Tentamen i Grundläggande Programvaruutveckling, TDA548

Benjamin Eriksson

Datum: 2020-10-28

Tid: 14.00-18.00

Hjälpmedel: Allt

Betygsgränser: U: -23

3: 24-37

4: 38-47

5: 48-60 (max 60)

Lärare: Benjamin Eriksson. Alla frågor under tentamen skickas med epost till beneri@chalmers.se

Granskning: Meddelas via Canvas.

Instruktioner:

- SKRIV ALLA SVAR I KODEN I RESPEKTIVE FIL: Q1, Q2, Q3, Q4, ... direkt i IntelliJ-projektet. Om du inte har tillgang till IntelliJ skriv lösningar i vilken typ av program som helst men lägg in lösningarna i IntelliJ-projektet. Filerna skall heta som ovan (Q1, Q2, 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 far användas anges for varje uppgift. Anges inget far man alltid använda de grundläggande språkliga konstruktionerna, arrayer, egna metoder och egna klasser (men inte t.ex. List/ArrayList/String/Math/m. .).

GÖR ERT BÄSTA...

1. Förklara med egna ord: 4p

   a) Vad är skillnaden på **int** och **Integer**?

   b) Vad avses med **int <: double**.

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

2. Förklara noggrant vilket och hur **x** får sitt slutliga värde. 2p

```
int x = 5;
try {
  x = 6;
  out.println(1/0);
  x = 7;
} catch (ArithmeticException e) {
  out.println("Error");
}
out.println("x = " + x);
```

3. Given en heltalsarray, med minst 2 tal, skriv en metod som hittar en plats (index) i arrayen att dela på så att summan av de två delarna blir lika. Om ej möjligt, returnera -1. Talet på index i läggs i den vänstra gruppen. Ordningen i arrayen får inte ändras. Implementera metoden **int balanceArray(int[] arr)**. 8p

```
Array                Resultat
-----                --------
[1, 2, 3, 4, 5, 5]    3         // 1+2+3+4 = 5+5
[1, 2, 3, 3, 2, 1]    2         // 1+2+3 = 3+2+1
[1, 1]                0         // 1 = 1
[1, 1, 3]            -1         // Går ej att balansera
```

4. Given en **N * N** matris och en sidlängd **side**, skriv en metod **boolean sameSubMatricies(int[][] matrix, int side)** som returnerar **true** om alla delmatriser med sidlängden **side** har samma summa. För full poäng krävs funktionell nedbrytning. Exempel: 12p

   - I första fallet nedan, med sidläng 2, får vi: 1+2+3+1 = 2+3+1+1 = 3+1+0+3 = 1+1+3+2
   - I andra fallet, med sidlängd 2, är första och andra submatriserna olika: 1+2+3+1 != 2+**5**+1+1
   - I tredje fallet, med sidlängd 1, blir varje element en submatris: 1 != 2
   - Till sist blir den enda submatrisen hela matrisen.

```
int[][] m1 = {{1,2,3},
              {3,1,1},
              {0,3,2}};
int[][] m2 = {{1,2,5},
              {3,1,1},
              {0,3,2}};
```

Anrop	Resultat
sameSubMatricies(m1, 2)	true
sameSubMatricies(m2, 2)	false
sameSubMatricies(m1, 1)	false
sameSubMatricies(m1, 3)	true

5. Givet en lista med ord (max 10 ord) skriv en metod **int longestSnake(List<String> words)** som hittar den längsta följden av ord där sista bokstaven i föregående ord är samma som först bokstaven i nästa ord. Alla ord i listan behöver inte användas. Anta att ni har en metod *permutations* som ger er alla permutationer av en lista av ord. **List<List<String>> permutations(List<String> words)**. Ni får också använda allt från Appendix. Exempel: 8p

   - För ordlistan [”java”, ”kod”, ”gillar”, ”akademisk”, ”debug”, ”okej”] har den längsta ormen längden **6.** Den längsta ormen är [”okej”, ”java”, ”akademisk”, ”kod”, ”debug”, ”gillar”]. Eftersom “oke**j**” slutar på **j** och nästa ord “**j**ava” börjar på **j,** osv för resterande ord.
   - **Varning:** Antalet permutationer växer väldigt snabbt, redan för 10 ord finns det över 3 miljoner permutationer. Jag rekommenderar inte att testa listor med fler än 6 element.

   Fler exempel:

```
Ordlista              Längsta orm längd
--------              --------------
["ab", "cd", "bc"]    3            (["ab", "bc", "cd"])
["abc"]               1            (Endast ett ord)
["a", "b", "c"]       1            (Varje ord blir en egen orm)
["ab", "bc", "xx"]    2            ("ab" och "bc" skapar en orm,
                                    "xx" används ej)
```

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

```
public void program() {
  A a = new A(2); // Before
  doIt(a);        // Call
                  // After
}
void doIt(A a) {
  B tmp = a.bs[0];
  a.bs[0] = a.bs[1];
  a.bs[1] = tmp;
  a.bs[0].i = 2;
}
public class A {
  int n;
  B[] bs;
  A(int n) {
    this.n = n;
    bs = new B[n];
    for(int i = 0; i < bs.length; i++) {
      bs[i] = new B(i);
    }
  }
}
public class B {
  int i;
  B(int i) {
    this.i = i;
  }
}
```

7. Vi vill skapa en modell för säker hantering av dokument från olika personer. I den här modellen finns det dokument och personer. Dokument har en heltals rank för hur hemligt dokumentet är. Personer har en liknande rank för hur hemliga dokument de kan läsa. 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.

   a) Skapa en klass **Document** som innehåller en titel på ett dokument, en tid då dokumentet skapades och en heltals rank för hur hemligt dokumentet är. 2p

   b) Skapa en klass **Person** som innehåller ett namn och en heltals rank för hur hemliga dokument personen kan läsa. Implementera också metoden **boolean canRead(Document document)** som returnerar **true** om personen får läsa dokumentet, alltså om personens rank är högre än eller lika med dokumentets rank. Annars returnerna **false**. 4p

   c) Till sist vill vi skapa en ACM (Access Control Matrix). Implemeterna klassen **ACM** som har en lista med dokument samt en lista med personer (List typer från Appendix får användas). Skapa också metod **void printMatrix()** som först printar titeln på alla dokument på en rad, sedan för varje person printar en rad med true/false beroende på om personen kan läsa dokumentet. Ni behöver inte justera texten i tabellen på något vis. Det gör alltså inget om texten inte hamnar i perfekta kolumner. Exempel: 6p

```
Vi har 3 dokument {hemsida, anställda, hemligaRecept} med
rankerna {1,2,3}.
Vi har också 3 personer {Emma, Kalle, ZeroCool} med
rankerna {2,1,3}.
Då bör printMatrix() skriva ut följande:
          hemsida anställda hemligaRecept
Emma        true    true      false
Kalle       true    false     false
ZeroCool    true    true      true
```

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

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

```
a) A a = new B(); a.doIt();
b) A a2 = new B(); ((B) a2).doIt();
c) C c = new D(); c.doIt();
d) A a3 = new B(); C c3 = (C) a3;
e) IX ix4 = (IX) new B(); ix4 = (IX) new C(); ix4.doIt();
f) Object o5 = new A();
g) ArrayList<Object> objects6 = new ArrayList<A>();
h) Object[] objects7 = new A[10];

-------------------------------------------------------
interface IX { void doIt();}
interface IY { void doOther(double d);}
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 d) { out.println("doOther C");
}
}
class D extends C {
   public 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 resultat < -1, 0 eller resultat > 1 om s1
                    är respektive mindre, lika med eller större än s2 i
                    lexikografisk ordning
- s.split(pattern), gör om strängen till en array av strängar.
                    Strängen avdelas av pattern t.ex.'' ``
                    (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.`