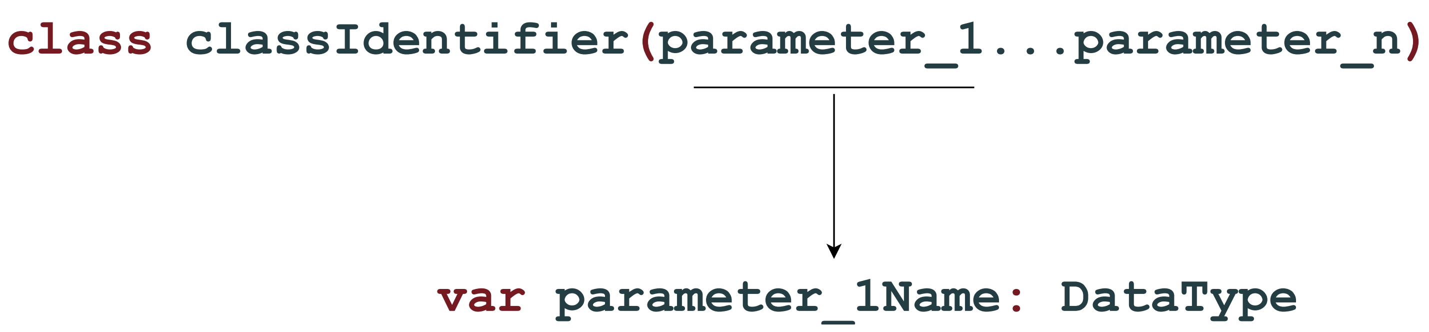
Creating an Object Using Constructor Parameters

In this lesson, we will look at another way of creating an object class and learn about constructors.

We can pass arguments to a class the same way we can pass arguments to functions. They are known as **constructor parameters** as they are assigned a value when the object is *constructed* using a class.

Let’s look at the syntax below:



```
class classIdentifier(parameter_1...parameter_n)

var parameter_1Name: DataType
```

As you can see, where we initially only had `class classIdentifier`, we now also have a parameter list. The rest of the code for creating a class is exactly the same.

Let’s now redefine our `Person` class using constructor parameters.

```
class Person(var name: String, var gender: String, var age: Int) {

  private var years = 15

  def walking = println(s"$name is walking")
  def talking = println(s"$name is talking")
  def yearsFromNow = {
    var newAge = years + age
    println(s"In $years years from $name will be $newAge")
  }
}
```

`name`, `gender`, and `age` are now constructor parameters. We also have a new field, `years`, which is assigned a value **15**. `years` is *private*, so it can only be accessed by the members of the class. `yearsFromNow` is a new method which calculates the age of a `Person` object `years` from their current age. As `yearsFromNow` is a member of the `Person` class, it can access `years`.

> Constructor parameters are also fields of that class; hence, they can be used

just as fields would be used.

We will now create an instance of the `Person` class using the constructor parameters.

```
// Creating an object of the Person class
val firstPerson = new Person("Sarah", "Female", 25)
```

The expression on **Line 2** is known as a **constructor** as it is *constructing* an instance of a class.

Just as before, we can access `name`, `gender`, and `age`.

This code requires the following environment variables to execute:

LANG C.UTF-8

```
class Person(var name: String, var gender: String, var age: Int) {

  private var years = 15

  def walking = println(s"$name is walking")
  def talking = println(s"$name is talking")
  def yearsFromNow = {
    var newAge = years + age
    println(s"In $years years from $name will be $newAge")
  }
}

// Creating an object of the Person class
val firstPerson = new Person("Sarah", "Female", 25)

// Accessing name, gender, and age
println(firstPerson.name)
println(firstPerson.gender)
println(firstPerson.age)
```

Let's see what happens when we call the `yearsFromNow` method.

This code requires the following environment variables to execute:

LANG C.UTF-8

```
class Person(var name: String, var gender: String, var age: Int) {

  private var years = 15

  def walking = println(s"$name is walking")
  def talking = println(s"$name is talking")
  def yearsFromNow = {
    var newAge = years + age
    println(s"In $years years from $name will be $newAge")
  }
}

// Creating an object of the Person class
val firstPerson = new Person("Sarah", "Female", 25)

// Calling yearsFromNow method on the object firstPerson
firstPerson.yearsFromNow
```

Our methods are simply printing an output, not returning anything. If your methods have a return value, you can store that value in a variable and use it whenever required.

In the next lesson, we will look at a unique Scala feature: singleton objects.