Tentamen i Objektorienterad Programmering, DAT043

Joachim von Hacht

Datum: 2021-03-13

Tid: 08.30-12.30

Hjälpmedel: Allt (dock inget samarbete tillåtet)

Betygsgränser:

U: -23, 3:24-37, 4:38-47, 5:48-60 (max 60)

Lärare: Joachim von Hacht. Alla frågor under tentamen skickas med epost till hajo@chalmers.se

Granskning: Meddelas via Canvas.

Instruktioner:

- SKRIV ALLA SVAR I KODEN I RESPEKTIVE FIL: Q1andQ2, Q3, Q4, ... direkt i IntelliJ-projektet. Om du inte har tillgång till IntelliJ skriv lösningar i vilken typ av program som helst men lägg in lösningarna i IntelliJ-projektet. Filerna skall heta som ovan (Q1and2, Q3, ..)
- Lösningarna måste klara de fall som anges *samt fall som är principiellt lika*. Lösningar som bara klarar exemplen räcker *inte*. Överkomplicerade lösningar kan ge poängavdrag.
- Färdiga klasser m.m. som får användas anges för varje uppgift. Anges inget får man alltid använda de grundläggande språkliga konstruktionerna, arrayer, egna metoder och egna klasser (men inte t.ex. List/ArrayList/String/Math/m.fl.).

LYCKA TILL...

1. Uppgift 1 och 2 är sammanslagna till en enda uppgift (denna). Antag att en person inte vill avslöja sin förbrukning av chips. Så fort det är slut i påsen köper personen en ny, i hemlighet, och lägger på samma ställe. Det verkar alltså som personen bara har en påse och äter väldigt sparsamt. En närstående vill dock få en ungefärlig uppfattning om förbrukningen och börjar att då och då räkna antal chips i påsen. Skriv ett program som låter den närstående ange värdena för antal chips i påsen för varje räkning och därefter räknar ut beviserligen minsta antal påsar personen har köpt. Exempel (ok att mata in rad för rad om du vill): 4p

```
Antal räkningar > 10
> 17 15 16 16 18 17 14 12 13 9      (antal chips i påsen matas in)
Beviserligen minsta antal köpta påsar: 3
```

2. -

3. Givet en array, arr, med positiva heltal, och längden $2n, n \geq 1$ (jämnt antal element). Skriv en metod som returnerar en ny array, *expanded*, där elementen i *expanded* ges av 7p

   $expanded = \{arr[1]_1, arr[1]_2, ..., arr[1]_{arr[0]}, ...arr[3]_1, arr[3]_2...arr[3]_{arr[2]}, ...\}$.

   D.v.s. jämt index i arr ger antal element och udda index ger värdet för dessa element i *expanded*. Exempel:

```
arr                  expanded
---                  --------
[1, 9, 1, 5, 1, 7]   [9, 5, 7]      // En av varje
[10, 2]              [2, 2, 2, 2, 2, 2, 2, 2, 2, 2] // Tio tvåor
[2, 9, 3, 21, 1, 7]  [9, 9, 21, 21, 21, 7]
```

4. Givet en kvadratisk matris med positiva heltal där varje rad består av unika element. Skriv en metod som returnerar en matris där raderna består av de element som är gemensamma för varje par av rader (snittet för två rader). Ordningen på raderna spelar ingen roll. Att snittet saknar element representeras med värdet 0. Antal snitt för en matris av storleken n ges av $\frac{n(n-1)}{2}$. För full poäng krävs en lämplig funktionell nedbrytning och att du för varje metod anger syftet med metoden (en kort kommentar vad den gör). Inte tillåtet med samlingar t.ex. List eller String o.s.v. Exempel: 12p

```
Matris              Resultat
-------             -----------
[1, 4, 7, 5]        [1, 5, 0, 0]     // Snittet för rad 1 o 2
[2, 1, 5, 3]        [1, 0, 0, 0]     // 1 o 3
[1, 3, 2, 9]        [7, 5, 0, 0]     // 1 o 4
[7, 5, 6, 8]        [2, 1, 3, 0]     // 2 o 3
                    [5, 0, 0, 0]     // 2 o 4
                    [0, 0, 0, 0]     // 3 o 4

[3, 4, 1]           [0, 0, 0]
[2, 7, 5]           [4, 1, 0]
[4, 1, 7]           [7, 0, 0]
```

5. Skriv en metod som givet två icke-tomma godtyckliga strängar, s och t, returnerar den kortaste delsträngen till s sådan att alla tecken i t finns i delsträngen. Finns det flera möjliga delsträngar returneras valfri (t.ex. den första). Skulle t innehålla tecken som saknas i s returneras hela s. Alla klasser/metoder i Appendix är tillåtna. TIPS: Bryt ned i flera metoder. Exempel: 10p

```
s = "aba accb accabc"    // OBS! Blankslag räknas som ett tecken

t               delsträng ur s
-----           --------------
"a"             "a"              // Finns många tänkbara
"aa"            "aba"            // Två a ("a a" också tänkbart)
"aaa"           "aba a"          // Tre a (unik lösning)
"abc"           "cab"            // Ett a, b och c ("abc" också tänkbart)
"babc"          "ba accb"        // "b accab" också tänkbart
"aaaaabbbccccc" "aba accb accabc" // Hela strängen
"ax"            "aba accb accabc" // Tecken saknas
```

6. Rita en bild som visar variabler, värden, referenser och objekt samt hur dessa förhåller sig till varann före, respektive efter anropet av metoden doIt. Rita som vi ritat under kursen, lådor, pilar o.s.v. 8p

```
A a = new A(new A[]{new A(-1), new A(1)}, 0); // Before
doIt(a);  // Call
          // After

void doIt(A a) {
   A[] tmp = new A[2];
   a.as[0].as = tmp;
   tmp[1] = a;
}

class A {
   int i;
   A[] as;
   A(A[] as, int i) {
      this.i = i;
      this.as = as;
   }
   A(int i) {this.i = i;}
}
```

7. Skapa en objektorienterad modell för ett verktyg för kommunikation och grupparbete (typ Slack). Klasserna skall vara så icke-muterbara som möjligt och dölja så mycket som möjligt av sin data (information hiding). Alla klasser/metoder i appendix är tillåtna. Följande klass skall användas.

```
public class User {            // A users of the system
   private String name;
   private String passwd;      // User password
   public User(String name, String passwd) {
      this.name = name;
      this.passwd = passwd;
   }
   // equals, hashCode, getter setter as needed here
}
```

   a) Skapa en klass för meddelanden. Ett meddelande har en meddelandetext och en avsändare. 1p

   b) Skapa en klass för en meddelandetråd. En tråd har ett id och antal meddelanden. Lägg till en metod som givet ett meddelande lägger till detta sist i tråden. 2p

   c) Skapa en klass Channel för kanaler. En kanal innehåller ett antal trådar och har ett id. Lägg till en metod som givet ett tråd-id returnerar tråden om den finns i kanalen annars null. 3p

   d) Skriv en klass Sluck som representerar hela systemet. Systemet har ett antal användare och ett antal kanaler. Lägg till en metod publish, som givet en användare, ett meddelande, en kanal och ett tråd-id, publicerar meddelandet (lägger sist i tråden) i given kanal och tråd. Metoden skall kontrollera att rimliga förutsättningar är uppfyllda. Om metoden lyckas returneras true annars false. 5p

8. Besvara a) och b) nedan.

   a) Varför släpper kompilatorn igenom de två sista raderna i koden nedan? 4p

```
interface A { ... };
class X { ... }

A a;
X x = new X();

a = x;      // Kompileringsfel
x = a;      // Kompileringsfel
a = (A) x;  // Inget kompileringsfel
x = (X) a;  // Inget kompileringsfel
```

   Ge kodexempel för de sista raderna som visar att det faktiskt kan fungera under körning. Gränssnittet A och classen X får inte ändras men du får skapa nya gränssnitt och/eller klasser för att motivera.

   b) Koden nedan skriver ut: "Happy" och "NOT happy". Motivera *mycket precist* de avgörande punkterna i programmet som påverkar att utskriften blir som den blir (t.ex. vilka metoder som körs (varför), returvärden, variabel/objekttyper som kan påverka, ev. m.m) 4p

```
void program() {
   B b1 = new B(123, 1);
   B b2 = new B(123, 2);
   A a1 = b1;
   A a2 = new A(123);
   if (a1.equals(b2)) {
      out.println("Happy");
   } else {
      out.println("NOT happy");
   }
   if (a1.equals(a2)) {
      out.println("Happy");
   } else {
      out.println("NOT happy");
   }
}

// Forst. nästa sida.
```

```
public class A {
   private int aNumb;
   public A(int aNumb) {
      this.aNumb = aNumb;
   }
   public boolean equals(A a) {
      if (this == a) {
         return true;
      } else if (a == null) {
         return false;
      } else if (getClass() != a.getClass()) {
         return false;
      }
      return aNumb == a.aNumb;
   }
}

public class B extends A {
   private final int bNumb;
   public B(int aNumb, int bNumb) {
      super(aNumb);
      this.bNumb = bNumb;
   }
   public boolean equals(B b) {
      if (this == b) {
         return true;
      } else if (b == null) {
         return false;
      } else if (!super.equals(b)) {
         return false;
      } else if (getClass() != b.getClass()) {
         return false;
      }
      return bNumb == b.bNumb;
   }
}
```

APPENDIX

```
Följande klasser/metoder får användas om så anges vid uppgiften.

Ur klassen String
- equals(s), avgör om en sträng innehåller samma tecken som en annan.
- charAt(int i), ger tecknet vid index i.
- indexOf(char ch), ger index för tecknet ch, -1 om tecknet saknas.
- length() ger längden av strängen.
- subString(int start, int end), ger en delsträng från
  start (inkl.) till end-1.
- subString(int start), ger en delsträng från start (inkl.)
  till strängens slut.
- toCharArray(), gör om strängen till en array med tecken
- s1.compareTo(s2), ger -1, 0 eller 1 om s1 är respektive mindre,
  lika med eller större än s2 i lexikografisk ordning
Ur klassen StringBuilder
- append(String s), lägger till strängen s sist i
  Stringbuilder-objektet.
- append( char ch ), som ovan
- indexOf(ch), ger index för tecknet ch
- deleteCharAt(i), raderar tecken vid index i.
- toString(), omvandlar StringBuilder-objektet till en String.
Ur klassen Character
- isDigit(ch),isLetter(ch) avgör om tecknet är en siffra repsltive bokstav
Ur List/ArrayList
- get(i), ger objektet för index i
- add(o), lägger till objektet o sist i listan
- set(i, o), lägger till objektet vid index i, flyttar övriga till höger.
- remove(o), tar bort objektet o ur listan, returnerar
  true om detta lyckades annars false
- remove(i), tar bort och returnerar objektet vid index i ur listan
- removeAll( list ), tar bort alla element i list.
- contains(o), sant om objektet o finns i listan.
- indexOf(o), ger index för objektet
- size(), ger längden på listan

Klassen Random med metoden nextInt() är alltid tillåten.
```