Handling Futures

Lesson Objectives

- After completing this lesson, you should be able to:
  - Describe how to use Futures to perform work asynchronously
  - Describe how to pattern match on the result of a Future
  - Outline how to use higher order functions on the result of a Future
  - Illustrate how to use for comprehensions to work with Futures

Futures

- Allow us to define work that may happen at some later time, possibly on another thread
- Futures return a Try of whether or not the work was successfully completed

ExecutionContext

- To use a Future, you must provide a thread pool that the Future can use to perform the work
- I can use an **`implicit val`** to declare it one time and automatically apply it to all usages within a scope

ExecutionContext

```
import scala.concurrent.ExecutionContext
import java.util.concurrent.ForkJoinPool

implicit val ec: ExecutionContext =
  ExecutionContext.fromExecutor(new ForkJoinPool())
```

Typesafe

Timeout

- Futures can have a defined amount of time before they “time out”, or fail because they have taken too long to do their work or be scheduled
- Scala has a nice DSL for creating such time-based values

Timeout

```
scala> import scala.concurrent.duration._
import scala.concurrent.duration._

scala> implicit val timeout = 1 second
timeout: scala.concurrent.duration.FiniteDuration = 1 second
```

Required Imports

```
import scala.concurrent.Future
import scala.concurrent.ExecutionContext
import java.util.concurrent.ForkJoinPool
import scala.util.Failure
import scala.util.Success
import scala.concurrent.duration._
```

Wrapping a Call in a Future

```
val f: Future[Int] = Future {
  inventoryService.getCurrentInventory(1234567L)
}
```

Typesafe

Pattern Matching on Future

```
scala> val f: Future[Int] = Future { 1 + 2 + 3 }
f: scala.concurrent.Future[Int] =
  scala.concurrent.impl.Promise$DefaultPromise@8b96fde

scala> f.onComplete {
     | case Success(i) => println("onComplete Success: " + i)
     | case Failure(f) => println("onComplete Failure: " + f)
     | }

onComplete Success: 6
```

Typesafe

Higher Order Functions and Futures

```
scala> val g: Future[Int] = Future { 1 + 2 + 3 }
g: scala.concurrent.Future[Int] = ...

scala> g.map(result => println(Mapping: " + result)
Mapping: 6
```

Typesafe

Higher Order Functions and Futures

```
val g: Future[Int] = Future { Thread.sleep(4000); 5 }
g: scala.concurrent.Future[Int] = ...

scala> g.map(result => println(Mapping: " + result)
```

For Expressions and Futures

```
val usdQuote = Future {
  connection.getCurrentValue(USD) }
val chfQuote = Future {
  connection.getCurrentValue(CHF) }
val purchase = for {
  usd <- usdQuote
  chf <- chfQuote if isProfitable(usd, chf)
  } yield connection.buy(amount, chf)
```

Typesafe

Lesson Summary

- Having completing this lesson, you should be able to:
  - Describe how to use Futures to perform work asynchronously
  - Describe how to pattern match on the result of a Future
  - Outline how to use higher order functions on the result of a Future
  - Illustrate how to use for comprehensions to work with Futures