CSC 347 - Concepts of Programming Languages

Scala Classes

Instructor: Stefan Mitsch

Learning Objectives

❓ Object-oriented programming

- Identify classes and objects in Scala
- Describe class parameters

Classes

- Define a class `C`

```
1 class C (f1:Int, val f2:Int, var f3:Int):
2   //...
```

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

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

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

Class Parameters

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

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

- Execute in REPL
- Output

```
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

```
1 class C1 (x:Int):
2   def double() = x+x
3 end C1
```

Expressed in Java

```
1 public class C1 {
2   private final int x;
3
4   public C1(int x) { this.x = x; }
5
6   public int double() { return x+x; }
7 }
```

Class Parameter Details

- Immutable class parameters can be initialized but not modified

Scala

```
1 class C2 (val x:Int):
2   def double() = x+x
3 end C2
```

Expressed in Java

```
1 public class C2 {
2   private final int x;
3
4   public C2(int x) { this.x = x; }
5
6   public int x() { return x; }
7
8   public int double() { return x+x; }
9 }
```

Class Parameter Details

- Mutable class parameters can be changed even after initialization

Scala

```
1 class C3 (var x:Int):
2   def double() = x+x
3 end C3
```

Expressed in Java

```
 1 public class C3 {
 2   private int x;
 3
 4   public C3(int x) { this.x = x; }
 5
 6   public int x() { return x; }
 7
 8   public void setX(int x) { this.x = x; }
 9
10   public int double() { return x+x; }
11 }
```

Class Body

- Class body contains
  - Field declarations (`val` or `var`)
  - Constructor code
  - Method definitions

```
1  class C (f1:Int, val f2:Int, var f3:Int):
2    val f4 = f1 * f2
3    var f5 = f2 * f3
4
5    println ("Constructing instance of C")
6
7    def m (x:Int) : Int =
8      // cannot reassign to f1, f2, f4
9      f3 = f3 + 1
10     f5 = f5 + 1
11     f1 * f3 * x
12   end m
13 end C
```

Class Body

- Can omit empty body

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

- Can omit empty class parameters

```
1 class E:
2   private var n: Int = 0
3   def get () : Int =
4     val tmp = n
5     n = n + 1
6     tmp
7 end E
```

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

Methods vs. Fields

- `def` can be used non-parameterized: `def x = 5`; non-strict, executed every time
- `val` declares a variable: `val x = 5`; strict, initialized once
- `lazy val`: memoized `def`, initialized on demand

Scala

```
1 class C:
2   val x = 1
3   lazy val y = 1 + 2
4   def z = 1
5 end C
```

Expressed in Java

```
 1 public class C {
 2   private final int x = 1;
 3   private Integer y = null;
 4   public int x() { return x; }
 5   public int y() {
 6     if (y == null) y = 1 + 2;
 7     return y;
 8   }
 9   public int z() { return 1; }
10 }
```

Objects

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

```
1 object C:
2   var count:Int = 0
3 end C
4
5 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 =
    //...
  end main
end C
```

Companion Objects

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

Java: static vs. non-static

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

```java
class C {
  private int f1;
  public int m1 () {
    C.f2 = 3;
    return f1;
  }
  private static int f2;
  public static int m2 (C other) {
    other.f1 = 5;
    return f2;
  }
}
```

Scala: Companion object replaces `static`

- All data and methods belong to objects
- Companion related to other instances

```scala
class C:
  private var f1:Int = 0
  def m1 () : Int =
    C.f2 = 3
    f1

object C:
  private var f2:Int = 0
  def m2 (other: C) : Int =
    other.f1 = 5
    f2
```

Summary

- Scala class parameters and factory methods replace constructors
- Scala has builtin support for the Singleton design pattern
- Scala has `object` and *companion objects* instead of `static`