Software Engineering
Design & Construction

TECHNISCHE UNIVERSITÄT DARMSTADT

Exercise Session 1: Closures

Dr.-Ing. Michael Eichberg

Software Technology Group
TU Darmstadt | FB Informatik

Introduction

- You have seen an introduction to Scala
- Goal of first exercise:
  - getting familiar with Scala, hands-on
  - make sure you have a solid understanding of closures and related concepts

- Now, a few specific constructs and basic terminology...

Basic Concepts

- Higher-order function
- Closure
- First-class value (object, citizen, etc)
- Anonymous class (in Java)

- Dynamic dispatch
- Recursive algorithm

Methods and Parameters

Java:

- data is defined by classes
- operations are defined by methods
  - the language does not have functions
- a method can be parameterized over values (and types).
- a method is **not** a first-class value

How do I pass an operation to a method?

Methods and Parameters

Example (from `java.util.Collections`)

```
<T> T max(Collection<? extends T> coll,
          Comparator<? super T> cmp)
```

Simplified (`T = String`):

```
String max(Collection<String> coll,
           Comparator<String> cmp)
```

Usage:

```
max(myStringColl, new Comparator() {
  int compare(String a, String b) {
    return a.compareToIgnoreCase(b);
  }
});
```

Methods and Parameters

In Scala:

```
def max[T](coll: Collection[T],
           cmp: (T, T) => Boolean)
```

Usage:

```
max(myStringColl,
    { (a,b) => a.compareToIgnoreCase(b) })
```

Methods and Parameters

More idiomatic Scala:

```
def max[T](coll: Collection[T])
          (cmp: (T, T) => Boolean)
```

Usage:

```
max(myStringColl) { (a,b) =>
  a compareToIgnoreCase b
}
```

Or even:

```
max(myStringColl) { _ compareToIgnoreCase _ }
```

Higher-Order Function

A higher-order function is a function that

- takes **function(s) as argument(s)**, or
- returns a function

Functions in Scala are **first-class values**.

Lists

```
List(1,2,3) map { x => x*x }
  == List(1, 4, 9)

List(1,2,3) filter { x => x % 2 == 0 }
  == List(2)

List(1,2,3) foreach { x => println(x) }
```

(prints “123”)

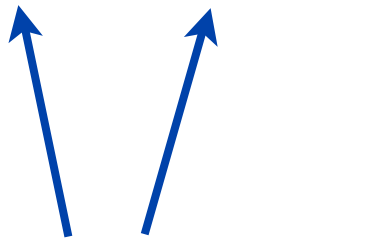
```
List(1,2,3).foldLeft(0){ (acc, x) => acc + x }
  == ((0 + 1) + 2) + 3 == 6

List(1,2,3).foldRight(0){ (x, acc) => acc + x }
  == 0 + (1 + (2 + 3)) == 6
```

Closures

```
sum += x
```

Closures

```
sum += x
```



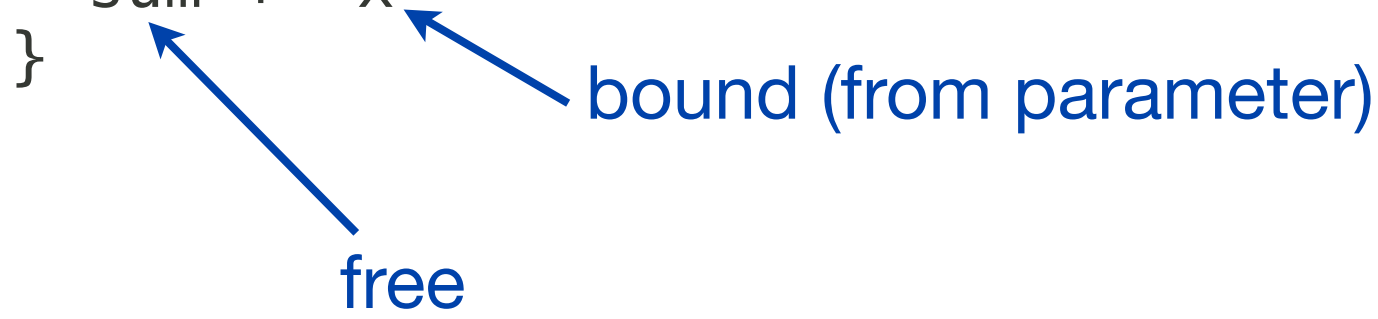
free variables

Closures

```
def add(x: Int) {
  sum += x
}
```

Closures

```
def add(x: Int) {
  sum += x
}
```

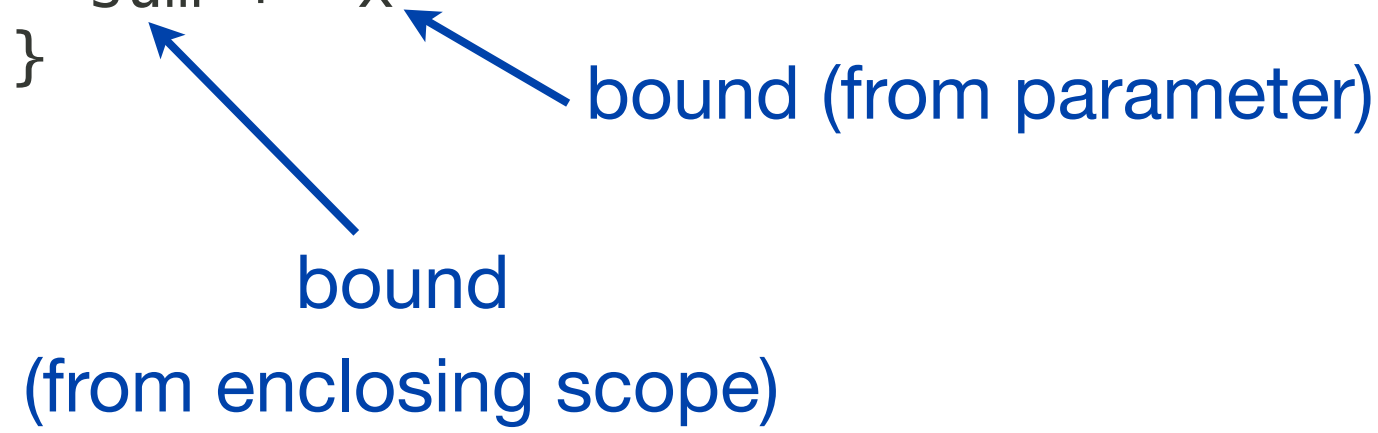


Closures

```
var sum = 0

def add(x: Int) {
  sum += x
}
```

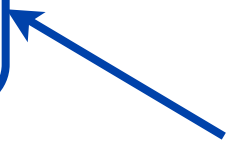
Closures

```
var sum = 0

def add(x: Int) {
  sum += x
}
```



Closures

```
var sum = 0

def add(x: Int) {
  sum += x
}
```



closed expression
(no free variables)

Closures

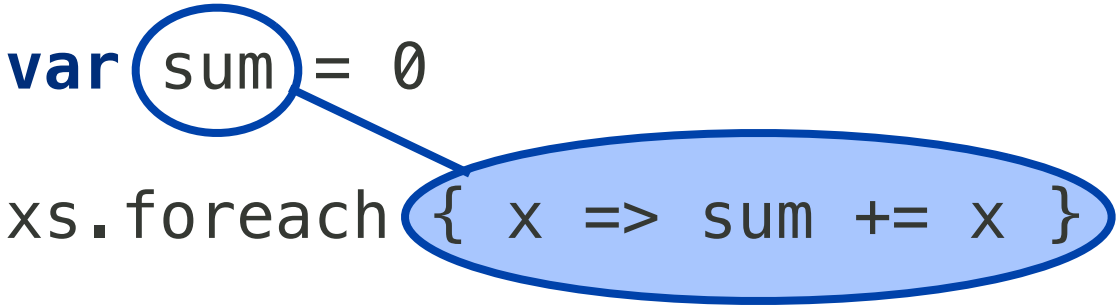
```
var sum = 0

xs.foreach { x =>
  sum += x
}
```

Closures

```
var sum = 0

xs.foreach { x =>
  sum += x
}
```

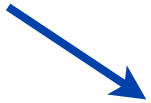
Closures

Closure:
function plus reference to its environment



Closures

```
var sum = 0

xs.foreach { x =>
  sum += x
}
```



```
val sum = new IntVar(0)

xs.foreach(new Function1[Int, Unit] {
  def apply(x: Int) = sum += x
})
```

```
val sum = new IntVar(0)

class Anon$0(sum$0: IntVar) extends Function1[Int, Unit] {
  def apply(x: Int) = sum$0 += x
}
val anon$0 = new Anon$0(sum)
xs.foreach(anon$0)
```

Closures (Scala) vs SAM Classes (Java)

Instead of a closure in Scala, you can sometimes use a class or interface with a single abstract method (SAM) in Java.

How are closures different from SAM class instances?

- More concise
- Non-local returns
- ...

Java 8 Closures

Java 8 has a restricted form of closures:

- They differ from Scala's closures in a number of ways (also see exercise)

Details on:

- https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html