Tail Calls

Josh Cough

What Will I Learn

- Learn what tail calls are
- Learn to avoid stack overflow on the JVM
  - with scalac
  - with @tailrec
  - with trampolining

Definitions

- Tail Call

  When the last thing a function does is call another function, and return that functions result.

Definitions

- Tail Call

  When the last thing a function does is call another function, and return that functions result.

- Tail Recursion

  Tail call, but with callee == caller.

Definitions

- Tail Call

  When the last thing a function does is call another function, and return that functions result.

- Tail Recursion

  Tail call, but with callee == caller.

- Tail Call Optimization

  Don't consume stack space for tail calls.

Tail Call Example

```
def x() = {
  y() // regular call
  z() // tail call
}
```

Tail Call Example

```
def x(i:Int) {
  if(i==5) y() // tail call
  else z()     // tail call
}
```

Not a Tail Call

```
def x(): Int = {
  y() // regular call
  z() // regular call
  7   // not a call.
}
```

Tail Recursion?

```
def fact(n:Int): Int =
  if(n==1) 1 else n * fact(n-1)
```

Not Tail Recursion!

```
def fact(n:Int): Int =
  if(n==1) 1 else n * fact(n-1)
```

Can We Fix It...?

Can We Fix It...?

Here's an ugly way.

Can We Fix It...?

Here's an ugly way.

```
def fact(n:Int): Int = {
  var tmp = n
  var acc = 1
  while(tmp>1) {
    acc = tmp * acc
    tmp -= 1
  }
  acc
}
```

Can We Fix It...?

Can We Fix It...?

There's a nicer way:

Refactor to tail recursion.

Can We Fix It...?

There's a nicer way:

Refactor to tail recursion.

```
def fact(n:Int, acc:Int=1): Int =
  if(n==1) acc else fact(n-1, n * acc)
```

But...

But...

The JVM doesn't support tail calls.

So how does this help?

scalac helps

scalac automatically converts tail recursion into a while loop*

scalac helps

scalac automatically converts tail recursion into a while loop*

* in most situations (I'll explain)

Let's decompile this

```
object Fact {
  def fact(n:Int, acc:Int=1): Int =
    if(n==1) acc else fact(n-1, n * acc)
}
```

Decompiled Java

```
public int fact(int n, int acc) {
  while (true) {
    if (n == 1) return acc;
    acc = n * acc;
    n -= 1;
  }
}
```

@tailrec

But...I've heard about @tailrec, now I'm confused.

@tailrec

http://www.scala-lang.org/api/current/scala/annotation/tailrec.html:

'A method annotation which verifies that the method will be compiled with tail call optimization. If it is present, the compiler will issue an error if the method cannot be optimized into a loop.'

Fact again

```
object Fact {
  def fact(n:Int, acc:Int=1): Int =
    if(n==1) acc else fact(n-1, n * acc)
}
```

Adding @tailrec

```
object Fact {
  @scala.annotation.tailrec
  def fact(n:Int, acc:Int=1): Int =
    if(n==1) acc else fact(n-1, n * acc)
}
```

Change object to class

```
class Fact {
  @scala.annotation.tailrec
  def fact(n:Int, acc:Int=1): Int =
    if(n==1) acc else fact(n-1, n * acc)
}
```

Error

could not optimize @tailrec annotated method fact:
it is neither private nor final so can be overridden

Recall the original fact

```
object Fact {
  def fact(n:Int): Int =
    if(n==1) 1 else n * fact(n-1)
}
```

Recall the original fact

```
object Fact {
  @scala.annotation.tailrec
  def fact(n:Int): Int =
    if(n==1) 1 else n * fact(n-1)
}
```

Error

could not optimize @tailrec annotated method fact:
it contains a recursive call not in tail position

Trampolining

`scala.util.control.TailCalls`

Recall

```
object PlainPopcornNormalCalls {
  def main = println("yum: " + askMrSalt)
  def askMrSalt = askMrButter
  def askMrButter = askMrPopper
  def askMrPopper = "plain popcorn"
}
```

Trampolining

```
import scala.util.control.TailCalls._

object PlainPopcornTailCalls {
  def main = println("yum: " + askMrSalt.result)
  def askMrSalt = tailcall(askMrButter)
  def askMrButter = tailcall(askMrPopper)
  def askMrPopper = done("plain popcorn")
}
```

Trampolining

```
import scala.util.control.TailCalls._

object PlainPopcornTailCalls {
  def main = println("yum: " + askMrSalt.result)
  def askMrSalt = tailcall(askMrButter)
  def askMrButter = tailcall(askMrPopper)
  def askMrPopper = done("plain popcorn")
}
```

Mutually Recursive Tail Call

```
def even(n:Int): Boolean =
  if(n==0) true
  else odd(n-1)

def odd(n:Int): Boolean =
  if(n==0) false
  else even(n-1)
```

Trampolining

```
import scala.util.control.TailCalls._

def even(n:Int): TailRec[Boolean] =
  if(n==0) done(true)
  else tailcall(odd(n-1))

def odd(n:Int): TailRec[Boolean] =
  if(n==0) done(false)
  else tailcall(even(n-1))
```