Comp 311
Functional Programming

Nick Vrvilo, Two Sigma Investments
Robert “Corky” Cartwright, Rice University

October 25, 2018

Review: Call-By-Name

Call-By-Value

- We could delay evaluation in these cases by wrapping arguments in function literals that take no parameters

```
def myOr(left: Boolean, right: () => Boolean) =
  if (left) true
  else right()
```

Call-By-Name

- Scala provides a way that we can pass arguments as thunks without having to wrap them explicitly

```
def myOr(left: Boolean, right: => Boolean) =
  if (left) true
  else right
```

We simply leave off the parentheses in the parameter's type

Traits

Traits

Traits provide a way to factor out common behavior among multiple classes and “mix” it in where appropriate

Trait Definitions

Syntactically, a trait definition looks like an abstract class definition, but with the keyword “trait”:

```
trait Echo {
  def echo(message: String) =
    message
}
```

Trait Definitions

- Traits can declare fields and full method definitions
- They must not include constructors

```
trait Echo {
  val language = "Portuguese"
  def echo(message: String) =
    message
}
```

Using Traits

- Classes "mix in" traits using either the `extends` or `with` keywords

```
class Parrot extends Echo {
  def fly() = {
    // forget to fly and talk instead
    echo("poly wants a cracker")
  }
}
```

Using Traits

- Classes "mix in" traits using either the `extends` or `with` keywords

```
class Parrot extends Bird with Echo {
  def fly() = {
    // forget to fly and talk instead
    echo("poly wants a cracker")
  }
}
```

Using Traits

- Classes "mix in" traits using either the `extends` or `with` keywords

```
trait Smart {
  def somethingClever() =
    "better a witty fool than a foolish wit"
}
```

Using Traits

- Classes can mix in multiple traits via multiple `with`s:

```
class Parrot extends Bird with Echo
with Smart {
  def fly() = {
    // forget to fly and talk instead
    echo(somethingClever())
  }
}
```

Using Traits

Classes can mix in multiple traits via multiple `with`s:

```
trait X
case class Foo()

new Foo() with X
```

*Must use the **new** keyword when creating a new class instance with a **mixin** trait*

Traits with Self-Types

- We can restrict a trait so that it's only valid when mixed-in with a specific type
  - Useful for declaring extra dependencies

```
trait SmartTalk { this: Echo with Smart =>
  def talk() =
    echo(somethingClever)
}
```

Self-Types vs Inheritance

- What is the difference between *extends* and self-types?

  Whereas *extends* introduces a subtype relationship, self-types only specify a dependency.

- When would you *need* to use a self-type
  (i.e., an example where *extends* wouldn't work)?

  Self-typing allows introduction of a *cyclic* dependency between two types. Cyclic subtyping is not possible.