EECS3311 Fall 2019
Lab Exercise 2
Implementing an Iterable Repository ADT

Chen-Wei Wang

Common Due Date for Sections A & E: 2pm, Friday, October 11

- **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 Java 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. Infer the necessary features (with correct headers) and classes from the given unit test.
2. Apply the Iterator Pattern.
3. Define a Generic Class.
4. Write unit tests to verify the correctness of your software.
5. Draw professional architectural diagram using the draw.io tool.

3 Background Readings

- Tutorial Videos on Generic Parameters and the Iterator Pattern
- Tutorial Videos on Information Hiding and the Iterator Pattern
- Lecture slides/recordings on `Complete Postcondition` and on `Generic Parameter and Iterator Pattern`: `https://www.eecs.yorku.ca/~jackie/teaching/lectures/index.html#EECS3311_F19`
- 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 Problem

The REPOSITORY ADT supports the storage and retrieval of a collection of data sets. Each data set consists two data items (which may be of different types) and can be uniquely identified by a search key (which can also be of a different type).

A client of the REPOSITORY ADT is expected to instantiate types of the search key and the two kinds of data items. For example, the declaration `repos: REPOSITORY[INTEGER, STRING, DATE]` intends to have a birthday book, where each data set consists of a person's string name and their birthday date, and each data set is uniquely identified by some integer key. You are required to implement the REPOSITORY via the following implementation strategy, where there are:

- Two separate linear structures (ARRAY and LINKED_LIST) storing the keys and the first data items
- A hash table mapping keys to the second data items

For example:

keys ⟶

1	2

data_items_1 ⟶

"Suyeon"	"Yuna"

data_items_2 ⟶

KEY	VALUE
1	2013-8-31
2	2016-6-20

The first data items (i.e., names) of a data set can be identified via the index of their associated search key, whereas the second data items (i.e., birthday dates) can be identified via inquiries to the hash table. In the above example, the data set ("Suyeon", 2013-8-31) can be identified by the search key 1.

5 Getting Started

- Go to the course moodle page for Sections A and E. Under Lab 2, download the file repository.zip which contains the starter project for this lab.
- Unzip the file, and you should see a directory named repository.
- Move the repository project into where you stored the Lab0 project (e.g., ∼/Desktop/eecs3311_workspace).
- Now it is assumed that the *project location* of the current lab is: ∼/Desktop/eecs3311_workspace.
- Add the repository project to EStudio and try to compile it.
- **It is expected that the starter project does not compile (and you'd have to encounter a few rounds of compilation errors until you fix them all).** This is because there are missing classes, inheritance declarations, generic parameter declarations, and feature definitions.
- **Study carefully the EXAMPLE_REPOSITORY_TESTS class, where test cases are provided to illustrate how the various features are expected for you to implement.** Pay attention to types of variables.
- This is the suggested workflow for you:
  - You should start by adding the necessary classes and feature declarations in order to at least make the entire project compile. You receive 0 marks if your submitted project does not compile.
    Any implementation classes that you add must be created under the model cluster.
  - Then, implement those features so that the provided tests pass.
  - Then, add your own tests until you are satisfied before submission.
    Any test classes that you add must be created in the STUDENT_TESTS class.

6 Your Tasks

6.1 Complete the REPOSITORY and DATA_SET Classes

- A generic class `REPOSITORY[KEY -> HASHABLE, DATA1, DATA2]` is already declared for you, where `KEY`, `DATA1`, and `DATA2` denote types of, respectively, the search key and its associated first and second data items.
- Notice that in Eiffel collection classes such as `ARRAY`, `LINKED_LIST`, `HASH_TABLE`, and `TUPLE`:
  - There is a Boolean attribute `object_comparison`, which indicates, when the $\sim$ operator is used, if items in the collections are to be compared using reference equality (i.e., via = when `object_comparison` is `false`) or object equality (i.e., via the user-redefined `is_equal` feature when `object_comparison` is `true`).
  - By default, the `object_comparison` attribute is `false`. If you mean it to be `true`, you can call the command `compare_objects`.
  - See the `test_array_comarison` feature in class `EXAMPLE_REPOSITORY_TESTS` for an example.
- Pay attention to the test `test_hash_table_retrieval`, which shows you some programming tips on retrieving items from a hash table.
- For this lab exercise, you are forced to implement the REPOSITORY ADT via a pre-determined solution using two linear data structures and hash table: a linked list of keys, an array of first data items, and a hash table mapping from keys to second data items. A *valid* combination of some first value (from the `data_items_1` array), some second value (from the `data_items_2` table), and some key (from the `keys` list) make an existing data set in the repository.
- You are required to make the `REPOSITORY` class *iterable*.
- You are forbidden to use any other additional attributes to implement the repository. Such a constraint is imposed to make you practice the "hard" case of implementing the Iterator Pattern.
- Identify all `-- TODO` comments which indicate what you should complete. Notice that there is a task for you to complete in the `DATA_SET` class.
- For this lab, it is OK for you to refer to the three implementation attributes `keys`, `data_items_1`, and `data_items_2` in the postconditions. As a hint, given that `REPOSITORY` is iteratble, try to use `across Current as ... end` whenever possible.
- Comments within the `REPOSITORY` class, together with the test queries in `EXAMPLE_REPOSITORY_TESTS`, should provide you with information sufficient to complete all the programming tasks.
- You must **not** change any of the feature names, parameters, or contract tags.
- You must **not** add any additional preconditions, postconditions, or class invariants.
- In the `STUDENT_TESTS` class, you are required to add as many tests as you judge necessary to test the correctness of `REPOSITORY` and `DATA_SET`.
  - You must add ***at least 10*** test features in `STUDENT_TESTS`, and all of the must pass. (In fact, you should write as many as you think is necessary.)
  - You will not be assessed by the quality or completeness of your tests (i.e., we will only check that you have at least 10 tests and all of them pass). However, **write tests for yourself** so that your software (implementation and contracts) will pass all tests that we run to assess your code.
  - For each test feature, always start with a call to `comment(...)`.

6.2 BON Architecture Diagram

– Use draw.io to draw a BON Design Architecture Diagram that details **all** classes in the `model` cluster (i.e., supplier classes) and the `EXAMPLE_REPOSITORY_TESTS` class (i.e., the client class). Your supplier classes should clearly illustrate the architecture of the Iterator Pattern (with the appropriate client-supplier and inheritance relations).

You do **not** need to show all features. We leave it to your judgement to summarize the critical classes/features/contracts for others to understand the design of the project in a single page.

– When completing your drawing, also export the diagram into a PDF.

– Move the source of your draw.io drawing (name it `EECS3311_lab2_bon.xml`) and its exported PDF (name it `EECS3311_lab2_bon.pdf`) to the `docs` directory of your project.

6.3 Short Report

– Compile (into a single PDF file named `Report.pdf`) including:

- A cover page that clearly indicates: 1) course (`EECS3311`) and section (A or E); 2) semester (`Fall 2018`); 3) name; and 4) CSE login;
- With the limit of **one page**:
  - ◇ Explain how the Iterator Pattern is implemented in the `model` cluster.
  - ◇ Explain how you implement the feature `another_cursor` in the `REPOSITORY` class.

7 Submission

7.1 Checklist before Submission

1. Complete all contracts and implementations indicated with "`-- TODO`" in the starter code.
   - Do **not** modify any tags of the contracts. Do **not** modify signatures (names and types) of existing features. You **may** add auxiliary features to existing classes if necessary.
   - 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_lab2_bon.xml`) and its exported PDF (name it `EECS3311_lab2_bon.pdf`) to the `docs` directory of your project.
3. Move the source of your report (`Report.pdf`) into the `docs` directory. Do **not** put any other format of the report (e.g., word).

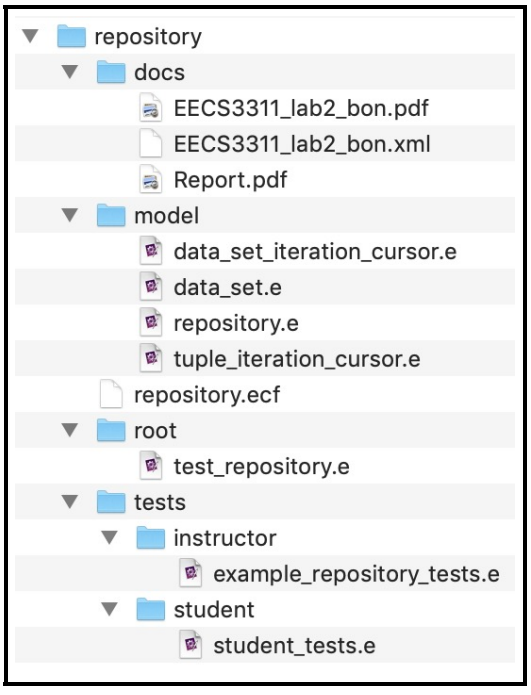
7.2 Submitting Your Work

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_workspace
eclean repository
```

- Make sure the directory structure of your project is consistent to (with **no** `EIFGENs`):



By the due date, submit via the following command:

```
cd ~/Desktop/eecs3311_workspace
submit 3311 Lab2 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 Lab2
```

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