Futures

Futures

Computations that will finish at some point with a value

```
// thread pool (Scala-specific)
given executionContext: ExecutionContext = ExecutionContext.fromExecutorService(executor)

val aFuture: Future[Int] = Future(calculateMeaningOfLife())
```

given executionContext passed here

Inspecting the value of a Future at this moment

- may be absent
- may be a failure

Callbacks: onComplete

- need to deal with potential failure
- evaluated on some other thread

```
aFuture.onComplete {
  case Success(value) => ...
  case Failure(ex) => ...
}
```

Functional Programming

onComplete is a hassle

- hard to read, understand, debug
- callback hell

Solution: functional programming

- map, flatMap, filter
- for comprehensions

```
val action = profileFuture.flatMap { profile =>
  SocialNetwork.fetchBestFriend(profile).map { bestFriend =>
    profile.sendMessage(bestFriend, message) // unit
  }
}
```

Falling back

- recover, recoverWith
- fallbackto

Blocking

Block the calling thread until the Future is completed

- returns the value inside
- throws if the Future is failed
- throws if the Future doesn't complete within the specified timeout

```
import scala.concurrent.duration.*

Await.result(transactionStatusFuture, 2.seconds)
```

Notes

- .seconds is an extension method*
- necessary import for the .seconds extension method

Blocking is not recommended unless you have no other option

Scala rocks