Kontrollskrivning
EDAA45 Programmering, grundkurs

2023-10-25, 14:00-19:00

Hjälpmedel: Snabbreferens för Scala & Java.

Instruktioner

Du får en lapp med en **identitetskod** som du ska skriva här: _____
Identitetskoden ska även skrivas på omslaget och på alla inlämnade blad. Ha legitimation redo.
Kontrollskrivningen är indelad i tre moment:

- *Moment 1*, ca 2 h 15 min: **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. Skriv endast på ena sidan av varje blad. Du får inte skriva med rödfärgad penna. Du ska inte lämna in kladdpapper eller anteckningar som inte ingår i dina svar. Du ska skriva din identitetskod 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, vänta under tystnad tills den individuella delen är över. Du får inte lämna lokalen. När tiden är ute samlas skrivningarna in.
- *Moment 2*, ca 1 h 15 min: **Kamratbedömning**. Du får utdelat en bedömningsmall som du läser igenom noga. Efter ett tag får du en annan persons skrivning som du poängsätter enligt anvisningarna i bedömningsmallen. Du blir också tilldelad en eller två andra studenter som diskussionspartner och ni ska hjälpa varandra att göra så bra bedömningar som möjligt. När tiden är slut samlas alla skrivningarna in.
- *Moment 3*, ca 0,5 h: **Kontrollera bedömningen**. 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.
- 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. Tolka uttryck. Totalt max 10 p.

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

```
class FlipFlop private (init: Boolean = true):
  private var flipFlop: Boolean = init

  def apply(n: Int): Unit = for _ <- 1 to n do flipFlop = !flipFlop

  def toggle(): FlipFlop = { apply(1); this }

  def isFlip: Boolean = flipFlop

  def isFlop: Boolean = !flipFlop

  override def toString: String = if flipFlop then "flip" else "flop"

object FlipFlop:
  val flip = new FlipFlop()

  val flop = new FlipFlop(false)

  val xs = Vector(flip, flop, flip, flop)

  def apply(n: Int): FlipFlop =
    require(n > 1)
    val result = new FlipFlop(false)
    result.apply(n)
    result

  def apply(s: String): FlipFlop = apply(s.length)
```

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.

```
import FlipFlop.{flip, flop, xs}
import scala.util.Try

val u0 = 41.0 + 1
val u1 = FlipFlop(0)
val u2 = Try(if FlipFlop.flip.flipFlop then "gurka" else "tomat").toOption
val u3 = { xs(2).toggle(); xs.last.toggle(); xs.head.isFlop }
val u4 = { flop.toggle(); xs.map(_.toString).drop(1).take(1).map(_.length) }
val u5 = Vector(FlipFlop(4), FlipFlop(7)).zipWithIndex.map(_._2)
```

	Vid kompilerings-fel sätt kryss.	Vid exekveringsfel sätt kryss.	Ange statisk **typ** som kompilatorn härleder om ej kompilerings- eller exekveringsfel.	Ange det **värde** som tilldelas vid exekvering, med samma format som vid utskrift av värdets `toString`, om ej kompilerings- eller exekveringsfel.
`u0`			`Double`	`42.0`
`u1`				
`u2`				
`u3`				
`u4`				
`u5`				

Del B. Skriva kod. Totalt max 40 p.

Du ska skriva ett program som delar in studenter i kompetensblandade grupper baserat på förkunskaper. I uppgift B1 implementerar du en klass som du ska använda i uppgift B2.

Uppgift B1. Moduloräknare. Totalt max 15 p.

Implementera klassen `ModuloCounter` enligt nedan krav:

1. Klassen ska ha en heltalsparameter vid namn `modulo`, som även är ett synligt, oföränderligt attribut.
2. Om klassparametern ej är större än noll så ska ett undantag kastas med felmeddelande enligt nedan exempel. *Tips:* Du har nytta av metoden `require`.
3. Klassen ska ha ett privat, förändringsbart heltalsattribut vid namn `counter`, med initialvärdet `0`.
4. Klassen ska ha en metod `next` med tom parameterlista och explicit angiven returtyp, som ska returnera nuvarande värde av `counter` och även öka `counter` med ett modulo `modulo`. Detta innebär att första anropet ska ge 0 och därefter ska varje nytt anrop ge efterföljande heltal upp till och med `modulo - 1`, varefter resultatet av nästa anrop börjar om på `0`, och så vidare.

Exempel på användning:

```
scala> val mc = ModuloCounter(4)
val mc: ModuloCounter = ModuloCounter@3b7b0b57

scala> mc.next()
val res0: Int = 0

scala> mc.next()
val res1: Int = 1

scala> Vector.fill(19)(mc.next())
val res2: Vector[Int] = Vector(2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0)

scala> mc.modulo
val res3: Int = 4

scala> ModuloCounter(-1)
java.lang.IllegalArgumentException: requirement failed: modulo=-1, must be > 0
```

Uppgift B2. Registrering och gruppindelning. Totalt max 25 p.

På sidan 5 visas den givna koden som du ska utgå ifrån. Du ska färdigställa de saknade delarna markerade med `???`, vars maximala poängvärde ges i kommentar. Den givna koden förklaras nedan. Du ska följa de krav och tips som ges på sidan 6.

Förklaring av given kod

Den givna koden innehåller följande färdigimplementerade abstraktioner:

- Den uppräknade typen `Experience` representerar olika nivå av förkunskaper hos studenter.
- Typaliaset `Lines` är ett kortare namn på typen av en strängvektor.
- Singelobjektet `Lines` innehåller två metoder för hantering av textfiler:
  - metoden `loadLines` för läsning av rader från en fil med sökvägen `path`
  - extensionsmetoden `saveToFile` sparar textrader till en fil med sökvägen `path`
- Klassen `Student` representerar en student med ett namn, en unik identitet och en förkunskapsnivå.
- I kompanjonsobjektet `Student` finns fabriksmetoden `fromString` som kan skapa en student utifrån en textrad med attribut som är separerade med `delim`.
- Typaliaset `Students` är ett kortare namn på typen av en studentvektor.

- Huvudprogrammet använder ovan abstraktioner för att läsa in en kommaseparerad textfil med data om studenter, skriva ut statistik om studenternas förkunskaper, samt skapa en gruppindelning som ger grupper med blandade förkunskaper och sparar gruppindelningen i en kommaseparerad textfil.

Given kod

Du ska implementera delarna markerade med ???, enligt krav och tips på sidan 6.

```scala
extension (d: Double) def toRoundedPercent: Int = ??? //3p

enum Experience { case None, Low, Medium, High }

type Lines = Vector[String]

object Lines:
  def loadLines(path: String): Lines =
    val source = scala.io.Source.fromFile(path)
    try source.getLines().toVector finally source.close()

  extension (lines: Lines) def saveToFile(path: String): Unit =
      val pw = java.io.PrintWriter(java.io.File(path), "UTF-8")
      try pw.write(lines.mkString("","\n","\n")) finally pw.close()

case class Student(name: (String, String), id: String, pre: Experience)

object Student:
  def fromString(s: String, delim: Char = ','): Student =
    val parts = s.split(delim)
    require(parts.length == 4, s"must be exactly 4 elems separated by $delim\n$s")
    Student(parts(0) -> parts(1), parts(2), Experience.fromOrdinal(parts(3).toInt))

type Students = Vector[Student]

object Students:
  def fromLines(lines: Lines, dropHeading: Boolean = true): Students = ??? //6p

  def registerExperience(students: Students): Vector[(Experience, Int)] = ??? //7p

  def allocate(nbrGroups: Int, students: Students): Vector[(Int, Student)] = ??? //9p

@main def createGroups(inPath: String, outPath: String, groupSize: Int) =
  import Lines.*, Students.*
  val students = fromLines(loadLines(inPath))
  println(s"Number of students in $inPath: ${students.length}")
  println("Experience:")
  for (x, n) <- registerExperience(students) do
    val percent = (n.toDouble / students.length).toRoundedPercent
    println(s"$x:${" " * (8 -x.toString.length)}$n, $percent%")
  val output =
    for (g, s) <- allocate(groupSize, students)
    yield s"$g,${s.name._1},${s.name._2},${s.id},${s.pre}"
  output.saveToFile(outPath)
  println(s"Groups saved to $outPath")
```

Krav och tips

Exempelkörning med testdata:

```
> cat students.csv
förnamn,efternamn,id,förkunskaper
Oddput,Clementin,od3928cl-s,2
Gusten,Grodslukare,gu7612gr-s,0
Napp,Flaska,na4636fl-s,1
Samma,Namn,sa2182na-s,0
Kim,Kardasjian,ki8971ka-s,1
Samma,Namn,sa2183na-s,0
Kringla,Två Efternamn,kr8456tv-s,1
Anna,Panna,an1324pa-s,3
Sven,Banansson,sv9876ba-s,1
Gurka,Tomatsson,gu9483to-s,2
Päron,Förälder,pä4321fö-s,3
> scala-cli run . -- students.csv groups.csv 3
Number of students in students.csv: 11
Experience statistics:
None:    3, 27%
Low:     4, 36%
Medium:  2, 18%
High:    2, 18%
Groups saved to groups.csv
> cat groups.csv
grupp,förnamn,efternamn,id,förkunskaper
1,Gusten,Grodslukare,gu7612gr-s,0
1,Napp,Flaska,na4636fl-s,1
1,Sven,Banansson,sv9876ba-s,1
1,Anna,Panna,an1324pa-s,3
2,Samma,Namn,sa2182na-s,0
2,Kim,Kardasjian,ki8971ka-s,1
2,Oddput,Clementin,od3928cl-s,2
2,Päron,Förälder,pä4321fö-s,3
3,Samma,Namn,sa2183na-s,0
3,Kringla,Två Efternamn,kr8456tv-s,1
3,Gurka,Tomatsson,gu9483to-s,2
```

Programmet på sidan 5 ska fungera enligt denna exempelkörning. Koden ligger i aktuell katalog tillsammans med indatafilen `students.csv`. Efter prompten > körs `cat` för att visa filer och `scala-cli` för att köra huvudprogrammet `createGroups`. Efter `--` ges programargument. De rader som inte börjar med > är utskrifter.

Du ska beakta följande krav och tips i dina implementationer av de saknade delarna:

1. Metoden `toRoundedPercent` ska omvandla decimaltal till avrundade heltalsprocent. Exempel:

   ```
   scala> (2 / 3.0).toRoundedPercent
   val res0: Int = 67
   ```

2. Metoden `fromLines` ska skapa en studentvektor ur en vektor med strängar. Strängvektorn innehåller rader med flera attributvärden avgränsade med `delim`, enligt testdata i ovan exempelkörning. Om `dropHeading` är sann så ska första strängen hoppas över, annars ska den tas med. *Tips:* Du har nytta av metoderna `drop` och `Student.fromString`.
3. Metoden `registerExperience` ska registrera antalet förekomster av olika förkunskapsnivåer. *Tips:* Du har nytta av `Experience.values.toVector` och samlingsmetoden `count`.
4. Metoden `allocate` ska ge en vektor med par av gruppnummer och student. Gruppnummer ska vara från 1 till och med `nbrGroups`. Du kan anta att `nbrGroups` är större än 1. *Lösningsidé:* Använd en instans av `ModuleCounter` från tidigare uppgift. Iterera över en studentvektor som sorterats i förkunskapsordning och skapa en vektor med par med gruppnummer och student. Sortera resultatet i gruppnummerordning. *Tips:* Du har nytta av metoden `ordinal` på enum-värden och samlingsmetoden `sortBy`.