CSC 347 - Concepts of Programming Languages

Scala Classes

Instructor: Stefan Mitsch

Learning Objectives

❓ Object-oriented programming

- Understand basics of object-oriented programming in Scala

Scala Object-Oriented Programming

- Scala supports both functional programming and object-oriented programming

Functional

```
(0 to 9).toList.partition ((x:Int) => x%3 == 0)
```

Object-oriented

```
class Counter:
  var n = 0
  def get () : Int = { val tmp = n; n = n + 1; tmp }

val c = new Counter()
c.get()
c.get()
```

Classes

- Define a class `C`

```
class C (f1:Int, val f2:Int, var f3:Int):
  //...
```

- Has one constructor with parameters `f1`, `f2`, `f3`
- Instantiate the class `C`

```
val c = new C (2, 3, 5)
```

- Instance of `C` is heap allocated
- `c` is a reference to instance

Class Parameters

```
class C (f1:Int, val f2:Int, var f3:Int):
  //...
```

- `f1` private and immutable
- `f2` public immutable
- `f3` public mutable

```
scala> val c = new C (2, 3, 5)
c: C = C@8bd1b6a

scala> c.f1
<console>:10: error: value f1 is not a member of C

scala> c.f2
res1: Int = 3

scala> c.f3
res2: Int = 5

scala> c.f2 = 10
<console>:9: error: reassignment to val

scala> c.f3 = 10
c.f3: Int = 10
```

Class Parameter Details

- Class parameters simultaneously declare constructors and accessor methods
- Components of the class body are `public` by default

Scala

```
class C1(x:Int):
  def double() = x+x
```

Java

```
public class C1 {
  private final int x;

  public C1(int x) {
    this.x = x;
  }

  public int double() {
    return x+x;
  }
}
```

Class Parameter Details

- Immutable class parameters can be initialized but not modified

Scala

```scala
class C2(val x:Int):
  def double() = x+x
```

Java

```java
public class C2 {
  private final int x;

  public C2(int x) {
    this.x = x;
  }

  public int x() {
    return x;
  }

  public int double() {
    return x+x;
  }
}
```

Class Parameter Details

- Mutable class parameters can be changed even after initialization

Scala

```scala
class C3(var x:Int):
  def double() = x+x
```

Java

```java
public class C3 {
  private int x;

  public C3(int x) { this.x = x; }

  public int x() { return x; }

  public void setX(int x) { this.x = x; }

  public int double() { return x+x; }
}
```

Class Body

- Class body contains
  - `val` or `var` field declarations
  - Constructor code
  - method definitions

```
class C (f1:Int, val f2:Int, var f3:Int):
  val f4 = f1 * f2
  var f5 = f2 * f3

  println ("Constructing instance of C")

  def m (x:Int) : Int =
    // cannot reassign to f1, f2, f4
    f3 = f3 + 1
    f5 = f5 + 1
    f1 * f3 * x
```

Class Body

- Can omit empty body

```
class D (f1:Int)
```

- Can omit empty parentheses

```
class E:
  private var n:Int = 0
  def get () : Int =
    val tmp = n
    n = n + 1
    tmp

val o : E = new E()
o.get()
```

Objects

- `object` declares a single instance, accessible through the object name
- Language support for the singleton design pattern

```
object C:
  var count:Int = 0

C.count = C.count + 1
```

Objects

- Singleton objects `object` are instantiated on program startup
- Method `main` must be declared in an `object`

Java

```java
public class C {
  public static void main (String[] args) {
    //...
  }
}
```

Scala

```scala
object C:
  def main (args:Array[String]) : Unit =
    //...
```

Companion Objects

- Many languages distinguish data and methods belonging
  - to an instance
  - to a class

Java

- `static` components belong to a class
- All other belong to an instance of the class

```
class C {
  int f1;
  int m1 () { return f1; }
  static int f2;
  static int m2 () { return f2; }
}
```

Scala: Companion object replaces `static`

- All data and methods belong to objects
- Companion object is related to other instances of the same class

```
class C:
  var f1:Int = 0
  def m1 () : Int = f1

object C:
  var f2:Int = 0
  def m2 () : Int = f2
```

Companion Objects

- Companion objects can be used to construct instances of companion class
- Implementation of the factory method design pattern
- Companion can invoke `private` constructor

```
class Point private (private var x:Int, private var y:Int):
  def translate (xDisp:Int, yDisp:Int) : Unit =
    x = x + xDisp
    y = y + yDisp

object Point:
  def apply (x:Int, y:Int) : Point =
    if 0 <= x && x <= 10 && 0 <= y && y <= 10 then
      new Point (x, y)
    else
      throw IllegalArgumentException (s"Both x and y must be in range [0,10], but x=$x and y=$y.")
```

```
// java.lang.IllegalArgumentException:
// both x and y must be in range [0,10], but x=1 and y=100.
val p = Point.apply (1, 100)
// succeeds
val q = Point.apply (1, 10)
// name of "apply" method can be omitted
val r = Point (1, 10)
// compile error: constructor Point in class Point
// cannot be accessed
val s = new Point (1, 100)
```

Scala Library Companion Objects

- Class `List` has a companion object

```
object List extends SeqFactory[List] with Serializable:
  def apply[A](xs: A*): List[A] = ...
  //...other methods...

List (1, 2, 3)          // means List.apply (1, 2, 3)
```

Summary

- Scala combines functional and object-oriented programming
- Scala has builtin support for the Singleton design pattern
- Scala has *companion objects* instead of `static`