Tentamen i Introduktion till objektorientering, DAT044

Joachim von Hacht

Datum: 2022-06-10

Tid: 08.30-12.30

Hjälpmedel: Inget förutom Svensk-Engelskt lexikon.

Betygsgränser:

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

Lärare: Joachim von Hacht.

Granskning: Meddelas via Canvas.

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 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. Redogör kortfattat för följande begrepp. Du får gärna illustrera med en skiss eller med kod. 4p

   a) Returtyp

   b) Referens kontra värdelikhet.

2. Programmeraren har nedan försökt skapa en "icke-muterbar" klass. D.v.s. ingen data i objekten skall kunna ändras efter det att objektet har skapats. Har programmeraren lyckats eller ej. Förklara. Motivera din förklaring. Om den är muterbar förklara vad som är problemet. 5p

```
class MyImmutableClass {
   private final int[] arr ;
   private final String str;
   public MyImmutableClass(){
      arr = new int[]{1,2,3};
      str = "abc";
   }
   public int[] getArr() { return arr; }
   public String getStr() { return str; }
}
```

3. En "hop" är en del av en array där alla element har samma värde. D.v.s. minst två intilliggande index måste ha samma värde. Exempel: 7p

```
{1, 2, 3, 4, 5}            // Ingen hop
{1, 1, 3, 4, 5, 6, 8, 9}   // En hop 1,1
{1, 1, 3, 4, 5, 8, 8, 8}   // Två hopar 1,1 och 8,8,8
{5, 5, 4, 4, 3, 8, 8, 8}   // Tre hopar 5,5 och 4,4 och 8,8,8
{1, 1}                     // En hop
```

   Skriv en metod som givet en heltals-array med *längd* ≥ 2, returnerar antalet hopar i fältet.

4. Givet en kvadratisk matris med icke-negativa heltal och med $sida > 2$. Skriv en metod som 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. 1,2,3 se understrukna
 1, 2, 3              finns flera andra t.ex. 2, 3, 5)
 3, 5, 2]

[2, 4, 2       false
 1, 3, 3
 3, 1, 2]

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

5. Vi skall implementera en simulering av en endimensionell värld. Världen representeras av en sträng med ettor och nollor. Nollor representerar tomma platser och ettor levande organismer. För varje tidsteg och position i världen (understruket värde) gäller följande regler. 8p

```
0_0_0 -> 0_0_0  // Tom plats omgiven av tomma, inget händer
0_0_1 -> 0_0_1  // Tom plats en granne, inget händer
1_0_0 -> 1_0_0
1_0_1 -> 1_1_1  // Tom plats med två levande grannar föder ny organism
0_1_0 -> 0_0_0  // Organism utan grannar dör
0_1_1 -> 0_1_1  // Organism med en granne lever vidare
1_1_0 -> 1_1_0
1_1_1 -> 1_0_1  // Två grannar, organism dör
```

För första och sista position gäller samma regler dock har dessa bara max en granne. Exempel på hur världen utvecklas ($0 - n$ är tidsteg):

```
Tid  Världen
---  --------
0:   01101001
1:   01110000
2:   01010000
3:   00100000
4:   00000000
```

Skriv en metod som givet en värld i form av en sträng returnerar nästa generation i världen som en sträng (ett tidssteg). Alla metoder i Appendix är tillåtna.

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: Olika lådor/boxar, variabelnamn, 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. Vi skall skriva ett objektorienterat program för epost.

   a) Skriv en klass för personer som kan skicka och ta emot mail. Personerna har en email-adress och ett namn. Allt initieras då en person skapas. Inga set/get metoder behöver anges vi antar att de finns. Samma sak med equals och hashCode. Allt i Apendix är tillåtet att använda (gäller även nedan). 1p

   b) Skriv en klass Mail för mail. Ett mail har ett unikt id, en sändare, en mottagare och ett innehåll (en text). Alla värden skall kunna sättas då man skapar ett brevobjekt. 2p

   c) Skriv en klass MailServer. Klassen skall ha en inbox, för inkomna brev och en outbox för skickade brev. Lägg till en metod forward, som givet ett mail och en mottagare, söker efter brevet i inbox:en. Om detta hittas skapas en kopia av brevet där mottagaren sätts till den angivna mottagaren. Kopian sparas därefter i outbox:en (hur själva sändningen går till bekymrar vi oss inte för). 5p

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

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

```
a) IX ix = new C(); IY iy = (IY) ix; iy.doIt(new B());
b) A a = new B(); a.doIt(1);
c) B b = new C(); b.doIt(1);
d) A a = new B(); IX ix = (IX) a; C c = (C) ix; c.doIt(1.0);
e) C c = new C(); c.doIt(new B());
f) IY iy = new C(); iy.doIt(iy);
g) List<A> as = new ArrayList<>(); List<B> bs = as; bs.get(0).doIt(5);
h) B[] bs = new B[2]; bs[0] = (B) new C();

// --- Interfaces and classes
interface IX { void doIt(double d); }
interface IY { void doIt(IX ix); }
class A implements IX {
    public void doIt(double d) {
        out.println("doIt A");
    }
}
class B extends A {
    public void doIt(int i) {
        out.println("doIt B");
    }
}
class C extends A implements IY{
    public void doIt(IX ix) {
        out.println("doIt C");
    }
}
```

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
- str1.contains(str2), ger sant om strängen s2 finns i s1

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.