EPFL

Case Classes

Effective Programming in Scala

Aggregating Values with Case Classes

The concept of a *rectangle*, which is defined by a *width* and a *height*, can be modeled as follows in Scala:

```scala
case class Rectangle(width: Int, height: Int)
```

Aggregating Values with Case Classes

The concept of a *rectangle*, which is defined by a *width* and a *height*, can be modeled as follows in Scala:

```scala
case class Rectangle(width: Int, height: Int)
```

This definition introduces a **type** and a **constructor**, both named `Rectangle`.

The type `Rectangle` has two members: `width` and `height`. Both members have type `Int`.

The constructor `Rectangle` takes two parameters: `width` and `height`.

Constructing and Manipulating Rectangles

```
case class Rectangle(width: Int, height: Int)
```

A value of type `Rectangle` can be constructed as follows:

```
val facade = Rectangle(5, 3)
```

The area of the value `facade` can be computed as follows:

```
facade.width * facade.height
```

Adding Operations to a Type

Because computing the area of a rectangle is a common operation in our program, we can define it as an operation of the case class `Rectangle`:

```
case class Rectangle(width: Int, height: Int):
  val area = width * height
```

Note that the area is not part of the information that defines a rectangle. Instead, it is *computed* from the width and the height of the rectangle. That's why it is defined as an operation rather than a parameter of the case class.

The operations of a class must be defined with a higher level of indentation than the class itself.

Complete Example

```scala
case class Rectangle(width: Int, height: Int):
  val area = width * height

val facade = Rectangle(5, 3)
facade.area
```

Class Body (Scala 2 Compatibility)

In Scala 2, the body of a class was delimited by braces:

```
case class Rectangle(width: Int, height: Int) {
  val area = width * height
}
```

Exercise

Model the concept of a square. *A square has a side* width.
Add an operation `area` *to the type* `Square`.

Exercise

Model the concept of a square. *A square has a side* width.
Add an operation `area` *to the type* `Square`.

```
case class Square(width: Int):
  val area = width * width
```

Exercise

Model a circle, *which has a* radius *and an operation* area.

Exercise

Model a circle, *which has a* radius *and an operation* area.

```
case class Circle(radius: Int):
  val area = radius * radius * 3.14
```

Case Classes Are Immutable

Once you construct a `Rectangle` you can't modify it:

```
val rectangle = Rectangle(width = 3, height = 4)
rectangle.width = rectangle.width * 2 // Error: Reassignment to val width
```

In Scala, it is idiomatic to work with **immutable data types**.

Case classes come with a handy method to create a copy of an existing value with some fields updated:

```
val smallRectangle = Rectangle(width = 3, height = 4)
val largeRectangle = smallRectangle.copy(width = smallRectangle.width * 2)
```

We will discuss later how to create mutable data types, and what their tradeoffs are.

Summary

A *case class* aggregates several concepts together in a new type.

Case classes define immutable data types.