ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

# Async await

Principles of Reactive Programming

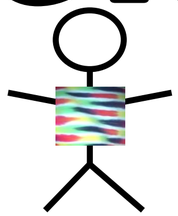
Erik Meijer

# Making effects implicit

```
T => Future[S]

T => Try[S] or even
T => S
```

We say one thing, but we really want…



# Async await magic

```
import scala.async.Async._

def async[T](body: =>T)
(implicit context: ExecutionContext)
:Future[T]
def await[T](future: Future[T]): T
```

**`async{ … await{…}…}`**

# Async, the small print

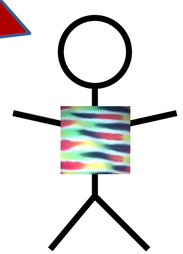
**Illegal Uses**

The following uses of await are illegal and are reported as errors:

- await requires a directly-enclosing async; this means await must not be used inside a closure nested within an async block, or inside a nested object, trait, or class.
- await must not be used inside an expression passed as an argument to a by-name parameter.
- await must not be used inside a Boolean short-circuit argument.
- return expressions are illegal inside an async block.
- **await should not be used under a try/catch**.

# Warning

**Getting async await to work,**
**Dealing with the compiler error messages,**
**Navigating the limitations,**
**…**
**can be frustrating.**
**But ultimately, it will pay off!**



# Retrying to send using await (an no recursion)

```
def retry(noTimes: Int)(block: ⇒Future[T]): Future[T] =
async {
  var i = 0
  var result: Try[T] = Failure(new Exception("…"))
  while (result.isFailure && i < noTimes) {
    result = await { Try(block) }
    i += 1
  }
  result.get
}
```

```
object Try {
  def apply(f: Future[T]):
Future[Try[T]] = {…}
}
```

# Reimplementing filter using await

```
def filter(p: T ⇒ Boolean): Future[T] = async {
  val x = await { this }
  if (!p(x)) {
    throw new NoSuchElementException()
  } else {
    x
  }
}
```

# Reimplementing flatMap using await

```
def flatMap[S](f: T ⇒ Future[S]): Future[S] = async {
  val x: T = await { this }
  await { f(x) }
}
```

# Reimplementing filter without await

```
def filter(pred: T ⇒ Boolean): Future[T] = {
  val p = Promise[T]()

  this onComplete {
    case Failure(e) ⇒
      p.failure(e)
    case Success(x) ⇒
      if (!pred(x)) p.failure(new NoSuchElementException)
      else p.success(x)
  }

  p.future
}
```

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

# End of Async await

Principles of Reactive Programming

Erik Meijer