Omtentamen för ID1004 Objektorienterad programmering HT11, 29 maj 2012, 09-13

Denna tentamen examinerar 3 högskolepoäng av kursen.

Inga hjälpmedel är tillåtna.

Tentamen består av 12 frågor. Varje fråga kan ge mellan 1 och 4 poäng. Tentamensbetyget A-F bestäms i huvudsak av den ackumulerade poängsumman (korrekt besvarade frågor) enligt tabellen nedan. Examinator förbehåller sig möjligheten att vid gränsfall justera tentamensbetyget med hänsyn till en bedömning av helhetsintrycket.

Poäng	0-10	11-13	14-17	18-21	22-25	26-28
Tentamens-betyg	F	E	D	C	B	A

Läs noga igenom alla frågorna först och planera tiden. Lycka till!

1 Grundläggande syntaktiska konstruktioner och begrepp

1.1 (1p) Föreslå en primitiv datatyp lämplig för att lagra heltal mellan 0 och en miljon.

1.1 int eller long, (men även float eller double)

1.2 (2p) Följande variabelinitiering har flera fel. Skriv om den så att den blir korrekt och följer stavningskonventionerna i Java.

```
private stattic double liter per minut == 3,8
```

1.2 private static double literPerMinut = 3.8;

```
void minMetod(int p) {…}
void minMetod(long p) {…}
void minMetod(char p) {…}
```

Kodfigur 1

1.3 (1p) Se kodfigur 1. Hur vet Java-kompilatorn vilken metod som skall anropas? (Vi bryr oss inte om vad metoden gör.)

1.3 Kompilatorn väljer rätt metod genom att titta på vilken datatyp parametern i det aktuella anropet har.

```
import java.io.*;
…
String someFile = …;
…
try {
  boolean success = true;
  FileInputStream fis = new FileInputStream(someFile);
  …
} catch (FileNotFoundException fex) {
  success = false;
}
if (success) { ... }
```

Kodfigur 2

1.4 (2p) Se kodfigur 2. I koden finns ett problem med felhanteringen. Vilket?

1.4: Variabeln success måste vara deklarerad utanför try-catch-satsen för att vara nåbar av den avslutande if-satsen.T ex:

```
boolean success = true;
try {
  FileInputStream fis = new FileInputStream(someFile);
  …
} catch (FileNotFoundException fex) {
  success = false;
}
if (success) { … }
```

2 Grundläggande datalogiska begrepp och relationer

```
int w = 314;
System.out.println(--w++); // rad 1
System.out.println(w); // rad 2
```

Kodfigur 3

2.1 (2p) Se kodfigur 3. De tre raderna exekveras i sekvens. Vilka värden kommer att skrivas ut av rad 1 och rad 2?

2.1 Rad 1 skriver ut 313, rad 2 skriver ut 314.

```
int limit = 0;
/* loop A */
```

```
for (int i = 0; i < limit; i++) { System.out.println(i); }
/* loop B */
int i = 0; while (i < limit) { System.out.println(i); i++;}
/* loop C */
int i = 0; do {System.out.println(i); i++;} while (i < limit);
```

Kodfigur 4

2.2 (3p) Se kodfigur 4. Vilken av dessa loopar gör flest varv, och varför är det så?

2.2: Loop C (1 gång) eftersom en DO-loop alltid exekverar loop-kroppen minst en gång. Loop A och B kommer aldrig att exekvera sin loop-kropp.

2.3 (3p) Se kodfigur 4. Hur många varv gör varje loop om variabeln limit initieras till 1?

2.3: Alla tre looparna gör 1 varv vardera.

```
if (a < b) if (b < c) f = true; else f = false; else f = false;
```

Kodfigur 5

2.4 (4p) Se kodfigur 5. Skriv om koden så att ingen eller högst en if-sats används.

2.4: Vi börjar med att skriva om koden med blockstruktur:

```
if (a < b) {
    if (b < c) {
      f = true;
    } else {
      f = false;
    }
} else {
    f = false;
}
```

Vi ser då att f alltid kommer att bli tilldelad antingen true eller false, och att den bara blir true om båda villkoren är sanna. Det ger oss möjlighet att använda ett &&-uttryck, och det finns flera sätt:

```
f = false;                              // alt. 1
if ((a < b) && (b < c)) f = true;

if ((a < b) && (b < c)) {               // alt. 2
  f = true;
} else {
  f = false;
}

f = (a < b) && (b < c) ? true : false; // alt. 3
```

```
f = (a < b) && (b < c);                // alt. 4
```

3 Grundläggande objektorienteringsbegrepp

3.1 (2p) Förklara varför man har deklarationerna `private`, `protected`, (inget), och `public` för metoder i klasser, och beskriv deklarationernas särdrag.

3.1 Dessa åtkomstdeklarationer används för att styra vilka metoder i en klass som skall vara synliga utåt för andra klasser (public), endast synliga för andra klasser i samma package (inget/default), endast synliga inom klassen och dess underklasser (protected) eller bara synliga i den egna klassen (private). Detta är en central del i begreppet inkapsling, dvs möjligheten att dölja det interna maskineriet i en klass för andra klasser (och deras programmerare).

3.2 (3p) Beskriv kortfattat (det är inget krav att skriva kod) hur man objektorienterat kan modellera varje individuellt spelkort i en kortlek. Vi antar en kortlek med 52 kort, inga jokrar, fyra färger (hjärter, spader, ruter, klöver) och 13 valörer (2-10, knekt, dam, kung, äss).

3.2 En klass Spelkort kan innehålla ett identitetsnummer (0-51) som exakt bestämmer vilket kort det är. Kortets id kan därefter översättas till ett index till färg genom att heltalsdividera med 13, och ett index till valör genom att beräkna id modulo 13. Dessa index kan därefter användas för att slå upp namn på färg och valör i arrayer eller enums.

Ett annat sätt är att skapa en klasshierarki med en superklass som innehåller valörerna och sedan fyra underklasser, en för varje färg. Det är dock inte självklart att göra så eftersom egenskaperna valör och färg är ortogonala från varandra; det vore precis lika korrekt att skapa en superklass med de fyra färgerna och en underklass för var och en av de tretton valörerna. Eftersom ingen av egenskaperna färg och valör är underordnad den andra så finns ingen naturlig hierarki att följa.

3.3 (3p) Med uppgift 3.2 i åtanke, beskriv kortfattat (det är inget krav att skriva kod) en objektorienterad modell för själva kortleken. Kortleken ska ha ett gränssnitt för att (a) samla in och blanda korten, (b) dela ut nästa kort och (c) informera om att inga kort finns kvar (alla är utdelade).

3.3 En kortlek kan modelleras som en egen klass som innehåller två listor. Den ena listan innehåller ej utdelade kort, den andra listan utdelade kort. (a) En metod blanda() för över alla utdelade kort till de ej utdelade och randomiserar ordningen. (b) Vid utdelning av kort flyttas nästa kort från ej utdelade till utdelade kort, och en referens till kortet returneras.(c) Om inget ej utdelat kort finns kvar kan metoden returnera null eller kasta en exception.

3.4 (2p) Antag att du som medlem i ett större mjukvaruprojekt har fått i uppgift att skriva ett flertal Java-klasser. Projektledningen har beslutat att varje sådan klass ska implementera interface-klassen MegaFace, vilket innebär att varje klass du skriver måste implementera alla metoder i MegaFace. Dessvärre så har MegaFace ganska många metoder, och det ser ut som om fler kommer att läggas till snart. Till saken hör att de klasser du skriver bara behöver tillhandahålla ett fåtal metoder i MegaFace, de övriga kommer aldrig att anropas och kan vara tomma implementationer. Du inser att det kommer att bli mycket extra skrivande och ändrande för att hålla dina klasser i synk med MegaFace.

Föreslå en strategi som minimerar mängden kod som du måste underhålla, och förklara varför det är ett lämpligt sätt.

3.4 Skriv en adapterklass, dvs en klass som implementerar alla metoder i MegaFace på enklaste sätt. Låt därefter projektklasserna ärva från adapterklassen. Varje projektklass behöver då endast implementera (override) de metoder som behövs i just den klassen. Ändringar i MegaFace behöver endast följas upp i adapterklassen och eventuellt även i de projektklasser som har egna implementationer.