Valfri tentamen
EDAA45 Programmering, grundkurs

2025-01-08, 8:00-13:00

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. Uppgift 1. Evaluera uttryck. Totalt max 20p.

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

```
1  sealed trait X:
2    protected def s: Set[Int]
3
4  object X:
5    var s = Set(0, 0, 0)
6
7    val a = A()
8
9    final class A extends X:
10     final override protected val s = Set[Int]()
11     final var m = Map[Int, Int]()
12   end A
13
14   final case class B(t: Set[Int] = X.s) extends X:
15     protected def s = t
16     val size = s.size
17   end B
18
19   def z(): Unit = 42
20
21   def me(): X.type = this
22
23   def apply() = Vector.tabulate(3)(i => i + 1)
24 end X
```

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`, `u2`, ... , 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.

```
1  val u0 = 42.0
2  val u1 = X.B().t
3  val u2 = { X.a.s = Set(1, 2, 3); X.B().t }
4  val u3 = { X.s = Set(1, 2, 3).map(_ / 2); X.B().t }
5  val u4 = { class AA extends X.A; (new AA).s}
6  val u5 = { X.a.m = Map(42 -> 0); (new X.A()).m.get(42) }
7  val u6 = { val y = X; y.a.m.keySet.toSeq.head }
8  val u7 = Vector(X.me().me().me().z())
9  val u8 = X().filter(_ > 3).head
10 val u9 = { val x = new X { override protected def s = Set() }; x.size }
11 val u10 = X.B(t = Set.fill(10)(Set.empty[Int]).flatten).size
```

	Vid kompileringsfel 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`				
`u6`				
`u7`				
`u8`				
`u9`				
`u10`				

Del B. Uppgift 2–4. Fyra-i-rad. Totalt max 80p.

Regler för spelet Fyra-i-rad

Spelet Fyra-i-rad (eng. *Connect 4*) är ett sällskapsspel för två personer som påminner om luffarschack. Deltagarna turas om att lägga en bricka (eng. *marker*) i taget i spelbrädet, som är ett lodrät placerat fackverk enligt bilden till höger. Brickorna är av olika färg, en för varje spelare, vanligtvis röda eller gula.

Det finns 7 staplar (kolumner) med vardera 6 platser (rader). Brickorna fylls på uppifrån i separata kolumner och trillar ned i en stapel. Brickorna får ej flyttas efter placering. Den som först får fyra av sina brickor i rad (vågrätt, lodrätt eller diagonalt) vinner. Spelet slutar oavgjort om det inte längre finns plats för fler brickor.

Huvudprogram

Det färdiga huvudprogrammet ser ut som följer:

```
1  val board = Board()
2  val player1 = HumanPlayer(Marker.Cross)
3  val player2 = GreedyComputerPlayer(Marker.Ring)
4
5  def otherPlayer(p: Player): Player = if p.eq(player1) then player2 else player1
6
7  @main def play: Unit =
8    var currentPlayer: Player = player1
9    println("New Game is starting")
10   var isGameOver = false
11   while !isGameOver do
12     println(board.show)
13     println(s"Current player: $currentPlayer")
14     val colOpt = currentPlayer.nextCol(board)
15     if board.availableCols.isEmpty then
16       isGameOver = true
17       println("Board is full. It's a tie.")
18     else if colOpt.isEmpty then
19       isGameOver = true
20       println(s"$currentPlayer did not select available column.")
21       println(s"${otherPlayer(currentPlayer)} wins!")
22     else
23       val c = colOpt.get
24       println(s"selected column $c")
25       val m = currentPlayer.marker
26       if board.isWinnerIfPut(m, c) then
27         isGameOver = true
28         println(s"Player $currentPlayer wins!")
29       else
30         currentPlayer = otherPlayer(currentPlayer)
31       board.put(m, c)
32   end while
33   println("Game over.")
34   println(board.show)
```

Början av en exempelkörning av detta huvudprogram visas till höger, där en mänsklig spelare spelar mot datorn. Jämför utskriften till höger med huvudprogrammet på föregående sida.

Röda brickor representeras med `X` och gula brickor representeras med `O`. Innan något drag har skett visas ett tomt bräde (eng. *board*). Spelarna gör omväxlande sitt drag som resulterar i en vald stapel där det finns lediga platser.

Första draget görs av människan som väljer kolumn `0`. Därefter väljer datorn kolumn `3`. I kolumn `0` finns en kryss-markering och i kolumn `3` finns en ring-markering. Därefter väntar huvudprogrammet på att människan igen ska välja kolumn, etc.

Datorspelaren *Greedy Computer Player* är en ”halvsmart” spelare som alltid väljer en av de kolumner som ger flest brickor i rad, men *utan* att aktivt förhindra att motspelaren får fyra brickor i rad. Detta gör att människan kan bli glad över att känna sig smart i förhållande till datorn.

Du ska i efterföljande uppgifter färdigställa påbörjade programdelar. Läs igenom huvudprogrammet och alla uppgifter *innan* du börjar eftersom du behöver se all given kod för att förstå hur systemet hänger ihop och vad som saknas.

```
New Game is starting
-------




-------
0123456
Current player: Human Player X
select one column of 0,1,2,3,4,5,6: 0
selected column 0
-------




X
-------
0123456
Current player: Greedy Computer Player O
selected column 3
-------




X  O
-------
0123456
Current player: Human Player X
select one column of 0,1,2,3,4,5,6:
```

Uppgift 2. `Marker`, `Pos`. (7p)

`Marker` representerar innehållet i en position `Pos` i brädet. `Direction` representerar en riktning.

```
1  enum Marker:
2    case Empty, Cross, Ring
3    def toChar: Char = Marker.chars(ordinal)
4    def nonEmpty: Boolean = this != Empty
5
6  object Marker:
7    val chars: Seq[Char] = Vector(' ', 'X', 'O')
8    def fromChar(c: Char): Marker = ??? //3p
9
10 enum Direction(val deltaRow: Int, val deltaCol: Int):
11   case UpDown    extends Direction(1, 0)
12   case LeftRight extends Direction(0, 1)
13   case Diag1     extends Direction(1, 1)
14   case Diag2     extends Direction(1, -1)
15
16 case class Pos(row: Int, col: Int):
17   def +(d: Direction): Pos = ??? //2p
18   def -(d: Direction): Pos = ??? //2p
```

Du ska implementera de saknade delarna ovan och beakta efterföljande krav och tips:

- Metoden `fromChar` ska ge `Marker.Cross` om argumentet är tecknet `X` och `Marker.Ring` om argumentet är tecknet `O`. För alla andra teckenargument ska `Marker.Empty` returneras.
- Metoderna `+` respektive `-` ska returnera en `Pos` där `row` och `col` är ökad respektive minskad med `deltaRow` och `deltaCol`. Exempel: `Pos(0, 1) - Direction.Diag2 == Pos(-1, 2)`

Uppgift 3. `Board`. (43p)

`Board` representerar spelbrädet med `nbrCols` staplar som kan innehålla maximalt `nbrRows` brickor där staplarna från början är tomma. Varje stapel representeras av en förändringsbar sekvens där längden motsvarar antalet icke-tomma brickor.

```
1  class Board:
2    val (nbrRows, nbrCols) = (6, 7)
3
4    private val board =
5      import collection.{immutable, mutable}
6      immutable.ArraySeq.fill(nbrCols)(mutable.ArrayBuffer.empty[Marker])
7
8    def canPut(col: Int): Boolean = ??? //5p
9
10   def availableCols: Seq[Int] = ??? //4p
11
12   def get(p: Pos): Marker = ??? //5p
13
14   def nextEmptyRow(col: Int): Option[Int] =  ??? //4p
15
16   def put(m: Marker, col: Int): Unit = ??? //4p
17
18   def countAdjacent(m: Marker, p: Pos, d: Direction): Int = ??? //10p
19
20   def countAllAdjacentIfPut(m: Marker, col: Int): Option[Seq[Int]] = ??? //7p
21
22   def isWinnerIfPut(m: Marker, col: Int): Boolean = ??? //4p
23
24   def show =
25     val sepLine = ("-" * nbrCols) + "\n"
26     val rowStrings = for r <- nbrRows - 1 to 0 by -1 yield
27       (for c <- 0 until nbrCols yield get(Pos(r, c)).toChar).mkString
28     sepLine + rowStrings.mkString("\n") + "\n" +
29       sepLine + (0 until nbrCols).mkString
```

Du ska implementera de saknade delarna ovan och beakta efterföljande krav och tips:

- Attributet `board` ska inte innehålla någon `Marker.Empty`. En stapel med lediga platser representeras genom att stapeln har färre brickor än `nbrRows`.
- `canPut` ska ge **`true`** om `col` är en giltig kolumn med plats för fler brickor annars **`false`**.
- `availableCols` ska ge en sekvens med alla staplar som har plats för fler brickor. Om brädet är fullt ska en tom sekvens returneras.
- `get` ska ge den bricka i brädet som finns på platsen `p`. Om `p` är en position utanför brädet eller om det inte finns någon bricka på platsen `p` så ska `Marker.Empty` returneras. Rader räknas nerifrån och kolumner räknar från vänster. Första position i första stapeln motsvarar `Pos(0, 0)`.
- `nextEmptyRow` ska ge en `Option` med den rad som är nästa lediga plats i stapeln `col`. Om stapeln är full eller `col` är utanför brädet så ska `None` returneras.
- `put` ska uppdatera brädet genom att lägga till `m` överst i stapeln `col`, d.v.s. genom tillägg sist i den sekvens som motsvarar kolumnen `col`. Med metoden `require` ska du säkerställa att ett undantag kastas om det inte går att placera en bricka i kolumn `col` eller om `m` är en tom bricka.
- `countAdjacent` ska räkna antalet till `p` närliggande brickor i rad som är lika `m` i riktningen `d`. Exempel: Om `d` är `Direction.UpDown` så ska det antal närliggande brickor som är lika `m` både uppåt och nedåt med utgångspunkt från `p`, tills första bricka som är olik `m` hittas. Eventuell bricka på plats `p` ska ej räknas.

- `countAllAdjacentIfPut` ska ge en `Option` som är `None` om `col` är full, annars ska returnerad `Option` innehålla en sekvens av längd 4 med antalet närliggande brickor lika `m` för varje möjligt riktning. Du har nytta av **`enum`**-metoden `values` som ger en `Array[Direction]` med alla möjliga riktningar. Exempel: nedan skapas ett bräde med tre röda brickor, två i första stapeln och en i tredje stapeln, varefter `countAllAdjacentIfPut` ger en `Option` med antalet närliggande brickor lika `m` för varje riktning givet att en röd bricka hade placerats i andra stapeln:

```
scala> val b = Board()
scala> b.put(Marker.Cross, 0); b.put(Marker.Cross, 0); b.put(Marker.Cross, 2)
scala> val optSeq = b.countAllAdjacentIfPut(Marker.Cross, 1)
val optSeq: Option[Seq[Int]] = Some(ArraySeq(0, 2, 0, 1))
```

- `isWinnerIfPut` ska ge **`true`** om spelaren med bricka `m` har minst 3 närliggande brickor lika `m` i någon riktning om den hade placerat en bricka i stapel `col`. Om `col` är full eller ogiltig ska **`false`** returneras. Du har nytta metoden `countAllAdjacentIfPut`.
- Endast metoden `put` får förändra `board`.

Uppgift 4. `Player`. (30p)

`Player` är en spelare med brickan `marker` som kan välja var den vill lägga sin nästa bricka.

```
1  trait Player:
2    def marker: Marker
3    def nextCol(board: Board): Option[Int]
4
5  class HumanPlayer(val marker: Marker) extends Player:
6    override def toString = s"Human Player ${marker.toChar}"
7    def nextCol(board: Board): Option[Int] = ??? //12p
8
9  class GreedyComputerPlayer(val marker: Marker) extends Player:
10   override def toString = s"Greedy Computer Player ${marker.toChar}"
11   def nextCol(board: Board): Option[Int] = ??? //18p
```

Du ska implementera de saknade delarna ovan och beakta efterföljande krav och tips:

- Du kan förutsätta att `marker` är icke-tom.
- `nextCol` ska ge en `Option` med en giltig stapel där spelaren vill placera sin nästa bricka eller `None` om brädet är fullt.
- `nextCol` ska inte förändra `board`.
- `nextCol` i `HumanPlayer` ska fungera så här:
  - Om det finns möjlig stapel så ska en prompt ges som börjar med `"select one column of "` följt av en kommaseparerad lista med alla möjliga staplar samt kolon och blanktecken.
  - Indata från användaren ska läsas med `scala.io.StdIn.readLine`.
  - Så länge indata inte är ett heltal eller indata inte är en giltig stapel så ska textraden `"Illegal column. Try again!"` visas och användaren ska få en ny chans att ge indata tills den lyckas ge giltigt val.
  - Du har nytta av strängmetoden `toIntOption`.
- `nextCol` i `GreedyComputerPlayer` ska fungera så här:
  - För varje möjlig stapel beräknas alla största värden av `countAllAdjacentIfPut`. Om det finns ett unikt största värde ska motsvararande kolumn väljas. Om det finns mer än ett största värde så ska motsvarande kolumn bland dessa väljas slumpmässigt med lika sannolikhet. Exempel: Antag att det finns tre staplar som har de högsta antalet närliggande brickor lika `marker` som är 2 st (alla andra möjliga staplar ger alltså ett lägre antal närliggande brickor lika `m`). Då ska en av de tre motsvarande staplarna väljas med lika hög sannolikhet.
  - Du har nytta av metoderna `zipWithIndex` och `scala.util.Random.nextInt`.