CS 520 Advanced Programming Languages
Fall Semester, 2009
Doc 23 Scala Assignment 2 Comments
Dec 10, 2009

Issues

```
class trimString(n:String){
def trimBlank(n:String):String={
return n.trim
}
def trimBlanks= trimBlank(n)
}
implicit def trimStr(n:String) = new trimString(n)
println(" a b c d ".trimBlanks)
```

Issues

```
def trimBlanks(s:String):String = {
val trimmed=s.trim
return trimmed
}
```

Issues

```
object StudentGrades {
def fromfile(f:String):List[StudentGrades]  = {
val lines = Source.fromFile(f).getLines.toList
val splitstring = lines.map(line => line.split(','))
val name = splitstring.map(name => name.apply(0))
val grades = splitstring.map(grade => grade.drop(1))
val gradesList = grades.map(gr => doubleList(gr))
val objects=lines.map(a => new
StudentGrades(name(name.indexOf(a)),gradesList(lines.indexOf(a))))
name.foreach(a=> println(a))
objects.foreach(a=>println(a.average +" \n" + a.grade))
return objects
}
```

Issues

```
def grd()={
val avg=average
var g=""

if(avg>90 && avg<=100)g="A"
if(avg>80 && avg<=90) g="B"
if(avg>70 && avg<=80) g="C"
if(avg>60 && avg<=70) g="D"
if(avg>0  && avg<=60) g="F"
g
}
}
```

Issues

```
def grade : Unit = {
val a: Double = average
if(a <= 100 )
if(a >= 90 )
println("A")
if(a <= 89 )
if(a >= 80 )
println("B")
if(a <= 79 )
if(a >= 70 )
println("C")
if(a <= 69 )
if(a >= 60 )
println("D")
if(a <= 59 )
if(a >= 0 )
println("F")
}

}
```

Issues

```
class StudentGrades(name:String,grades:Seq[Double]){
def gradeSum:Double=grades.foldLeft(0.0) { (x,y) => x + y }
def average:Double=gradeSum/grades.length
def grade: Char={if(average >= 90) 'A'
                else if( average >= 80) 'B'
                else if( average >= 70) 'C'
                else if( average >= 60) 'D'
                else 'F'}
}
```

Issues

```
object StudentGrades {

  def fromfile(fileName: String):List[StudentGrade]={
    var student = fromFile(fileName).getLines.toList
    val temp = student.map(line=>line.split(','))
    val name =temp.map(each=>each.apply(0))
    val grade = temp.map(line=>line.drop(1))
    val grades = grade.map(marks => { marks.map(mk=>mk.toDouble)} )

    var j = grades.length-1
    var i=1
        var x = new StudentGrade(name.apply(j), grades(j))
      var list =List(x)
      j=j-1
      if(j>=0){
      do{
        var y = new StudentGrade(name.apply(j),grades(j))
       list = y ::list
        j=j-1

      }while(j>=0)
        }
        return list
  }
}
```

Issues

```
def trimBlanks(a: String):String = {
                    b:String;
                    b=a.trim;
                    b
        }
```

Issues

```
def add():Double={
        val sum = grades.reduceLeft(_+_)
        return(sum)
}
```

Issues

```
if(tempStudent==null) {}
        //Good input.  Add it to the return list. Note the concatenation.
else
        returnList = returnList ::: List(tempStudent)
}
```

A Verses B

```
def grade():String={
      val avg = average()
      if (90<=avg && avg <= 100)
        "A"
      else if (80<=avg && avg< 90)
        "B"
      else if (70<=avg && avg< 80)
        "C"
      else if (60<=avg && avg< 70)
              "D"
      else
        "F"
}
```

```
def grade = average match {
      case n if n > 90 => "A"
      case n if n > 80 => "B"
      case n if n > 70 => "C"
      case n if n > 60 => "D"
      case _ => "F"
}
```

Issues

```
def fromFile(fileName:String):List[StudentGrades]={
      try{
            val entireFile = Source.fromFile(fileName).getLines.reduceLeft(_+_)
            val line = entireFile.split("\n").filter(s=> s.length!=1)
            val ObjectMap = line.map(s=>getObject(s))
            ObjectMap.toList
      }
      catch{
            case ex : FileNotFoundException =>
                  println("Error: File Not Found/Invalid File")
                  null
      }

}
```

Issues

```
def trimBlanks(s: String): String = {
  var str: String = "";
  if(s.startsWith(" ")){
      str = s.trim();
  }

  if(s.endsWith(" ")){
      str = s.trim();
  }
  return str;
}
```

Issues

```
def fromfile(x:String):StudentGrades ={
    val lines=Source.fromFile(x).getLines.toList
    val a=lines.map(each=>each.split(','))
    a(0).foreach(println)
    val names=a.map(each=>each.apply(0))
    names.foreach(n=>println(n))
    val grades=a.map(each=>each.drop(1))
    grades.foreach(g=>println(g))
    val sgrades = grades.map(grd => change(grd))
    val x1=new StudentGrades(names(0), sgrades(0))
    x1

  }
}
```

Issues

```
def grade{
        if ( average >= 90) println("A")
        else if ( average >= 80) println("B")
        else if ( average >= 70) println("C")
        else if ( average >= 60) println("D")
        else println("F")
}
```

Issues

```
object StudentGrades{

    def fromFile(fileName: String):List[StudentGrades] = {
        var studinfo: List[StudentGrades] = List()
        var line:Iterator[String]= Source.fromFile(fileName).getLines //returns a iterator
        line.foreach(line=>if(line != '\n' ) {
            val list:List[String] = line.replaceAll(" ","").split(",").toList
            val Studname:String = list.head
            val Studmarks:List[Double] = list.tail map(_.toDouble)
            studinfo = List(new StudentGrades(Studname,Studmarks))::: studinfo
                                    })//foreach closes here
        studinfo
    }
}
```

Solutions

Problem 1

```
def trimBlanks(string: String) = {
    def trimFront(aString: String):String = aString.charAt(0) match {
        case ' ' => trimFront(aString.drop(1))
        case _ => aString
    }

    def trimBack(aString: String) = trimFront(aString.reverse).reverse

    return trimBack(trimFront(string))
}
```


This solution is a bit inefficient for a string library that people would use all the time.

Problem 2

```
class TrimString(str :String) {
    def trimBlanks():String = str.trim
}

implicit def stringToTrimString(x: String) = new TrimString(x)
```

Problem 3

```
case class StudentGrades(val name: String, val grades: Seq[Float]) {
    private def sum(items: Seq[Float]) = items.reduceLeft(_+_)

    def average = sum(grades)/grades.length

    override def toString = "name " + name + " average " + grade

    def grade = average match {
        case n if n > 90 => "A"
        case n if n > 80 => "B"
        case n if n > 70 => "C"
        case n if n > 60 => "D"
        case _ => "F"
    }
}
```

Problem 4

```
object StudentGrades {
	def fromFile(fileName: String) = {
		val studentsGradeData:List[String] = fileLines(fileName)
		studentsGradeData.map(studentFromString(_))
	}

	def studentFromString(line:String) ={
		val (name:String, grades:List[Float]) = stringToNameGrades(line)
		new StudentGrades(name, grades)
	}
	def stringToNameGrades(line:String) = {
		//line format = name,grade1,grade2,...,gradeN
		//string.toFloat removes blanks from ends of string
		val tokens = line.split(",")
		val name = tokens(0)
		val grades = tokens.drop(1).map(_.toFloat).toList
		(name, grades)
	}

	def fileLines(fileName: String):List[String] =
		scala.io.Source.fromFile(fileName).getLines.toList
}
```