Exercise Set 9: Loops with Iterators

Exercise 9.1 Study the given `IRange interface` for an iterator and its use with the list of `Student`. An example shows how to use the iterator to find whether a student with the given name is in the list and how to print all items in the list.

- Draw the UML diagram of this collection of classes.
- Develop another example of the use of this iterator to determine whether a student with gpa greater than 3.5 is in the list.
- Use a similar technique to design a method in the test suite, which computes the best gpa of all `Students` in this list.
- Use a similar technique to design a method in the test suite, which counts the number of students in this list.
- Use a similar technique to design a method in the test suite, which computes the average gpa of all students in this list.

Exercise 9.2 The given code defines also an `ArrayRange` iterator and a `TreeRange` iterator for binary trees. Use the `ArrayRange` iterator and the `TreeRange` iterator to perform the same tasks as in the previous example.

Exercise 9.3 The given code defines a binary search tree of `Integer`s (BST), and an iterator which implements `IRange` to traverse the tree in *inorder*.

- Develop a test suite that tests the iterator on a BST with at least 7 nodes.
- Design and test a `ReverseTreeRange` iterator which traverses the BST in ***reverse inorder***.

Exercise 9.4 Design the class `DataSet` which has as its member data a data collection of `Comparable` objects, such as list of `Integer`s, or an array of `String`s, and a corresponding iterator. Design the following methods in this class:

- `findItem` method, which determines whether a given object appears in the data collection
- `count` method, which counts the number of items in the collection
- `minimum` method, which returns the minimum item in the collection. The method may assume that it will only be invoked by with a non-empty collection.

Make sure you develop the tests for these methods that use at least two different data collection and their corresponding iterators.

Exercise 9.5 (Will be avilable later.) Use the given `IRange interface` and its `FileRange` implementation to perform the following tasks:

- Develop the classes to represent a Binary Search Tree (`BST`)of arbitrary `Comparable` objects.
- Develop the method which reads the data from a file using the `FileRange` iterator and builds a `BST`. Test your result using the code from previous exercises.

Exercise Set 10: Loops with Counters

Exercise 10.1 Design the class ArrayAlgorithms. Its member data is an array on Comparable objects. Develop the following methods for this class:

- `find` method which determines whether a given object is one of the elements of this array.
- `findMinLocation` method which returns the index for the smallest item in this array.
- `floor` method which consumes another array of the same size and returns a new array of the same size in which each element is the smaller of the two corresponding elements in the original arrays.
- `filter` method which consumes a Comparable data item and produces a new array which contains only those items from the original array that are smaller that the given item.
- `sort` method which consumes an array of Comparable data items and produces a new array which contains the same elements, but sorted in ascending order.

Write the tests for these methods in the test suite.
Test your code on arrays of `String`s and arrays of `Integer`s.

Exercise 10.2 The given code specifies a `Double2Double interface` and a `Plot class`. The constructor for the `Plot` class consumes `Rectangle2D` object which specifies the region for the display of the function graph. The `Plot` class also includes the methods `plotAxes()` which plots the axes for the graph, and `plotValue(double x, double y)` which plots the value `y` for the point `x`.
Write the class `FunctionPlot` as follows. The member data specify the function to plot - an object in the class which implements the `Double2Double interface`. Develop the following methods in the class `FunctionPlot`:

- `minimum` method which consumes the `double` values `x1` and `x2` - the two ends of the interval on which the function should be plotted and an `int` value `n` which specifies the number of points to plot and returns the minimum value of this function among the `n` points.
- `maximum` method which consumes the `double` values `x1` and `x2` - the two ends of the interval on which the function should be plotted and an `int` value `n` which specifies the number of points to plot and returns the maximum value of this function among the `n` points.
- `plotFunction` method which consumes the `double` values `x1` and `x2` - the two ends of the interval on which the function should be plotted and an `int` value `n` which specifies the number of points to plot. The function returns `void`, but displays the graph with axes in the graphics window.

Exercise 10.3 This is an independent continuation of the previous exercise. Develop the method `integral` which for the given (`double`) interval `(x1, x2)` computes the value of the integral of this function, approximated to the value of a given `epsilon`.

Exercise Set 11: The Meaning of Equality

Exercise 11.1 Create a class `Person`: with name, eyecolor, date of birth, and address. Create a class `Address` with city and zip code only.

- Define three a `Address` objects and four `Person` objects. Design examples to illustrate the problem with assignment and mutation. Write comments explaining the problem.
- Define a shallow copy constructor for the class `Person` and show on examples when it fails.
- Define a `copy` method in the class `Person`, which returns a new copy of the given object. Design and run test that verify that your code works properly.
- Define the method `equals`, which compares two person objects and returns true if the two people have the same name, eyecolor, date of birth, city, and zip, even if they are not represented by the same object. Design tests to verify that your method works correctly.

Exercise 11.2 Start with an array of Person. Experiment with making copies of the array, modifying people in the first array, observe what happens in both. First make the copy by assignment, then using the incorrect copy constructor, then using the copy method developed in the previous exercise.