Ordinarie tentamen för ID1004 Objektorienterad programmering, 30 maj 2013, 8-13

Denna tentamen examinerar 3.5 högskolepoäng av kursen.

Inga hjälpmedel är tillåtna.

Tentamen består av tre sektioner. För ett godkänt betyg på hela tentamen måste varje sektion ha minst ett godkänt svar. 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	0-11	12-16	17-21	22-26	27-31	32-35
Tentamens-betyg	F	E	D	C	B	A

Läs noga igenom alla frågorna först och planera tiden.

Om inget annat uttryckligen sägs så visas och efterfrågas kod och uttryck ur Java version 6 eller 7.

Grundläggande syntaktiska konstruktioner och begrepp

1. (3p) Inspektera deklarationerna nedan och anmärk för varje om något är fel eller inte.

a)
```
private static final double log10/log2 =
Math.log(10d)/Math.log(2d);
```

Identifieraren log10/log2 innehåller ett otillåtet tecken: ’/’

b)
```
Integer i, j, k = 0;
```

Detta är en korrekt deklaration och initiering. Datatypen är en referenstyp till klassen Integer. Variablerna i och j initieras till null. Med hjälp av s k autoboxing initieras k på ett sätt som motsvaras av

```
Integer k = new Integer(0);
```

c)
```
public String castor = "pollux".length();
```

Det går inte att initiera en strängtyp till en int. Längden på strängen ”pollux” är 6, och det är detta värde som ges som initiering.

2. (3p) Inspektera följande tre kodfragment och anmärk på konkreta och möjliga problem.

a)
```
for (int i = 0; i < numbers.length; i++) {
  if (numbers[i] == wins) {
    int winner = i;
  }
}
System.out.println("Vinnaren är lott " + winner);
```

Variabeln winner deklareras inuti if-satsen, men refereras till utanför, i println-satsen. Antingen kommer detta att leda till ett kompileringsfel, eller så finns det annan variabel med samma namn som blir skuggad av den inre winner. Vi måste också anta att variabeln wins är deklarerad tidigare i programmet.

b)
```
public class Hello {
  String helloWorld = "Hello world";
  public static void main(String[] args) {
    System.out.println(helloWorld);
  }
}
```

Detta lilla program kommer inte att kompilera eftersom den statiska main-metoden försöker referera en icke-statisk variabel. Om man ändrar argumentet till println så fungerar det: `System.out.println(new Hello().helloWorld);`

c)
```
String[] names = // … kommer från användaren eller en fil
public int findIndexOf(String name) {
  for (int i = 0; i < names.length; i++) {
    if (name == names[i]) return i;
  }
}
```

Användandet av identitetsoperatorn == innebär att parametern name måste referera till ett objekt i arrayen names för att hittas. Om man vill hitta strängar som har samma lydelse men som finns i olika objekt bör man istället använda `String.equals(String)`

3. (5p) Fyll i de primitiva datatypernas namn:

Datatyp	Representation	Minsta värde	Högsta värde
byte	heltal	-128	127
double	flyttal	-1.7E+308	1.7E+308
int	heltal	-2 147 483 648	2 147 483 647
short	heltal	-32 768	32 767
float	flyttal	-3.4E+38	3.4E+38

4. (3p) Vad händer när följande programsatser exekveras?

a) `for(;;) break;`

Den oändliga for-loopen går in i sitt första varv och möter omedelbart ett `break` som lämnar loopen. I praktiken händer ingenting med programmet.

b) `int i = 1; while (i < 0) System.out.println("I am stuck!");`

Variabeln i kommer att initieras till 1. Värdet kommer att testas mot 0, men loopen kommer inte att köras eftersom 1 inte är mindre än 0.

c) `{;}`

Ingenting händer, det är ett block med en tom programsats och därmed ett tomt block.

Grundläggande datalogiska begrepp och relationer

5. (4p) Kasta om ordningen på följande programsatser (a) så att variabeln a har värdet 1 när den skrivs ut. Svara med en radnummersekvens, koden nedan har sekvensen 1 2 3 4 5 6.

a)
```
1: int b = 4;
2: int c = b + 1;
3: int a = 16;
4: a /= c * b;
5: System.out.println(a);
6: a += b;
```

316245 123645 132645 312645 136245 (notera att alla dessa slutar med 245 eller 645)

b) Det finns tre sekvenser som skriver ut 0 (noll). Vilka är de?

123456 132456 312456 (dessa slutar alla med 456. Dessa rader måste komma sist och det finns bara tre olika sätt att ordna initieringen av variablerna)

6. (3p) Antag att ett program innehåller tre variabler deklarerade som:

```
boolean p, q, r;
```

Med Java-syntax, skriv uttryck som skulle kunna användas i en if-sats eller en while-loop för att testa:

a) Minst en av variablerna har värdet `true`

```
(p || q || r)
```

b) Alla variablerna har värdet `false`

```
(!p && !q && !r)
```

eller

```
(!(p || q || r))
```

c) Två (och endast två) av variablerna har värdet `true`

```
((p && q && !r) || (p && !q && r) || (!p && q && r))
```

7. (2p) Antag att variabeln `double u` innehåller ett stickprov (sample) av en signal. Det förväntade området är -1.0 — +1.0. Skriv med Java-syntax uttryck som testar om:

a) Stickprovet är svagare än -60 dB (±0.001)

```
((-0.001 < u) && (u < 0.001))
```

eller

```
(Math.abs(u) < 0.001)
```

b) Stickprovet är starkare än 0 dB (±1.0)

```
((u < -1.0) || (1.0 < u))
```

eller

```
(1.0 < Math.abs(u))
```

8. (3p) Arrayen `double [] buf` innehåller stickprov (samples) av en signal (som i uppgift 7).

a) Skriv en loop som beräknar medelvärdet av stickproven i arrayen.

```
double sum = 0d;
for (double v : buf) sum += v;
double m = sum / (double) buf.length;
```

b) Skriv en loop som med start i första elementet sätter var fjärde element i arrayen till noll. Tänk på risken att skapa en ArrayIndexOutOfBoundsException.

```
for (int i = 0; i < buf.length; i += 4) buf[i] = 0.0;
```

c) Skriv en **while-loop** som går från första elementet till dess att signalen byter tecken eller sista elementet nås. Tips: Metoden `public static double Math.signum(double a)` i standardbiblioteket ger -1, 0 eller 1 beroende på om a är negativt, noll eller positivt.

```
double sign = Math.signum(buf[0]);
int i = 1;
while ((i < buf.length) && (sign == Math.signum(buf[i]))) i++;
```

Grundläggande objektorienteringsbegrepp och komplexitet

9. (1p) Redogör kortfattat för relationen mellan begreppen *klass* och *objekt*.

Klassen definierar egenskaperna hos alla objekt som är instanser av klassen. De enskilda objekten är lika varandra eftersom de tillhör samma klass, de har samma metoder och variabler. Samtidigt kan de vara olika varandra, eftersom de data som instansvariablerna innehåller kan skilja sig åt eller referera till olika objekt.

10. (3p) Beskrev tre viktiga egenskaper hos *inkapsling* i programmeringsspråket Java.

Inkapsling (1) skyddar instansvariabler från yttre åtkomst, (2) erbjuder definierade operationer i form av metoder för att förändra och läsa instansvariablerna på ett kontrollerat sätt och (3) detta åstadkoms med hjälp av skyddsnivåer som private, protected, default och public.

11. (3p) Redogör så gott du kan för hur *arv, interface-klasser* och *abstrakta* klasser hänger ihop.

Arv – interface: En klass kan bara ha en enda superklass som den ärver (extends) från. Klassen kan däremot implementera mer än ett interface. Till skillnad från klasser så kan interface ärva från mer än ett interface.
Abstrakt klass – interface: En abstrakt klass innehåller minst en abstrakt metod, och kan innehålla konkreta metoder och variabler. Ett interface innehåller bara publika abstrakta metoder och kan inte ha variabler (men kan ha statiska konstanter).
Arv – abstrakt klass: En klass som ärver från en abstrakt klass måste skriva implementationer för alla abstrakta metoder, annars blir den ärvande klassen också abstrakt.

12. (2p) Vilken av dessa två alternativa metoder för beräkning av fakultetsfunktionen anser du vara mer effektiv, och varför det?

```
public int fakultetI(int n) {
          int r = 1;
          for (int i = 1; i <= n; i++) r *= i;
          return r;
}

public int fakultetR(int n) {
          return (1 < n) ? n * fakultetR(n - 1) : 1;
}
```

Fakultetsfunktionen skrivs vanligen n!, där 0! = 1! = 1, n!= 1*2*3*…*n. T ex 3! = 1*2*3 = 6.

Funktionen `fakultetI` är iterativ och använder sig av tre variabler i beräkningen. I varje iteration så ska i och r updateras (en addition, en multiplikation), i och n jämföras. Mängden minne är alltså densamma oavsett antalet iterationer.

Funktionen `fakultetR` är rekursiv och använder sig av en variabel, en jämförelse mot en konstant och ett metodanrop. Även här behövs en addition och en multiplikation. Att göra det rekursiva metodanropet tar också lite tid. Varje anrop av metoden behöver sin egen version av parametervariabeln n, vilket betyder att mängden minne som krävs för att utföra beräkningen beror på värdet av parametern n.

Båda funktionerna använder iteration för beräkningen, men eftersom `fakultetR` kräver minne och ett antal metodanrop som beror på parametern så är `fakultetI` mer effektiv.

Det kan också sägas att även om principen i resonemanget är korrekt, så är i just detta exempel skillnaden försumbar. Detta beror på att det största värdet på n i dessa implementationer är 12.

12!= 479 001 600 vilket ryms i en int vars största värde är 2 147 483 647, medan 13! = 6 227 020 800 och alltså inte får plats.

En long håller 2^{63} = 9 223 372 036 854 775 807 men den räcker bara till att lagra 20!. När man har så få tillåtna parametrar kan man lika gärna kan använda en array med förberäknade resultat.

Lycka till!