22c:022 Object-Oriented Software Development
Fall 2013

Homework 3

Due: Tuesday, Oct 8 by 9pm

This is a programming assignment in Scala to be done *individually.* The homework is in two parts. Download from ICON the accompanying file `hw3.scala`, complete as instructed below, and submit them through ICON.

Your files *must generate no syntax or type errors.* You may receive no credit for each subproblem whose code contains such errors. *Pay close attention to the problem specification and the restrictions imposed on its solution.* Solutions ignoring the restrictions may receive no credit.

This assignment is meant to exercise your general programming skills, your understanding of basic OO programming concepts, your newly acquired knowledge of Scala, and your ability to read and understand specifications. It is in your best interest to you do all the assigned Scala readings so far and the posted sample solutions to previous assignments before working on the problems below.

Problem

In this assignment you will develop Scala classes implementing generic and immutable finite sets similar to the predefined immutable `Set` type in Scala. The main class, `FinSet[T]`, is an abstract class already specified for you in the file `hw3.scala`. It declares methods for converting the set to a list (`toList`) and to a string (`toString`), as well as methods for comparing two sets for inclusion (`<=`) and equality (`==`), producing the union (`U`), the intersection (`^`) and the difference (\) of two sets, adding an element to a set (`+`), and checking that a set contains a given element (`contains`). All these methods are specified in more detail in `hw3.scala`.

The methods `toString` and `==` are already implemented in `FinSet`. *You are to implement the method* `<=` *within* `FinSet`.

The remaining methods are meant to be implemented in concrete subclasses of `FinSet`. You will develop two of them: `ListSet` and `MapSet`.

Class `ListSet`

This class implements a finite set by storing its elements in an internal (i.e., private) list with no repetitions.

Its main constructor is private and takes as input the list of elements that constitute the set. Its only public constructor takes no (value) parameters and generates the empty set. Since `ListSet` is itself generic, the constructor does take a type parameter, for the type of the elements. For example, `new ListSet[Int]` generates the empty set of integers, `new ListSet[String]` generates the empty set of strings, and so on.

Similarly to Homework 2 for matrices, to construct non-empty sets one can use operators like `+` and `U`. For instance,

- `(new ListSet[Int]) + 3` generates the set containing just 3, which in math notation we would write as $\{3\}$;
- `(new ListSet[Int]) + 3 + 4` generates the set $\{3, 4\}$;
- `s1 U s2` generates the set $\{1, 2, 3, 4\}$ if `s1` is $\{1, 2\}$ and `s2` is $\{1, 3, 4\}$, say.

Implement in `ListSet` *all the abstract methods of* `FinSet`. You can use any methods of the predefined `List` type that you find convenient, but you cannot use values of type `Set` (which would defeat the purpose of this assignment).

Note: Methods like `U`, which take another set as input, must accept an input of general type `FinSet[T]`, not just `ListSet[T]`. This is necessary to satisfy the substitution principle, so that sets generated by `ListSet` methods can be combined indifferently via union, intersection and so on with sets generated by other concrete implementations of `FinSet` (such as `MapSet` below). The same applies to the same methods within `MapSet`.

Class `MapSet`

This concrete class is similar to `ListSet` but implements a finite set in a completely different way by storing its elements in an internal immutable map of type `Map[T, Unit]` where `T` is the type of the set elements. We have seen immutable maps in Chapter 3 of the Scala textbook. More information about their operations can be found in Chapter 17.2.

The idea in this case is that the internal map associates a key x of type `T` with the `Unit` value `()` if and only if x is in the set represented by the map. Note that the actually value associated with x is irrelevant.[1] The important point is whether x has an associated value in the map or not, as in the first case it means that x is in the set, and in the second case that it is not.

Implement in `MapSet` *all the abstract methods of* `FinSet`. You can use any methods of the predefined immutable `Map` type that you find convenient. But again you cannot use values of type `Set`.

As usual, in both subclasses you are allowed to used auxiliary methods as long as they are local or private.

[1] That is why we chose the type `Unit` for that, which has just one value: `()`.