EECS-3311 – Lab – Sorted Maps

L2

12/31/18 5:15:53 PM

1 Goals

```
require
     across 0 |..| 1 as i all lab_completed(i.item) end
     read accompanying document: Eiffel101[1]
ensure
     submitted on time
     no submission errors
rescue
     ask for help during scheduled labs
     attend office hours for TA William
```

Implement Inheritance
- Declare inheritance
- Implement inherited, deferred features
- Use of commands from the Mathmodels library

Specify Contracts
- Use of queries from the Mathmodels library

[1] See: https://www.eecs.yorku.ca/~eiffel/eiffel101/Eiffel101.pdf

2 Getting started

These instructions are for when you work on one of the EECS Linux Workstations or Servers (e.g *red*). You should not compile on *red* as it is a shared server; compile on your workstation. Invoke the Eiffel IDE from the command line as `estudio18.11` (aliased to `estudio`) and the command line compiler is `ec18.11` (aliased to `ec`).

2.1 Retrieve and compile Lab2

```
> ~sel/retrieve/3311/lab2
```

This will provide you with a starter directory `sorted-map`. The directory has the following structure:

```
sorted-map/
├── model
|   ├── sorted_map_adt.e
|   └── util.e
├── root
|   └── root.e
├── sorted-map.ecf
└── tests
    ├── instructor
    |   ├── int_key_tests.e
    |   ├── sorted_model_tests.e
    |   ├── string_key_tests.e
    |   └── test_util.e
    └── student
        └── student_tests.e
```

Table1 1 Lab2 Directory Structure

The three classes in **red** are **incomplete**, and you are required to:

- In class `UTIL`, complete all contracts and implementations with the **TO DO** labels.
- In class `STUDENT_TESTS`, add as many (at least four) test cases as you judge necessary for testing the correctness of your software.
- There is a missing class, which is indicated by the class `SORTED_MODEL_TESTS`, and implement features in these classes.

You can now compile the Lab.

```
> estudio sorted-map/sorted-map.ecf &
```

where `sorted-map.ecf` is the Eiffel configuration file for this Lab. The root file looks like this:

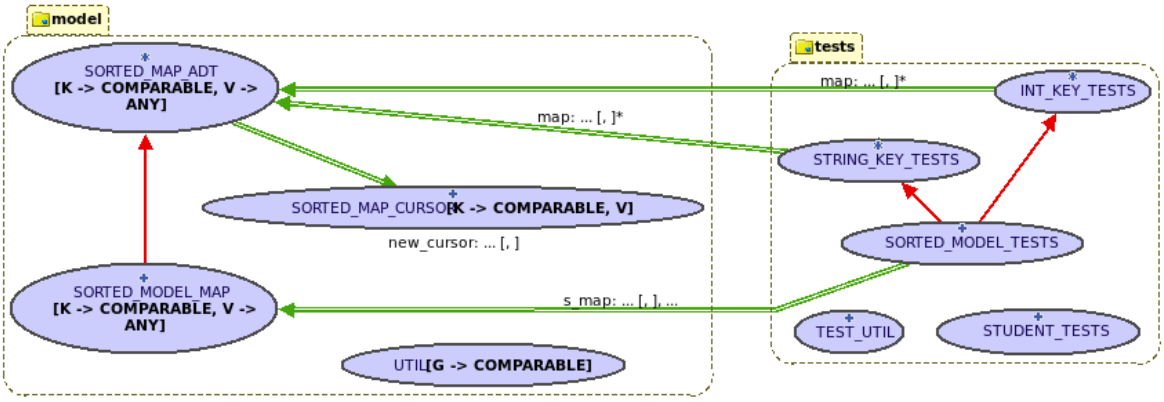
```
class
    ROOT

inherit
    ARGUMENTS
    ES_SUITE

create
    make

feature {NONE} -- Initialization

    make
            -- Run application.
        do
            add_test (create {TEST_UTIL}.make)
            add_test (create {STUDENT_TESTS}.make)
--          add_test (create {SORTED_MODEL_TESTS}.make)
            show_browser
            run_espec
        end

end
```

Some of the tests are commented out (i.e., those specified in class `SORTED_MODEL_TESTS`). The project will compile and when you execute the Lab in workbench mode (Control-Alt-F5), you will see the following *ESpec* Unit Testing report:

FAILED (7 failed & 0 passed out of 7)		
Case Type	**Passed**	**Total**
Violation	0	0
Boolean	0	7
All Cases	0	7
State	**Contract Violation**	**Test Name**
Test1	TEST_UTIL	
FAILED	Postcondition violated.	t1: test merge sort of empty array
FAILED	Precondition violated.	t2: merge two empty arrays
FAILED	Postcondition violated.	t3: concat two empty arrays
Test2	STUDENT_TESTS	
FAILED	NONE	t1: describe test t1 here
FAILED	NONE	t2: describe test t2 here
FAILED	NONE	t3: describe test t3 here
FAILED	NONE	t4: describe test t4 here

Notice that the four default tests given to you in class STUDENT_TESTS all **fail** initially, simply because their Boolean results are all set to *false*. You are expected to modify these tests, and add as many other tests as you judge necessary, that would suitably test your implementation.

The BON diagram, generated by EiffelStudio, is here for your reference:



2.2 Get the initial Unit Tests working

The Red Bar means that some of the tests fail. You must get all the tests to work and obtain a Green Bar.

- `STUDENT_TESTS`: you must write your own tests and we will check your tests. You may insert as many tests as you wish in this class. The *minimum* number of tests to add in this class is 4.
- `TEST_UTIL`: We provide you with some *very* basic tests to get features in class `UTIL` working correctly. See the `To Do` hints in this class. Many more tests on features of the `UTIL` class are expected to be developed by you.

In the `UTIL` class, complete the following features and their contracts as hinted by the `To Do` comments:

- `concatenate (a, b: ARRAY [G]): ARRAY [G]`
  You are required to complete both the implementation and contracts (postcondition) for this query, which intends to return the concatenation of the two input arrays "a" and "b".
- `merge_sort(a: ARRAY[G]): ARRAY[G]`
  The implementation for this feature is already completed for you, and it calls another helper query `merge` which you are required to complete. This query intends to implement a standard, recursive merge sort routine. You are only required to complete the postcondition for this merge sort routine.
- `merge (left, right: ARRAY[G]): ARRAY[G]`

You are required to complete both the implementation and contracts (precondition and postcondition) for this query. This query is called as a helper routine in the `merge_sort` query.

Once you have all the incomplete features in class `UTIL` working, and added a reasonable number of tests in the `STUDENT_TESTS` class, you should now obtain a Green Bar:

PASSED (7 out of 7)

Case Type	Passed	Total
Violation	0	0
Boolean	7	7
All Cases	7	7

State	Contract Violation	Test Name
Test1	TEST_UTIL	
PASSED	NONE	t1: test merge sort of empty array
PASSED	NONE	t2: merge two empty arrays
PASSED	NONE	t3: concat two empty arrays
Test2	STUDENT_TESTS	
PASSED	NONE	t1: describe test t1 here
PASSED	NONE	t2: describe test t2 here
PASSED	NONE	t3: describe test t3 here
PASSED	NONE	t4: describe test t4 here

2.3 Get all the remaining Unit Tests working

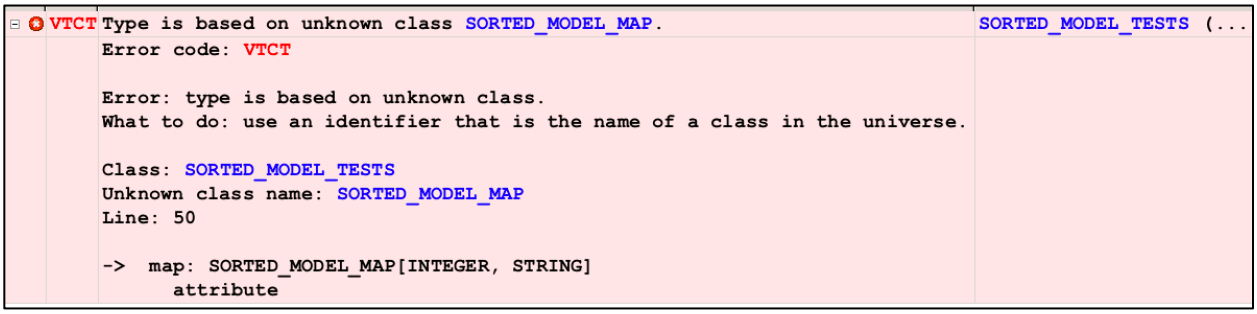
To complete the remaining tests, uncomment the line

add_test (**create** {`SORTED_MODEL_TESTS`}.make)

in class ROOT:

```
class
    ROOT

inherit
    ARGUMENTS
    ES_SUITE

create
    make

feature {NONE} -- Initialization

    make
            -- Run application.
        do
            add_test (create {TEST_UTIL}.make)
            add_test (create {STUDENT_TESTS}.make)
            add_test (create {SORTED_MODEL_TESTS}.make)
            show_browser
            run_espec
        end

end
```

Now save and re-compile the lab project. It is *expected* that the compilation would fail with the following error message:



What does the above error message mean to say? Error messages from a programming IDE (e.g., EiffelStudio, Eclipse, *etc.*) are always meant to hint on how a developer should fix the corresponding compilation errors. Try to interpret the above error message before continuing to the next page.

In class SORTED_MODEL_TESTS, there is a reference to some class SORTED_MODEL_MAP which does not actually exist. Double-clicking on the first line "Type is based on unknown class SORTED_MODEL_MAP" would bring you to where the error occurs:

SORTED_MODEL_TESTS

```
map: SORTED_MODEL_MAP[INTEGER, STRING]
    attribute
        create Result.make_empty
    end
```

The return type SORTED_MODEL_MAP of query map denotes a class that was not given to you. Your task now is to create this class and implement all its features:

```
class
    SORTED_MODEL_MAP [K -> COMPARABLE, V -> ANY]

inherit
    SORTED_MAP_ADT[K,V]

end
```

As can be seen from the BON diagram in the *docs* directory, SORTED_MAP_ADT is a child class of SORTED_MODEL_MAP. Consequently, all *deferred* (i.e., unimplemented) features inherited from the parent class SORTED_MODEL_MAP should be *effected* (i.e., implemented), otherwise the child class SORTED_MAP_ADT should also be declared as *deferred*. In our case, we do not want to declare SORTED_MODEL_MAP as a deferred class, so you must implement *all* deferred features inherited from SORTED_MAP_ADT.

To get you started, consider the following two features in SORTED_MODEL_MAP:

```
feature -- model

    model: FUN [K, V]
            -- abstraction function
        do
            Result := implementation
        end

feature{NONE} -- attributes

    implementation: FUN[K,V]
            -- inefficient but abstract implementation of sorted map
        attribute
            create Result.make_empty
        end
```

In the above fragment of code in SORTED_MODEL_MAP:

- Feature implementation is declares as an attribute of type FUN[K, V] (see this class in Mathmodels for its API) and initialized as an empty function.
- You may then want to use this implementation attribute to implement all remaining features in SORTED_MODEL_MAP. To see which features are supported on implementation, study the API of the FUN class (for mathematical functions) in the Mathmodels library.
- For example, query model is declared as deferred in the parent class SORTED_MAP_ADT, and in the current class SORTED_MODEL_MAP, we choose to implement it as an alias (via a reference assignment :=) to the implementation attribute.
  Note. Observe that contracts in the parent class SORTED_MAP_ADT are specified in terms of the model query. For example:

```
extend (key: K; val: V)
        -- inserts an element of `key' and `value' into map
    require
        key_unique: not has (key)
    deferred
    ensure
        extended: model ~ ((old model.deep_twin) + [key, val])
    end
```

  At the level of deferred class SORTED_MAP_ADT contracts that are specified using model are not executable, because model itself is deferred. However, at the level of an effective descendant, such as SORTED_MODEL_MAP, given that we implement the deferred query model, these contracts are not only inherited from the parent class (which is called *subcontracting* and we will discuss this in detail later in class) but also executable and testable at runtime.

You then are required to supply implementations for all inherited deferred features in the SORTED_MODEL_MAP class and get all these remaining tests working.

Important Note: *To check syntax, a compile (shortcut F7) is sufficient. But it is best to freeze (Control-F7) before running unit tests. Run the unit tests often! (even after very small changes to your code). When you compile, ensure that the compilation succeeded (reported at the bottom of the IDE). When you run unit tests, ensure that all your routines terminate (can also be checked in the IDE). If you keep running the tests without halting the current non-terminating run, you will keep adding new non-terminating processes to the workstation, and the workstation will choke on all the concurrently executing processes. Study how to use your tools effectively and efficiently.*

3 To Submit

1. Add correct implementations as specified.
2. Work incrementally one feature at a time. Run all regression tests before moving to the next feature. This will help to ensure that you have not added new bugs, and that the prior code you developed still executes correctly.
3. Add at least 4 tests of your own to `STUDENT_TESTS`, i.e. don't just rely on our tests.
4. Don't make any changes to classes other than the ones specified.
5. Ensure that you get a green bar for all tests. Before running the tests, always freeze first.

You must make an electronic submission as described below.

1. On Prism (Linux), *eclean* your system, freeze it, and re-run all the tests to ensure that you get the green bar.
2. *eclean* your directory *sorted-map* again to remove all EIFGENs.

Submit your Lab from the command line as follows:

```
submit 3311 Lab2 sorted-map
```

You will be provided with some feedback. Examine your feedback carefully. Submit often and as many times as you like.

Remember

- Your code must compile and execute on the departmental Linux system (Prism) under CentOS7. That is where it must work and that is where it will be compiled and tested for correctness.
- Equip each test t with a *comment* ("t: …") clause to ensure that the ESpec testing framework and grading scripts process your tests properly. (Note that the colon ":" in test comments is mandatory.). An improper submission will not be given a passing grade.
- The directory structure of your folder *sorted-map* **must** be a superset of Table1 1

4 Appendix: Study the Implemented Iterator Pattern

How is SORTED_MODEL_MAP *iterable* automatically? There is no magic:

```
deferred class
    SORTED_MAP_ADT [K -> COMPARABLE, V -> ANY]

inherit

    ITERABLE [TUPLE [K, V]]
        redefine
            is_equal,
            out
        end

feature --iteration
    new_cursor: ITERATION_CURSOR [TUPLE[key: K; value: V]]
            -- Fresh cursor associated with current structure
        do
            Result := as_array.new_cursor
        end
```

The above code indicates that:

- The parent class SORTED_MAP_ADT is declared as *iterable.*
- The required feature new_cursor is already implemented (and also inherited to SORTED_MODEL_MAP), by leveraging the return value from query as_array, which is also *iterable*.

Consequently, we may iterate through an instance of SORTED_MODEL_MAP using the "across" construct. For example, here is a test that illustrates the *iterable* behavior of SORTED_MODEL_MAP (where the map is iterated in a sorted order of its keys):

```
t: BOOLEAN
        -- Test
    local
        m: SORTED_MAP_ADT[INTEGER, STRING]
        tuples: LINKED_LIST[TUPLE[k: INTEGER; v: STRING]]
    do
        comment("t: test iterable model map")
        create {SORTED_MODEL_MAP[INTEGER, STRING]} m.make_empty
        m.extend (2, "two")
        m.extend (1, "one")
        create tuples.make
        across
            m as cursor
        loop
            tuples.extend (cursor.item)
        end
        Result :=
            tuples.count = 2
            and
            tuples[1].k = 1 and tuples[1].v ~ "one"
            and
            tuples[2].k = 2 and tuples[2].v ~ "two"
    end
```

Exercise: Add the above test to your project and make sure it passes. Can you create more tests that illustrate the *iterable* behavior of SORTED_MODEL_MAP, possibly with a different type of keys?