Scala Notes

Features

- Singleton object: a class with a single instance.
- There is no static memebers for Scala, instead, Scala declear those members in singleton objects.
- Everything in Scala is a object, no matter it is a number or function. Java distinguishes primitive types (such as int vs. Integer, boolean vs. Boolean), and does not enable one to manipulate a function as a value.
- Scala lexer uses a longest match rule(greedy mode) for tokens
- Case classes differ from standard classes in following respects:
  - keyword **new** is not mandatory to create instances of case classes.
  - getter functions are automatically defined for the constructor parameters (still confusing)
  - default definitions for methods euqals and hashcode are provided, which will work on the structure of the instances and not on their identity.
  -

Scala Version Hello World

```scala
//Hello world for Scala
object HelloWorld {
  def main(args : Array[String]) {
    println("Hello World!")
  }
}
```

Interact with Java

```scala
 1 //Scala can work with Java seamlessly, directly import the library from Java with more powerful import syntax
 2 import java.util.{Data, DataFormat}
 3 //Scala use the underscore(_) to represent every name in oen package, not asterisk(*) because the asterisk is a valid Scala identifier
 4 import java.util._
 5
 6 object FrenchDate {
 7   def main(args: Array[String]) {
 8     val now = new Date
 9     val df = getDateInstance(LONG, locale.FRANCE)
10     print(df format now)
11   }
12 }
```

Explanation for line 10 `print(df format now)`:

This is a interesting property of Scala's syntax, if methods take one argument can be used with an infix syntax.

Everything is an *Object*

Numbers are objects

An arithmetic expression like `1 + 2 * 3 / x` could be written as `(1).+(((2).*(3))./(X))`, the reason add brackets for 1 is Scala's lexer uses a longest match rule for tokens, which will break expression `1.+(2)` into `1.` and `(2)`. The reason that this tokenization is chosen is because `1.` is a longer valid match.

Functions are objects

Example Code:

```scala
object Timer {
  def oncePerSecond(callback: () => Unit) {
    while (true) { callback(); Thread sleep 1000 }
  }
  def timeFlies() {
    println("time flies like tomorrow...")
  }
  def main(args: Array[String]) {
    oncePerSecond(timeFlies)
  }
}
```

The type of the function `oncePerSecond` is written `() => Unit` inad is the type of all functions which take no arguments and return nothing.

Anonymous functions

The reason we use anonymous functions is to improve the readability. The above code can be rewritten as following:

```scala
object TimerAnonymous {
  def oncePerSecond(callback: () => Unit) {
    while (true) { callback(); Thread sleep 1000 }
  }
  def main(args: Array[String]) {
    oncePerSeond(() =>
      println("time flies like an arrow..."))
  }
}
```