Kontrollskrivning
EDAA45 Programmering, grundkurs

2016-10-25, 14:00-19:00, Kårhusets Gasquesal

Hjälpmedel: Snabbreferens för Scala & Java.

Instruktioner

Du får ett identitetsnummer som du ska ange här, på omslaget och på alla inlämnade blad: _____
Kontrollskrivningen är indelad i tre moment:

- *Moment 1*, ca 2,5 h: **Lösning av uppgifterna**. Du löser uppgifterna individuellt; del A direkt i detta häfte och del B på separat papper. Du får endast lämna in *ett* svar per uppgift. Skriv med blyertspenna. Du får inte skriva med rödfärgad penna. Du ska inte lämna in kladdpapper eller anteckningar som inte ingår i dina svar. Du ska skriva ditt identitetsnummer i övre högra hörnet på *alla* inlämnade blad. När du är klar, eller när tiden är ute, lägger du dina svar inklusive detta häfte inuti omslaget och låter skrivningen ligga på din bänk. Är du klar innan sluttiden får du under tystnad ta fram en bok, din telefon, eller liknande.
- *Moment 2*, ca 1,5 h: **Parvis kamratbedömning**. Ni sätter er parvis enligt muntliga instruktioner och får ut bedömningsmallen som ni läser igenom noga. Efter ett tag får ni ut två andra personers skrivningar som ni poängsätter enligt anvisningarna i bedömningsmallen. När tiden är slut samlar lärare in alla skrivningarna.
- *Moment 3*, ca 0,5 h: **Bedömning av kamratbedömningen**. Du får nu inspektera din egen skrivning och värdera poängsättningen. Är du inte nöjd med poängsättningen skriver du ett kryss i motsvarande ruta på omslagets baksida och beskriver utförligt på baksidan av omslaget vad i poängsättningen du anser bör korrigeras och varför. Du får *inte* ändra i själva skrivningen, bara skriva på omslagets baksida (och omslagets insida om du behöver extra utrymme). Efter ett tag samlas skrivningen in och du kan därefter lämna lokalen. Om du vill gå i förtid, kontakta skrivningsansvarig.

Kontrollskrivningen motsvarar i omfång en halv ordinarie tentamen och är uppdelad i två delar; del A och del B. Följande poängfördelning gäller:

- Del A omfattar 20% av den maximala poängsumman och innehåller uppgifter med korta svar.
- Del B omfattar 80% av den maximala poängsumman och innehåller uppgifter med svar i form av programkod som du ska skriva på separat papper.
- Maximal poäng: 50
- Om du erhåller `p` poäng på kontrollskrivningen bidrar du med `(p / 10.0).round.toInt` i individuell bonuspoäng inför sammanräkningen av samarbetsbonus.

Den diagnostiska kontrollskrivningen påverkar inte om du blir godkänd eller ej på kursen, men det samlade poängresultatet för din samarbetsgrupp ger möjlighet till *samarbetsbonus* som kan påverka ditt betyg.

Del A. 10 p

Följande kod finns kompilerad utan kompileringsfel och tillgänglig på din *classpath*:

```
1  trait Movable {
2    def move(dx: Double, dy: Double): Movable
3  }
4
5  case class Point(x: Double = 0.0, y: Double = 0.0) extends Movable {
6    def distanceTo(other: Point): Double = math.hypot(x - other.x, y - other.y)
7    def move(dx: Double, dy: Double): Point = Point(x + dx, y + dy)
8  }
9
10 case class Polygon(pts: Vector[Point]) extends Movable {
11   def append(p: Point): Polygon = Polygon(pts :+ p)
12   def move(dx: Double, dy: Double): Polygon = Polygon(pts.map(_.move(dx, dy)))
13 }
14 object Polygon {
15   def apply(pts: Point*): Polygon = new Polygon(pts.toVector)
16 }
```

Du ska fylla i tabellen på nästa sida enligt följande. Antag att du skriver in nedan kod i Scala REPL rad för rad. För varje variabel med namn `u1` ... `u5`, ange statisk **typ** (alltså den typ kompilatorn härleder), samt det **värde** variabeln får efter initialisering, *eller* sätt i stället kryss i rätt kolumn om det blir ett **kompileringsfel** respektive **exekveringsfel**. Vid frånvaro av fel, svara på samma sätt som Scala REPL skriver ut typ respektive värde, enligt exempel `u0` i tabellen. Det förekommer varken kompilerings- eller exekveringsfel på raderna 1–5 nedan, medan efterföljande rader kan innehålla fel.

```
1  val p1 = Polygon(Point(10,10), Point(30, 10), Point(30, 20), Point(10, 20))
2  val p2 = p1.move(100, 100)
3  val ps: Vector[Movable] = Vector(p1, Point())
4  def moveAll(xs: Seq[Movable], dx: Double, dy: Double): Seq[Movable] =
5    xs.map(_.move(dx, dy))
6
7  val u0 = Point(0)
8  val u1 = p1.pts(0)
9  val u2 = p2.pts.apply(100)
10 val u3 = moveAll(ps, 100, 100).drop(1).head
11 val u4 = Point(100, 100).distanceTo()
12 val u5 = {
13   val z = Point()
14   val pts = p1.pts ++ p2.pts
15   if (pts.length > 0) {
16     var p = pts(0)
17     var i = 1
18     while (i < pts.length) {
19       if (pts(i).distanceTo(z) > p.distanceTo(z))
20         p = pts(i)
21       i += 1
22     }
23     p
24   } else z
25 }
```

	Vid kompilerings-fel sätt kryss.	Vid exekve-ringsfel sätt kryss.	Ange statisk **typ** som kompilatorn härleder om ej kompilerings- eller exekveringsfel.	Ange dynamiskt **värde** som tilldelas vid exekvering om ej kompilerings- eller exekveringsfel.
`u0`			`Point`	`Point(0.0,0.0)`
`u1`				
`u2`				
`u3`				
`u4`				
`u5`				

Del B. 40 p

Uppgift B1. Slumpmässiga identitetsnummer. 10p

På en diagnostisk kontrollskrivning i programmering ska studenterna ges unika slumpmässiga identitetsnummer. Dessa nummer ska skrivas ut på lappar som sedan ska delas ut på skrivningsbänkarna. För detta ändamål utvecklas ett program som kan skriva ut `n` stycken unika slumptal i intervallet `min` till och med `max` i slumpvis ordning. Nedan visas huvudprogrammet:

```
object RandomId {
  def main(args: Array[String]): Unit = {
    if (args.length == 4) {
      val seed           = args(0).toInt
      val sequenceLength = args(1).toInt
      val minValue       = args(2).toInt
      val maxValue       = args(3).toInt
      if (maxValue - minValue + 1 < sequenceLength)
        println("Error: The interval is smaller than the sequence length")
      else {
        val randomGenerator = new java.util.Random(seed)
        IdPrinter.print(randomGenerator, sequenceLength, minValue, maxValue)
      }
    } else println("Usage: scala RandomId <seed> <n> <min> <max>")
  }
}
```

Exempelkörning av ovan huvudprogram visas nedan, där första argumentet är slumptalsfrö, andra argumentet anger antalet identitetsnummer som ska genereras, och de två sista argumenten utgör minsta och största möjliga värde:

```
$ scala RandomId 42 7 1001 9999
7753
1809
6290
1325
5642
5338
8626
```

Implementera `IdPrinter`

Du ska implementera singelobjektet `IdPrinter` med metoden `print` enligt nedan specifikation:

Specification **`IdPrinter`**

```
object IdPrinter {
  import java.util.Random

  /** Prints n unique random integers within [min, max] in random order */
  def print(rnd: Random, n: Int, min: Int, max: Int): Unit = ???
}
```

Tips: Använd en mängd för att hålla reda på vilka slumptal som redan är dragna. Skriv ut unika slumptal allteftersom de dras.

Uppgift B2. Rotera array på plats. 10p

När skrivningstiden är slut samlar läraren in skrivningarna och lägger dem i en hög för varje bänkrad, där ordningen i högen motsvarar hur de skrivande sitter i respektive bänkrad. Nu ska skrivningarna delas ut igen inom samma bänkrad för att rättas av studenterna. Skrivningarna ska delas ut på ett sådant sätt att man inte tilldelas sin egen skrivning, samtidigt som den ursprungliga ordningen i högen enkelt ska kunna återställas. För att åstadkomma detta ”roteras” skrivningshögen genom att ett visst, hemligt antal skrivningar tas överst ur högen och placeras underst i högen en efter en. Därefter delas skrivningarna i ut i bänkraden enligt ordningen i den roterade skrivningshögen.

Som stöd för att under denna procedur hålla reda på studenternas unika identitetsnummer efter rotation, har läraren utvecklat Java-programmet nedan. Du ska översätta programmet till Scala och samtidigt förbättra det så att det klarar några extra specialfall enligt nedan instruktioner.

```
1  public class JRotate {
2      public static void rotateArrayOfStrings(String[] xs, int fromPos){
3          String temp = xs[fromPos];
4          for (int i = fromPos; i < xs.length - 1; i++) {
5              xs[i] = xs[i + 1];
6          }
7          xs[xs.length - 1] = temp;
8      }
9
10     public static void main(String[] args){
11         int numberOfSteps = Integer.parseInt(args[0]);
12         for (int i = 1; i <= numberOfSteps; i++){
13             rotateArrayOfStrings(args, 1);
14         }
15         for (int i = 1; i < args.length; i++){
16             System.out.println(args[i] + " ");
17         }
18     }
19 }
```

En exempelkörning av ovan huvudprogram visas nedan, där första argumentet är det hemliga antalet rotationssteg (här 3 st), medan efterföljande argument är de identitetsnummer som ska roteras:

```
1  $ java JRotate 3 7753 1809 6290 1325 5642 5338 8626
2  1325
3  5642
4  5338
5  8626
6  7753
7  1809
8  6290
```

Översätt `JRotate` till Scala och förbättra programmet.

Du ska översätta programmet ovan till ett singelobjekt i Scala med namnet `Rotate`. Du ska även förbättra programmet enligt följande:

- Om `args` har noll eller ett element ska huvudprogrammet skriva ut ett meddelande och sedan avslutas utan att exekveringsfel sker. Meddelandet ska vara:
  `Error: Not enough arguments.`
- Om metoden `rotateArrayOfStrings` anropas med tom array, en array med endast 1 element, eller `fromPos` som är negativ eller för stor, så ska metoden inte göra något (ej heller ge exekveringsfel).

Uppgift B3. Registrering av bonuspoäng. 20p

Studenterna som skriver kontrollskrivningen är indelade i samarbetsgrupper. Poängresultatet från kontrollskrivningen ska fördelas så att varje individ får *medelvärdet* av samarbetsgruppens poäng dividerat med 10, vilket sedan blir bonuspoäng till den ordinarie tentamen.

Du ska implementera klassen `BonusRegister` som beräknar individernas bonuspoäng, enligt specifikation på nästa sida. Nedan visas hur klassen `BonusRegister` används med indata, huvudprogram och utdata. Indata utgörs av en textfil som innehåller en och endast en rad per student, i godtycklig ordning. Varje rad har följande tre kolumner separerade med ett blanktecken, med detta innehåll i respektive kolumn:

- studentens unika identitet, t.ex. `dat16abc`
- gruppen som studenten tillhör, t.ex. `D01a`
- antalet poäng på kontrollskrivningen dividerat med 10 avrundat till närmaste heltal mellan 0 och 5.

Exempel: Indatafilen `points.txt` innehåller följande:

```
eko14def D01b 3
eko14efg D02a 4
eko14abc D01a 1
dat16efg D01a 3
dat16fgh D01b 4
eko14cde D01b 5
dat16def D01a 3
dat16hij D02a 5
dat16ijk D02a 0
dat16cde D01a 2
```

Huvudprogrammet ser ut så här:

```
1  object Bonus {
2    def main(args: Array[String]): Unit = {
3      val lines = scala.io.Source.fromFile(args(0)).getLines.toVector
4      val reg = new BonusRegister
5      for (line <- lines) {
6        val xs = line.split(' ')
7        reg.add(studentId=xs(0), group=xs(1), points=xs(2).toInt)
8      }
9      val bonusMap = reg.calculateBonus
10     val sortedPairs: Vector[(String, Int)] = bonusMap.toVector.sorted
11     sortedPairs.foreach(println)
12   }
13 }
```

Följande utdata ges vid körning av ovan huvudprogram med `points.txt` som indata:

```
1  $ scala Bonus points.txt
2  (dat16cde,2)
3  (dat16def,2)
4  (dat16efg,2)
5  (dat16fgh,4)
6  (dat16hij,3)
7  (dat16ijk,3)
8  (eko14abc,2)
9  (eko14cde,4)
10 (eko14def,4)
11 (eko14efg,3)
```

Implementera `BonusRegister`

Du ska implementera `BonusRegister` enligt denna specifikation:

Specification **`BonusRegister`**

```
/** A mutable register of individual points for bonus calculation */
class BonusRegister {
  /** Add data for one student */
  def add(studentId: String, group: String, points: Int): Unit = ???

  /** Returns an immutable Map with the bonus points of each studentId */
  def calculateBonus(): Map[String, Int] = ???
}
object BonusRegister {
  /** Returns the average bonus calculated from a sequence of integers */
  def collaborationBonus(points: Seq[Int]): Int =
    (points.sum / points.size.toDouble).round.toInt
}
```

Klassen `BonusRegister` ska innehålla följande attribut:

```
import scala.collection.mutable
private val groupOfStudent: mutable.Map[String, String] = mutable.Map.empty
private val pointsOfGroup: mutable.Map[String, Vector[Int]] = mutable.Map.empty
```

Tips:

- Metoden `add` ska lägga till element i tabellerna `groupOfStudent` och `pointsOfGroup`, så att det efter alla tillägg med indata enligt exemplet på föregående sida gäller att:

  `groupOfStudent("dat16cde")` har värdet `"D01a"`, och

  `pointsOfGroup("D01a")` har värdet `Vector(1, 3, 3, 2)`.

- Uttrycket `pointsOfGroup.isDefinedAt(g)` ger **`true`** om tabellen `pointsOfGroup` har ett värde definierat för nyckeln `g`, annars **`false`**.

- Uttrycket `groupOfStudent.keySet` ger en mängd med alla nycklar i tabellen; här alla studenter.

- Om du anropar metoden `toMap` på en sekvens av par, får du som resultat en oföränderlig tabell med motsvarande nyckel-värde-par.