Extension methods let you add methods to a type after the type is defined, i.e., they let you add new methods to closed classes. For example, imagine that someone else has created a Circle class:

```
case class Circle(x: Double, y: Double, radius: Double)
```

Now imagine that you need a circumference method, but you can't modify their source code. Before the concept of term inference was introduced into programming languages, the only thing you could do was write a method in a separate class or object like this:

```
object CircleHelpers:
  def circumference(c: Circle): Double = c.radius * math.Pi * 2
```

Then you'd use that method like this:

```
val aCircle = Circle(2, 3, 5)

// without extension methods
CircleHelpers.circumference(aCircle)
```

But with extension methods you can create a circumference method to work on Circle instances:

```
extension (c: Circle)
  def circumference: Double = c.radius * math.Pi * 2
```

In this code:

Circle is the type that the extension method circumference will be added to

The c: Circle syntax lets you reference the variable c in your extension method(s)

Then in your code you use circumference just as though it was originally defined in the Circle class:

aCircle.circumference

Import extension method

Imagine, that circumference is defined in package lib, you can import it by

import lib.circumference

aCircle.circumference

The compiler also supports you if the import is missing by showing a detailed compilation error message such as the following:

value circumference is not a member of Circle, but could be made available as an extension method.

The following import might fix the problem:

import lib.circumference

Discussion

The extension keyword declares that you're about to define one or more extension methods on the type that's put in parentheses. To define multiple extension methods on a type, use this syntax:

```
extension (c: Circle)
  def circumference: Double = c.radius * math.Pi * 2
  def diameter: Double = c.radius * 2
  def area: Double = math.Pi * c.radius * c.radius
```