Type Classes and Extension Methods

Effective Programming in Scala

Motivation

In the previous lectures, we have seen that type classes could be used to retroactively add operations to existing data types.

However, when the operations are defined outside of the data types, they can't be called like methods on these data type instances.

```
case class Rational(num: Int, denom: Int)
given Ordering[Rational] = ...
val x: Rational = ...
val y: Rational = ...

x < y
//   ^
// value '<' is not a member of 'Rational'
```

Reminder: Extension Methods

Extension methods make it possible to add methods to a type, outside the type definition.

```
extension (lhs: Rational)
  def < (rhs: Rational): Boolean =
    lhs.num * rhs.denom < rhs.num * lhs.denom

val x: Rational = ...
val y: Rational = ...

x < y // It works!
```

Type Classes and Extension Methods

How can we add the < operation to any type `A` for which there is a given `Ordering[A]` instance?

Type Classes and Extension Methods

How can we add the < operation to any type `A` for which there is a given `Ordering[A]` instance?

We define the extension method in the trait `Ordering[A]`:

```
trait Ordering[A]:
  def compare(x: A, y: A): Int
  extension (lhs: A)
    def < (rhs: A): Boolean = compare(lhs, rhs) < 0
```

And then:

```
def sort[A: Ordering](xs: List[A]): List[A] =
  ...
  ... if x < y then ...
  ...
```

Applicability of Extension Methods

Extension methods on an expression of type `T` are applicable if

1. they are visible (by being defined, inherited, or imported) in a scope enclosing the point of the application, or
2. they are defined in an object associated with the type `T`, or
3. they are defined in a given instance associated with the type `T`.

Applicability of Extension Methods (2)

```
trait Ordering[A]:
  def compare(x: A, y: A): Int
  extension (lhs: A)
    def < (rhs: A): Boolean = compare(lhs, rhs) < 0

def sort[A](xs: List[A])(using ordering: Ordering[A]): List[A] =
  ...
  ... if x < y then ...
  ...
```

In our case, the compiler rewrites the call `x < y` to:

```
ordering.<(x)(y)
```

Summary

Leverage extension methods to provide a nice syntax to work with your type classes.

Extension methods are applicable on values of type `T` if there is a given instance

- that is visible at the point of application,
- or that is defined in a companion object associated with `T`.