

Lecture 34: Scala Traits

CSCI 131
Spring, 2011

Kim Bruce

Approximating Multiple Inheritance

- No interfaces in Scala
- Traits are new construct -- more powerful
- Purely abstract trait is like interface
- But traits can contain implementations

Iterator Traits

```
trait AbsIterator[T] {  
  def hasNext: boolean  
  def next: T  
}
```

Can also contain behavior:

```
trait RichIterator[T] extends  
  AbsIterator[T] {  
  def foreach(f: T => unit): unit =  
    while (hasNext) f(next)  
}
```

Traits can be used as Classes

- But traits have no constructors
- Can also be used as “mixins”
- Class can extend at most one class or trait
 - but can mixin many traits

Trait as Superclass

```
class StringIterator(s: String) extends AbsIterator[Char] {  
  private var i = 0  
  def hasNext = i < s.length  
  def next = { val x = s.charAt(i); i = i + 1; x }  
}
```

Traits applied in Order

```
object Test {  
  def main(args: Array[String]): Unit = {  
    class Iter extends StringIterator(args(0))  
      with RichIterator[Char]  
    val iter = new Iter  
    iter.foreach(System.out.println)  
  }  
}
```

- Solves multiple inheritance problems because linearize order
 - last definition wins
 - super refers to previous trait/superclass
 - Can override in body or in any trait

Another Example

```
trait SyncIterator[T] extends AbsIterator[T] {  
  abstract override def hasNext: Boolean =  
    synchronized(super.hasNext)  
  abstract override def next: T =  
    synchronized(super.next)  
}
```

abstract because not implemented in AbsIterator

```
StringIterator(someString) with RichIterator[Char]  
  with SyncIterator[Char]
```

- Order not matter here, but could in other examples
- Super calls go to StringIterator, not AbsIterator

Evaluating OOLs

Evaluation of OOL's

- Pro's (e.g., with Eiffel and Java)
 - Good use of information hiding. Objects can hide their state.
 - Good support for reusability. Supports generics like Ada, run-time creation of objects (unlike Ada)
 - Support for inheritance and subtyping provides for reusability of code.

Evaluation of OOL's

- Con's
 - Loss of locality.
 - Type-checking too rigid, unsafe, or requires link time global analysis. Others require run-time checks.
 - Semantics of inheritance is very complex.
 - Small changes in methods may make major changes in semantics of subclass.
 - Must know definition of methods in superclass in order to predict impact on changes in subclass. Makes provision of libraries more complex.
 - Weak or non-existent support of modules.

Concurrent & Parallel Programming Constructs

What is the difference?

- Parallel programming is about using additional computational resources to produce an answer faster.
- Concurrent programming is about correctly and efficiently controlling access by multiple threads to shared resources.
 - Includes providing reasonable response times.

Definitions by Dan Grossman.

Why Important

- Speed-ups limited w/single processors
 - dual/quad/oct processors now standard
- Required for distributed processing
- Concurrency required for event-driven programming