Tentamen i Imperativ Programmering med Grundläggande Objektorientering, DIT012

Joachim von Hacht

Datum: 2018-04-04

Tid: 08.30-12.30

Hjälpmedel: Engelskt-Valfritt språk lexikon

Betygsgränser:

U: -23

G: 24-43

VG: 44-60 (max 60)

Lärare: Joachim von Hacht, tel. 031/772 10 03. Någon besöker ca 09.30 och 11.00

Granskning: Tentamen kan granskas på studieexpeditionen. Vid ev. åsikter om rättningen eposta mig och ange noggrant vad du anser är fel så återkommer jag (ev. ta en bild och skicka med).

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. Förklara med en eller ett par meningar. Du får gärna ge kodexempel eller rita. 4p

   a) Referenslikhet

   b) Synlighetsområde

2. Vad skriver koden nedan ut? Motivera! 4p

```
Double d1 = 5.0;
Double d2 = 5.0;
out.println(d1 == 5.0);
out.println(d1 == d2);
out.println(d1 == 1 * d2);
out.println(d1.equals(d2));
```

3. Skriv en method som returnerar det n:te heltalet vars siffersumma är lika med 10. Metoden skall ta n som parameter. Exempel: 6p

```
n           Tal
1           19  (första talet med siffersumma 10)
2           28
3           37
11          118
123         2044
```

4. Givet två sekvenser av heltal med samma längd ≥ 2, och ett heltal $n \geq 2$. Skriv en metod, commonSeq, som returnerar någon eventuell gemensam delsekvens av längden n. Sekvenserna skall representeras med heltalsarrayer och resultatet skall vara en array med den gemensamma delsekvensen (orginal-arrayer får inte ändras). Saknas gemensam sekvens returneras en tom array. För full poäng krävs: En lämplig funktionell nedbrytning och att du för varje metod anger vad metoden gör, indata samt utdata. OBS! Inte tillåtet att använda samlingar (t.ex. List). Exempel: 12p

```
Sekvenser
a1 = [3, 1, 2, 3, 0, 1, 5]
a2 = [9, 4, 7, 1, 2, 6, 7]
a3 = [1, 2, 3, 4, 7, 3, 8]
a4 = [3, 0, 1, 5, 3, 0, 5]

commonSeq( a1, a2, 2) ger [1, 2]
commonSeq( a1, a2, 3) ger []
commonSeq( a1, a3, 3) ger [1, 2, 3]
commonSeq( a1, a4, 4) ger [3, 0, 1, 5]
commonSeq( a1, a1, 7) ger [3, 1, 2, 3, 0, 1, 5]
```

5. Vi vill översätta svenska meningar till ”allspråket”. Detta sker genom att lägga till ”all” efter den första konsonanten i alla ord. Exempel: 7p

```
"Hej på dig"        ->   "Hallej pallå dallig"
"Ost är otrevligt"  ->   "Osallt ärall otallrevligt"
"Bo bodde på en ö"  ->   "Ballo ballodde pallå enall ö"
```

   Skriv en metod String toAll(String orig) som utför översättningen. Parametern orig är den svenska meningen och resultatet är en ny mening med den översatta texten. Ni kan bortse från skiljetecken och vi antar att alla ord i meningen åtskiljs av exakt ett mellanslag. Färdiga metoder (från olika klasser) som får användas se appendix.

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. Ni *måste* rita! 6p

```
A a1 = new A(1);
A a2 = new A( 2, a1); // Before
doIt(a2);             // Call
a1 = null;           // After

void doIt(A a) {
    A t = new A(3);
    t.n = a.n;
    a.n = t;
}

class A {
    int i;
    A n;
    A( int i, A n ){
        this.i = i;
        this.n = n;
    }
    A( int i ){
        this.i = i;
    }
}
```

7. Implementera en OO-modell för spelet That's life, se bild. Spelet har en spelplan bestående av brickor där spelarna flyttar sina pjäser utifrån tärningsresultat. OBS! Du behöver aldrig skriva set/get-metoder, vi antar att dessa finns. Färdiga metoder som får användas se appendix.

   a) Skriv en klass Tile för brickor. En bricka har ett visst antal poäng. Poäng anges då man skapar brickan. 2p

   b) Skriv en klass Player för en spelare. En spelare har ett namn och en lista med brickor (spelaren kan samla på brickor). Det skall finnas en metod som lägger till en bricka i spelarens lista. Namn anges då man skapar en spelare. 2p

   c) Skriv en klass Piece som skall representera en spelpjäs. En Piece har en färg, en spelare som äger den och en position, d.v.s. den Tile den står på. Alla data skall anges då pjäsen skapas. 3p

   d) Skriv en klass Game för hela spelet. Game har en spelplan i form av en lista med Tiles, en lista med Piece för pjäserna samt en aktuell pjäs (den som skall flyttas). All data sätts då ett Game-objekt skapas. 3p

   e) Lägg till en metod move(int diceResult) i Game. Metoden flyttar aktuell pjäs enligt parametern diceResult. Om det efter förflyttningen är tomt på brickan där pjäsen stod skall metoden ta bort brickan från spelplanen och tilldela denna till spelaren som äger pjäsen (spelplanen krymper) 5p

   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 kodavsnitt //a-//f: 6p

   a) Om det kompilerar eller...

   b) ... om det ger körningsfel eller ...

   c) ... om det fungerar och i så fall vad som skrivs ut.

   Du måste ge en kort motivering!

```
//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();

// --- Interfaces and classes
public interface IA { public void doA(); }
public interface IX { public 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

```
// Tillåtna metoder för uppg. 5
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.
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.
- toString(), omvandlar StringBuilder-objektet till en String.

// Tillåtna metoder för uppg. 7
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
```