Fall 2018 Midterm Solutions for Q8 & Q9 (asst-cache module)

Q.8 Cache.scala

```
case class MyCache[K, V](fulfill: K=>Future[V]) extends Cache[K, V] {

  private def put(k: K, v: V): Unit = cache.+=((k,v))

  override def apply(k: K): Future[V] = if (cache.contains(k)) Future(cache(k)) else for (v <- fulfill(k); _ = put(k, v)) yield v

  def expire(k: K): Unit = cache.-=(k)

  val cache: mutable.Map[K, V]  = mutable.Map.empty

  def empty: Unit = cache.empty
}
```

The apply method of an immutable List gives you the element at an index. Similarly, the apply method of MyCache should give the value (Future[V]) for the corresponding key k. MyCache internally uses a mutable Map to store key-value pairs. So, the apply method will return the value for key k from the mutable Map. If the key-value pair is not present in the Map, it will generate the value using the 'fulfill' constructor parameter.

Q.9 Portfolio.scala

Position case class is designed holds a stock symbol and its quantity. For eg, Position('MSFT', 100) would mean 100 stocks of MSFT (Microsoft).

Portfolio case class is designed to hold a sequence of Position, which is to say it holds several stock symbols and their corresponding quantities.

Also, here an instance of MyCache is used to store the stock symbol as key and its price as value.

```
case class Portfolio(positions: Seq[Position]) {

  def value(cache: Cache[String, Double]): Future[Double] = {
    val xfs = for (p <- positions) yield for (v <- p.value(cache)) yield v
    for (xs <- Future.sequence(xfs)) yield xs.sum
  }

}
```

Having said the above, value method of Portfolio (shown above) should give the sum of values of all the Position instances that it holds.

```
case class Position(symbol: String, quantity: Double) {
  def value(cache: Cache[String, Double]): Future[Double] = for (v <- cache(symbol)) yield v * quantity
}
```

The value method of Position case class (shown above) should give the value of the stock that it holds. The price of the stock multiplied by quantity would give the value. Note, here a Cache[String, Double] is used to get the price (value) for a stock symbol (key).

```
object Position {
  val positionR = """(\w+)\s+(\d+(\.\d+))""".r
  def parse(w: String): Try[Position] = w match {
    case positionR(a, b, _) => Try(Position(a,b.toDouble))
    case _ => Failure(new Exception(s"cannot parse $w as a Position"))
  }

  def value(cache: Cache[String, Double])(w: String): Future[Double] = flatten(for (p <- parse(w)) yield p.value(cache))

  private def flatten[X](xfy: Try[Future[X]]): Future[X] =
    xfy match {
      case Success(xf) => xf
      case Failure(e) => Future.failed(e)
    }

}
```

The value method of object Position (shown above) takes a parameter w of type String. If you look at the test case for this, you will notice w is something like “MSFT 100.0”. Now, we have been given a method parse that takes a String and returns a Try[Position]. Therefore, we do *parse(w)* to get this Try[Position], which will hold symbol = “MSFT” and quantity = 100.0

The value method should return the value of this Position instance. Therefore, we do *p.value(cache)* to get the Position’s value as Future[Double]. The *flatten* is used to bring this value into appropriate shape.