EECS3311 Fall 2019
Lab Exercise 5
Specifying the Iterable Repository ADT

Chen-Wei Wang

Common Due Date for Sections A & E: **11:59PM, Wednesday, December 4**

- **Check the Amendments section of this document regularly for changes, fixes, and clarifications.**
- **Ask questions on the course forum on the moodle site for Sections A and E.**

1 Policies

- **Your (submitted or un-submitted) solution to this lab exercise (which is not revealed to the public) remains the property of the EECS department. Do not distribute or share your code in any public media (e.g., a non-private Github repository) in any way, shape, or form. The department reserves the right to take necessary actions upon found violations of this policy.**
  - You are required to **work on your own** for this lab. **No** group partners are allowed.
  - When you submit your lab, you claim that it is **solely** your work. Therefore, it is considered as **an violation of academic integrity** if you copy or share **any** parts of your Eiffel code during **any** stages of your development.
  - When assessing your submission, the instructor and TA may examine your code, and suspicious submissions will be reported to the department/faculty if necessary. **We do not tolerate academic dishonesty**, so please obey this policy strictly.
- You are entirely responsible for making your submission in time.
  - You may submit **multiple times** prior to the deadline: only the last submission before the deadline will be graded.
  - Practice submitting your project early **even before it is in its final form**.
  - No excuses will be accepted for failing to submit shortly before the deadline.
  - Back up your work **periodically**, so as to minimize the damage should any sort of computer failures occur. Follow this tutorial series on setting up a **private** Github repository for your Eiffel projects.
  - The deadline is **strict** with no excuses: you receive **0** for not making your electronic submission in time.
  - Emailing your solutions to the instruction or TAs will not be acceptable.
- You are free to work on this lab on your own machine(s), but you are responsible for testing your code at a Prims lab machine before the submission.

2 Learning Outcomes of this Lab Exercise

1. Using the mathematical model library, create an abstract, mathematical abstraction for an ADT.
2. Write unit tests to verify the correctness of your software.
3. Draw professional architectural diagram using the draw.io tool.

3 Background Readings

- Mathematical Library
    - Lecture Slides, its recording, and example codes.
- Generic Parameters and Iterator Pattern:
- You can also find an abundance of resources on DbC, TDD, ESpec tests, and Eiffel code examples from these two sites:
    - `http://eiffel.eecs.yorku.ca`
    - `https://wiki.eecs.yorku.ca/project/eiffel/`

Contents

4 Getting Started

- Download the starter project archive `repository.zip` from the course moodle, unzip it, and launch EStudio.
- Choose `Add Project`, browse to the project configuration file `repository/repository.ecf`, then select `Open`.
- Study carefully the `EXAMPLE_REPOSITORY_TESTS` class, where:
  - Initially only the test feature `test_model` fails, because the abstraction function `model` in class `REPOSITORY` is not yet implemented.
  - All other tests pass, simply because:
    1. Correct implementations of all commands/queries in `REPOSITORY` and `TUPLE_ITERATION_CURSOR` have been given to you.
    2. All required contracts in `REPOSITORY` are all specified with the default expression **true**.

5 Tasks

In this lab exercise, you are given the correct implementation of all commands and queries in the `REPOSITORY` and `TUPLE_ITERATION_CURSOR` classes in Lab 2. You are then required to:

- Implement the abstraction function of `REPOSITORY`:

```
model: FUN[KEY, TUPLE[d1: DATA1; d2: DATA2]]
```

  Class `FUN` is from the `MATHMODELS` library. Study the `test_model` test feature in `EXAMPLE_REPOSITORY_TESTS` class to see how to manipulate the basic features of `FUN`. Make sure that you set the `MATHMODELS` environment variable (as done in Lab 0), so that you can also examine the entire API of `FUN` in Eiffel Studio.

- Specify postcondition of the `model` query by referring to the three implementation attributes: `keys`, `data_items_1`, and `data_items_2`.

  Requirement: This is the **only** place in which you can reference these three implementation attributes.

- Specify the pre- and post-conditions of all other commands and queries (e.g., `check_in`, `check_out`, `matching_keys`) in `REPOSITORY` as hinted by `-- TODO:`.

  Requirements:

  - You must specify each postcondition here by referring to `model` (or `Result` if applicable) **only**. Any reference to the three implementation attributes `keys`, `data_items_1`, or `data_items_2` is forbidden.
  - Do **not** add any auxiliary features to the `REPOSITORY` class.

  Any violation of the above requirements will result an **immediate zero** for this lab.

5.1 Programming

- Do not touch the given class `TUPLE_ITERATION_CURSOR`.
- Complete all contracts marked with `-- TODO:` in class `REPOSITORY`.
- Comments within the `REPOSITORY` class, together with the test queries in `EXAMPLE_REPOSITORY_TESTS` (in particular `test_model`), should provide you with information to complete all the programming tasks.

  Where contracts are expected to be written (except the case of the postcondition of `model`), you are forbidden to use the attributes (`keys`, `data_items_1`, and `data_items_2`); instead, you are only supposed to use the `model` query (and `Result` if applicable) to complete your contracts. Any violations of this rule will result in an immediate **zero** of the programming part of this lab.

5.2 BON Architecture Diagram

- Use draw.io to draw a BON Architecture Diagram that details the relationship between the following classes:
  - REPOSITORY
  - FUN
  - EXAMPLE_REPOSITORY_TESTS
- For the REPOSITORY class, show its detailed view with all model contracts.
- For the other two classes, show only the compact view (in ovals).
- When completing your drawing, also export the diagram into a PDF.
- Move the source of your draw.io drawing (name it `EECS3311_lab5_bon.xml`) and its exported PDF (name it `EECS3311_lab5_bon.pdf`) to the `docs` directory of your project.

6 Writing Your Own Tests for Contract Violations

- You are recommended (but **not required**) to write and submit tests testing the pre- and post-conditions you specify.

  However, your submitted code will be run against grading tests that test your pre- and post-conditions.
- All tests you write must go into the given STUDENT_TESTS class.
- Here is a short tutorial lecture on testing for contract violations:

  `https://www.eecs.yorku.ca/~jackie/teaching/lectures/2019/F/EECS3311/slides/TDD.pdf`
- Here is the example source code mentioned in the tutorial:

  `https://www.eecs.yorku.ca/~jackie/teaching/lectures/2019/F/EECS3311/codes/testing_of_contracts.zip`
- For the idea about writing postcondition tests, refer to part of this lecture recording (from **43:52** to **54:33**):

  `https://www.youtube.com/watch?v=JBP_s8rhF3M&list=PL5dxAmCmjv_4MZr4VH364zzRtsMvr1BLk&index=11`
- To test your postconditions, you are **allowed** to add additional "bad descendant" classes, but only to the `tests/student` cluster (where STUDENT_TESTS) is placed.
- Before submission, you must make sure that the STUDENT_TESTS is de-referenced from the root class TEST_REPOSITORY:

```
make
	do
		add_test (create {EXAMPLE_REPOSITORY_TESTS}.make)

		-- When submitting, make sure that you comment out this line
		-- to take out the reference to STUDENT_TESTS.
		-- add_test (create {STUDENT_TESTS}.make)}


		show_browser
		run_espec
	end
```

7 Submission

7.1 Checklist before Submission

1. Complete all contracts indicated with "`-- TODO:`" in the `REPOSITORY` class.
   - Do **not** modify any tags of the contracts. Do **not** modify signatures (names and types) of existing features. Do **not** add any auxiliary features to the `REPOSITORY` class.
   - You should exercise the test-driven development method that was taught.
   - Write and pass as many tests as possible on your software, as only passing the given tests will **not** ensure good quality of your development.
2. Use draw.io to draw a BON Architecture Diagram.
   - When completing your drawing, also export the diagram into a PDF.
   - Move the source of your draw.io drawing (name it `EECS3311_lab5_bon.xml`) and its exported PDF (name it `EECS3311_lab5_bon.pdf`) to the `docs` directory of your project.

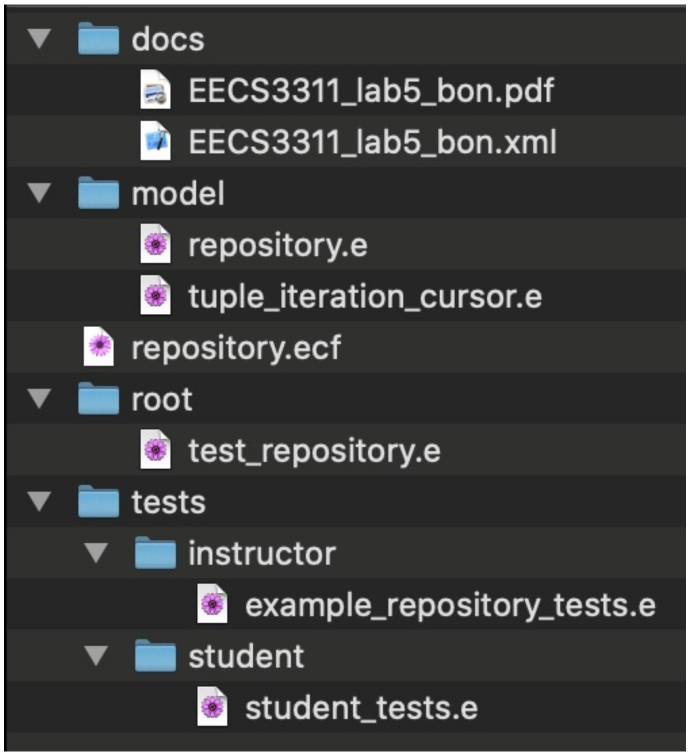
7.2 Submitting Your Work

To get ready to submit:

- Close EStudio
- Type the following command (only available via your lab account) to clean up the `EIFGENs` directory:

```
cd ~/Desktop/EECS3311_Labs/Lab_5
eclean repository
```

- Make sure the directory structure of your project is identical to Fig. 1 (with **no** `EIFGENs`). There might be new classes that you add into `tests/student` to test postconditions.



```
▼ docs
      EECS3311_lab5_bon.pdf
      EECS3311_lab5_bon.xml
▼ model
      repository.e
      tuple_iteration_cursor.e
  repository.ecf
▼ root
      test_repository.e
▼ tests
   ▼ instructor
         example_repository_tests.e
   ▼ student
         student_tests.e
```

Figure 1: Lab 5: Expected Diretory Structure

– You must make sure that the STUDENT_TESTS is de-referenced from the root class TEST_REPOSITORY:

```
make
	do
		add_test (create {EXAMPLE_REPOSITORY_TESTS}.make)

		-- When submitting, make sure that you comment out this line
		-- to take out the reference to STUDENT_TESTS.
		-- add_test (create {STUDENT_TESTS}.make)}


		show_browser
		run_espec
	end
```

By the due date, submit via the following command:

```
cd ~/Desktop/EECS3311_Labs/Lab_5
submit 3311 Lab5 repository
```

A check program will be run on your submission to make sure that you pass the basic checks (e.g., the code compiles, passes the given tests, *etc*). After the check is completed, feedback will be printed on the terminal, or you can type the following command to see your feedback (and later on your marks):

```
feedback 3311 Lab5
```

In case the check feedback tells you that your submitted project has errors, you must fix them and re-submit. Therefore, you may submit for as many times as you want before the submission deadline, to at least make sure that you pass all basic checks.

Note. You will receive zero for submitting a project that cannot be compiled.

8 Amendments

– The only contract that you can mention the attributes, of course, is the postcondition of the `model` feature.