. Supporting evidence about the code quality, including own test suites and description of the development process.

We are aware of the fact that some groups will work with automatic test-case extraction, while some other do not. The grading will be adjusted accordingly.

Your implementations should be hosted on the Origo development platform, available at [http://origo.ethz.ch/](http://origo.ethz.ch/). If you have not done yet, please open a project and add all group members as developers. Notify your assistant with the name of the project. Also, add your group assistant(s) and the user “aleitner” into the project group if it is a closed source project.