CSC 347 - Concepts of Programming Languages

Scala Introduction

Instructor: Stefan Mitsch

Learning Objectives

❓ How to combine multiple programming paradigms in a single language?

- Identify Scala basic syntax
- Identify and describe tuples in Scala
- Identify and describe lists in Scala

Scala

- Functional and object-oriented PL
- Java + ML + more
- Scalable Component Abstractions
- Compiles to JVM
- Interop: Scala calls Java; Java calls Scala
- Examples
  - Twitter Scala School
  - Apache Spark (Scala, Java, Python, R)
  - Chicago Scala Meetup

Scala

- Scala has a read-evaluate-print loop (REPL)
- Boolean literals: `false || true`
- Numeric literals: `1 + 2`
- String literals: `("hello" + " " + "world").length`
- Use of Java's libraries

```
1 val dir = java.io.File ("/tmp")
2 dir.listFiles.filter (f => f.isDirectory && f.getName.startsWith ("c"))
```

Everything is an Object

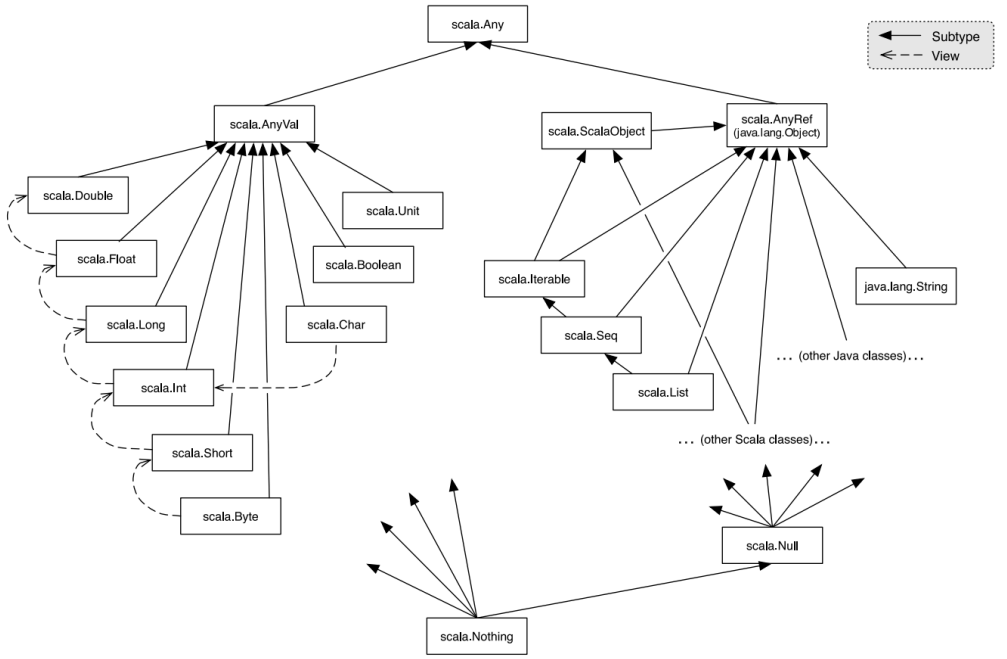
- `5:Int` is an object of type `Int` with methods: `5.toDouble`
- Methods can have symbolic names (see `scala.Int`): `5.+ (6)`
- `scala.runtime.RichInt` adds more methods: `5.max (6)`
- Any unary function `e1.f(e2)` can be written as `e1 f e2`
  - `5 + 6` is `5.+ (6)`
  - `5 max 6` is `5.max (6)`

Scala Type Checking

- Scala performs static type checking

```
1 def f () = 5 - "hello"   // rejected by type checker
```

- REPL prints types of expressions
- Java type hierarchy is embedded in Scala
  - Java primitive types are Scala value types
  - Java reference types are Scala reference types
  - `java.lang.Object` is `scala.AnyRef`



Mutable and Immutable Variables

Mutable Variables

- Scala

```scala
var x = 10          // declare and initialize x
x = 11              // assignment to x OK
```

- ❓ Java
- ❓ C

Immutable Variables

- Scala

```scala
val x = 10          // declare and initialize x
x = 11              // assignment to x fails
// error: reassignment to val
```

- ❓ Java
- ❓ C

Mutable and Immutable Variables

Mutable Variables

- Scala

```scala
var x = 10          // declare and initialize x
x = 11              // assignment to x OK
```

- Java

```java
int x = 10;         // declare and initialize x
x = 11;             // assignment to x OK
```

- C

```c
int x = 10;         // declare and initialize x
x = 11;             // assignment to x OK
```

Immutable Variables

- Scala

```scala
val x = 10          // declare and initialize x
x = 11              // assignment to x fails
// error: reassignment to val
```

- Java

```java
final int x = 10;  // declare and initialize x
x = 11;            // assignment to x fails
// error: cannot assign a value to final variable x
```

- C

```c
const int x = 10;  // declare and initialize x
x = 11;            // assignment to x fails
// error: assignment of read-only variable ‘x’
```

Expression Sequencing

C and Java Statements

```
1 int f() {
2   s_1;
3   s_2;
4   ...
5   return e_n;
6 }
```

Scala Expressions

```
1 def f() : Int =
2   e_1
3   e_2
4   ...
5   e_n
6 end f
```

Also allowed:

```
1 def f() : Int = {
2   e_1;
3   e_2;
4   ...
5   return e_n;
6 }
```

Methods

- Parameters require type annotations

```
1 def plus (x:Int, y:Int) : Int = x + y
2 def times (x: Int, y:Int)     = x * y
```

- Return types
  - can often be inferred
  - but are required for recursive methods
- Body of a method is an expression; its value is returned

Methods

- Conditional expressions (recursive)

```
1 def fact(n: Int) : Int =
2   if n <= 1 then 1
3   else n * fact (n - 1)
4 end fact
```

- For-expressions (non-recursive)

Range generator and iterator

```
1 def fact(n: Int) : Int =
2   var result = 1
3   for i <- 1 to n /* by 1 */ do
4     result *= i
5   result
6 end fact
```

Collection iterators

```
1 def sum(xs: List[Int]) : Int =
2   var result = 0
3   for n <- xs do
4     result += n
5   result
6 end sum
```

Scala Collections

- Scala collections guide
- `scala.collection`
- `scala.collection.immutable`
- `scala.collection.mutable`
- `java.util` is available
- Scala has arrays `Array[Int]`

Tuples

💡 Tuples: fixed number of immutable heterogeneous data items

Scala Tuples

```scala
val p = (5, "hello")
// val p : (Int, String) = (5, "hello")
val x = p(0)
// val x : Int = p(0)
```

Expressed in Java

```java
public class Pair<X,Y> {
  final X x;
  final Y y;
  public Pair (X x, Y y) { this.x = x; this.y = y; }
}

Pair<Integer, String> p = new Pair<> (5, "hello");
int x = p.x;
```

Expressed in C++

```cpp
#include <tuple>
auto p = std::make_tuple(5, std::string("hello"));
// std::tuple<int, std::string> p = std::make_tuple(5, std::string("hello"))
auto x = std::get<0>(p)
// int x = std::get<0>(p)
```

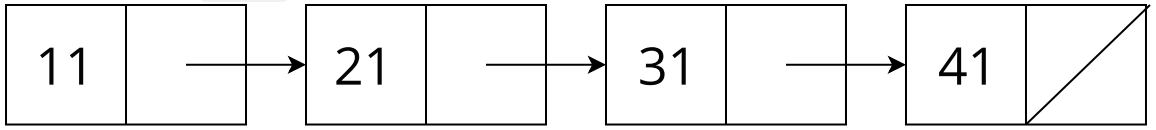
Linked Lists

💡 Lists: variable number of immutable homogeneous data items

- List constructors

```
1 List (1, 2, 1 + 2)
2 1 :: 2 :: (1+2) :: Nil
```

- Scala's `::` is an *infix* operator to construct lists




```
1 // List(11, 21, 31, 41)
2 val xs = 11 :: 21 :: 31 :: 41 :: Nil  // :: is right-associative
3 // with explicit parentheses
4 val xs = 11 :: (21 :: (31 :: (41 :: Nil)))
5 // method-call style, not encouraged!
6 val xs = Nil.::(41).::(31).::(21).::(11)
```

List Projections

- Projections extract components of a list: often called `head` and `tail`

```
1 val xs = List(1, 2, 3, 4)
2 val x  = xs.head // x  == 1
3 val ys = xs.tail // ys == List(2, 3, 4)
```

Summary

- Scala combines functional and object-oriented programming

- **Everything is an expression** with a result value (even loops)
- **Type inference**: `val x = 10` is `val x: Int = 10`
- **Immutable** vs **mutable** variables: `val x = 10` vs. `var x = 10`

- **Methods**
  - require argument types
  - can infer return types
  - last expression in body determines return value
- **Tuples** `(1, "hello")` or `1 -> "hello"`
- **Immutable lists** `List(1,2,3)` or `1 :: 2 :: 3 :: Nil`