CS-3710 Vocational Languages

Nov 14, 2012

foldr

```
Array.prototype.foldr=function(f,terminal)
{
  function recCall(result,list){
    if(list.length==0){return result;}
    else{
         result= recCall(result,list.slice(1));
         document.writeln(result+" ,"+list);
         return f(list[0],result);
        }
    }
  return recCall(terminal,this);
}
```

Fixed-length Arrays

- Declare fixed-length arrays, initialized with Scala default values: 0 for numbers, null for objects.

```
scala> val nums=new Array[Int](2)
newNum: Array[Int] = Array(0, 0)
scala> nums=newNum
<console>:8: error: reassignment to val
  num=newNum
```

- Declare fixed-length array, initialized with user defined value

```
scala> var values=Array(1,2,3,4,5)
scala> values=nums;
```

- Access Array element:

```
scala> nums(0)=100 //values(0)becames 100
```

Flexible-length Arrays

- Declare flexible-length arrays, with length 0

```
scala> import scala.collection.mutable.ArrayBuffer
scala> val b=ArrayBuffer[Int]()
```

- Add 1 element

```
scala> b+=1
res3: b.type = ArrayBuffer(1)
```

- Add multiple elements

```
scala> b+=(1,2,3,5)
res5: b.type = ArrayBuffer(1, 1, 2, 3, 5)
```

- Concatenate two arrays:

```
scala> b++=nums
res6: b.type = ArrayBuffer(1, 1, 2, 3, 5, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0)
```

Working with Arrays and Array Buffers

```
scala> var values=Array(1,2,3,4,5)
scala> for(i <- 1 until values.length) print(values
  (i))
res10: 2345
scala> for(e <- values) print(e+",")
res11: 1,2,3,4,5,
scala> for (e <- values) yield 2*e
res12: Array[Int] = Array(2, 4, 6, 8, 10)
//equivalent to values.map(2*_)
scala> for (e <- values if e % 2 == 1) yield e
res13: Array[Int] = Array(1, 3, 5)
//equivalent to values.filter(_%2==1)
```

Multiple Dimensional Arrays

- Implemented as arrays of arrays.
- Create a 3x4 array:

```
scala> val matrix = Array.ofDim[Double](3,4)
matrix: Array[Array[Double]] = Array(Array(0.0, 0.0,
  0.0, 0.0), Array(0.0, 0.0, 0.0, 0.0), Array(0.0,
  0.0, 0.0, 0.0))
```

- Access Array element:

```
scala> matrix(1)(2)=23
scala> matrix(1)(2)
res21: Double = 23.0
```

Class

- In Scala, a class is not declared as public. A Scala source file can contain multiple classes and all of them have public visibility.

```
class Counter{//default is public
    private var value=0;
    //the code above is a part of the default primary
      constructor
    def increment(){value +=1;} //methods are
    def current()=value; //public by default
}
scala> val myCounter=new Counter(); //call default primary
   constructor.
myCounter: Counter = Counter@260ec729
scala> myCounter.increment
scala> myCounter.current
res23: Int = 1
```

Primary Constructor

- Every class has a primary constructor which is defined with the class definition.

```
scala> class Person(val name: String, var age: Int){}
//define a class, its primary constructor and two
  public members
defined class Person
scala> val John=new Person("John",20);
scala> John.name
res25: String = John
scala> John.age
res27: Int = 20
scala> John.age=30
John.age: Int = 30
scala> John.name="Jo"
<console>:10: error: reassignment to val
       John.name="Jo"
```

Auxiliary Constructors

- When no primary constructor is provided, compiler provides one that takes no parameter, hence no implicitly defined members.
- The class can define the members with initial value explicitly like the `Counter` class.
- Alternatively, the class can define overloaded auxiliary constructors, whose name is this, that either calls the default primary constructor or other auxiliary constructors to construct a new object.

Auxiliary Constructors - Example

```
scala> class Person{
   | private var name="";
   | private var age=0;
   | def this(name: String){this();this.name=name;};
   | def this(name:String,age:Int){this(name);
   this.age=age}
}
defined class Person
scala> var p1=new Person //call?
p1: Person = Person@3f9b12fb
scala> var p2=new Person("John") //call?
p2: Person = Person@c2dd10a
scala> var p3=new Person("John",20)//call?
p3: Person = Person@1d8299fd
```

Object Construct

- Scala has no static methods or fields.
- We can use object construct to define a single instance of a class with the methods and fields we need.

```
object Q1 {
    def main(args: Array[String]){
        val a = Array.ofDim[Int](3,4);
         //fill in a value here
        val b = Array.ofDim[Int](4,5);
         //fill in b value here
        val c = matrixProduct(a,b);
         //print c here
   }
   def matrixProduct(a: Array[Array[Int]],
  b :Array[Array[Int]]):Array[Array[Int]]={a;}
}
```

Companion Objects

- Class definition

```
class Account{
val id=Account.newUniqueNumber()
private var balance=0.0;
def deposit(amount: Double): Double={balance+=amount;
   balance;}
}
```

- Companion Object

```
object Account{
private var lastNumber=0;
private def newUniqueNumber()={lastNumber+=1; lastNumber;}
}
var b=new Account();
println(b.id); //>
println(b.balance); //?
println(b.deposit(100)); //?
```