Valfri tentamen
EDAA45 Programmering, grundkurs

2023-01-07, 08:00-13:00

Hjälpmedel: Snabbreferens för Scala & Java.

Instruktioner

- Skriv din anonymkod + personlig identifierare här: _____
  Om du skriver icke-anonymt ange personnummer + namn i stället.
- Tillåtet hjälpmedel: Snabbreferens för Scala & Java.
- **Del A** (uppgift 1) ska besvaras genom att fylla i en tabell i *detta häfte*.
- **Del B** innehåller uppgift 2, 3, ... med svar i form av programkod som du ska skriva på *separata vita papper*. Skriv bara på *ena* sidan av varje inlämnat blad. Skriv din anonymkod + personlig identifierare (eller personnummer + namn om du skriver icke-anonymt) *överst* på *varje* inlämnat blad. Det ska tydligt framgå vilken (del)uppgift du löser.
- Detta häftet ska lämnas in tillsammans med ifyllt omslag och svaren på uppgifterna i del B.
- Innan du går måste du lämna in detta häfte och ifyllt omslag, även om du inte löst några uppgifter. Du får inte lämna salen innan du lämnat in. Du får tidigast lämna in 1 timme efter start.

Upplysningar

- Tentamen är valfri och kan, om du är tentamensberättigad, resultera i överbetygen 4 eller 5 om du uppfyller kraven för överbetyg, enligt nedan.
- För att vara tentamensberättigad ska du före tentamens start vara godkänd på alla obligatoriska moment (laborationer, projekt och muntligt prov), samt ha genomfört diagnostisk kontrollskrivning. Om du uppfyller dessa krav får du minst betyg 3 oavsett resultat på denna tentamen.
- Tentamen kan maximalt ge 100p, varav del A omfattar 20p och del B omfattar 80p.
- Preliminära krav för överbetyg:
  - För överbetyg krävs minst 10p på del A, samt totalt minst 67p för betyg 4 och 83p för betyg 5.
  - Om du erhåller färre än 10p på del A så bedöms inte del B och överbetyg medges ej.
  - Poäng och delpoäng som anges ovan och i uppgifterna är preliminära och kan komma att justeras när den slutliga bedömningen fastställs.
- Du ska *inte* tentera om du ej är tentamensberättigad. Om du ändå tenterar utan att vara tentamensberättigad annulleras din skrivning utan att bedömas. 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 inom en månad.
- Lösningar läggs ut på kursens hemsida efter tentamen.
- Resultatet läggs in i Ladok när bedömningen är klar.

Del A. Tolka uttryck. Totalt max 20 p.

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

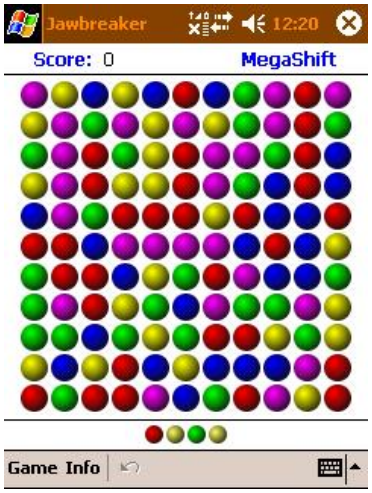
```
val area = Vector("SE4", "SE3", "SE2", "SE1")

abstract class Price(val min: Double, val max: Double = 16.0)
case object LowStable        extends Price(0.2, 0.5)
case object Low              extends Price(0.7, 1.5)
case object Medium           extends Price(1.0, 2.0)
case object High             extends Price(2.0, 4.5)
case object Ultra            extends Price(4.5)

val priceCustomerDay = Map[String,Map[Int, Price]]("Lund A" ->
  Map(1 -> LowStable, 15 -> Medium, 16 -> High, 18 -> Ultra),
  "Kiruna A" -> Map(4 -> LowStable, 15 -> Medium, 16 -> Medium, 24 -> Low)
)

def test(a: Boolean): String = if (!a) a.toString.reverse else a.toString
```

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.

```
val u0 = 40 + 2.0

val u1 = area(1)
val u2 = area.sortBy(_.map(_ + 1).sum).head
val u3 = area.reverse.filter(_ contains "SE").headOption
val u4 = (for (i <- area.indices) yield area(i - 1)).apply(0)

val u5 = u2(2).toString.toInt match {
  case i if i <= Medium.max => Medium.min
  case i if i <= High.max => High.min
  case i if i <= Ultra.max => Ultra.min
  case _ => LowStable.min
}

val u6 = (new Price(4.2, 8.4)).min
val u7 = priceCustomerDay("Kiruna A")(21 % 10).min < 0.8
val u8 = priceCustomerDay("Lund A")(15).max > 2.0
val u9 = scala.util.Try{ area(area(1).length) }.getOrElse(42.toString)
val u10 = test("false")
```

	Vid kompi-leringsfel sätt kryss.	Vid exekve-ringsfel 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`				
`u6`				
`u7`				
`u8`				
`u9`				
`u10`				

Del B. Spelet ”Bubblor”. Totalt max 80 p.



I denna uppgift ska du implementera en variant av det klassiska spelet **SameGame** som ursprungligen släpptes under namnet (eng. *Chain Shot!*) 1985. Spelet består av ett bräde med olikfärgade bubblor. I varje drag väljer spelaren en ö, intilliggande bubblor av samma färg. Ön tas bort och man får poäng baserat på öns storlek. Efter att ön tagits bort faller bubblor ner och fyller igen tomrummet. Kolumner skiftas även åt höger för att fylla igen helt tomma kolumner. Tomma rader och kolumner fylls sedan på med nytt slumpmässigt innehåll. Detta beskrivs i detalj på nästa sida. Spelet är slut när alla öar har storleken 1. SameGame portades till en mängd olika plattformar varav en variant hette **Jawbreaker** och var implementerad för PocketPC. Alla varianter har olika regler så vi kommer definera vår variant i denna tentan.

Varje ruta på spelbrädet har 4 grannar, höger, vänster, upp och ner. En ö består av alla intilliggande bubblor med samma färg, dvs du kan gå runt på en ö utan att behöva passera en bubbla med annan färg. Öar har alltid maximal storlek, dvs två grannar med samma färg ingår alltid i samma ö, se tips och krav 16 för en förklaring på hur du räknar ut vilka bubblor som ingår i en ö. När en ö med n bubblor tas bort får du $n \cdot (n-1)$ poäng, se tips och krav 3.

I exemplet består brädet av 10 x 10 rutor, men din koden ska kunna hantera brädor av godtycklig storlek. Rader benämns med bokstäver, `A`–`J` i exemplen, och kolumner benämns med siffror, `0`–`9` i exemplen. Innan spelet börjar fylls alla rutor med en slumpmässigt färglagd bubbla (blå, grön, röd och orange). Alla rutor utanför brädet betraktas som tomma när du skapar öar och räknar ut poäng, dvs till vänster om kolumn 0 finns en kolumn med tomma rutor.

```
@main def run = BubbleGame.play()
```

Ovan huvudprogram körs nedan.

```
Score: 0
  0 1 2 3 4 5 6 7 8 9
A b b g b b b o b b o
B r b r g o b b o o b
C g o o b r o g o b r
D o r b o g r b r r o
E g o r g r b o b r r
F g b r o b o g o b b
G b g o g b g r b g r
H r b b g b o b b o g
I g b g r g o o b r g
J b r o r b g b o o b

Burst bubbles at (e.g A4): A0
Score: 6
  0 1 2 3 4 5 6 7 8 9
A     g b b b o b b o
B r   r g o b b o o b
C g o o b r o g o b r
D o r b o g r b r r o
E g o r g r b o b r r
F g b r o b o g o b b
G b g o g b g r b g r
H r b b g b o b b o g
I g b g r g o o b r g
J b r o r b g b o o b

Burst bubbles at (e.g A4): G4
Score: 12
  0 1 2 3 4 5 6 7 8 9
A     g b   b o b b o
B r   r g   b b o o b
C g o o b   o g o b r
D o r b o b r b r r o
E g o r g o b o b r r
F g b r o r o g o b b
G b g o g g g r b g r
H r b b g r o b b o g
I g b g r g o o b r g
J b r o r b g b o o b

Burst bubbles at (e.g A4): I3
Score: 14
  0 1 2 3 4 5 6 7 8 9
A     g     b o b b o
B r   r     b b o o b
C g o o b   o g o b r
D o r b g b r b r r o
E g o r b o b o b r r
F g b r o r o g o b b
G b g o g g g r b g r
H r b b o r o b b o g
I g b g g g o o b r g
J b r o g b g b o o b

Burst bubbles at (e.g A4): I3
Score: 26
  0 1 2 3 4 5 6 7 8 9
A           b o b b o
B r   g     b b o o b
C g o r     o g o b r
D o r o     r b r r o
E g o b b b b o b r r
F g b r g o o g o b b
G b g r b r g r b g r
H r b o o g o b b o g
I g b b g r o o b r g
J b r o o b g b o o b
```

Brädeoperationer

Efter att en ö tagits bort faller bubblorna neråt (`shiftDown`). Det kan då hända att det bildas helt tomma rader i toppen av brädet. När detta sker så fylls hela översta rader med nya bubblor med slumpmässig färg som sedan de faller ner. Detta upprepas tills tills det finns åtminstone en kolumn som fyller hela vägen upp.

```
  0 1 2 3 4 5 6 7 8 9
A g o o g o b b b b b
B r b r b g r o r g g
C g b b r g o b b o r
D r b b b r r r b r b
E b b g o g b o b r r
F o g o b r o g g r o
G b b b r o o o b o o
H g o o r o r r g o b
I g o r g o g g b b b
J r o r b b g b o o g

Score: 0, Burst bubbles at (e.g A4): G5

// alla bubblor i ön där G5 ingår tas bort

  0 1 2 3 4 5 6 7 8 9
A g o o g o b b b b b
B r b r b g r o r g g
C g b b r g o b b o r
D r b b b r r r b r b
E b b g o g b o b r r
F o g o b r . g g r o
G b b b r . . . b o o
H g o o r . r r g o b
I g o r g . g g b b b
J r o r b b g b o o g

// Därefter faller ovanstående
// nedåt för att täcka hålet.

  0 1 2 3 4 5 6 7 8 9
A g o o g       b b b
B r b r b     b r g g
C g b b r   b o b o r
D r b b b o r b b r b
E b b g o g o r b r r
F o g o b g r o g r o
G b b b r r b g b o o
H g o o r g r r g o b
I g o r g r g g b b b
J r o r b b g b o o g
```

Kolumner av bubblor faller åt höger (`compactColumnsRight`). *Exempel:* om kolumn 5 blir tom flyttas kolumn 1-4 till 2-5. Efter det finns en eller flera tomma kolumner till vänster. Dessa fylls med nya bubblor. Till skillnad från rader fylls inte alltid hela kolumnen med nya bubblor, utan både antalet bubblor och dess färger slumpas fram. De staplas alltid i botten på kolumnen.

```
Score: 96, Burst bubbles at (e.g A4): J1

  0 1 2 3 4 5 6 7 8 9         0 1 2 3 4 5 6 7 8 9
A               b b b       A               b b b
B             b r g g       B             b r g g
C       g   b o b o r       C       g   b o b o r
D g     b o r b b r b       D g     b o r b b r b
E r     r g o r b r r       E r     r g o r b r r
F g     o g r o g r o  -->  F g     o g r o g r o
G r   o b r b g b o o       G r   o b r b g b o o
H o   r r g r r g o b       H o   r r g r r g o b
I b   g r r g g b b b       I b   g r r g g b b b
J r o o g r g b o o g       J r . . g r g b o o g


// shiftDown (alla bubblor i kolumn 2 föll ner)

  0 1 2 3 4 5 6 7 8 9
A   .           b b b
B   .         b r g g
C   .   g   b o b o r     . = tom ruta
D g .   b o r b b r b
E r .   r g o r b r r
F g .   o g r o g r o
G r .   b r b g b o o
H o . o r g r r g o b
I b . r r r g g b b b
J r . g g r g b o o g

// compactColumnsRight      // ny slumpmässig kolumn
                            // längst till vänster
                            // antalet bubblor varierar

  0 1 2 3 4 5 6 7 8 9         0 1 2 3 4 5 6 7 8 9
A               b b b       A               b b b
B             b r g g       B r           b r g g
C       g   b o b o r       C g     g   b o b o r
D   g   b o r b b r b       D r g   b o r b b r b
E   r   r g o r b r r       E b r   r g o r b r r
F   g   o g r o g r o  -->  F o g   o g r o g r o
G   r   b r b g b o o       G b r   b r b g b o o
H   o o r g r r g o b       H g o o r g r r g o b
I   b r r r g g b b b       I g b r r r g g b b b
J   r g g r g b o o g       J r r g g r g b o o g
```

Given kod

Nedan visas de delvis färdiga klasserna `Bubble`, `BubbleBoard`, `BubbleIsland`, `Pos` samt singelobjektet `BubbleGame`. Du ska färdigställa de saknade delarna markerade med `???`, vars maximala poängvärde ges i kommentar. Abstraktionerna i modellen förklaras övergripande på sidan 9. Du ska följa de krav och tips som ges på sidan 9.

```scala
import scala.util.Random

enum Bubble(symbol: String):
    case Empty extends Bubble(" ")
    case Red extends Bubble("r")
    case Blue extends Bubble("b")
    case Green extends Bubble("g")
    case Orange extends Bubble("o")

    override def toString(): String = this.symbol

object Bubble: // Uppgift 2, Totalt: 4p
    def length = values.length
    def random() = random(Random)
    def random(rand: Random): Bubble = ??? // 4p
```

```scala
case class Pos(x: Int, y: Int) extends Ordered[Pos]: // Uppgift 3, Totalt: 4p
    def compare(that: Pos): Int = ??? // 4p

    override def toString = s"${(y + 'A').toChar}$x"
```

```scala
case class BubbleIsland(bubbles: Vector[Pos]):  // Uppgift 4, Totalt: 1p
    require(bubbles.length > 1
            && bubbles.zip(bubbles.tail).forall((a,b) => a.compare(b) <= 0))

    def score = ???  // 1p
```

```scala
import scala.util.Random

object BubbleBoard: // Uppgift 5, Totalt: 4p
    val Size = 10

    def neighbors(pos: Pos): Vector[Pos] =
        Vector(
            Pos(pos.x-1, pos.y),
            Pos(pos.x+1, pos.y),
            Pos(pos.x, pos.y-1),
            Pos(pos.x, pos.y+1)
        )

    def create(): BubbleBoard = new BubbleBoard(Size, Size)

    def randomBubbles(n: Int, rand: Random): Vector[Bubble] = ??? // 4p


class BubbleBoard(width: Int, height: Int): // Uppgift 6, Totalt: 67p
    import BubbleBoard._
    import scala.collection.mutable
```

```scala

    private var random = new scala.util.Random()
    private var board: Vector[Vector[Bubble]] = ??? // 3p

    def apply(pos: Pos): Bubble = apply(pos.x, pos.y)

    def apply(x: Int, y: Int): Bubble = ??? // 4p

    def set(pos: Pos, bubble: Bubble): Unit = set(pos.x, pos.y, bubble)

    def set(x: Int, y: Int, bubble: Bubble): Unit = ??? // 3p

    def isEmptyRowAtTop: Boolean = ???     // 3p

    def isEmptyColumnAtLeft: Boolean = ??? // 3p

    def addNewRowToTop(): Unit = ???    // 3p

    def insertNewColumn(): Unit = ???   // 5p

    def clearOut(island: BubbleIsland): Unit = ??? // 3p

    def setColumn(x: Int, col: Vector[Bubble]): Unit = ??? // 4p

    def shiftDown(): Unit = ??? // 6p

    def compactColumnsRight(): Unit = ??? // 10p

    def findIsland(start: Pos): Option[BubbleIsland] = ??? // 14p

    def hasIsland: Boolean = ??? // 6p
```

```scala
object BubbleGame:
   var board = BubbleBoard.create()
   var gameScore = 0

   def ask(): Option[Pos] =
        print("Burst bubbles at (e.g A4): ")
        val response = scala.io.StdIn.readLine()
        scala.util.Try({
            val row = response(0) - 'A'
            val col = response.substring(1).toInt
            Pos(col, row)
        }).toOption

    def isGameOver(board: BubbleBoard) = !board.hasIsland

    def stepContinuousMode(board: BubbleBoard): Unit =
        board.shiftDown()
        board.compactColumnsRight()

        // Add new rows if there is no bubbles at top
        while board.isEmptyRowAtTop do
            board.addNewRowToTop()
            board.shiftDown()

```

```
        // Add new columns if there are no bubbles in the most left column
        while board.isEmptyColumnAtLeft do
            board.insertNewColumn()
            board.compactColumnsRight()

    def printWorld(): Unit =
        println(s"Score: ${gameScore}")

        val colHead = Seq.tabulate(board.width)(c => s"$c ").mkString
        println(s"  $colHead")

        def row(row: Int) =
            val bubbles = for x <- 0 until board.width yield board(x, row)
            bubbles.map(s => s"$s ").mkString

        for r <- 0 until board.height do
            val rowHead = ('A' + r).toChar.toString
            println(s"$rowHead ${row(r)}")

        println()

    def play(): Int =

        while !isGameOver(board) do
            printWorld()

            val pos = ask()
            if pos.isDefined then
                board.findIsland(pos.get) match {
                    case Some(island) =>
                        board.clearOut(island)
                        stepContinuousMode(board)
                        gameScore += island.score
                    case None => println("No island could be found at location.")
                }
            else
                println("Invalid position.")

        println(s"GameOver: ${gameScore}")
        gameScore
```

Förklaring av abstraktionerna i modellen

1. Enumerationen `Bubble` representerar en bubbla. Färgerna `Red`, `Blue`, `Green` och `Orange` indikerar att en färgad bubbla finns i en given ruta. `Empty` innnebär att ingen bubbla finns i en given ruta.

2. Case-klassen `Pos` representerar en position på brädet. Den ärver från `Ordered` som definerar en metod `compare` vars uppgift är att jämföra två `Pos` object så att dessa kan ordnas. `compare` ordnar först utifrån `x` och sedan efter `y` när `x` är lika.

3. Case-klassen `BubbleIsland` representerar en ö av bubblor som hänger ihop på brädet. Det finns en metod `score` som anger hur många poäng denna ö av bubblor är värd. Minst 2 element måste finnas för att klassas som en ö.

4. Klassen `BubbleBoard` representerar brädet. Här finns alla operationer som kan utföras på ett bräde. `apply` hämtar ut läget på brädet. `shiftDown` och `compactColumnsRight` utför de stora förflyttningarna av bubblor.

5. Singelobjektet `BubbleGame` modellerar själva spelet och har två attribut `board` och `gameScore` som representerar brädet och nuvarande poäng. Metoden `play` startar en spelomgång.

Krav och tips

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

1. Metoden `random` i kompanjonsobjektet `Bubble` ska ge en slumpmässigt färgad bubbla. *Tips:* metoden `fromOrdinal` är användbar för att lösa denna metod. *Tänk på:* att `Empty` aldrig ska kunna ges tillbaka och att paramentern `rand` ska användas.

2. Metoden `compare` i Case-klassen `Pos` ska jämföra två positioner `this` och `that` och ge tillbaka avståndet mellan punkterna enligt följande: Om `this` och `that` ligger på olika kolumner returneras avståndet i x-led och om de ligger på samma kolumn returneras avståndet i y-led. Om `this` ligger ovanför, eller till vänster på samma rad är avståndet negativt.

3. Metoden `score` i `BubbleIsland` beräknar antalet poäng en ö är värd enligt: $n \cdot (n-1)$, där n är antalet bubblor i ön.

Allt nedan gäller enbart klassen `BubbleBoard`:

4. Metoden `randomBubbles` ska ge n slumpmässigt färgade bubblor. Använd rätt `Bubble.random`. För full poäng krävs att parametern `rand` skickas vidare.

5. Attributet `board` inuti `BubbleBoard` ska initeras till ett slumpmässigt bräde med storleken `width` * `height`. Alla bubblor ska vara färgade, inga tomma rutor ska finnas.

6. Metoden `apply` ska ge tillbaka färgen för vald position eller värdet för en tom ruta om positionen är utanför brädet.

7. Metoden `set` ska ändra brädet. *Tänk på* att vektorer är immutable och måste uppdateras på korrekt sätt.

8. Predikatet `isEmptyRowAtTop` ska vara sant om alla rutor på översta raden är tomma.

9. Predikatet `isEmptyColumnAtLeft` ska vara sant om den första kolumnen (x=0) längst till vänster består av endast tomma rutor.

10. Metoden `addNewRowToTop` ska fylla översta raden (y=0) med slumpmässigt likafördelade färgade bubblor.

11. Metoden `insertNewColumn` ska fylla kolumnen längst till vänster (x=0) med slumpmässigt valt antal, från 1 till `height`, bubblor av slumpmässig färg. Både antalet och färgerna ska vara likafördelade.

12. Metoden `clearOut` ska sätta alla angivna positioner i `BubbleIsland` till att bli tomma rutor på brädet.

13. Metoden `setColumn` ska ersätta innehållet i kolumn x med de celler som finns i vektorn `col`. Metoden ska även fylla ut början av vektorn med tomma bubblor om vektorn består av färre element än höjden på brädet.

    Tips: Du kan ha nytta av denna metod när du löser andra metoder nedanför.

14. Metoden `shiftDown` ska se till att alla bubblor faller nedåt (positiv y-led) så att inga tomma hål finns mellan bubblor. Tomma rutor får finnas från toppen och neråt. Se exempel på sidan 5. Tomma rader kan finnas efter att operationen är slutförd.

15. Metoden `compactColumnsRight` ska förflytta hela kolumner till höger så att inga tomma kolumner mellan två kolumner finns. Tomma kolumner kan finnas till vänster efter att operationen är slutförd. Se exempel på sidan 5.

    Tips: Välj ut de kolumner som inte är tomma och sätt in dessa på rätt plats. Missa inte att det eventuellt kan finnas tomma kolumner till vänster och att de då måste sättas till enbart bestå av tomma rutor.

16. Metoden `findIsland` ska implementeras enligt följande algoritm:

    a) Verifiera att *start* har färg annars kan inte en ö hittas

    b) Skapa en mängd som kommer representera besökta positioner, denna är initialt tom.

    c) Skapa en lista som initialt innehåller *start* positionen. Denna lista kommer hädanefter vara kallad *arbetslistan*.

    d) **`while`** arbetslistan inte är tom gör följande:

        i. Ta ur det första elementetet i arbetslistan och lägg till den i mängden för besökta. Detta element kallar vi *candidate*.

        ii. För varje granne (eng. *neighbor*) till *candidate* som har samma färg som *start* positionen och inte är besökt (tillhör alltså inte den besökta mängden): lägg till i arbetslistan.

    e) Nu har vi lämnat while-loopen och arbetslistan är tom. Om den besökta mängden innehåller fler än ett element, ge tillbaka en ö av alla besökta positioner annars kan inte en ö hittas. *Obs!* Positionerna ska vara sorterade när ön skapas.

17. Predikatet `hasIsland` avgör om det finns åtminstone en ö med **2** eller fler bubblor. Denna ska implementeras effektivt och det görs genom att undersöka om det finns en godtycklig bubbla på brädet som har åtminstone en granne med samma färg. *Tänk på* att tomma rutor inte innehåller några bubblor.