Tentamen
EDAA45 Programmering, grundkurs

2021-01-11, 14:00-19:00

- *Tillåtet hjälpmedel:* **Snabbreferens för Scala & Java**. Inga andra hjälpmedel är tillåtna. Du får alltså inte använda REPL, kompilator, editor eller andra programmeringsverktyg. Du får inte ta hjälp av internet eller någon annan människa. Din inlämning innebär att du försäkrar på heder och samvete att du ensam har skapat dina lösningar och att du inte använt otillåtna hjälpmedel eller på annat sätt har fuskat.
- Tentan innehåller 3 uppgifter med svar i form av programkod som du ska skriva på papper. Börja på nytt blad för varje uppgift.
- Uppgift 1 och 2 innehåller deluppgifter som utgörs av saknade implementationer i given kod markerade med `???`. I uppgift 3 ska du implementera en komplett klass som uppfyller den givna specifikationen.
- Det ska tydligt framgå vilken deluppgift du löser. Börja därför för varje deluppgift med deklarationen som du implementerar, t.ex. metodhuvudet eller attributdeklarationen. Du ska *inte* lämna in avskrifter av övriga färdigimplementerade delar i given kod.
- Dina papper med lösningar ska scannas, t.ex. via mobil med med Adobe Scan, och lämnas in som en pdf-fil enligt instruktioner i Canvas.
- Tentan rättas anonymt i Canvas. Du ska därför *inte* skriva ditt namn på inlämnade blad.

Preliminär poängfördelning

- Maximalt ges 80p, varav 40p krävs för godkänt. För betyg 4:a krävs minst 54p och för 5:a krävs minst 67p.
- De poäng och delpoäng som anges ovan och i uppgifterna är preliminära och kan komma att justeras när den slutliga, samlade bedömningen fastställs.

Upplysningar

- För att vara tentamensberättigad ska du vara godkänd på alla obligatoriska laborationer och projekt, samt ha genomfört diagnostisk kontrollskrivning.
- Om du tenterar utan att vara tentamensberättigad annulleras din skrivning. För att undvika att någon skrivning annulleras av misstag kommer alla som, enligt institutionens noteringar, tenterat utan att vara tentamensberättigade att kontaktas via epost. Felaktigheter i institutionens noteringar kan därefter påtalas fram till nästa tentamenstillfälle då resterande skrivningar annulleras.
- Lösningar läggs ut på kursens hemsida efter tentamensdagen.
- Resultatet läggs in i Ladok när bedömningen av alla tentamina är klar.

Uppgift 1: Matris, totalt 22p. (`get`, 10p; `update`, 12p)

Klassen `Crossword` hanterar en teckenmatris och användas för att konstruera korsord så här:

```
scala> val c = new Crossword(rows = 4, cols = 10)
val c: Crossword =
************
*          *
*          *
*          *
*          *
************
scala> c.update(1,5,"gurka", Horizontal)  // andra raden, sjätte kolumnen
val res0: Boolean = true
scala> val u = (c.update(1,4,"tomat", Horizontal), c.update(0,9,"kaka", Vertical))
val u: (Boolean, Boolean) = (false,true)
scala> println(c)
************
*         k*
*     gurka*
*         k*
*         a*
************
```

Implementera de metoder nedan som har `???`. Metoden `get` ska ge en sträng med med $*$ på de positioner som eventuellt hamnar utanför matrisens indexgränser via användning av `charAt`, som tar hand om all indexgränskontroll. Metoden `update` ska endast göra uppdatering om `s` passar enligt `canFit`.

```scala
sealed trait Direction
case object Horizontal extends Direction  // vågrät riktning
case object Vertical   extends Direction  // lodrät riktning

class Crossword(val rows: Int, val cols: Int){
  private val xss = Array.fill(rows, cols)(' ')

  def charAt(row: Int, col: Int): Char =
    if (row >=0 && col >= 0 && row < rows && col < cols) xss(row)(col) else '*'

  /** Ger strängen på plats row, col med längd length i riktning dir. */
  def get(row: Int, col: Int, length: Int, dir: Direction): String = ???

  def canFit(row: Int, col: Int, s: String, dir: Direction): Boolean = {
      val current = get(row, col, s.length, dir)
      current.indices.forall(i => current(i) == ' ' || current(i) == s(i))
  }

  /** Ändrar strängen på plats row, col i riktning dir till s om den passar.
    * Returnerar true om updatering gjordes, annars false. */
  def update(row: Int, col: Int, s: String, dir: Direction): Boolean = ???

  override def toString = {
    def rowString(r: Int) = (-1 to cols).map(charAt(r, _)).mkString + "\n"
    (-1 to rows).map(rowString).mkString
  }
}
```

Uppgift 2: Registrering, totalt 34p. (`fromString`, 4p; `updated`, 4p; `statusOf`, 3p; `studentIds`, 3p; `labIds`, 2p; `add`, 12p; `labStatusCount`, 6p)

Du ska implementera metoderna med `???` i `labreg` nedan som hanterar registrering av status på laborationer. Användningen av `labreg` definieras av testprogrammet på nästa sida. Använd inbyggd sortering.

```scala
object labreg {
  type LabId = String
  type StudentId = String

  sealed trait Status
  case object Approved  extends Status // godkänd
  case object Residual  extends Status // kompletteras
  case object Postponed extends Status // uppskov
  case object Missing   extends Status // saknad eller okänd status

  object Status {
    val values = Vector[Status](Approved, Residual, Postponed, Missing)

    /** Om s är "godkänd","kompletteras","uppskov" returneras motsvarande status:
      * Approved, Residual, Postponed. I alla andra fall returneras Missing. */
    def fromString(s: String): Status = ???
  }

  case class Student(id: StudentId, labStatus: Map[LabId, Status]){
    /** Returnerar en ny Student med uppdaterad labStatus */
    def updated(lab: LabId, status: Status): Student = ???

    /** Returnerar status för laboration lab. Ger Missing om status saknas. */
    def statusOf(lab: LabId): Status = ???
  }

  class LabReg {
    import scala.collection.mutable
    private val reg = mutable.Map[StudentId, Student]()
    private val labs = mutable.Set[LabId]()

    def apply(sid: StudentId): Student = reg(sid)

    /** En sorterad vektor med alla studentidentiteter. */
    def studentIds: Vector[StudentId] = ???

    /** En sorterad vektor med alla laborationsidentiteter. */
    def labIds: Vector[LabId] = ???

    /** Uppdaterar status för en viss laboration för en viss student.
      * Om sid saknas så registreras ny Student med lab -> status.*/
    def add(sid: StudentId, lab: LabId, status: Status): Unit = ???

    /** Räknar för en viss laboration totala antalet förekomster av status. */
    def labStatusCount(lab: LabId, status: Status): Int = ???
  }
}
```

I filen `test-labdata.csv` nedan finns kommaseparerad laborationsdata där varje rad innehåller en statusuppdatering med studentidentitet (unik per student), laborationsidentitet (unik per lab) och status:

```
ab1234-s,w01-kojo,uppskov
cd5678-s,w01-kojo,kompletteras
ef9012-s,w01-kojo,???
ab1234-s,w03-irritext,godkänd
ab1234-s,w01-kojo,godkänd
cd5678-s,w03-irritext,kompletteras
ef9012-s,w03-irritext,godkänd
gh3425-s,w03-irritext,uppskov
ab1234-s,w04-blockmole,godkänd
```

Följande huvudprogram används för att testa singelobjektet `labreg`:

```scala
object TestLabReg {
  import labreg._

  def printReport(labReg: LabReg): Unit = {
    val (labs, students) = (labReg.labIds, labReg.studentIds)
    println(s"--- NUMBER OF LABS: ${labs.size}")
    for (lab <- labs) {
      val row = Status.values.map(stat =>
        s"$stat:${labReg.labStatusCount(lab, stat)}"
      )
      println(row.mkString(s"$lab: ", " ", ""))
    }
    println("--- NUMBER OF STUDENTS: ${students.size}")
    for (stud <- students) {
      val row = labs.map(lab => s"$lab:${labReg(stud).statusOf(lab)}")
      println(row.mkString(s"$stud: ", " ", ""))
    }
  }

  def main(args: Array[String]): Unit = {
    val labReg = new LabReg
    val lines = scala.io.Source.fromFile(args(0)).getLines()
    lines.foreach{ line =>
      val row = line.split(',').toVector
      labReg.add(sid = row(0), lab = row(1), status = Status.fromString(row(2)))
    }
    printReport(labReg)
  }
}
```

Vid exekvering av `TestLabReg` med `test-labdata.csv` som argument ska följande utskrift ges:

```
--- NUMBER OF LABS: 3
w01-kojo: Approved:1 Residual:1 Postponed:0 Missing:2
w03-irritext: Approved:2 Residual:1 Postponed:1 Missing:0
w04-blockmole: Approved:1 Residual:0 Postponed:0 Missing:3
--- NUMBER OF STUDENTS: 4
ab1234-s: w01-kojo:Approved w03-irritext:Approved w04-blockmole:Approved
cd5678-s: w01-kojo:Residual w03-irritext:Residual w04-blockmole:Missing
ef9012-s: w01-kojo:Missing w03-irritext:Approved w04-blockmole:Missing
gh3425-s: w01-kojo:Missing w03-irritext:Postponed w04-blockmole:Missing
```

Uppgift 3. Sortering i Java. 24p

Du ska implementera klassen `HighScoreList`, som hanterar en lista med spelresultat, enligt specifikationen på nästa sida. I Java-programmet `HighScoreListTest` nedan testas klassen `HighScoreList`. Klassen `GameResult` representerar poängresultatet för en spelare vid en viss spelomgång.

```
import java.util.ArrayList;

public class HighScoreListTest {
    public static void main(String[] args){
        HighScoreList hsl = new HighScoreList();
        hsl.insert(new GameResult("Sandra", 4242));
        hsl.insert(new GameResult("Anna",   4242));
        hsl.insert(new GameResult("Björn",  1042));
        hsl.insert(new GameResult("Patrik", 2242));
        hsl.insert(new GameResult("Roger",    2242));
        ArrayList<GameResult> tl = hsl.topList(10);
        for (GameResult gr : tl) {
            System.out.println(gr);
        }
    }
}
```

Klassen `GameResult` är implementerad enligt nedan:

```
public class GameResult {
  private String name;
  private int score;

  public GameResult(String name, int score){
    this.name = name;
    this.score = score;
  }

  public String getName(){
    return name;
  }

  public int getScore(){
    return score;
  }

  public String toString(){
    return name + ": " + score;
  }
}
```

Då huvudprogrammet i `HighScoreListTest` körs ska följande utskrift ges:

```
Anna: 4242
Sandra: 4242
Patrik: 2242
Roger: 2242
Björn: 1042
```

Implementera klassen `HighScoreList` i Java enligt nedan specifikation.

class HighScoreList

```
/** Konstruktor som skapar en tom lista med spelresultat. */
HighScoreList();

/** Ger true om gr1 ordnas före gr2, annars false.
    Ordningen sker med avseende på högst poäng i första hand och
    bokstavsordning på namn i andra hand. */
static boolean isBefore(GameResult gr1, GameResult gr2);

/** Sorterar in ett spelresultat gr på rätt plats i listan. */
void insert(GameResult gr);

/** Returnerar en sorterad lista med de n stycken bästa spelresultaten.
    Om listan är kortare än n ges en sorterad lista med alla spelresultat. */
ArrayList<GameResult> topList(int n);
```

Krav och tips:

- Du ska implementera valfri sorteringsalgoritm från grunden och alltså inte använda inbyggd sortering.
- Använd en ArrayList för att lagra elementen i listan (se snabbreferens för Java).
- Din sorteringsalgoritm ska använda sig av `isBefore`.
- Bokstavsordning av strängar ges av jämförelse med `compareTo` (se snabbreferens för Java).
- Det är lättast att hålla hela listan sorterad genom att sätta in nya spelresultat på rätt plats i `insert`, men det går även bra att åstadkomma en korrekt sortering av topplistan på andra sätt.
- Samma person ska kunna förekomma flera gånger i resultatlistan, även med en och samma poäng. Du behöver alltså inte särskilt beakta detta fall.