Comp 311
Functional Programming

Nick Vrvilo, Two Sigma Investments
Robert “Corky” Cartwright, Rice University

October 1, 2019

Announcements

- Homework 1 was due today before class
- Homework 2 will be posted after class today
- Assignment description PDF on Piazza
- No provided “skeleton” code
- Simple interface (compilation/linking) check provided
- Due in 2 weeks

Additional Syntax for Homework 2

Repeated Parameters

- Scala allows the last entry in a parameter list to stand for zero or more arguments

- Arguments are placed in a sequence of the given type

```
def squares(xs: Int*) =
  xs.map(x => x*x)
```

Repeated Parameters

- Scala allows the last entry in a parameter list to stand for zero or more arguments

- Arguments are placed in a sequence of the given type

```
squares(4,2,6,5,8)
squares()
squares(4,2,6,8)
squares(3)
squares(4,3,7)
```

Repeated Parameters

- Scala allows the last entry in a parameter list to stand for zero or more arguments

- Arguments are placed in a sequence of the given type

```
def fnName(a1: T1, ..., aN: TN*) = expr
```

Repeated Parameters

- Scala allows the last entry in a parameter list to stand for zero or more arguments

- Arguments are placed in a sequence of the given type

```
squares(1, 2, 3, 4, 5) ↦*
ArraySeq(1, 4, 9, 16, 25)
```

ArraySeq

- ArraySeq[T] is an immutable view of an Array[T]
  - Random access to elements
  - Similar idea to Java's ArrayList class, but immutable
- ArraySeq[T] is the default type for Scala varargs sequences
  - Can pass in other Seq types (e.g., List or Vector)
  - Note that varargs are not raw Arrays as in Java

Repeated Parameters

- If you have an array and you wish to pass it to a repeated parameter, include the suffix `:_*`

```
val myList = List(1, 2, 3)
squares(myList: _*)
```

- The `_*` syntax also works in sequence patterns:

```
List(1, 2, 3) match {
  case List(x, _, y, zs @ _*) => ???
}
```

Guidelines on Repeated Parameters

- Use repeated parameters to provide factory methods for collections classes

```
List(1,2,3,4,5)
```

- Use repeated parameters for methods that map over an immediately provided set of values

```
squares(1,2,3,4,5)
```

- Use repeated parameters for folds over an immediately provided set of values

```
sum(1,2,3,4,5)
```

Named Arguments

- With *named arguments*, the arguments to a function can be passed in any order
- Each argument must be prefixed with the name of the parameter and an equals sign:

```
def speed(distance: Double, time: Double) =
  distance/time

speed(distance = 2.0, time = 5.0)
```

Named Arguments

If positional arguments are mixed with named arguments, the positional arguments must come first

```
def speed(distance: Double, time: Double) =
  distance/time

speed(2.0, time = 5.0)
```

Guidelines on Named Arguments

- Named arguments add bulk to function applications
- Use when it's unclear which arguments correspond to which parameters, e.g.:
  - There are multiple arguments of the same type
  - There is no natural order for the arguments
  - The expected order of the arguments is difficult to remember

Default Parameter Values

- Function parameters can include default values:

```
case class Circle(radius: Double = 1) extends Shape {
  val pi = 3.14

  def area = { pi * radius * radius }
  def makeLikeMe(that: Shape): Circle = this
}
```

- The argument for a parameter with a default value can be omitted at the call site:

```
Circle()
```

Guidelines of Default Parameter Values

- Consider default parameter values instead of static overloading
- Use when there is a common argument value that is usually used
  - A default I/O source, file location, etc.

Non-trivial Postconditions

```
def add(x: Int, y: Int) = {
    x + y
 } ensuring(result => {
      if (x >= 0 && y >= 0)
        result >= max(x, y)
      else if (x < 0 && y < 0)
        result < min(x, y)
      else
        result >= min(x, y)
}, "integer overflow during addition")
```

String Interpolation

Scala s-strings support *interpolation* of values:

Prefix the string with an s

```
s"My name is $name."

s"My name is ${name}.“

s"Max value: ${max(x, y)}."
```

Prefix variable names with $

General expressions must be wrapped in braces

String Formatting

Scala strings support `printf`-style formatting:

```
"Customer %-20s ordered %07d units"
    .format(accountName, 500)
```

F-String Interpolator

Scala f-strings support `printf`-style formatting:

```
f"Customer $account%-20s ordered $n%07d units"
```

Raw-String Interpolator

Scala strings support “raw” input
(useful for regular expressions):

```
raw"^[a-zA-z][-.\w]+\s+\d{1,5}$“
```

Triple-quoted strings are both *raw* and *multi-line*

```
"""^[a-zA-z][-.\w]+ "quote" \d{1,5}$"""

"""String
   spanning
   four
   lines"""
```

Package Declarations

An entire source file can be "placed" within a *package* (i.e., a module) using a `package` statement at the beginning of the file:

```
package comp311.hw2
```

Scala also supports scoping declarations to specific packages using braces to enclose the package contents:

```
package comp311 {
  package hw2 {
    def fn() = ???
  }
}
```

See also https://docs.scala-lang.org/tour/packages-and-imports.html