Tentamen 2006-05-06
Objekt-orienterad programmering i Java, 5p
distanskurs

Uppsala Universitet
Instutitionen för informationsteknologi
Avdelningen för datalogi

Kursansvarig: Sven-Olof Nyström

May 2, 2007

Läs igenom följande instruktioner noggrant:

- Skrivtiden är fem timmar.
- Inga extra hjälpmedel är tillåtna.
- Skriv endast en lösning per blad och skriv namn på varje blad.
- Skriv inte på baksidan av bladen.
- Hänvisa inte mellan uppgifterna.
- Använd inte rödpenna.
- Oläsbara eller otydliga lösningar ger noll poäng per uppgift.
- Om tvetydigheter uppstår, gör intelligenta antaganden och ange dem.
- Om du har svårt att komma ihåg exakt hur någon metod eller klass i Javas standardbibliotek ser ut och fungerar, ange noggrant ditt antagande.

Maxpoäng är 40. För godkänd krävs vanligtvis 20 poäng och för väl godkänd 30 poäng.
Läs igenom hela tentan innan du börjar.
Om du fastnar på en uppgift, gå vidare till nästa—uppgifterna är inte nödvändigtvis ordnade i svårighetsgrad. Om en uppgift består av flera deluppgifter och du fastnar på en av dem, gå vidare till nästa deluppgift—det kan hända att efterföljande deluppgifter kan lösas utan det fullständiga svaret på tidigare uppgifter.

1. Frukt (6p)

Citroner kostar 1,50 kronor per styck, bananer kostar 11 kronor per kilo. Bananer och citroner är frukter.

(a) Skriv klasserna `Citron`, `Banan` och `Frukt`. Alla frukter ska ha en metod `double pris()` som returnerar fruktens pris, samt en metod `double vikt()` som returnerar fruktens vikt (gör ett rimligt antagande om vikten på citroner).

Ingen av klasserna får innehålla instansvariabler, förutom `Banan` där det finns en instansvariabel som lagrar bananens vikt. Klassen `Banan` har en konstruktor där vikten på bananen anges.

(b) Skriv en klass `Påse` som implementerar följande metoder:

- `void läggTill(Frukt b)` — placera en frukt i påsen
- `pris()` — beräkna det sammanlagda priset på frukterna i påsen.
- `totalVikt()` — beräkna den totala vikten av frukterna i påsen.

(c) Skriv ett testprogram som skapar en påse, lägger frukter av olika slag i påsen, och beräknar påsenspris och vikt.

Ge en lösning som använder arv och polymorfi.

2. Referenssemantik (5p)

(a) Betrakta följande program:

```
public class Box {
    String contents;
    Box (String s1) {
        contents = s1;
    }
    public void setContents(String s1) {
        contents = s1;
    }
    public String getContents() {
        return contents;
    }
}
```

Vad definierar klassen? Vad gör de olika metoderna?

(b) Med definitionen ovan, betrakta följande klass:

```
public class A {
  public static void main(String args[]) {
    Box y = new Box("hund");
    Box z = new Box("katt");
    // (1)
    y = z;
    // (2)
    y.setContents("giraff");
    // (3)
  }
}
```

Vilka värden har variablerna `y` och `z` vid positionerna (1), (2) och (3)?

(c) Med samma definition av klassen `Box` som ovan, betrakta följande klassdefinition:

```
public class B {
  public static void main(String args[]) {
    Box y = new Box("ko");
    Box z = new Box("kalv");
    // (4)

    Box w = m(y, z);
    // (5)

  }

  public static Box m(Box r1, Box r2) {
      r1 = new Box ("elefant");
      r2.setContents("en "+ r2.getContents() + " i lådan");
      // (6)
      return r2;
  }
}
```

Vilka värden har variablerna `y`, `z` och `w` vid positionerna (4), (5) och (6)?
Vilka värden har `r1` och `r2` vid position (6)?

3. Arv (5p)

Vi har tre klassdefinitioner:

```
class A { }
class B extends A { }
class C extends A { }
```

Anta att variablerna `a`, `b` och `c` är initialiserade enligt:

```
A a = new A();
B b = new B();
C c = new C();
```

Ange för var och en av följande satser om satsen *a*) kompilerar och exekverar korrekt, *b*) kompilerar utan fel men ger felmeddelande vid körning, eller *c*) ger fel vid kompilering. Motivera!

(a) `a = b;`

(b) `b = c;`

(c) `b = (B)c;`

(d) `b = (B)a;`

(e) `a = (A)b;`

Tips. Det kan hjälpa om du ersätter klasserna `A`, `B` och `C` med klasser där arvshierarkin är mera naturlig.

4. Primitiva typer, undantag, (5p)

(a) Java har ett antal primitiva datatyper (Skansholm kallar dem inbyggda enkla typer) som inte definieras av någon klass.
Ge några exempel på primitiva datatyper.

(b) På vilka sätt skiljer sig de värden av de primitiva datatyperna från andra objekt?

(c) Hur definieras nya typer av undantag (exceptions) i Java?

(d) Vissa undantag måste alltid fångas eller deklareras. Vilka är dom? Ge exempel.

(e) Betrakta följande kodfragment (du behöver inte bry dig om hur koden mellan "try" och första catch-klausulen fungerar).

```
try {
  String os = System.getProperty("os.name");
  if(os != null && os.startsWith("Windows")) {
    UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
  }
}
catch(NullPointerException aNullPointerException)
  {}
catch(Exception e)
  {}
```

Vad är resultatet av att man kapslar in koden med en try-catch sats på det här sättet? Vad händer om koden kastar (tex) en `NullPointerException` eller en `IOException`?

5. Figurer (7p)

Givet följande gränssnitt:

```
interface GraphicsObject {
  public double area();
  public void translate(double dx, double dy);
  public boolean inside( double x, double y);
}
```

Du ska implementera två grafiska objekt (figurer), `Square` och `Group`. (Ett mera realistisk exempel skulle förstås ha fler typer av grafiska objekt.) Objekt av klassen `Group` innehåller en mängd grafiska objekt. Du behöver inte definiera metoder för att addera grafiska objekt till en grupp, men det ska förstås vara möjligt att skriva såna metoder.

Du bör använda collection classes ur standardbiblioteket, och utnyttja polymorfi där det är lämpligt.

Metoden `translate()` ska förflytta figuren med dx i x-led och dy i y-led.

Metoden `inside()` ska returnera sant om punkten (x, y) ligger inne i figuren, falskt annars. För en grupp av figurer gäller det att kolla om punkten ligger inom någon av de figurer som ingår i gruppen.

(a) Skriv klasserna `Square` och `Group` som ska implementera GraphicsObject.

(b) Skriv en klass `GraphicsWindow` som ska innehålla ett antal grafiska objekt (`Square` och `Group`). Skriv metoderna

```
läggTill(GraphicsObject go);// lägg in go i fönstret
taBort(int x, int y);       // ta bort alla figurer som innehåller
                            // (x,y) enligt metoden inside()
double area();              // returnera summan av alla figurers yta
translate(int dx, int dy);  // förflytta alla figurer
```

Metoden `area()` ska helt enkelt summera alla figurers yta (dvs ni behöver inte ta hänsyn till att figurer eventuellt överlappar varandra).

6. På bio (7p)

Anta att en biograf lagrar en databas över filmvisningar.

Varje filmvisning innehåller information om vilken film som visades och vilka personer som såg filmen.

(a) Beskriv en lämplig representation av databasen (helst med collection classes). Du bör förstås beskriva de olika klasserna som ingår.

(b) Skriv en metod som givet en samling (collection) av personer och en samling av filmer returnerar de filmer som inte setts av någon av personerna.

(c) Antag att man vill veta vilken eller vilka filmer som setts av flest människor (det räknas inte om någon sett samma film flera gånger).

Skriv en metod som givet en databas (enligt ovan) tar fram en lista (dvs en collection) av de filmer som setts av flest personer. Du bör skriva metoden så att den bara går igenom filmvisningarna en gång, oavsett hur många filmer eller personer det finns i databasen.

I det fall flera filmer har setts av lika många människor returnerar du alltså flera filmer, annars bara en.

7. Utvecklingsmodeller (5p)

Beskriv de olika utvecklingsmodellerna vattenfallsmodellen, rational unified process (iterativ utveckling) och extreme programming.

Nämn några önskemål man kan ha på en utvecklingsprocess (det finns ett antal ganska självklara) och diskutera hur de olika utvecklingsmodellera uppfyller dem.

LYCKA TILL!

Sven-Olof

Appendix: något om samlingar och iteratorer

Metoder i gränssnittet Collection:

- add(obj): lägg in obj
- addAll(coll): lägg in alla objekt i coll
- clear(): tar bort samtliga objekt
- contains(obj): returnerar sant om obj finns i samlingen
- containsAll(coll): returnerar sant om alla objekt i coll finns i samlingen
- equals(coll): returnerar sant om coll och aktuell samling är lika.
- isEmpty(): returnerar sant om samlingen är tom
- iterator(): returnerar en Iterator som kan användas för att loopa genom samlingen
- remove(obj): tar bort obj ur samlingen
- removeAll(coll): tar bort objekten i coll ur samlingen
- retainAll(coll): tar bort alla objekt utom de som finns i coll ur samlingen
- size(): returnerar antal element i samlingen
- toArray(): returnerar en array med alla element i samlingen

Klasser som implementerar Collection:

- ArrayList (implementerar List)
- LinkedList (implementerar List)
- Vector (implementerar List)
- TreeSet (sorterad mängd, implementerar SortedSet)
- HashSet (mängd, implementerar Set)

Metoder i iteratorer:

- hasNext(): returnerar sant om det finns fler element
- next(): returnerar nästa element, flyttar ”pekaren”
- remove(): tar bort aktuellt element

Exempel på hur man använder iteratorer:

```
static void loop (Collection c) {
       for(Iterator i = c.iterator(); i.hasNext(); ) {
            Object obj = i.next();
            System.out.println( obj );
       }
   }
```

Metoder i gränsnittet Map (som implementeras av HashMap och TreeMap):

- int size() - returnerar antal (nyckel-värde)-par.
- void clear() - tar bort alla par i avbildningen.
- Object put(Object nyckel, Object värde) - lägger in (nyckel,värde)-paret, ersätter det gamla (det med samma nyckel) om det finns.
- Object get(Object nyckel) - returnerar värdet för nyckeln nyckel eller null om nyckel ej finns.
- Set entrySet() - returnerar ett Set med varje (nyckel-värde)-par i form av ett Map.Entry-element (se vidare nedan i loop-exemplet).
- Set keySet() - returnerar ett Set med bara nycklarna.
- Collection values() returnerar en samling med bara värdena.

Loop genom avbildning (mappen i exemplet) med utskrift av nyckel och värde:

```
Set mapAsSet = mappen.entrySet();
for (Iterator i = mapAsSet.iterator(); i.hasNext();) {
  Map.Entry me = (Map.Entry) i.next();
    System.out.println(me.getKey() + ": " + me.getValue());
}
```