Tentamen i Grundläggande Programvaruutveckling, TDA548

Joachim von Hacht

Datum: 2020-01-07

Tid: 08.30-12.30

Hjälpmedel: Lexikon Engelskt-Valfritt språk.

Betygsgränser:

U: -23

3: 24-37

4: 38-47

5: 48-60 (max 60)

Lärare: Joachim von Hacht. Någon besöker ca 09.30 och 11.00, tel. 031/7721003

Granskning: Anslås på kurssida.

Instruktioner:

- För full poäng på essäfrågor krävs ett läsbart, begripligt och heltäckande svar. Generellt 1p för varje relevant aspekt av problemet. Oprecisa eller alltför generella (vaga) svar ger inga poäng. Konkretisera och/eller ge exempel.
- Det räcker med enbart relevanta kodavsnitt, övrig kod ersätts med "..." (aldrig import, main-metod, etc....). Vi utgår från att användaren alltid skriver rätt och/eller gör rätt (d.v.s ingen felhantering behövs). Om felhantering skall ingå anges detta specifikt.
- 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.

LYCKA TILL...

1. Vad avses med: 4p

   a) super().

   b) static (klass) variabel.

   Förklara med en eller ett par meningar, du får gärna förtydliga med en skiss eller med kod.

2. Koden nedan har ett kompileringsfel. Vilket? 2p

```
int i = 0;
for (int j = 1; j < 10000; j++) {
   if (j % 2 == 0 && j % 3 == 0 && j % 5 == 0) {
      i++;
      if( i > 3){
         break;
      }
   }
}
out.println("Found: " +  j + " " + i);
```

3. Given en icke-tom array av heltal. Skriv en metod som expanderar arrayen enligt: 8p

   - Det första värdet behåller sin plats d.v.s finns först i resultatet.
   - Varje element utom det sista anger hur många av nästa element som skall finnas i resultatet.

```
Array             Resultat
-----             --------
[1]               [1]
[1, 2, 3]         [1, 2, 3, 3]
[5, 1]            [5, 1, 1, 1, 1, 1]
[2, 3, 4, 3]      [2, 3, 3, 4, 4, 4, 3, 3, 3, 3]
```

   Ni kan anta att ni har en metod int getExpandedLength(int[] arr) som ger storleken för resultat-arrayen.

4. Givet en icke tom heltals-array och en kvadratisk heltalsmatris. Skriv en metod, hasSubmatrixWith(), som avgör om det i matrisen finns en kvadratisk delmatris med samma värden som i arrayen. Vi antar att användaren anger en rimlig array (längden är en kvadrat o.s.v.). För full poäng krävs: En lämplig funktionell nedbrytning och att du för varje metod anger syftet med metoden (en kort kommentar vad den gör). OBS! Inte tillåtet att använda samlingar (t.ex. List). Metoden Math.sqrt() är tillåten. Exempel: 12p

```
int[] a0 = {4};
int[] a1 = {1, 4, 0, 2};
int[] a2 = {4, 0, 2, 0};
int[] a3 = {1, 4, 7, 2, 0, 5, 3, 3, 1};

int[][] m = {
    {1, 4, 7},
    {2, 0, 5},
    {1, 3, 3}
};
```

Anrop	Resultat
hasSubmatrixWith(a0, m)	true
hasSubmatrixWith(a1, m)	true
hasSubmatrixWith(a2, m)	false
hasSubmatrixWith(a3, m)	true

5. Ett girlangord är ett ord som inleds och avslutas med en likadan delsträng t.ex. ”onion” som inleds och avslutas med ”on”. Delsträngen är alltid kortare än hela ordet. Ordets grad bestäms av längden på delsträngen, ”onion” har alltså grad två. Exempel: 8p

   - ”underground” är girlangord av grad tre därför att det börjar och slutar på ”und” som har längden tre.
   - ”aaaa” är ett girlangord av grad tre eftersom delsträngen skall vara kortare än hela ordet d.v.s. ”aaa”

   Med hjälp av ett girlangord kan man skapa en girlangsträng på följande sätt:

   a) Ta bort delsträngen från början av girlangordet
   b) Det som blir kvar av girlangordet lägger men till på slutet av (hela) girlangordet lika många gånger som graden för girlangordet.

   Exempel: ”onion” delsträng: ”on” (grad två)

   - Ta bort inlednad ”on” ger ”ion”
   - Lägg till ”ion” två gånger på slutet ger ”onionionion” som är girlangsträngen.

   Fler exempel (se vidare nedan):

```
Inträng       Girlangsträng
--------      --------------
""            ""           (inte ett girlangord)
"jackhammer"  "jackhammer" (inte ett girlangord)
"zvioz"       "zviozvioz"  (delsträng z,  zvioz + vioz)
"aaaa"        "aaaaaaa"    (delsträng aaa, aaaa + a + a + a,  )
"alfalfa"     "alfalfalfalfalfalfa" (delstäng alfa,
                           alfalfa + lfa + lfa + lfa + lfa)
```

   a) Skriv en metod som givet en inträng returnerar den inledande/avslutande delsträngen om inträngen är ett girlangord. Om ej returneras tomma strängen. Alla klasser/metoder i Appendix är tillåtna, gäller även b)
   b) Skriv en metod som givet en inträng skapar en girlangsträng om inträngen är ett girlangord. Om ej returneras inträngen.

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. Strängar kan ritas förenklat som t.ex. “abc”. 8p

```
int[] arr = new int[]{1, 2, 3};
XM xm1 = new XM("abc", arr);
XM xm2 = new XM("def", arr);  // Before
doIt(xm1, 0, xm2.s, xm2.a[2]);  // Call
                                // After

void doIt(XM xm, int i, String s, int v) {
   xm.a = new int[]{7,8,9};
   xm.a[i] = v;
   xm.s = s;
}

class XM {
   int[] a;
   String s;
   XM(String s, int[] a) {
       this.s = s;
       this.a = a;
   }
}
```

7. Skapa en objektorienterad modell av en affär. 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 klasser skall användas.

```
class Customer {
   private String id;
   private String name;
   // Constructor, setter/getter, equals omitted but
   // available, also for classes below
}

class Item {  // Something the customer can buy
   private String id;
   private String description;
   private double price;
}

class OrderItem {  // How many items to order
   private Item item;    // The item
   private int qty;      // Number of items (quantity)
   public OrderItem(Item item, int qty) {
      this.item = item;
      this.qty = qty;
   }
}
```

   a) Skapa en klass Order. Klassen skall ha en köpare och en lista med OrderItem:s som köparen vill köpa. All data sätts då ordern skapas. Vi antar att alla setters/getters/equals/hashCode vi ev. behöver finns (gäller även nedan). 2p

   b) Skapa en klass ItemQuantity. Klassen representerar tillgängligheten (hur många) av en viss produkt som finns i lager. Vilken produkt det gäller sätts då objektet skapas. Klassen skall ha en metod för att minska antalet av produkten. 3p

   c) Skapa en klass Store för affären. Affären har en lista med tillgängligheten för alla produkter, en lista med kundordrar (orderings) och en lista med restordrar (backorders). Klassen skall ha en metod som tar en kund och en inköpslista med OrderItem (som kunden alltså vill köpa). För alla produkter i inköpslistan som finna i tillräckligt antal skapas en order med dessa för den aktuella kunden. Ordern sparas i listan med kundordrar. Är det till gängliga antalet produkter för litet skapas på samma sätt en order men den läggs i restorderlistan (antingen finns tillräckligt många produkter eller ej, inget annat beaktas). 5p

8. Betrakta koden nedan och ange för varje rad a) - g) *en* av följande. 8p

   - Kompilerar ej därför att ...
   - Körningsfel därför att ...
   - Om inget av ovan, ange vad som skrivs ut.

```
a) D d = new D(); C c = d; c.doIt();
b) IY iy = new D(); C c1 = (C) iy; c1.doOther();
c) A a = new B(); a.doIt(1);
d) IX ix = new B(); IY iy1 = new C(); ix = (IX) iy1; ix.doIt(); // 2p
e) A a1 = new C(); D d1 = (D) a1; d1.doIt(1.0);
f) C c2 = new D(); B b = (B) c2;
g) C c3 = new C(); A a2 = c3; a2.doOther();

-------------------------------------------------------
interface IX { void doIt();}
interface IY { void doOther(double);}
class A {
   public void doIt(double d) { out.println("doIt A " + d); }
}
class B extends A implements IX {
   public void doIt() { out.println("doIt B");}
   public void doIt(int i) { out.println("doIt B " + i); }
}
class C extends A implements IY {
   public void doIt() { out.println("doIt C"); }
   public void doOther(double) { out.println("doOther C");
}
}
class D extends C {
   void doIt() { out.println("doIt D");}
   public void doOther(int i) { out.println("doOther D");}
}
```

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
- endsWith(s), sant om strängen avslutas med s.
- s1.compareTo(s2), ger -1, 0 eller 1 om s1 är respektive mindre, lika med eller stö
  lexikografisk ordning
- s.split(pattern), gör om strängen till en array av strängar. Strängen avdelas av p
  (mellanslag) eller ‘‘X’’ tecknet X. Pattern kommer att försvinna.
Ur klassen StringBuilder
- append(String s), lägger till strängen s sist i
  Stringbuilder-objektet.
- append( char ch ), som ovan
- setLength(), sätter aktuell längd, setLength(0)
  raderar alla tecken.
- lenght() ger aktuell längd
- toString(), omvandlar StringBuilder-objektet till en String.
- deleteCharAt(int i), radera tecknet på plats i

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.
```