Tentamen i Grundläggande Programvaruutveckling, TDA548

Joachim von Hacht

Datum: 2019-10-30

Tid: 14.00-18.00

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 15.00 och 16.30, 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) Parameter.

   b) Explicit typomvandling.

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

2. Koden nedan ger ett körningsfel. Vilken rad ger felet och varför uppstår det? 2p

```
Integer i1 = null;
Integer i2 = 5;
if (i1 == i2) {
    out.println("Equal");
}
if (2 * i1 == 2 * i2) {
    out.println("Also equal");
}
```

3. Given en array av heltal där vissa tal kan finnas i dubbletter, skriv en metod som returnerar antalet dubbletter. Alla dubbletter är unika d.v.s. finns det t.ex. ett par av treor så finns det inte fler par av treor. Exempel: 7p

```
{1, 2, 2, 3} metoden ger 1. Paret är (2,2)
{1, 2, 4, 3, 5, 2, 7, 1} metoden ger 2. Paren är (1,1) och (2,2)
{2, 2, 4, 3, 5, 3, 7, 7} metoden ger 3. Paren är (2,2), (3,3) och (7,7)
```

   Ingen felhantering krävs, vi förutsätter att arrayen är rimlig.

4. Skriv en metod som givet en kvadratisk matris med icke-negativa heltal avgör om det finns en strikt stigande sekvens $a_1, a_2, ...a_n$ av tal sådana att varje tal, a_i finns på rad i. För full poäng krävs funktionell nedbrytning. Du kan anta att det finns metoder minValue och maxValue (som du får använda) som ger minsta respektive största element i en heltals-array. Exempel: 12p

```
 Matris       Finns sekvens
[0, 1, 2          true     (t.ex. 0, 1, 2
 1, 2, 3                    eller 2, 3, 5)
 3, 5, 2]

 [2, 4, 2         false    (inget på rad 3 är
  1, 3, 3                   större än något på rad 2)
  3, 1, 2]

[4, 8, 3, 9        true   (3, 6, 7, 8)
 1, 3, 6, 7
 3, 7, 2, 6
 8, 1, 4, 2]
```

5. Skriv en metod, String format(String str, int nCols), som formaterar en sträng av ord (utan nyradstecken) så att alla rader håller sig inom ett givet antal kolumner (nCols). Alla ord i inströngen är åtskilda av exakt ett mellanslag och längden av det längsta ordet överstiger inte antalet kolumner. Orden får inte delas. För alla rader utom sista gäller att: 9p

   - om den sammanlagda längden av orden på en rad överstiger antalet kolumner flyttas det sista ordet till nästa rad d.v.s ett nyradstecken läggs in (nyradstecknet räknas inte med i antal kolumner).
   - om en rad inte fylls ut helt läggs mellanslag till sist på raden så att den blir nCols bred och därefter läggs ett nyradstecken till.

   Exempel (mellanslag visas som '.' för att förtydliga):

```
Inströng                    Kolumner    Utströng
"aaa.bb.cc.dddd"              6         aaa.bb (osynligt nyradstecken
                                        cc....  sist på varje rad
                                        dddd    utom sista)

"aaaa.bbbb.cc.ddd.eeee.fff"  5          aaaa.
                                        bbbb.
                                        cc...
                                        ddd..
                                        eeee.
                                        fff
```

   Alla metoder i appendix är tillåtna. Följande kod får användas:

```
final char NL = '\n'; // New line (last on all rows except the final one)
final char SPACE = '.';

// Return string of n spaces (i.e n=3 gives "...")
String getSpaces(int n);
```

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

```
XX x = new XX();
x.i = 1;
x.ys = new YY[]{new YY(2), new YY(3)}; // Before
YY y = doIt(x);  // Call
                 // After
YY doIt(XX x) {
    x.i = 2 * x.ys[0].i;
    YY tmp = x.ys[1];
    x.ys[1] = null;
    return tmp;
}
class XX {
    int i;
    YY[] ys;
}
class YY {
    int i;
    public YY(int i) {
       this.i = i;
    }
}
```

7. Vi skall skapa en objektorienterad modell av en Bank. Klassen och enum:en nedan är givna och skall användas:

```
enum Type { REAL, PERSONAL, NOMINAL } // Type for Account

public class Person {                 // Class for customers
    private final String id;
    private String name;
    public Person(String id, String name) {
        this.id = id;
        this.name = name;
    }
    // getter/setter, equals, hashCode omitted (but exists)
}
```

   a) Skriv en klass Account för ett bankkonto. Ett konto har ett id, en typ och ett belopp. Id och typ skall anges då man skapar ett kontoobjekt. Vi antar att alla setters/getters/equals/hashCode vi ev. behöver finns (gäller även nedan). 2p

   b) Skriv en klass Association som kopplar ihop ett konto (Account) med en kund (Person). Konto och kund skall anges då man skapar ett objekt. 3p

   c) Skriv en klass Bank. En bank har ett antal kunder, ett antal konton och en lista med associationer mellan kunder och konton. Lägg till en metod createAccount som givet en kund och en kontotyp skapar ett nytt konto och kopplar ihop kunden med med kontot. Om detta lyckas så finns kontot och associationen sparade i banken och metoden returnerar true. Misslyckas det returneras false. Ni kan anta att det finns en metod long getUniqueAccoutnNumber() som ger ett unikt kontonummer. Följande restriktioner gäller: Kunden måste finnas i banken. En kund får bara ha ett konto av varje typ. 5p

   Alla metoder i appendix är tillåtna. Klasserna skall vara så icke-muterbara som möjligt och dölja så mycket som möjligt av sin data (information hiding).

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

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

```
a) B b = new B(); b.doA();
b) IA a = new X(); a.doA();
c) C c = new B(); c.doC();
d) B b1 = new C(); b1.doX();
e) IX x = new C(); X x1 = (X) x; x1.doA();
f) IX x2 = new C(); B b2 = (B) x2; b2.doC();
g) A a1 = new B(); a1.doA();
h) IA a2 = new A(); a2.doA();

public interface IA { void doA();}
public interface IX { void doX();}
public abstract class A implements IA {
    public void doA() { out.println("A.doA()");}
    public abstract void doC();
}
public class B extends A {
    public void doC() { out.println("B.doC()"); }
}
public class C extends B implements IX {
    public void doA() { out.println("C.doA()"); }
    public void doX() { out.println("C.doX()"); }
    public void doC() { out.println("C.doC()"); }
}
public class X implements IX {
    public void doX() { out.println("X.doX()"); }
    public void doA() { out.println("X.doA()"); }
}
```

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

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.