Valfri tentamen
EDAA45 Programmering, grundkurs

2024-01-04, 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*:

```
class Paket(var ärMjukt: Boolean = false)

sealed trait Hittepå:
  def superkraft: Boolean

class Tomte private (var ps: Array[Paket]) extends Hittepå:
  override val superkraft = false
  def delaUt(): Unit = ps = null
object Tomte:
  def apply(ps: Paket*): Tomte = new Tomte(ps.toArray)

class Gandalf private (trollFormel: String) extends Hittepå:
  def superkraft = trollFormel.head == '!'
  def trolla: Tomte =
    if !(superkraft && superkraft) || (superkraft && superkraft) then
      Tomte(Vector.tabulate(4)(i => Paket(i % 2 == 0))*)
    else Tomte(null)
object Gandalf:
  val trollkarlen = Gandalf("abrakadabra")
  val trollat = trollkarlen.trolla

case class TeleTubby(namn: String = "TinkyWinky") extends Hittepå:
  def superkraft = scala.util.Random.nextBoolean()
  def trolla(g: String => Int): Int = g(namn)
```

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.

```
val u0 = 41.0 + 1
val u1 = Gandalf.trollkarlen.trollformel.headOption.getOrElse('*')
val u2 = { Gandalf.trollat.delaUt(); Gandalf.trollat.ps }
val u3 = { Gandalf.trollat.ps = Array(Paket()); Gandalf.trollat.ps(1).ärMjukt}
val u4 = !Gandalf.trollat.ps(0).ärMjukt
val u5 = { val h = new Hittepå { override def superkraft = true }; h.superkraft }
val u6 = { val p = Paket(true) -> Paket(false); p._2.ärMjukt }
val u7 = Gandalf.trollkarlen == new Gandalf("abrakadabra")
val u8 = TeleTubby() == TeleTubby(namn = "TinkyWinky")
val u9 = { val t: Hittepå = TeleTubby(); t.namn.reverse.endsWith(s"""y""") }
val u10 = TeleTubby().trolla(s => s.reverse.head - 'x')
```

	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. Uppgift 2–4. Simulering av kortspel: ”finns i sjön”. Totalt max 80p.

Teknologen Oddput Clementin ska göra sitt årliga julbesök i mörka norrland för att träffa släkten. Varje jul vill de små syskonbarnen Kim och Noa spela kortspelet ”finns i sjön” (eng. *go fish*), enligt dessa regler:

1. En vanlig kortlek (eng. *deck*) med 52 kort blandas, varefter varje spelare får 7 kort på hand. Resterande kort sprids ut på bordet med baksidan uppåt i det som kallas ”sjön” (eng. *pond*).
2. En slumpmässigt vald spelare börjar fråga valfri annan spelare efter kort av viss valör (eng. *rank*), till exempel genom att uppfordrande yttra *”ge mig alla dina sjuor!?”*.
3. Den frågande spelaren får bara fråga efter en valör som den själv har på hand.
4. Om den frågade spelaren har minst ett kort av den efterfrågade valören på hand så måste den ge alla dessa till den frågande spelaren. Annars om frågaren misslyckas med att erhålla några kort så yttrar den frågade spelaren förnöjsamt: *”finns i sjön!”*.
5. Den frågande spelaren för fortsätta fråga valfri annan spelare efter kort så länge frågan lyckas.
6. Om frågan misslyckas måste den frågande spelaren ta upp ett kort ur sjön och turen går vidare till nästa spelare.
7. Om den frågande spelaren erhåller kort efter en fråga som gör att den kan bilda ett komplett fyrtal (eng. *book*) så ska dessa fyra kort av samma valör läggas ned på bordet med framsidan upp så att alla kan se dem.
8. Om turen går vidare till en spelare som saknar kort på hand så måste den spelaren ta upp 7 nya kort från sjön innan den börjar fråga. Om det finns färre än 7 kort kvar erhålls de återstående sjökorten.
9. Spelet är över när alla kort i sjön är slut.
10. Vinnare är den spelare som vid spelets slut har flest upplagda fyrtal. Om det finns mer än en spelare med flest fyrtal är spelet oavgjort.

Oddput har observerat att varken Kim eller Noa agerar med optimal strategi. Ingen av dem tar hänsyn till vad som hänt tidigare i spelet. Noa verkar ha en helt slumpmässig strategi medan Kim alltid girigt frågar en slumpässigt vald motspelare efter den valör som Kim har flest av. Oddput undrar nyfiket vilken strategi som oftast leder till vinst och påbörjar ett Scala-program som kan simulera ett stort antal spelomgångar för att reda ut om den slumvisa eller giriga strategin är bäst, med syfte att upplysa syskonbarnen.

Huvudprogram

Det färdiga huvudprogrammet ser ut som följer:

```scala
package gofish

@main def simulate(n: Int) =
  println(s"Simulerar finns-i-sjön $n spelomgångar...")
  def createPlayers() =
    Vector(new Player.Greedy("Kim"), new Player.Random("Noa"))
  val games: Vector[Game] = Vector.fill(n)(Game(createPlayers()))
  val winners: Vector[Player] = games.flatMap(_.play())
  val wins: Map[String, Int] = winners.groupBy(_.name).map((x, xs) => (x, xs.size))
  println(s"Antal vinster:\n${wins.mkString("\n")}")
```

En körning av detta huvudprogram med argument `10000` ger lite olika resultat (p.g.a slumpen), men det står klart att Noas strategi vinner i de flesta fall:

```
Simulerar finns-i-sjön 10000 spelomgångar...
Antal vinster:
Noa Random -> 7128
Kim Greedy -> 2380
```

Du ska i efterföljande uppgifter färdigställa påbörjade programdelar för att hantera kortleken, spelaren och själva spelet. Läs igenom alla uppgifter innan du börjar eftersom du behöver se all given, delvis färdig kod för att förstå hur systemet hänger ihop.

Uppgift 2. Kortlek. (21p)

Du ska implementera `rankCount`, `partitionBooksRemaining` och `take`.

```scala
package gofish

enum Suit:
  case Spades, Clubs, Hearts, Diamonds

enum Rank:
  case Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten, Jack, Queen, King, Ace

case class Card(rank: Rank, suit: Suit)

extension (cs: Vector[Card])
  def rankCount: Map[Rank, Int] = ??? // 6p

  def partitionBooksRemaining: (Set[Rank], Vector[Card]) = ??? // 10p

object Deck:
  def shuffled(): Deck =
    val orderedCards =
      (for r <- Rank.values; s <- Suit.values yield Card(r, s)).toVector
    val shuffledCards = util.Random.shuffle(orderedCards)
    new Deck(shuffledCards)

class Deck private (private var cards: Vector[Card]):
  def isEmpty: Boolean = cards.isEmpty

  def take(n: Int): Vector[Card] = ??? // 5p
```

Du ska beakta följande krav och tips:

- Extensionsmetoden `rankCount` ska registrera antalet kort i en kortvektor som är av samma valör och returnera en nyckelvärdetabell med registreringen.
- Extensionsmetoden `partitionBooksRemaining` ska ge ett par med (1) en mängd med rankerna för de fyrtal som ev. finns i kortvektorn och (2) en ny vektor med resterande kort utan fyrtalen.
- Metoden `take` ska ge `n` kort ur kortleken om det finns så många kort, annars resterande kort. Om kortleken är tom eller `n <= 0` så ges tom vektor. Attributet `cards` uppdateras så att givna kort är borttagna. *Tips:* Du har nytta av samlingsmetoderna `take` och `drop`.
- Dina implementationer ska fungerar enligt nedan REPL-evalueringar:

```
scala> import gofish.*, Suit.*, Rank.*, util.Random.shuffle

scala> def book(r: Rank) = Suit.values.map(s => Card(r, s)).toVector

scala> val xs = shuffle(Vector(Card(Two, Spades), Card(Two, Hearts)) ++ book(Ace) ++ book(Jack))

scala> val r = xs.rankCount
val r: Map[Rank, Int] = HashMap(Ace -> 4, Jack -> 4, Two -> 2)

scala> val p = xs.partitionBooksRemaining
val p: (Set[Rank], Vector[Card]) = (Set(Ace, Jack),Vector(Card(Two,Spades), Card(Two,Hearts)))

scala> val (cs, b) = { val d = Deck.shuffled(); d.take(51); (d.take(3), d.isEmpty) }
val cs: Vector[Card] = Vector(Card(Ten,Clubs))
val b: Boolean = true
```

Uppgift 3. Spelare. (20p)

Du ska implementera `give`, `receive`, `others`, samt `ask` i klassen `Greedy`.

```scala
package gofish

extension [T](xs: Seq[T]) def pickRandom: T = xs(util.Random.nextInt(xs.length))

trait Player(val name: String):
  protected var hand  = Vector.empty[Card]
  protected var books = Set.empty[Rank]

  def hasNoCards: Boolean = hand.isEmpty

  def nbrBooks: Int = books.size

  def give(rank: Rank): Vector[Card] = ??? // 5p

  def receive(cards: Vector[Card]): Unit = ??? // 5p

  def others(players: Vector[Player]): Vector[Player] = ??? // 4p

  def ask(players: Vector[Player]): (Player, Rank)

object Player:
  class Random(name: String) extends Player(s"$name Random"):
    override def ask(players: Vector[Player]): (Player, Rank) =
      (others(players).pickRandom, hand.pickRandom.rank)

  class Greedy(name: String) extends Player(s"$name Greedy"):
    override def ask(players: Vector[Player]): (Player, Rank) = ??? // 6p
```

Du ska beakta följande krav och tips:

- Metoden `give` ska ge en vektor med alla spelarens kort av valör `rank` och uppdatera `hand` så att de givna korten ej finns kvar.
- Metoden `receive` ska ta emot en sekvens av kort och uppdatera `hand` och `books` så att valörerna för de nya fyrtal som ev. kan bildas ur `hand ++ cards` läggs till i `books`. Den uppdaterade `hand` ska utökas med mottagna korten förutom kort som bildar fyrtal. *Tips:* Använd `partitionBooksRemaining`.
- Metoden `others` ska ge en spelarvektor med alla spelare i `players` utom den spelarinstans som metoden `other` anropas på. *Tips:* Du har nytta av `this` och `eq`. Exempel:

```
scala> import gofish.*, Player.*

scala> val ps = Vector.tabulate(4)(i => Greedy(i.toString))
val ps: Vector[Greedy] =
  Vector(Greedy@69c0bae6, Greedy@220f6a3c, Greedy@5eb041b5, Greedy@2648aa1b)

scala> val notMe = ps(2).others(ps)
val notMe: Vector[Player] = Vector(Greedy@69c0bae6, Greedy@220f6a3c, Greedy@2648aa1b)
```

- Metoden `ask` i klassen `Greedy` ska returnera ett par med (1) en slumpmässigt vald annan spelare och (2) den valör som den egna handen har flest kort av. Om det finns flera valörer med störst antal kort så ges godtycklig valör bland dessa. Du kan förutsätta att det finns minst ett kort på handen. *Tips:* Du har nytta av `rankCount` och samlingsmetoden `maxBy`.

Uppgift 4. Simulering av ett spel. (39p)

Du ska implementera `askForCards`, `nextPlayerIndex` och `winnerOpt`.

```
package gofish

class Game(val players: Vector[Player], val initNbrCards: Int = 7):
  require(players.length >= 2, "must be at least 2 players")

  private val pond = Deck.shuffled()

  def askForCards(currentPlayer: Player): Int =  ??? // 24p
    /* Pseudokod:
        så länge kortlös och sjön ej tom:
          ta emot initNbrCards ur sjön

        om kortlös så returnera 0 annars
          fråga efter kort
          ta emot kort om fick några annars ta ett kort ur sjön
          returnera antalet kort som frågan gav  */

  def nextPlayerIndex(currentIndex: Int): Int =  ??? // 2p

  def winnerOpt: Option[Player] = ??? // 13p

  def play(): Option[Player] =
    for p <- players do p.receive(pond.take(initNbrCards))
    var i = util.Random.nextInt(players.length)
    while !pond.isEmpty do
      if askForCards(players(i)) == 0 then i = nextPlayerIndex(i)
    end while
    winnerOpt
```

Du ska beakta följande krav och tips:

- Metoden `askForCards` ska implementera en enskild fråga efter kort enligt spelarens strategi. Din implementation ska uppfylla reglerna på sidan 4, och baseras på pseudokoden i kommentaren ovan. *Tips:* Notera att upprepad fråga vid erhållna kort (se regel 5 på sidan 4) hanteras av `play`.
- Metoden `nextPlayerIndex` ska ge det heltal som följer efter `currentIndex` modulo antalet spelare i `players`. Du kan förutsätta att `currentIndex >= 0`.
- Metoden `winnerOpt` ska ge `None` om det inte finns en unik vinnare (alltså om det finns mer än en spelare med maximalt antal fyrtal, vilket innebär oavgjort enligt regel 10 på sidan 4), annars en `Option` med den unika spelare som har flest antal fyrtal. Du kan förutsätta att det finns minst två spelare i `players`.