Ordinarie tentamen för ID1004 Objektorienterad programmering, 15 december 2012, 9-13

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 | 12-15 | 16-19 | 20-25 | 26-29 | 30-33 |
|---|---|---|---|---|---|---|
| Tentamens-betyg | F | E | D | C | B | A |

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

## Grundläggande syntaktiska konstruktioner och begrepp

1. (3p) Inspektera deklarationerna nedan och anmärk på hittade fel.

```
(a)    private lång jätteStortTal = 999887765439L
(b)    final double PI = 3,141 592 653 589 793;
(c)    protected getElement(int index) {  }
```

(a) Namnet *lång* är inte en numerisk primitiv datatyp i Java. Deklarationen avslutas inte med semikolon.
(b) Det ska vara decimalpunkt istället för komma. Det går inte att dela upp en numerisk konstant med mellanslag.
(c) Metoden saknar returtyp.

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

```
(a)    if (n = 99)
         n = 0;
         listIsEmpty = true;
(b)    for(i = 0; i < 10; i++)
         for(j = 0; j < 10; i++) mx[i][j] = -1;
(c)    public class Pony {
         private final int strength;
         public Pony (int strong) { strength = strong; }
         public boolean isStrongerThan(Pony otherPony) {
           int strength = otherPony.getStrength();
```

```
            return strength > strength;
        }
    }
```

(a) If-satsen saknar explicit satsblock '{ }' vilket gör det svårt att veta om kodraden "listIsEmpty = true" är tänkt att vara beroende av villkoret eller inte. Indenteringen tyder på att så kan vara fallet. Dessutom är villkorsuttrycket en tilldelning av en numerisk typ vilket kommer att leda till kompileringsfel. Operatorn borde vara '=='.

(b) For-looparna saknar satsblock (vilket dock inte krävs men är bra för ökad tydlighet). Den inre loopen ökar också på variabeln i (det ser ut som om det borde vara j) vilket kommer att få effekten att den inre loopen rusar iväg med mx[0][0], mx[1][0]… ända tills den orsakar en ArrayIndexOutOfBoundsException med mx[i][0].

(c) I metoden isStrongerThan så skapas en lokal variabel strength fast det redan finns en klassvariabel med samma namn. Skuggningen av klassvariabeln gör att returvillkoret alltid kommer att vara falskt, och alla ponnyer verkar vara precis lika starka. Dessutom så finns ingen implementation av metoden getStrength så klassen kommer inte ens att kompilera.

3. (2p) Vilken eller vilka av variablerna i klassen X_Files går att läsa (från vilket package som helst) utan att först skapa en instans av klassen?

```
public class X_Files {
  private int secret = 0;
  public static long hidden = 0;
  String denial = null;
  public int getSecret() { return secret; }
  protected final short proof = 0;
  private static final double bluff = 0;
}
```

Variabeln *hidden* är public och static och därför åtkomlig utan en instans av X_Files. De andra variablerna är skyddade. Åtkomstmetoden getSecret() räknas inte heller eftersom den kräver en instans att anropa metoden i.

4. (2p) Vilken eller vilka av följande metoder går att använda som parameter i ett metodanrop eller på den högra sidan om '=' i en variabeltilldelning?

```
(a)    public int setValue(int value)
(b)    private short maxSampleInRange(int lower, int higher)
(c)    public void apply(Lambda ell, List lst)
(d)    int unterminate(String s)
(e)    protected void calculateSums(int [] ar)
```

För att kunna användas till höger om tilldelningsoperatorn (=) eller i en parameteröverföring så måste metoden returnera ett värde. (a), (b),(d) ovan gör detta men inte (c) och (e) för de är deklarerade som *void,* dvs ingen returtyp.

## Grundläggande datalogiska begrepp och relationer

5. (3p) I tabellen nedan finns tre separata och ömsesidigt oberoende exempel på sekvenser av programsatser. För varje sekvens:

(a) Har programsatsernas ordningsföljd någon betydelse?

(b) Motivera svaret på (a)

Variablerna i sekvenserna kan antas vara deklarerade av typen int, om inte annat anges.

| Sekvens 1 | Sekvens 2 | Sekvens 3 |
|---|---|---|
| `a = a + b;`<br>`a += c;`<br>`a++;` | `d = d + e;`<br>`f += d;`<br>`d++;` | `String s;`<br>`s = ”Nummer: ” + i;`<br>`n = s.size();` |

I sekvens 1 spelar ordningen ingen roll. Sekvensen går att skriva om som:

a = a + b + c + 1

och eftersom ordningen inte spelar någon roll för addition så kan satserna utföras i godtycklig följd.

I sekvens 2 spelar ordningen roll eftersom f tilldelas värdet av d *efter* att d har uppdaterats med e, men *innan* d har ökats med 1. Om raden

f += d

placeras först så kommer f att ha ett lägre värde (f + d), och om den placeras sist så kommer f att ha ett högre värde (f + d + e + 1).

I sekvens 3 spelar ordningen roll eftersom variabeln s måste deklareras innan den tilldelas, och därefter tilldelas ett strängobjekt innan man kan referera till objektets metod size.

6. (5p) Nedan presenteras ett antal exempel på programmeringssituationer som brukar lösas med iteration. För varje situation, vilken typ av iterationssats (”loop statement”) anser du passar i just den situationen och varför då?

(a) I en array av String så ska elementet med index n tas bort och alla element med större index än n flyttas ett steg åt vänster (mot n).
(b) En metod ska anropas och dess returvärde inspekteras. Detta ska upprepas tills dess att returvärdet är noll.
(c) Alla element i en ArrayList ska skrivas ut.
(d) Så länge en List innehåller minst ett element så ska första elementet tas ut och bort från listan och behandlas.
(e) Vartannat element i en array ska skrivas ut.

(a) En for-loop med initiering, villkor och uppdatering. Detta eftersom man kommer att behöva ett index i arrayen.

(b) Om inspektionen av returvärdet inskränker sig till att kontrollera mot noll, så räcker det med:

```
while(method() != 0);
```

men det kan ju vara en mer ingående kontroll och saker som ska göras. I så fall är det kanske bättre med en do-loop:

```
int rtn = 0;
do {
  rtn = method();
  if (rtn…) /* Gör komplicerade saker */ ;
} while (rtn != 0);
```

(c) En for-each loop, till exempel:

```
for (T e : ArrayList<T>) { System.out.println(e);} // T står för en klass
```

eftersom den sortens loop är speciellt lämpad att gå igenom alla element i en datasamling.

(d) En while-loop, till exempel:

```
while (!myList.isEmpty()) {
  behandla (myList.remove(0));
}
```

eftersom vi börjar med att testa om det finns något element i listan.

(e) Här behöver vi återigen ett index, så då är en for-loop med initiering, villkor och uppdatering bäst. Till exempel:

```
for (int i = 0; i < ar.length; i += 2) {
  System.out.println(ar[i]);
}
```

7. (3p) Konstruera boolska uttryck (villkor) för följande specifikationer:

(a) Veckonumret v är utanför perioden vecka 22 – 35.

(b) Vattentemperaturen t är 40 – 80 grader samtidigt som trycket p är mindre än 9.6 men minst 3.3.

(c) Värdet b är någonstans mellan a och c men vi vet inte vilken som är störst av a och c.

*Exempel: ”Snödjupet h är djupare än 20 cm”*
*Svar: (20 < h)*

```
(a)    ((v < 22) || (35 < v))  alt. !((22 <= v) && (v <= 35))
(b)    ((40 <= t) && (t <= 80) && (3.3 <= p) && (p < 9.6))
(c)    ((a <= b) && (b<= c)) || ((c <= b) && (b <= a))
```

8. (2p) Skriv om de två oberoende programsatserna nedan med hjälp av if- eller if-else-satser:

*Metoden int java.lang.Math.min(int a, int b) returnerar det mindre av två värden.*
*Metoden int java.lang.Math.max(int a, int b) returnerar det större av två värden.*

```
(a)    y = Math.min(size.height - 12, y);
(b)    x = Math.max(0, Math.min(x, 100));
```

```
(a)  if (size.height – 12 < y) {
       y = size.height – 12;
     }
```

```
(b)   if (x < 0) {
        x = 0;
      }
      else if (100 < x) {
        x = 100;
      }
```

## Grundläggande objektorienteringsbegrepp och komplexitet

9. (3p) Vad är polymorfism och vilka mekanismer kan man använda för att åstadkomma det?

Polymorfism (flerformighet) är ett sätt att styra exekveringen till olika kod beroende på objektens typ, snarare än genom switch- och if-satser. Den främsta mekanismen är att vid arv åsidosätta (*override*) vissa ärvda metoder med implementationer som är specifika för den egna typen (klassen). Detta gör att ett metodanrop till en metod deklarerad i en superklass kan styras ner i en underklass för särskild behandling.

En snarlik mekanism är implementation av interface varvid varje implementerande klass är tvungen att implementera alla metoder i interface-deklarationen. Detta medför att alla klasser som implementerar ett visst interface kan behandlas uniformt i termer av interfacet, men ändå ha ett eget beteende i sina implementationer (t ex interfacet java.util.List och dess implementationer i ArrayList, LinkedList osv).

10. (2p) Vilken nytta ger överlagrade (*overloaded*) metoder?

En överlagrad metod är en metod i en klass som finns i flera versioner med samma namn men med olika parametrar. Det gör att parameterlistan bestämmer vilken metod som kommer att anropas. Överlagring är en lokal form av polymorfism, i och med att data bestämmer vilken kod som skall exekveras. Den främsta fördelen är dock att det går att erbjuda den som skriver den kod som skall anropa metoden flera alternativ och att slippa tänka på vad metoden heter för en viss parametertyp. Ett bra exempel är metoderna System.out.print och System.out.println som finns i varianter för alla primitiva datatyper och objekt.

11. (3p) Inkapsling är ett viktigt begrepp. Besvara följande frågor:

(a) Vad är det som kapslas in?
(b) Hur kapslas det in, och i vad?
(c) Hur kommer man åt det som är inkapslat när man behöver göra det?

(a) Det är variabler (data) som kapslas in. Detta görs för att skydda variablerna från oönskad (okontrollerad) manipulation, som visserligen inte behöver vara fientlig, men som kan vara oavsiktlig och därför leda till buggar som är svåra att hitta.
(b) Variablernas åtkomst i en klass regleras av deklarationerna *public, (default), protected och private*. Konstanter deklareras med ordet *final* och går inte att ändra efter sin första tilldelning. Data placeras i klassvariabler och lever inuti en eller (oftast) flera instanser av sin klass.

(c) Åtkomst till data sker genom metoder. Så kallade accessor-metoder ger tillgång till informationen (läsning) och så kallade mutator-metoder ger möjlighet att uppdatera deras värden.

12. (2p) En array av int är sorterad i stigande ordning. En metod ska skriva ut alla element som är mindre än ett visst element, det s k gränselementet. Man vet inte var i arrayen gränselementet ligger. Vilket av följande alternativ är mest effektivt, och varför då?

(a) Gå igenom arrayen från slutet till början och börja skriva ut element så snart gränselementet har passerats.
(b) Sök efter gränselementet, ta dess index och skriv ut alla element mellan 0 och index-1.
(c) Gå igenom arrayen från början mot slutet och skriv ut element. Avbryt när gränselementet har hittats.

Alternativ (a) måste gå igenom hela arrayen varje gång. Alternativ (b) måste först göra en sökning vilket betyder läsningar i arrayen utöver dem som behövs för själva utskriften. Alternativ (c) går bara igenom de element som eftersöks och gör inte mer arbete än så. Därför är (c) den mest effektiva.

Visserligen är det så att (c) måste jämföra storleken på varje element för att se om det är dags att sluta, medan en binärsökning i (b) behöver långt färre jämförelser. Dock kräver sökningen ytterligare administration i form av metodanrop, iteration och variabler vilket blir en konstant kostnad. Det krävs därför ett väldigt stort antal element innan man tar igen det minskade antalet jämförelser och (b) blir mer effektiv. Ett så stort antal är det osannolikt att man vill skriva ut.

Lycka till!