Programming Paradigms Tutorials
Nominal and structural type system

Duck Typing

Dynamic typing in general cuts down development time. There's less boilerplate code and it is easier to write a code. With static typing, compile-time checks slow down the development process. Static typing could be used at a later point to get better performance.

While it's true that compile-time checks are useful to catch potential problems early, duck typing enforces following coding conventions. Here is example of Scala implementation for duck typing:

```
def quacker(duck: {def quack(value: String): String}) {
  println (duck.quack("Quack"))
}
```

Sorting

The sorted function is used to sort the sequence in Scala like (List, Array, Vector, Seq). The sorted function returns new Collection which is sorted by their natural order.
Signature:

```
def sorted[B >: A](implicit ord: Ordering[B]): Repr
```

Example:

```
> val seq = Seq(2,3,5,4,1)
seq: Seq[Int] = List(2, 3, 5, 4, 1)
> seq.sorted
res0: Seq[Int] = List(1, 2, 3, 4, 5)
```

To sort in descending order then, use this signature :

```
Seq.sorted(Ordering.DataType.reverse)
```

To sort on the basis of an attribute of a case class using sorted method, it needs to extend Ordered trait and override abstract method compare. In compare method attributes used by sorting need to be selected.

```
case class Person(name: String, age: Int) extends
Ordered[Person] {
  def compare(that: Person) = this.name compare that.name
}
```

There are other methods which can be used for sorting:

- sortBy(attribute)
  The sortBy function is used to sort one or more attributes. The signature:
  `def sortBy[B](f: A => B)(implicit ord: Ordering[B]): Repr`
- sortWith(function)
  The sortWith function sorts the sequence according to a comparison function provided as argument. The signature:
  `def sortWith(lt: (A, A) => Boolean): Repr`

The below code presents their usage for previously defined Person class

```
val p1 = Person("Michael", 34)
p1: Person = Person(Michael,34)
val p2 = Person("Stan", 5)
p2: Person = Person(Stan,5)
val p3 = Person("Sophia", 8)
p3: Person = Person(Sophia,8)
val p4 = Person("Agnes", 33)
p4: Person = Person(Agnes,33)

var pList = List(p1,p2,p3,p4)
pList: List[Person] = List(Person(Michael,34), Person(Stan,5),
Person(Sophia,8), Person(Agnes,33))
pList.sorted
res0: List[Person] = List(Person(Agnes,33), Person(Michael,34),
Person(Sophia,8), Person(Stan,5))
pList.sortBy(_.age)
res1: List[Person] = List(Person(Stan,5), Person(Sophia,8),
Person(Agnes,33), Person(Michael,34))
pList.sortWith(_.age > _.age)
res2: List[Person] = List(Person(Michael,34), Person(Agnes,33),
Person(Sophia,8), Person(Stan,5))
```

Exercise:

1. Based on duck typing sample write a method that can be used by any animal which implement function `makeNoise`.
2. Scala classes are nominally typed, that why below code will generate error in the last line.

   ```
   class Foo {
    def method(input: String) = input
   }
   class Bar {
    def method(input: String) = input
   }

   var foo: Foo = new Bar(); // <- error
   ```

   rewrite the code in way which allow to use classes structurally
3. Create a class `Person` with fields `firstName` and `lastName` using Ordered trait add needed methods which allow to sort list of Persons by lastName (if same then firstName define the order).
4. Define the method:

   ```
   wordCounter(text: String) scala.collection.mutable.Map
   [String, Int]
   ```

   that counts occurring words in a given text. Please assume that the words are separated by spaces. The result needs to be a modifiable dictionary (Map), in which words are keys and number instances of words are values.