Intermediate Scala

Practical Exercises

Lab 1b About Scala Types

1. A MultiSet, sometimes called a Bag, is a collection that allows duplicates, while not imposing any ordering on the elements.

   Define a class to represent a MultiSet in Scala. Use whatever existing collection type(s) you need to underpin the collection types. For each element in the collection, you should maintain an “occurrence” count, reflecting on how many times the element has been added.

   Make the MultiSet invariant initially. What changes would be required to your implementation to make the MultiSet covariant in its element types?

2. Investigate the effects on your implementation of the MultiSet if you use an abstract type member rather than a type parameter.

3. Return to your types from the previous exercise. See if you can organise the types into a type hierarchy, using type parameters where necessary, and see if this allows you to remove some of the boilerplate code you had in the first exercise whilst retaining type safety.

   For example it should be possible to write the conversion functions that allow each unit of temperature to be converted into each of the others, but not to a unit of any other type like the distance units (and vice versa).

   Also, it should be possible to add or subtract values of any of the unit types, again in a type safe way. For example, it should be possible to do something like the following:

   ```
   val result = Metres(3.0) + Centimetre(40.0)
   ```

   which should result in 3.4 metres.

   Warning: this may be quite tricky!!