Exercise 1: Scala Basics

TECHNISCHE UNIVERSITÄT DARMSTADT

Software Engineering Design & Construction
WS 2016/17 - Dr. Michael Eichberg

Although this exercise is not graded, it is highly recommended to do it on your own. Just looking at a solution is much easier in comparison to actually coming up with it. Support can be found in the forum: `https://www.fachschaft.informatik.tu-darmstadt.de/forum/viewforum.php?f=234`

SBT

Your exercise is provided as an 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 1 Introduction to Scala

In the lecture as well as in the exercises we will use the Scala[1] programming language to solve various software design problems. As a prerequisite for this lecture, we expect you to already be familiar with Java 8. Please read the following paper to learn about the basics of Scala. `http://www.scala-lang.org/docu/files/ScalaTutorial.pdf`

Task 2 Scala Traits vs Java 8 Interfaces

The following class shows how logging routines are usually set up in Java. In case that the debug statement takes time to evaluate, it is preferable to first check if the desired log level is available in order to avoid unnecessary computations.

```
class MyClass {
  final static Logger logger = LoggerFactory.getLogger(this.getClass());

  ...

  log.debug("Some␣debug␣message␣" + complexComputationRequired());

  ...

  if (log.isDebugEnabled()) {
    log.debug("Some␣debug␣message␣" + complexComputationRequired());
  }
}
```

However, it is quite cumbersome to always check for the log level being enabled. It would be good if we could write this routine once and reuse it in all classes that require logging functionality.

Task 2.1 In Scala

Encapsulate the logging in a Scala trait that can be reused by mixing it into classes requiring logging functionality. Provide methods for the log levels `debug`, `info`, `warn`, and `error` that contain appropriate checks if the respective log levels are enabled. The logger should be cached inside of the trait and it should not be looked up every time it is used.

Task 2.2 In Java

Prior to Java 8, all methods defined in an interface needed to be abstract. Starting with Java 8, Java interfaces can also contain implementations, by defining `default methods` or static methods. Now, implement the same in a Java 8 interface using `default methods`. What are the commonalities/differences between Scala traits and interfaces with `default methods` in Java 8?

[1] `http://scala-lang.org`

Task 3 Object-Oriented Sets

There are many ways to represent sets – one way is to implement them as binary search 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 3.1 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): Set
```

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 3.2 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/javaOO/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 3.3 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
```