Part 1 - Maps (8pts)

The package `scala.collection` in the Scala standard library defines a number of data structures that represent different collections of objects. So far we have seen List, Array, and Map. There are usually two different versions for each collection, an immutable version (once an object is created it cannot be changed) and a mutable version.

In this problem we will work with mutable Maps. We will discuss a solution with immutable Maps in class. To make sure you use the right version of Map import

```scala
import scala.collection.mutable.Map
```

Assume we have the following Map:

```scala
val fruit_to_color: Map[String, String] = Map("banana" -> "yellow",
   "blueberry" -> "blue",
   "cherry" -> "red",
   "lemon" -> "yellow",
   "kiwi" -> "green")
```

In the file `Part1.scala`, write a function `reverse[A,B](map: Map[A,B]): Map[B,List[A]]` that returns a Map that maps each value in the original map to a list of keys.

The function should behave as follow:

```scala
scala> reverse(fruit_to_color)
res0: scala.collection.mutable.Map[String,List[String]] =
   Map(yellow -> List(lemon, banana), green -> List(kiwi),
   red -> List(cherry), blue -> List(blueberry))
```

Hints:

- Mutable maps have a method `put(key, value)` to add a (key,value) pair.
- Map is a trait. To create a new mutable Map you need to instantiate a concrete implementation such as HashMap.
Part 2 - Aquarium Simulator Revisited (12pt)

In this problem we will modify the Aquarium Simulator from Problem Set 2. You can base your solution on your own code, or you can use the sample solution for Problem set 2 [2].

In Problem 2, you added functionality (moving and eating) to the abstract class BaseFish by repeatedly extending the class (first to Fish then to BaseFish). Other LifeForms in the aquarium should be able to eat and move as well, so it makes sense to provide this functionality as traits.

(a) - 4 pts Refactor your classes for Fish and HungryFish in the following way. Move the eat and move methods into their own traits Moving and Eating. Then mix in Moving into the definition of Fish and Eating into the definition of HungryFish. HungryFish should still extend Fish. Note that the Eating trait needs to extend LifeForm or AquariumElement.

(b) - 4 pts In Problem Set 2, the only purpose of the class BaseFish was to provide a default implementation for the eat method required by the abstract class AquariumElement. Remove the declaration for eat and move from AquariumElement and get rid of the BaseFish class. This won't compile because the Aquarium class' attemptMove and handleCollision methods call eat and move for any AquariumElement.

Modify the attemptMove to only call move on objects of type Moving. Use pattern matching for the type checks (patterns can contain type restrictions). Modify handleCollision to only call eat on objects of type Eating.

(c) - 2 pts Create a class Crab that supports both Eating and Moving and is represented by the symbol C.

(d) - 2 pts Draw a class diagram indicating the inheritance relations between different AquariumElements. Indicate mixins using dashed lines and class inheritance using solid lines.