Objects of a Class

In this lesson, you will create your first object of a class and learn how to set its fields and call its methods.

We'll cover the following

Instantiating a Class

Once a class has been defined, you can create objects from the class blueprint using the `new` keyword followed by the class identifier.

```
new classIdentifier
```

We usually don't create objects for the sake of just creating them, rather, we want to work with them in some way. For this reason, we assign the object to a variable. Let's instantiate our `Person` class.

For ease, the code for creating the class is also provided below.

```
class Person{
  var name: String = "temp"
  var gender: String = "temp"
  var age: Int = 0

  def walking = println(s"$name is walking")

  def talking = println(s"$name is talking")

}
// Creating an object of the Person class
val firstPerson = new Person
```

Setting the Field Values

Now that we have our object `firstPerson`, let's set its field values. To do so, we call the field we wish to set on the object whose field we wish to set.

`objectName.field = setNewValue`

Let's set the `name`, `gender`, and `age` of the object `firstPerson`.

```
firstPerson.name = "Sarah"
firstPerson.gender = "female"
firstPerson.age = 25
```

To print the field values, we pass `object.field` to a print method.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
class Person{
  var name: String = "temp"
  var gender: String = "temp"
  var age: Int = 0

  def walking = println(s"$name is walking")

  def talking = println(s"$name is talking")

}
// Creating an object of the Person class
val firstPerson = new Person

firstPerson.name = "Sarah"
firstPerson.gender = "female"
firstPerson.age = 25


println(firstPerson.name)
println(firstPerson.gender)
println(firstPerson.age)
```

Calling a Class Method

To access the methods of a class, we follow the same syntax we use to access fields. Let's call the `walking` and `talking` methods and see what happens.

This code requires the following environment variables to execute:

LANG C.UTF-8

```
class Person{
  var name: String = "temp"
  var gender: String = "temp"
  var age: Int = 0

  def walking = println(s"$name is walking")

  def talking = println(s"$name is talking")

}
// Creating an object of the Person class
val firstPerson = new Person

firstPerson.name = "Sarah"
firstPerson.gender = "female"
firstPerson.age = 25

firstPerson.walking
firstPerson.talking
```

Since our methods don't have any parameters, we don't need to pass any arguments when calling them.

`firstPerson.walking` is called on the object `firstPerson` and prints a variable `name`. Since the method is called on an object, it determines if the variable used is one of the properties of that object. As `name` is defined for the object `firstName`, the method will print its value.

In the code above, `firstPerson.walking` takes the value of `Sarah` and prints `Sarah is walking`. `firstPerson.talking` performs the same operation.

Multiple Objects of the Same Class

As classes provide reusable code, it makes sense that we can create multiple objects using the same class blueprint. Along with `firstPerson`, let's create more objects of the `Person` class and set their field `name`.

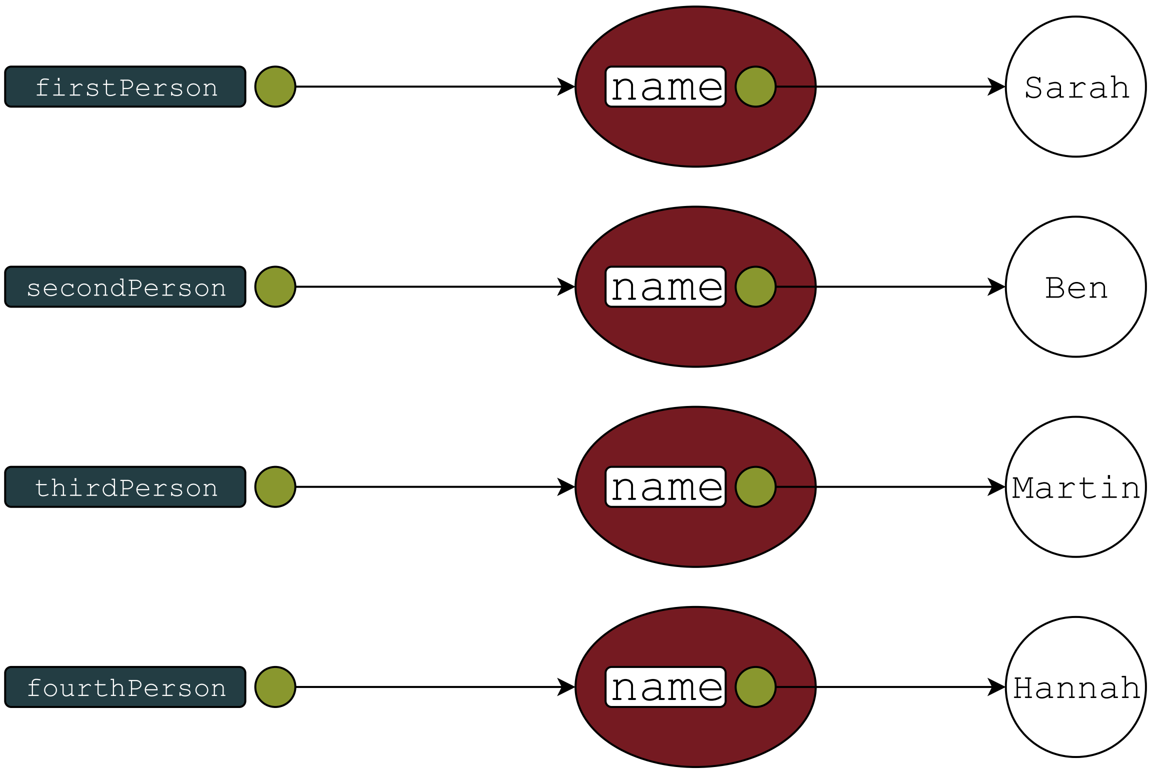
This code requires the following environment variables to execute:

LANG C.UTF-8

```
class Person{
  var name: String = "temp"
```

```scala
  var gender: String = "temp"
  var age: Int = 0


  def walking = println(s"$name is walking")

  def talking = println(s"$name is talking")

}
// Creating an object of the Person class
val firstPerson = new Person

firstPerson.name = "Sarah"
firstPerson.gender = "female"
firstPerson.age = 25

// Creating an object of the Person class
val secondPerson = new Person
secondPerson.name = "Ben"

// Creating an object of the Person class
val thirdPerson = new Person
thirdPerson.name = "Martin"

// Creating an object of the Person class
val fourthPerson = new Person
fourthPerson.name = "Hannah"

// Driver Code
println(firstPerson.name)
println(secondPerson.name)
println(thirdPerson.name)
println(fourthPerson.name)
```

Even though there are multiple `name` variables, they all are referenced by different objects, hence, modifying one doesn't modify the others. This is why fields are known as instance variables, because each object has its own set of those variables.



Private and Public Fields

By default, the fields defined in a class are **public**, i.e., they can be used by the object outside of the class itself. However, if you want to make sure that fields remain valid, it is better to prevent outsiders from accessing those fields. This is done by making the fields **private**. Private fields can only be accessed by methods defined in the same class. To create a private field, place the private access modifier in front of the field.

`private field`

Let's redefine our `Person` class so that its fields are private.

```
class Person{
  private var name: String = "temp"
  private var gender: String = "temp"
  private var age: Int = 0

  def walking = println(s"$name is walking")

  def talking = println(s"$name is talking")
}
```

Now, let's create an object and try to set the value for `name`.

```
class Person{
  private var name: String = "temp"
  private var gender: String = "temp"
  private var age: Int = 0

  def walking = println(s"$name is walking")

  def talking = println(s"$name is talking")
}

// Creating an object of the Person class
val firstPerson = new Person

firstPerson.name = "Sarah"
```

When you run the above code, you will get an error. This is because `firstPerson` is an object outside of the class. `fistPerson` is trying to access `name`, which is a *private* property. Hence accessing it outside the class will result in an error. Only members defined within the block of the class have access to private fields. However, in our case `name`, `gender`, and `age` are specifically assigned for each `Person` object. For such cases, Scala provides **constructor parameters**.

In the next lesson, we will look at constructor parameters and see how they are used to create an object of a class.