Exercise 1: Closures

TECHNISCHE UNIVERSITÄT DARMSTADT

Software Engineering Design & Construction
SS 2015 - Dr. Michael Eichberg

Your task in this exercise is to implement two different versions of immutable integer sets using elements of object-oriented and functional programming. You must use Scala, except for tasks Task 2.2 and Task 2.3. Your solution should be as simple as possible. As a first step, make yourself familiar with the code. Add your implementations in the corresponding places and test them thoroughly.
Support can be found in the forum:
`https://www.fachschaft.informatik.tu-darmstadt.de/forum/viewforum.php?f=234`

Task 1 SBT & Java 8

Your exercise is provided as a SBT project (`http://www.scala-sbt.org`). SBT can be used to easily run and test your Scala projects. Install sbt on your platform and make sure you have the Java 8 SDK installed. You can run the project using `sbt run` and run all tests using `sbt test`. If you like to develop in Eclipse, just run `sbt eclipse` and use "Import existing project" in Eclipse on your project directory.

Task 2 Object-Oriented Sets

There are many ways to represent sets – one way is to implement them as ordered binary trees: an element in the set is represented by a node in the tree, with its key being the value of the element. The tree is ordered so that the key of the left child is always strictly less than the key of the parent, that in turn is always strictly less than the key of the right child.
We can model the base class of all nodes as follows:

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

It has two methods, method `incl` adds an element e to a set S such that the resulting set is $S \cup e$ (it leaves S unchanged!) and method `contains`, which checks whether a given element is in the set. There are two kinds of nodes:

- Inner nodes that store an integer key, a left child and a right child (`NonEmpty`).
- Leaf nodes that represent empty sets (`Empty`).

Task 2.1 Set operations

Add the following methods to the abstract base class and implement them in the concrete subclasses (unless otherwise noted):

a) A `foreach` method that applies a function for every element in the set:

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

Override method `toString` in `IntSet` and implement it using `foreach`. The method should return a string that contains the elements of the set, separated by a single space, e.g., "3 5 11 " for a set containing elements 3, 5 and 11.

b) A filter method that, for a given set S and predicate p, returns a subset of S with precisely those elements for which p holds, i.e., the resulting set is $\{x \in S \mid p(x)\}$:

```
def filter(p: Int => Boolean): IntSet
```

Note that it might be necessary to add a helper method to class `IntSet`.

Using `filter`, implement a method in `IntSet` that intersects two sets S and T such that the result is $S \cap T$:

```
def intersect(that: IntSet): IntSet
```

Task 2.2 Set operations in Java 7

Implement the set class hierarchy and all its operations (`incl`, `contains`, `foreach`, `toString`, `filter`, `intersect`) from above in Java 7. Stay as close to the Scala version as possible. You will have to replace Scala closures with (anonymous) class instances. For this purpose, you also need to define one or more SAM (Single abstract method) classes for function type(s) such as `Int => Boolean`.

Task 2.3 Set operations in Java 8

Java 8 supports a limited form of closures. Their design is discussed in detail on `https://docs.oracle.com/javase/tutorial/java/java00/lambdaexpressions.html`. Read this document carefully and try to identify how Java 8 closures are different from Scala closures (just for yourself, no need to write anything).
Implement the set class hierarchy and all its operations (`incl`, `contains`, `foreach`, `toString`, `filter`, `intersect`) from above in Java 8. Stay as close to the Scala version as possible. Replace Scala closures with Java 8 closures.
Implement two variants of the `toString` method (call the second one `toString2`). The `toString` variant should use a `String` and `toString2` should use a `StringBuilder` to accumulate its result. One variant will require more manual work and maybe a helper class.

Task 2.4 Pattern Matching

For practice, implement `incl` and `contains` as functions (instead of methods in `IntSet`) using pattern matching instead of dynamic dispatch. You are not allowed to call `IntSet.incl` or `IntSet.contains`. They have the following signatures:

```
def incl(s: IntSet, x: Int): IntSet
def contains(s: IntSet, x: Int): Boolean
```

Task 3 Functional Sets

Another way to represent sets is via a single function `Set => Boolean` that defines whether a given element is in the set. In Scala, we can define a type alias, so that we can use the alias `Set` instead of the longer function type:

```
type Set = Int => Boolean
```

We can write a `contains` function as follows:

```
def contains(s: Set, elem: Int): Boolean = s(elem)
```

Since a set of type `Set` is just a function, `contains` simply applies that function to determine whether the element is in the set or not.

Task 3.1 Implementation (3 points)

Implement a constructor, with the following signature, that creates a set containing a single element:

```
def Set(elem: Int): Set
```

Implement set union, intersection and difference with functions of the following signatures:

```
def union(s: Set, t: Set): Set
def intersect(s: Set, t: Set): Set
def diff(s: Set, t: Set): Set
```

Implement a filter function for our functional set representation similar to the filter method above:

```
def filter(s: Set, p: Int => Boolean): Set
```

Implement a function that maps every element in a given set S using a function f, so that the result is the set $\{f(x) \mid x \in S\}$:

```
def map(s: Set, f: Int => Int): Set
```

You can assume that for all elements $x \in S$: $-1000 \leq x \leq 1000$.

Solve it on your own!

Although this exercise is not graded, it is highly recommended to do it by yourself. Just looking at a solution is much easier in comparison to actually coming up with it.

Requirements if you want to submit your solution

You can, once in the semester, submit your solution to get it corrected. Send your solution to `weiel@st.informatik.tu-darmstadt.de`. Make sure you zip your complete sbt project and make sure that is out of the box working by running `sbt run` and `sbt test`. If the project doesn't compile properly you will not receive any feedback!