Kontrollskrivning
EDAA45 Programmering, grundkurs

2022-10-26, 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*:

```
enum Base:
  case A, G, C, T

class Virus(private var dna: Array[Base]):
  require(dna.length > 0)
  def apply(i: Int): Base = if i == 42 then Base.A else dna(i)
  def show: String = dna.mkString
  def mutate(i: Int): Unit =
    dna(i) = Base.values((dna(i).ordinal + 1) % Base.values.length)

object Virus:
  val delta = Virus(Base.values ++ Base.values.reverse)
  val omikron = { delta.mutate(1); new Virus(delta.dna) }
```

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 Virus.*
import scala.util.Try

val u0 = 41.0 + 1
val u1 = delta == omikron
val u2 = omikron.show.indices.find(_ % 2 == 1).getOrElse(-1)
val u3 = omikron(42).ordinal + delta(41).ordinal
val u4 = Try{delta.mutate(42); delta.show}.toOption
val u5 = Base.values.indices.map(i => delta.dna.drop(i).take(i).head)
```

	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. Spelet ”sänka skepp”. Totalt max 40 p.

Det klassiska marina strategispelet **sänka skepp** (eng. *sea battle, battleships*) för 2 spelare går ut på att först upptäcka och sänka hela motståndarens flotta. Bilden ovan visar en analog variant av spelet. I förberedelserna ingår att varje spelares skepp placeras ut antingen lodrätt eller vågrätt. Under spelets gång turas spelarna om att ange en position på spelplanen där de tror att motståndaren har ett skepp, t.ex. G2. Efter beskjutning ska den som sköt få reda på om det var en miss (eng. *miss*), en träff (eng. *hit*) eller om ett skepp är sänkt (eng. *sunk*). Varje skepp har en längd mellan 2 och 5 block som ockuperar motsvarande antal positioner på respektive spelares spelplan. Ett skepp är sänkt om alla dess block har träffats.

Du ska i denna uppgift göra klart en digital variant enligt körningsexempel till höger, där en mänsklig användare spelar mot datorn. Människa och dator har varsitt hav som består av 10 * 10 positioner. Rader benämns `A`–`J` och kolumner benämns `0`–`9`. Innan spelet börjar placeras på slumpvisa platser i slumpvisa riktningar (lodrät eller vågrät) 5 stycken skepp vardera till människan och datorn, med längderna 5, 4, 3, 3, och 2.

I exempelkörningen som visas här till höger har människan t.ex. ett lodrätt skepp med längd 3 i positionerna `A6 B6 C6` och ett vågrätt skepp med längd 2 i positionerna `G6 G7`. För att människan enklare ska kunna skilja sina närliggande skepp åt, markeras startpositionen för lodräta skepp med `V` och vågräta med `>`, medan övriga positioner markeras med `|` för lodräta, respektive `-` för vågräta.

Människa får inte se datorns skepp, men alltefter som människan beskjuter datorns hav framträder skaderapporter med bokstäverna `M` för miss, `H` för träff och `S` om ett helt skepp är sänkt.

Indata `cheat` ger i avlusningssyfte en utskrift av datastrukturen med datorns skepp, medan `exit` ger en förtida avslutning av spelet.

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

Ovan huvudprogram körs nedan där användaren skriver `G0 G1 cheat G2 exit` efter varje kolon.

```
Human                     Computer
  0 1 2 3 4 5 6 7 8 9       0 1 2 3 4 5 6 7 8 9
A             V           A
B             |           B
C             |     V     C
D     V > - - - -   |     D
E     |             |     E
F     |             |     F
G             > -         G
H                         H
I                         I
J                         J
Fire at pos (e.g. A0): G0
You fired at G0 was Miss
Computer fired at F7 was Miss
Human                     Computer
  0 1 2 3 4 5 6 7 8 9       0 1 2 3 4 5 6 7 8 9
A             V           A
B             |           B
C             |     V     C
D     V > - - - -   |     D
E     |             |     E
F     |         M   |     F
G             > -         G M
H                         H
I                         I
J                         J
Fire at pos (e.g. A0): G1
You fired at G1 was Hit
Computer fired at H2 was Miss
Human                     Computer
  0 1 2 3 4 5 6 7 8 9       0 1 2 3 4 5 6 7 8 9
A             V           A
B             |           B
C             |     V     C
D     V > - - - -   |     D
E     |             |     E
F     |         M   |     F
G             > -         G M H
H     M                   H
I                         I
J                         J
Fire at pos (e.g. A0): cheat
Ship(Vector((J0,false), (J1,false), (J2,false),
(J3,false), (J4,false)),true),Ship(Vector((C3,false),
(D3,false), (E3,false), (F3,false)),false),
Ship(Vector((B0,false), (B1,false), (B2,false)),true),
Ship(Vector((H5,false), (H6,false), (H7,false)),true),
Ship(Vector((G1,true), (G2,false)),true)
Fire at pos (e.g. A0): G2
You fired at G2 was Sunk
Computer fired at D3 was Hit
Human                     Computer
  0 1 2 3 4 5 6 7 8 9       0 1 2 3 4 5 6 7 8 9
A             V           A
B             |           B
C             |     V     C
D     V H - - - -   |     D
E     |             |     E
F     |         M   |     F
G             > -         G M S S
H     M                   H
I                         I
J                         J
Fire at pos (e.g. A0): exit
Goodbye!
```

Given kod

Nedan visas de delvis färdiga klasserna `Pos`, `Ship`, `Sea` samt singelobjektet `SeaBattle`. Du ska färdigställa de saknade delarna markerade med `???`, vars maximala poängvärde ges i kommentar. Abstraktionerna i modellen förklaras övergripande på sidan 6. Du ska följa de krav och tips som ges på sidan 7.

```scala
case class Pos(row: Int, col: Int):
  override def toString = s"${(row + 'A').toChar}$col"

object Pos:
  def random(rowSize: Int = Sea.Size, colSize: Int = Sea.Size): Pos =  ??? // 2p
```

```scala
case class Ship(blockIsHit: Vector[(Pos, Boolean)], isHorizontal: Boolean):
  val isSunk: Boolean = blockIsHit.forall(pair => pair._2)
  val blocks: Vector[Pos] = blockIsHit.map(_._1)

  def isOverlap(other: Ship): Boolean = ??? // 3p

  def hitBlock(i: Int): Ship = ??? // 5p

  def isHitAt(p: Pos): Boolean = ??? // 4p

object Ship:
  def create(length: Int, at: Pos, isHorizontal: Boolean): Ship =
    val positions: Vector[Pos] = ??? // 4p
    Ship(positions.map(_ -> false), isHorizontal)

  def random(length: Int): Ship =
    val isHorizontal = scala.util.Random.nextBoolean()
    val at: Pos =  if isHorizontal
      then Pos.random(colSize = Sea.Size - length)
      else Pos.random(rowSize = Sea.Size - length)
    create(length, at, isHorizontal)
```

```scala
enum Damage { case Miss, Hit, Sunk }

object Sea:
  val Size = 10
  val ShipLengths = List(5, 4, 3, 3, 2)

  def create(): Sea =
    var ships = Vector(Ship.random(ShipLengths.head))
    for len <- ShipLengths.drop(1) do
      var candidate = Ship.random(len)
      while ships.exists(_.isOverlap(candidate)) do candidate = Ship.random(len)
      ships = ships :+ candidate
    new Sea(ships.toArray)

class Sea(val ships: Array[Ship]):
  var misses = Vector.empty[Pos]

  def isAllSunk: Boolean = ??? // 1p

  def firedAt(p: Pos): Damage  = ??? // 12p

  def show(p: Pos, isEnemy: Boolean): Char =
```

```
    ships.indexWhere(_.blocks.contains(p)) match
      case i if i < 0 => if misses.contains(p) then 'M' else ' '
      case i if ships(i).isSunk => 'S'
      case i if ships(i).isHitAt(p) => 'H'
      case i if isEnemy => ' '
      case i if ships(i).blocks.indexOf(p) == 0 =>
        if ships(i).isHorizontal then '>' else 'V'
      case i => if ships(i).isHorizontal then '-' else '|'
```

```
object SeaBattle:
  val (human, computer) = (Sea.create(), Sea.create())

  def printWorld(): Unit =
    def colHead: String = Seq.tabulate(Sea.Size)(c => s"$c ").mkString
    def row(s: Sea, r: Int, isEnemy: Boolean = false): String =
      (for c <- 0 until Sea.Size yield s"${s.show(Pos(r,c), isEnemy)} ").mkString
    println(s"Human${" " * 19}Computer\n  $colHead    $colHead")
    for r <- 0 until Sea.Size do
      val s = ('A' + r).toChar.toString
      println(s"$s ${row(human, r)}  $s ${row(computer, r, isEnemy = true)}")

  def humanMove(): Unit =
    scala.io.StdIn.readLine("Fire at pos (e.g. A0): ") match
      case "exit"  => println("Goodbye!"); sys.exit() // avbryter exekveringen
      case "cheat" => println(computer.ships.mkString(",")); humanMove()
      case input if input.length == 2 =>
        val p = Pos(input(0).toUpper - 'A', input(1) - '0')
        if p.row >= 0 && p.row < Sea.Size && p.col >= 0 && p.col < Sea.Size then
          println(s"You fired at $p was ${computer.firedAt(p)}")
        else
          println(s"Illegal coordinates: $input")
          humanMove()
      case input =>
        println(s"Illegal input: $input")
        humanMove()

  def computerMove(): Unit = ??? // 9p

  def play() =
    while !human.isAllSunk && !computer.isAllSunk do
      printWorld(); humanMove(); computerMove()
    if human.isAllSunk then println("You lost!") else println("You won!")
```

Förklaring av abstraktionerna i modellen

- Case-klassen `Pos` representerar en position. En position består av två heltalsattribut `row` och `col` där det första representerar radpositionen och det andra kolumnpositionen.
- Klassen `Ship` representerar ett skepp av block som ockuperar ett antal positioner. Attributet `blockIsHit` är en sekvens av par av positioner och ett booleskt värde som anger om positionen är träffad eller ej. Attributet `isHorizontal` anger om skeppet är placerat vågrätt eller ej.
- Enumerationen `Damage` representerar en skaderapport efter beskjutning och anger om skottet missade, `Miss`, träffade, `Hit` eller sänkte, `Sunk`, det beskjutna skeppet.

- Klassen `Sea` representerar ett föränderligt hav. Attributet `ships` hanterar skeppen i havet. Attributet `misses` håller reda på skott mot havet som inte träffat något skepp.
- Singelobjektet `SeaBattle` modellerar själva spelet och har två attribut `human` och `computer` som hanterar den mänskliga användarens respektive datormotståndarens hav. Metoden `play` startar en spelomgång där människan och datorn omväxlande skjuter tills någon vinner.

Krav och tips

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

1. Metoden `random` ska ge en slumpmässig position vars rad och kolumn är likafördelade i intervallen `0 until rowSize` respektive `0 until colSize`.
2. Predikatet `isOverlap` ska vara sant om skeppet har några gemensamma positioner med `other`. *Tips:* Du har nytta av sekvensmetoden `intersect`.
3. Metoden `hitBlock` ska ge ett nytt skepp där paret på plats `i` i sekvensen `blockIsHit` har ett andra element som är **`true`**, vilket representerar att positionen är träffad.
4. Predikatet `isHitAt` ska vara sant om en position `p` ingår bland skeppets positioner och är träffad. *Tips:* Du har nytta av linjärsökning, t.ex. med `indexOf`.
5. Värdet `positions` i metoden `create` i singelobjektet `Ship` ska vara en sekvens med alla positioner för ett skepp med längd `length` och startposition `at`, där resterande positioner är till höger eller nedanför startpositionen beroende på värdet av `isHorizontal`.
6. Predikatet `isAllSunk` ska vara sant om alla skeppen i havet är sänkta.
7. Metoden `firedAt` ska uppdatera havet i enlighet med utfallet av beskjutningen och lämna en skaderapport. Om skottet mot position `p` inte träffar något skepp ska denna position läggas till i slutet av `misses`. Om skottet träffar något skepp i havet så ska detta skepp uppdateras så att motsvarande block träffas. Skaderapporten ska ange om skottet var en miss eller om träffat skepp sänktes eller ej. *Tips:* Du har nytta av linjärsökning, t.ex. med `indexWhere` och `indexOf`.
8. Metoden `computerMove` ska genomföra ett drag av datorspelaren enligt följande algoritm:

   a) låt p vara en slumpmässig position inom människans hav

   b) **`while`** p ingår i redan missade eller träffade skott **`do`** dra en ny slumpposition

   c) beskjut människans hav

   d) skriv meddelande enligt exempelkörning på sidan 4 inklusive position och skaderapport

   Tips: Du har nytta av sekvensmetoderna `contains` och `exists`.