Quick Intro To Case Classes

The case class in scala is the most relevant feature in scala for handling data effectively.

```
final case class Employee(firstName: String, lastName: String, ssn: String)
```

The **final** qualifier makes sense because extending a case class may lead to inconsistencies and performance issues as well (

https://gist.github.com/chaotic3quilibrium/58e78a2e21ce43bfe0042bbfbb93e7dc)

A case class provides a swiss knife of features to make your life easier when handling data.

Covering The Key Features

Here is the list of the most popular features provided by the case class:

- A convenient toString() method that will display all it's field contents
- A compare by-field-values, not be reference
- A copy method for handling immutable data
- Pattern matching for field extraction (perhaps better avoided)

Convenience toString Method

With case classes, the toString method is invoked when you need to evaluate any object to a string eg

```
final case class Employee(firstName: String, lastName: String, ssn: String)
class EmployeeClass(firstName: String, lastName: String, ssn: String)

val employee = Employee("john", "wick", "111-11-1111")

val employeeClass = new EmployeeClass("john", "wick", "111-11-1111")

println(employee)
// Cool
// Employee(john,wick,111-11-1111)

println(employeeClass)
// Not cool. Prints a representation of the reference for this object
```

```
// Main$EmployeeClass$1@3f3afe78
```

Equality By Structure Not By Reference

With case classes, you can compare objects by their structure not by reference (default when using plain classes). Here is an illustration on how it works vs plan classes:

```
final case class Employee(firstName: String, lastName: String, ssn: String)
class EmployeeClass(firstName: String, lastName: String, ssn: String)



val employee1 = Employee("john", "wick", "111–11–1111")
val employee2 = Employee("john", "wick", "111–11–1111")
val employee3 = Employee("robert", "mccall", "222–222–2222")
println(employee1 == employee2)
// true
println(employee1 == employee3)
// false

val employeeClass1 = new EmployeeClass("john", "wick", "111–11–1111")
val employeeClass2 = new EmployeeClass("john", "wick", "111–11–1111")
println(employeeClass1 == employeeClass2
// false
```

Built-In copy Method

Case classes are immutable by default. This means that modifying fields is not possible. However, you can copy-modify case classes. eg:

```
val employeeWithModifiedLastName = employee1.copy(firstName = "Jon")
println(employeeWithModifiedLastName)
// Employee(Jon,wick,111–11–1111)
println(employee1)
// FYI, employee1 is not changed
// Employee(john,wick,111–11–1111)
```

Note: cases classes may not be mutable by default, but they can be mutable eg:

```
final case class MutableEmployee(var firstName: String, var lastName:
  String, var ssn: String)
```

```
val mutableEmployee = MutableEmployee("gravik", "skrull", "333-33-3333")
println(mutableEmployee)
// MutableEmployee(gravik,skrull,333-33-3333)

mutableEmployee.firstName = "talos"
println(mutableEmployee)
// MutableEmployee(talos,skrull,333-33-3333)
```

Using Pattern Matching For Field Extraction

You can extract the fields of a case class by using pattern matching eg:

```
employee1 match {
  case Employee(firstName, lastName, ssn) =>
    println(s"Name is $firstName $lastName and ssn is $ssn")
  // Name is john wick and ssn is 111-11-1111
}

// No need to define unused fields
employee1 match {
  case Employee(firstName, _, _) =>
    println(s"First name is $firstName")
  // First name is john
}
```

The reason I discourage pattern match extractions for scala case classes is that correct extractions depends on the correct order of the fields. Also, adding an extra field to the case class will cause a compile error. Here is an example where the wrong order creates a bug:

```
// wrong order!
employee1 match {
  case Employee(lastName, firstName, ssn) =>
println(s"First name is $firstName, last name is $lastName and ssn is $ssn")
  // First name is wick, last name is john and ssn is 111-11-1111
}

final case class WideCaseClass(name: String, s1: String, s2: String, s3:
String,
                               r1: String, r2: String, r3: String, t1: String, t2: String,
```

```
t3: String)


val wideClass: WideCaseClass = WideCaseClass("a", "b", "c", "d", "e", "f", "g", "h", "i", "j")
wideClass match
case WideCaseClass(_, _, _, _, _, r1, _, _, _, _) => println(r1)
// You've got it wrong by one position! Too bad!!!
// Type safety won't help you here.
```

In this page we did the basic thing: introduced case classes and it's basic features: The toString method, the equality by structure, the copy method and my least favorite field extraction by pattern matching.

Revision #7

Created Sun, Aug 6, 2023 12:21 AM by hernan saab

Updated Wed, Aug 9, 2023 7:20 PM by hernan saab