Exercise Sets: How to Design Class Hierarchies

Viera K. Proulx and Matthias Felleisen

February 28, 2003

Exercise Set 1A: Exploring Numbers and Arithmetic

Exercise 1A.1 Use the function `computeInt` to determine the result of the following expressions:

```
27 % 4,
-27 % 4,
27 % -4,
-27 % -4,
27.5 % 4.
```

Exercise 1A.2 Use the function `computeInt` to determine the result of the following expressions:

```
150 / 3 * 10
150 / (3 * 10)
10 / 3 * 10.
```

Exercise 1A.3 Use the function `computeInt` to determine the result of the following expressions:

```
150 - 20 - 30 - 10
150 - (20 - 30) - 10)
(150 - 20) - (30 - 10).
```

Exercise 1A.4 Use the function `computeInt` to determine the result of the following expressions. Experiment with the second expression to find out what is the largest integer Java can represent.

```
4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4
16 * 16 * 16 * 16 * 16 * 16 * 16 * 16 * 16 * 16.
```

Exercise 1A.5 Use the function `computeInt` to determine the result of the following expressions. Experiment with the second expression to find out what is the largest magnitude of a negative integer that Java can represent.

```
-4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4
-16 * 16 * 16 * 16 * 16 * 16 * 16 * 16 * 16 * 16.
```

Exercise 1A.6 Use the function `computeDouble` to determine the result of the following expressions:

```
27.6 % 4
```

```
-27.6 % 4
27.6 % -4
-27.6 % -4.
```

Exercise 1A.7 Use the function `computeDouble` to determine the result of the following expressions:

```
10 / 3 * 3.0)
10 / 3.0 * 3
10.0 / 3 * 3
10.0/3/3/3/3/3 *3*3*3*3*3
10.0/3/3/3/3/3/3/3/3/3/3 *3*3*3*3*3*3*3*3*3*3
10.0 *3*3*3*3*3*3*3*3*3*3 /3/3/3/3/3/3/3/3/3/3
1E20 /3/3/3/3/3/3/3/3/3/3 *3*3*3*3*3*3*3*3*3*3
1E20 *3*3*3*3*3*3*3*3*3*3 /3/3/3/3/3/3/3/3/3/3.
```

Exercise 1A.8 Use the function `computeDouble` to determine the result of the following expressions:

```
1.0 / 0
-1.0 / 0
0.0 / 0
1 / 0.
```

Exercise 1A.9 Find out whether the function `areaOfDisk` allows the input to be an integer.

Exercise 1A.10 Find out what happens if you use `int` return type in the function `areaOfDisk` instead of `double`.

Exercise Set 1B: Developing Functions; Function Composition

Exercise 1B.1 Test the following four functions with the same set of inputs. Try to predict the outcome. Explain the differences between the different results.

```
c1(f):  ((f-32)*5)/9
c2(f):  (f-32)*(5/9)
c3(f):  (((f-32)/9)*5)
c4(f):  (f-32*5/9)
```

Exercise 1B.2 Develop the function `fToC` that converts the temperature given in Fahrenheit to a temperature in Celsius. In this case represent all temperatures as integers, which is sufficient for weather reports.

Exercise 1B.3 Develop the function `travelTime` that computes how long a car trip takes (in hours and fractions of an hour), given the distance in miles and the estimated average speed in miles per hour.

Exercise 1B.4 Develop the function `travelTimeInMinutes` that computes how long a car trip takes (in minutes), given the distance in miles and the estimated average speed in miles per hour. Use the function `travelTime` as a helper function.

Exercise 1B.5 Develop the function `numberOfPages` that estimates the number of pages in a document, given the number of words in the document. Assume that a page has on the average 60 lines and a line has on the average 15 words in it.

Exercise 1B.6 Develop the function `timeToDownload` that estimates the time for downloading an image file from the web, given the pixel size and the download speed. The size of the picture is given by the number of pixels in each row and the number of rows of pixels. The download speed is given in the number of bytes per second. Each pixel requires three bytes.

Exercise 1B.7 Develop the function `unitPrice` that computes the unit price of a grocery item, given the weight and price of an item. The weight of the item is given in pounds and ounces. The price of the item is given in cents (i.e., for an item that costs $2.45 the price is given as the integer value 245).

Exercise 1B.8 Develop the function `howManyTiles` that computes how many tiles are needed to cover the floor during kitchen renovation, given the area of the kitchen and the size of the tile. The area of the kitchen is given in square feet. The size of the tile is given in inches. Assume the tiles are square.

Exercise Set 1C: Functions with Conditionals

Exercise 1C.1 Develop the function `taxRate` that determines the tax rate for a given income level. If the income is under $6000, there is no tax. For an income under $20000 the tax is 15%, for income under $70000 the tax is 24%, for $70000 and over the tax is 33%.
Develop the function `computeTax` that will compute the amount of tax for the given income, using the tax rates as determined earlier.

Exercise 1C.2 Develop the function `ticketType` that produces an age label (a `String`) for a ski ticket, given the age of the skier. The result is `"Child"` for those under six years old, `"Youth"` for those under twelve, `"Student"` for those under 21, `"Adult"` for those under 65, and `"Senior"` for anyone 65 and older.

Exercise 1C.3 Develop the function `letterGrade` that consumes the exam score (between 0 and 100) and produces a character representing the earned grade as follows: 85 and above is A, 70 and above is B, 60 and above is C, 50 and above is D, below 50 is F.

Exercise 1C.4 Develop the function `passing` that consumes the exam scores from two exams (a midterm and a final) and produces a `Boolean` value that indicates whether the student passes the course. Student who either failed both exams, or got F on one exam and D on the other, fails the course, otherwise student passes the course.
Hint: Use the function `letterGrade` as a helper.

Exercise 1C.5 Modify the function `letterGrade`, so that if the reported score is negative, or greater than 100, an exception with the message ”Score out of range” is raised.

Exercise 1C.6 Develop the function `computeToll` that computes the toll for a vehicle passing through the toll booth, given information about the vehicle. The vehicle driver hands in a ticket with one character code as follows: A is for a passenger car (automobile), B is for bus, T is for a truck, W is for a car towing a small trailer or a boat or an airplane (glider in a box). The ticket also has the code that determines the distance travelled. The toll structure is as follows. Passenger cars pay $0.25 per 10 miles of travel, rounded down, car with a trailer pays $0.35 per 10 miles. Bus pays $0.50 per 10 miles and an additional surcharge of $5 if the distance is greater than 100 miles. Trucks pay $0.40 per 10 miles, with a minimum of $5.
Remember the basic guidelines and develop two auxiliary functions to deal with bus and truck tolls.
Your purpose statement must specify clearly the range of acceptable input values for the vehicle code and the distance travelled.

Exercise Set 2: Simple Classes

Exercise 2.1 A class to represent information about one person.

- Define the member data for the `Class Person` that will include the date of birth (year only). Do not use more than seven attributes.
- Define the constructor and the `toString` method for the `Class Person`.
- Develop the purpose and the contract (header) for at least three methods for the `Class Person`.
- Develop the method isOlderThan that determines whether this person is older that a given age.
- Draw the UML diagram for this class.

Exercise 2.2 A class that represents weather data for one day.

- Run the code for the tests of the `Class WeatherData`.
- Add the code to test the method `tempPhrase`.
- Develop the method `howWet` for the `Class WeatherData`. It should return the String `"dry"` if the precipitation is below 0.01, `"wet"`, if the precipitation is below 0.25, `"soggy"` if the precipitation is below 1.0 and `"very wet"` otherwise.

Exercise 2.3 A class that represents automobiles.

- Design the `Class Car` that records the make and model of the car, the fuel tank capacity in gallons, and the estimated fuel consumption given in miles per gallon.
- Develop the constructor and the `toString` method for this class.
- Develop the method `maxDistance`, which computes the distance the car can travel on one tank of gas.
- Develop the method `canReach`, which determines whether a destination (distance given in miles) is reachable on one tank of gas.
- Develop the method `goFartherThan`, which determines whether this car can travel farther on one tank of gas than some other given car.
- Draw the UML diagram for this class.

Exercise 2.4 A class that represents one type of item in a grocery store (a can of coffee).

- Design the `Class Coffee` that represents a coffee can in the grocery store. The relevant information is the brand name, the weight of the can, given in ounces, and the price of the can, given in cents.

- Develop the constructor and the `toString` method for this class.
- Develop the method `unitPrice`, which computes the price per ounce of this grocery item.
- Develop the method `isCheaperThan`, which determines whether the unit price is lower that some given price.
- Develop the method `betterPriceThan` for this class, that determines whether this coffee is cheaper (in terms of the unit price) than some other given `Coffee`.
- Draw the UML diagram for this class.

Exercise 2.5 Run the existing code for the tests of the `Class DayData`. Complete the test suite for the `Class DayData`.

Exercise 2.6 A class that represents inventory information about an item in a grocery store (a can of coffee).

- Design the `Class InventoryItem` that keeps a record of the number of specific Coffee items in stock, as well as the number that has been sold.
- Develop the constructor and the `toString` method for this class.
- Develop the method `itemValue`, which computes the value of the current stock of Coffee items.
- Develop the method `grossIncome`, which determines the amount of money received for the Coffee items already sold.
- Draw the UML diagram for this class.

Exercise Set 3: Simple Class Hierarchy

Exercise 3.1 Given a class hierarchy to represent information about people at a university, including the classes `Person`, `Instructor`, and `Staff`:

- design `class Student` to be a new subclass of the `Class Person`. It should contain information about the student's GPA and major.
- Develop the method `withHonors`, which determines whether the student's GPA is greater than 3.5
- Complete the test suite for this collection of classes.
- Complete the UML diagram for this collection of classes.

Exercise 3.2 Given a UML class hierarchy to represent items in the grocery store, develop the classes `Grocery`, `Coffee`, `Juice`, and `IceCream` as specified in the UML diagram.

Exercise 3.3 Develop a class hierarchy to represent three different types of vehicles: cars, trucks, and buses. For all vehicles we need to record the size of the fuel tank in gallons, and the estimated fuel consumption given in miles per gallon. Just like the `class Car` in an earlier exercise, we can compute:

- the maximum distance the vehicle can travel on one tank of fuel
- whether it can travel a given distance
- whether it can travel farther than some other vehicle.

Exercise 3.4 *Warning:* This exercise can be completed independently of the previous exercise.
Develop a class hierarchy to represent three different types of vehicles: cars, trucks, and buses. Different kinds of vehicles need to represent additional information as follows:

- A car may or may not pull a trailer.
- A bus can carry some number of passengers (capacity) and has some number of passengers on board.
- A truck has some specified maximum load and some current load.

Develop the method `computeToll`, which will compute the toll for each vehicle according to the following rules:

- A car without a trailer pays $0.25 per mile.
- A car towing a trailer pays $0.35 per mile.
- A bus pays $0.35 per mile and additional $0.20 for each passenger.

- A truck pays $0.40 per mile and in addition $0.05 for each 1000 lb of load it currently carries.

Exercise 3.5

- Create a class to represent the address of some person. Address should include the street, the city, the state and the zip code.
- Extend the class hierarchy from Exercise 3.1, so that it records the address for each kind of person (`Person`, `Instructor`, `Staff`).
- Develop the method `thisZip`, which determines whether this person lives in a given zip code.
- Draw the UML diagram for this collection of classes.

Exercise 3.6

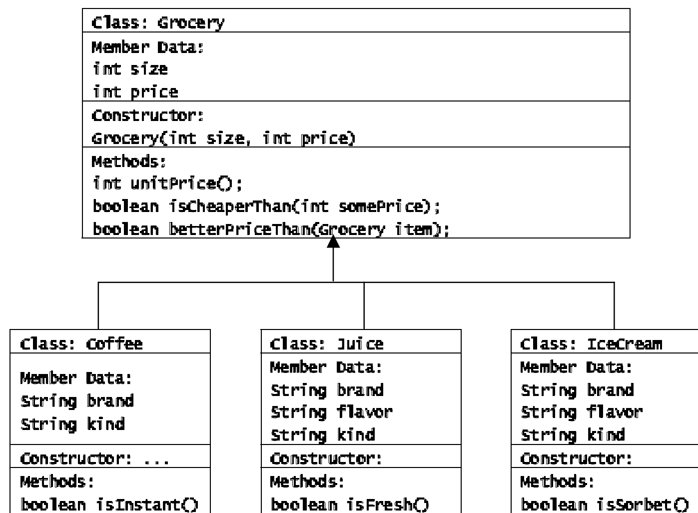
- Modify the `class InventoryItem` from Exercise 2.6 to represent the inventory information about any grocery item in the class hierarchy from Exercise 3.2.
- Draw the UML diagram for this collection of classes.
- Explain briefly how the method `grossIncome` performs the calculations.
- Explain briefly how the method `betterPriceThan` performs the calculation, when we choose to compare a `Coffee` item with an `IceCream` item.

Exercise Set 4: Self Referential Class Hierarchies

Exercise 4.1 Given the shopping list of groceries and the UML diagram (Figure 4.1), develop the following methods:

- `isInList`, which determines whether a given item is already on the list;
- `countItems`, which counts how many items are on the shopping list;
- `removeFromList`, which produces a new list with the given item removed;
- `addToList`, which adds a new item to the shopping list.

Figure 4.1



Exercise 4.2 Develop the collection of classes for representing a list of actors in a cast. We assume, each actor is listed only once. It is sufficient to just record actor's full name as one `String`: the last name followed by the first name.

- Develop the method `isInCast`, which determines whether some actor is in the cast.
- Develop the method `castSize`, which determines how many actors are in the cast.
- Develop the method `sortList`, which return a sorted list of actors.
- Develop the method `sortedIsInCast`, which determines whether some actor is in the cast, and takes advantage of the fact that the list is already sorted.

Exercise 4.3 The playbill is a list of actor-role pairs, each actor and each role can be represented as one `String`. Every actor plays only one role.

- Develop the collection of classes to represent a playbill for a play.
- Develop the method `whosePart`, which determines the actor playing the given role.
- Develop the method `whatPart`, which determines the role played by a given actor
- Develop the method `substitute`. The method consumes an actor and a role. It produces a new playbill where the given role is played by the given actor. If the given role does not appear in the playbill, the playbill is unchanged.

Exercise 4.4 The course information in the registrar's database contains the course number (four digits), the course title, and the number of credits.

- Develop the classes to represent a list of courses.
- Develop the method `departmentList`, which produces a list of all courses offered in the given department. A department is identified by the first two digits of the course number.
- Develop the method `credits`, which determines the number of credits student earns in a given course.

Exercise 4.5 A town soccer team has a phone tree to notify players about game cancellations and other changes in the schedule. Each player is assigned to call at most two other players.

- Develop the classes to represent the phone tree, recording each player's name and phone number.

- Develop the method `phoneNumber`, which returns the phone number of a given player.
- Develop the method `atFault`, which returns the list containing the names of all players in the tree that are in the calling chain leading to the given player.
- Develop the method `myPhoneTree`, which returns the phone subtree starting with the given player.
- Develop the method `countPlayers`, which counts how many players are in the phone tree.
- Develop the method `checkList`, which determines whether every player in a given list of players also appears in the given phone tree.

Exercise 4.6 Design the `class Item` that contains some *attribute* that defines an ordering of Item objects.

- Develop the classes to represent a binary search tree of `Item`s.
- Develop the method `insert`, which inserts a new `Item` into the BST.
- Develop the method `inBST`, which determines whether an `Item` with the given *attribute* appears in the BST.
- Develop the method `find`, which produces the `Item` with the given *attribute* in the BST, or `null` object if not found.

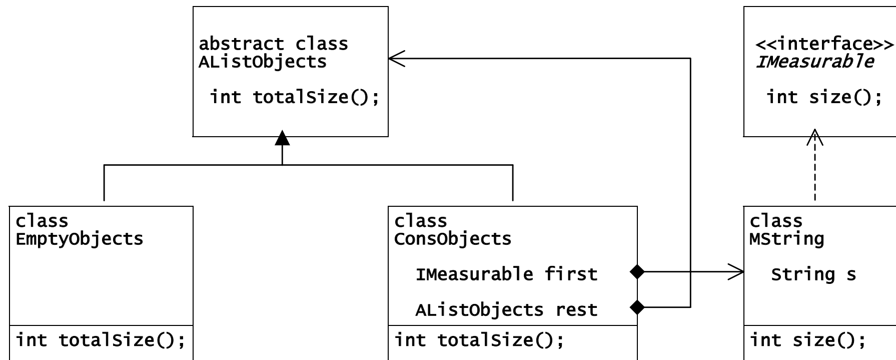
Exercise 4.7 The grocery store keeps the inventory as a list of lists of grocery items of different kind (`Coffee`, `IceCream`, `Juice` - see an earlier exercise).

- Develop the classes to represent the inventory for this grocery store.
- Develop the method `grossIncome`, which computes the gross income from all products in the inventory.
- Develop the method `incomeForClass`, which determines the gross income for a given class of products, given the `class` name for the products. Use `item.getClass().getName()` to retrieve a `String` that is the name of the `class`. Your method should consume a similar `String`.
- Develop the method `incomeForBrand`, which determines the gross income for all items in the given `class` and with the given brand name.
- Draw the UML diagram for this collection of classes.

Exercise Set 5: Interfaces

Exercise 5.1 The given UML diagram describes classes that define a list of measurable objects with a sample method `totalSize` and a sample `class MString` on which the list is tested.

- Develop the method `minItem`, which returns the minimum (`IMeasurable`) item in the list, as determined by the `size()` method. Test the `minItem` method using the `class MString`
- Develop the method `sortList`, which implements the insertion sort on the list.



Exercise 5.2 The Java `Comparable interface` is designed to support uniform comparison of objects.

- Develop a new collection of classes `CompList, EmptyCompList, ConsComp` that define a list of `Comparable` objects.
- Develop the method `minItem`, which returns the minimum (`Comparable`) item in the list, as determined by the `compareTo()` method. Test the `minItem` method using the `class String`
- Develop the method `sortList`, which implements the insertion sort on the list. Use the `class String` to test your code.
- Develop the `class CompString`, which is based on the Java `class String`, but re-implements the `Comparable interface` so that it compares `String`s by their length. Test your implementation of the `CompList` using the `Class CompString`.
- Draw the UML diagram of this collection of classes.

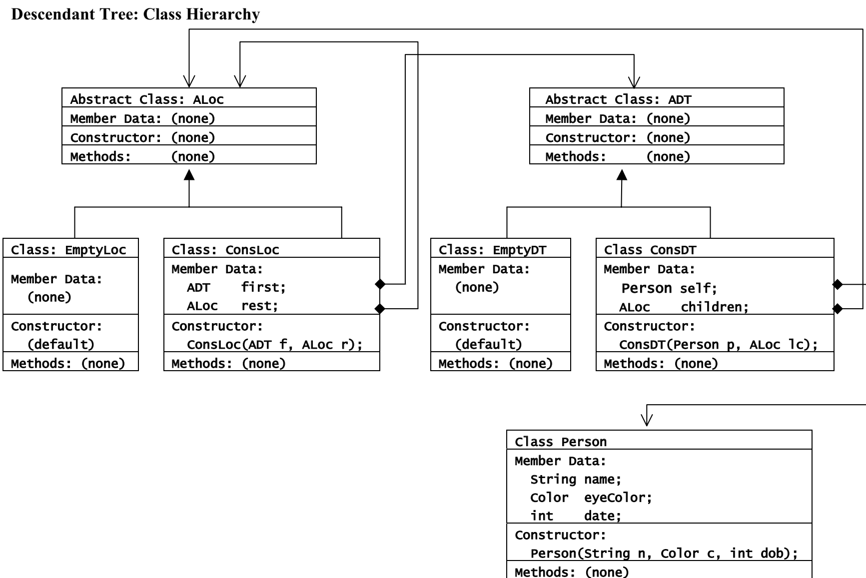
Exercise 5.3 Develop further tests and uses of the list of `Comparable` objects from the previous exercise. Develop the `class CompRectangle` which extends the Java `class Rectangle` by implementing the `Comparable interface`. The comparison is based on the area of the rectangle. Test the `class CompList` using the `class CompRectangle`. Draw the UML diagram of this collection of classes.

Exercise Set 6: Complex Class Hierarchies

Exercise 6.1 Given a UML diagram to represent the descendant tree, and the code that implements the method which counts the number of people in the descendant tree, develop the following methods:

- `blueEyes`, which determines whether there is a person with blue eyes in the descendant tree.
- `countBlueEyes`, which count the number of persons with blue eyes in the descendant tree.
- `find`, which produces the year of birth for a person with the given name, or `0`, if no person in the tree has this name.
- `children`, which produces a list of children for a person with the given name, or an empty list of children, if a person with this name is not in the tree.

Add the missing templates to all classes and continue to develop the templates as you work on this problem. Add one more generation (at least three grandchildren) to the test cases.



Exercise 6.2 A GUI `Component` is one of the following:

- BooleanView
- TextFieldView
- OptionsView
- ColorView
- Table

Each component contains a label and some additional data.
The data for a `Table` is one of

- empty
- list of `Rows`

A `Row` is one of

- empty
- list of `Components`

Data for each of the remaining components is the default value to be displayed, specified as a `String`, and the preferred width and height.

- Draw the UML diagram for this collection of classes.
- Develop the templates for methods needed to count the number of primitive GUI elements in a given GUI `Component`.
- Develop the templates for the methods needed to determines the height of a given GUI `Component`. The height of the table is the sum of the heights of the `Rows`. The height of a `Row` is the maximum size of components in the list of `Components`.

Note: Do not write the code!!!

Exercise 6.3 A `WebPage` consists of a

- `String header`,
- and a `Body b`.

A `Body` is a list of `HTML` elements.
A `HTML` element is either

- a `String word`
- or a `Link`.

A `Link` consists of

- a `String word`
- and a `WebPage`.

Perform the following tasks:

- Draw the UML diagram for the collection of classes that represent web pages.
- Develop the classes.
- Develop the method `allWords`, which produces a list of all words in the web page
- Develop the method `pages`, which produces the list of *immediate* words on a page. That is, it consumes a `WebPage` and produces a list of `String`. An `immediate` word on a list of `HTML` elements is defined as follows:
  - an `HTML` element that is a `word` is the *immediate* word
  - for an `HTML` element that is a `Link`, the method extracts the `word` from the `Link`.
- Develop the method `occurs`, which determines whether the given `word` occurs in the web page or its embedded pages.

Exercise Set 7: Anonymous Inner Classes (Lambda)

Exercise 7.1 The given code implements a list of `String`s with four methods: `count` which counts the number of items in the list, using the *traditional* technique; `mapStringToString` and `mapStringToBool`, each of which creates a new list by applying the specified method to every item in the original list; and `orMapString` which returns true if one of the list items satisfied the given predicates. The last three methods take as argument an object which implements one of the two *functional* interfaces. The examples in the test suite illustrate the implementation which uses the anonymous inner classes.

- In the test suite develop the test which consumes a list of `String` and produces a new list of `String`s in which each `String` has been converted to all upper case letters.
- In the test suite develop the test which consumes a list of `String`s and determines whether a given `String` is one of the items in this list.
- Develop a new method `andMap` in the class `AListStrings` and its variants. The method will take as argument an object which implements the `Obj2Boolean interface` and returns `true` if every item in the list satisfies the predicate defined in the class which implements this interface.
  Hint: Think of Scheme *andmap*.
- Develop two tests for this method - selecting the predicates on your own. Make sure you explain clearly the purpose of your test!

Exercise 7.2 The given example sorts the list of Persons by their full names, first names, and by the year of birth, using the anonymous inner classes to implement the desired Comparator interface.
Using the same technique, first develop a list of CDs, where each CD has a title, artist name and the number of tracks. Implement three different ways of sorting this list: by titles, by artists, and by the number of tracks.

Exercise Set 8: Loops with Iterators

Exercise 8.1 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 8.2 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 8.3 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 8.4 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 8.5 (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 9: Loops with Counters

Exercise 9.1 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 9.2 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 9.3 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 10: The Meaning of Equality

Exercise 10.1 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 10.2 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.

Exercise Set 11: Assignment for Primitive Types

Exercise 11.1 Create a table which displays the values of the variables x and y after each of the following statements, assuming the statements are executed in the order shown.

- (a)

```
x = 40;              // line 1
y = 55;              // line 2
x = y;               // line 3
/** ------------------------------------- */
```

- (b)

```
x = 25;              // line 1
y = 40;              // line 2
x = x + 20;          // line 3
y = x - 10;          // line 4
/** ------------------------------------- */
```

- (c)

```
x = 25;              // line 1
y = 40;              // line 2
y = x - 10;          // line 3
x = x + 20;          // line 4
/** ------------------------------------- */
```

- (d)

```
x = 15;              // line 1
y = 20;              // line 2
x = x + y;           // line 3
/** ------------------------------------- */
```

- (e)

```
x = 55;              // line 1
y = 99;              // line 2
x = x + y;           // line 3
y = x - y;           // line 4
x = x - y;           // line 5
/** ------------------------------------- */
```

Exercise 10.2 Create a table ...