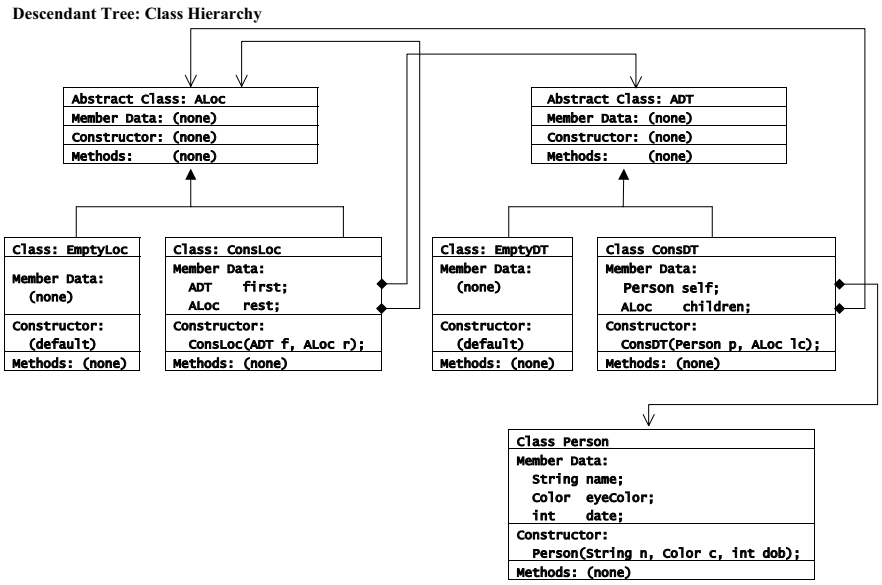
Exercise Set 7: Complex Class Hierarchies

Exercise 7.1 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 7.2 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 `Row`s

A `Row` is one of

- empty
- list of `Component`s

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 `Row`s. The height of a `Row` is the maximum size of components in the list of `Component`s.

Note: Do not write the code!!!

Exercise 7.3 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 8: Anonymous Inner Classes (Lambda)

Exercise 8.1 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 8.2 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 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.