Assignment 3 – Functional data structures

Advanced programming paradigms

In this assignment, you will add several methods to the `IntSet` that we discussed in class. You will be able to practice object-oriented programming mixed with functional-programming, a first example of multi-paradigm programming. Let us first consider the following code, which corresponds to the implementation of the `IntSet` that was given during the lecture:

```scala
abstract class IntSet() {
  def add(x: Int): IntSet
  def contains(x: Int): Boolean
}

class NonEmpty(elem: Int, left: IntSet, right: IntSet) extends IntSet {
  def add(x: Int): IntSet = {
    if (x < elem) new NonEmpty(elem, left add x, right)
    else if (x > elem) new NonEmpty(elem, left, right add x)
    else this
  }

  def contains(x: Int): Boolean = {
    if (x < elem) left contains x
    else if (x > elem) right contains x
    else true
  }
}

object Empty extends IntSet() {
  def contains(x: Int): Boolean = false
  def add(x: Int): IntSet = new NonEmpty(x, Empty, Empty)
}
```

Question 1 – *Displaying the content of the set*

To help visualizing and debugging of your code, we will start by devising a way to display a set.

(a) Begin by overriding the `toString` method for the two sub-classes so that the following code prints as follows:

```scala
println(Empty)                  // prints -
println(Empty.add(3))           // prints (-|3|-)
println(Empty.add(3).add(2))    // prints ((-|2|-)|3|-)
```

(b) In the second part of this exercise, we will not make a method only for printing a set but we will instead develop a more general method that we will then specialize later for displaying the content of a set.

The general method we will develop is called `foreach`. This is a standard method for collections in functional programming and its purpose is to apply a function to every element of the collection. The prototype of the method is as follows:

```scala
def foreach(f: Int => Unit): Unit
```

Please note that the `Unit` type declared here is a bit special. This type has a single value, called unit as well – denoted `()` –, which is used when the return value of a function is of no interest. Contrary to the Java `void` type, it has a value and **can** therefore be used in an expression. Test that your method is working as expected on a set `s` by calling

```
s.foreach(println)
```

(c) Apply the `foreach` method to display every element of the collection incremented by one. Each element should be separated by a comma (ignore the fact that the last element shows a comma as well). For instance:

```
(Empty.add(3).add(2).add(6).add(1)) foreach (...)
```

should print `"4, 3, 2, 7,"`. ⚠ You should be able to write this code as an one-liner!

(d) In the last question, why is the result `"4, 3, 2, 7, "` and not `"4, 3, 7, 2, "`, which corresponds to the insert order?

Because we're doing root-left-right in the for each, insertion order doesn't count, only the place in which they fit in the BST

Question 2 – *Union and intersection of sets*

Very common operations in set theory consist in computing the union and intersection of two sets. You will now implement those methods on `IntSet`.

(a) Add a new method called `union` for forming the union of two sets. Modify the abstract class accordingly. The prototype of the method is as follows:

```
def union(other: IntSet): IntSet
```

(b) Add a new method to compute the intersection of two integer sets. The intersection of two sets should comprise all the elements of the first set which also belong to the second set.

Question 3 – *Adding new operators*

We will now experiment with the implementation of new operators on the set we have defined.

(a) Start by adding to `IntSet` the following method:

```
def excl(x: Int): IntSet
```

which should return the given set without the element `x`. It might be useful (but it is not required) to implement the following helper method as well:

```
def isEmpty(): Boolean
```

(b) Make the necessary changes to your code so that it becomes possible to write the following expression (express you solution in terms of the existing methods):

```
val o1 = Empty + 3 + 4 + 12 + 5
val o2 = (o1 - 3 - 4)
```

The `toString` method of the resulting set should yield:
`((-|5|-)|12|-)`