Biomedical Software Engineering: BMI2002

Fall 2017
Mount Sinai School of Medicine
Prof. Arthur Goldberg

Assignment 7: Program with exceptions and unit tests

Due Fri., Dec. 15

Logistics

Pass in your code by providing access to a Git repository.

Introduction

Exceptions are an extremely effective method for handling errors and other exceptional conditions in modern programming languages. And unit testing is an extremely effective method for detecting software bugs.

This assignment asks you to use exceptions to handle errors and unit tests to detect bugs.

Program

You will use an OO program on which you recently worked, the `Person` class. To ensure that all of you start with the same code, use a new version of the `Person` class I have written -- don't use code you wrote for the OO assignment. In particular, download a new `person.py` which contains example exceptions and a new `test_person.py` which contains example unit tests from the assignment 7 folder on our BMSE GitHub repo.

Exceptions

Read `person.py`. Following the example I showed in class, I've defined `PersonError` as an exception for reporting errors in `person.py`.

The class `Gender` defines gender constants. It includes a constant that represents an unknown gender, which is needed if a subject's gender is not known. The method `get_gender()` converts input gender synonyms into the appropriate constant. `get_gender()` raises a `PersonError` exception if an input gender does not map onto a standard gender or the unknown gender constant. Gender avoids data duplication by defining the relationships between gender constants and alternative values just once. The constructor for `Person` will raise a `PersonError` exception if the gender does not map to a reference gender value. `help(Gender)` and `Gender.genders_string_mappings()` provide public explanations of the mappings from gender synonyms to gender constants.

`Person` defines `set_mother()` which sets the mother of a person and includes the person in the mother's children. `set_mother()` raises an exception if the mother is not female.

Add the following code to `person.py`:

1. Create a method `set_father()` that is analogous to `set_mother()`. Have it raise an exception if the father is not male.
2. Create a method `add_child(self, child)` that adds `child` to a person's children. `add_child()` should use `set_father()` and `set_mother()` to establish parent-child relationships. Have `add_child()` raise an exception if the person does not have a known gender. Commented out tests for `add_child()` are already in `test_person.py`.
3. Create a method `remove_father()` which removes a person's father. Have it raise an exception if the person does not have a father or the person is not one of their father's children. A commented out test for `remove_father()` is already in `test_person.py`. Also write an analogous `remove_mother()` method.

Unit tests

As is customary, the unit tests for `person.py` are in `test_person.py`. Run the tests with the command

```
python test_person.py
```

Review `test_person.py`. Following common unit testing approaches, each `Person` method is tested by a different test method, and `Person` method `x()` is tested by unit test `test_x()`. In some cases, the exceptions thrown by method `x()` are tested by test `test_x_error()`. Follow this pattern for the rest of `Person`'s methods.

In general, test `test_X()` tests `X()` by calling `X()` and ensuring that it returns the correct values. The `assert*()` methods provided by `unittest` support dozens of ways of comparing the actual with the expected result of running the code under test. My example tests use just `assertEqual()`, `assertRaises()`, `assertIn()`, and `assertNotIn()`. I expect that these will be sufficient for your code.

`setUp()` is run by unittest before each test method to initialize state in the `unittest` that can be used by the test. My example creates a few `Person` instances.

Make your unit tests do the following:

1. Test all of the methods you have added to `Person`.
2. Write tests for these existing Person methods:

   ```
   get_persons_name()
   grandparents()
   all_grandparents()
   all_ancestors()
   ancestors()
   ```

   The code after the comment "make a deep family history" which is commented out creates a 4 generation deep family which you can use for testing these last 4 methods. Try to make the tests concise.

   Test the exception raised by `ancestors()`.

If your unit tests find bugs, fix them and note the fixes with comments in your code.