Data och Informationsteknik / Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
Erland Holmström Göteborg 25 okt 2012

Examination in

Objektorienterad programvaruutveckling IT1 TDA545

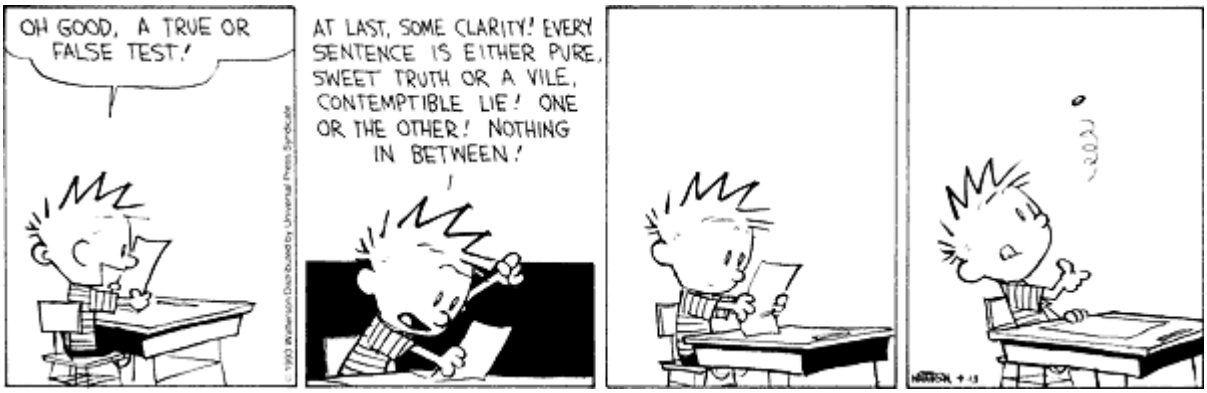
DAY: THURSDAY DATE: 2012-10-25 TIME: 8.30-12.30 ROOM: M

Responsible teacher:	Erland Holmström phone 1007, home 0708-710600
Results:	Are sent by mail from Ladok.
Solutions:	Are eventually posted on homepage.
Inspection of grading:	The exam can be found in our study expedition after posting of results.
	Time for complaints about grading are announced on homepage after the result are published or mail me and we find a time.
Grade limits:	CTH: 3=28p, 4=38p, 5= 48p, max 59p
Aids on the exam:	En sida **"Tentamneshjälpmedel – språkkonstruktioner i Java"**
utanför	på denna får man också skriva vad man vill **på de vita ytorna** textrutorna. **Lappen lämnas in med tentan.**

Observe:

- Start by reading through all questions so you can ask questions when I come. I usually will come after appr. 2 hours.
- All answers must be motivated when applicable.
- Write legible! Draw figures. Solutions that are difficult to read are not evaluated!
- Answer concisely and to the point.
- The advice and directions given during course must be followed.
- Programs should be written in Java, indent properly, use comments and so on.
- Start every new problem on a new sheet of paper.

Good Luck!



Problem 1. Uppvärmning

a) Testar: parameteröverföring, lämplighetsval

Metoderna i följande program utgör olika (något konstlade) försök att få en metod att lämna ifrån sig två tal satta till 5 respektive 6. Vad skrivs ut? Motivera (gärna med figur)! Gradera metoderna efter hur bra dom löser uppgiften.

```
import java.awt.*;
public class UPoint {
  public static void m1(int i, int j) {
    i = 5; j = 6;
  }
  public static void m2(int[] a) {
    a[0] = 5; a[1] = 6;
  }
  public static int[] m3() {
    int[] p = {5,6};
    return p;
  }
  public static Point m4() {
    return new Point(5,6);
  }
  public static void m5(Point p) {
    p.x = 5; p.y = 6;
  }
  public static void main(String[] par) {
    int s = 0, t = 1;
    m1(s, t); System.out.println(s);

    int[] q = new int[]{0,1};
    m2(q); System.out.println(q[0]);

    int[] p = m3(); System.out.println(p[0]);

    Point pt = new Point(0,1);
    pt = m4(); System.out.println(pt.x);

    pt = new Point(0,1);
    m5(pt); System.out.println(pt.x);
  }
}
```

(b-e Från bokens ”self study questions”)

b) Testar Scanner:

Antag att en indataström innehåller raderna 10_ft_↩12_ft ↩ (”_” står för mellanslag och ”↩” står för return) och att du läser med en Scanner som heter input. Skriv satser som tilldelar talet på första raden (10) till variablen width och talet på andra raden (12) till variabeln length.

c) Vad är skillnaden mellan följande två definitioner av Comparable?

```
interface Comparable {...}
public interface Comparable{...}
```

d) Vad är skillnaden mellan följande två definitioner av metoden graterThan i interfacet Comparable?

```
boolean graterThan(Comparable other);
public abstract boolean graterThan(Comparable other);
```

e) *Testar equals, överlagring.* Antag att `equals` metoden är överlagrad i `Circle` och `ColoredCircle` enligt:

```
public class Circle  {
  private int radius;
  public boolean equals(Object rhs) {
    if (rhs instanceof Circle) {
      return radius == (Circle)rhs.radius;
    } else {
      return false;
    }
  }
}

public class ColoredCircle extends Circle {
  private Color col;
  public boolean equals(Object rhs) {
    if (rhs instanceof ColoredCircle) {
      return super.equals(rhs)
                 && col.equals(ColoredCircle)rhs.col;
    } else {
      return false;
    }
  }
}
```

Dvs två cirklar är lika om deras radie är lika och två färgade cirklar är lika om dom är lika som cirklar och deras färg är lika. Giver definitionerna

```
Circle c1 = new Circle(3)
Circle c2 = new ColoredCircle(3, Color.Blue);
```

hur beräknas och vad är resultatet av uttrycken

```
c1.equals(c2); c2.equals(c1);
```

(10p)

Glöm inte svara på kursenkäten.

Exam Help lappen skall lämnas in med tentan,
ni får tillbaka den med tentan igen.

Problem 2. Testar: enkel klass, konstruktorer, metoder, booleska villkor, kasta exceptions, synlighet, switch-satsen, logiska operatorer, fält, strängar, överlagring, equals mm

Vi vill ha ett program som givet ett datum (tex 20121015 i form av ett heltal) skriver ut nästa datum på formen "tomorrow is 16 oktober 2012". Om datumet som anges inte är ett riktigt datum (tex månad=13) skall en felutskrift ske. Dvs det skall se ut ungefär som

```
% java TestaDatum
Enter a date (yyyymmdd): 20121231
tomorrow is 1 january 2013
```

Eftersom jag vill testa era kunskaper på olika områden är uppgiften ganska styrd enligt nedan. För att underlätta för rättarna är även namnen bestämda. När du skriver en metod skall du använda de andra metoderna (om dom behövs) även om du inte skrivit dem (så läs hela uppgiften innan du börjar). Du skall *inte* använda klasser från Javas API i den här uppgiften.

Först skall du skriva en modellklass för att hantera ett datum (Datum). Klassen skall också innehålla ett antal metoder. Samtliga är instansmetoder och skall placeras inuti klassen om inget annat sägs.

a) I uppg. c behöver du en metod i klassen som avgör hur många dagar det är i en månad.
Skriv metoden `public int daysInMonth()` som avgör hur många dagar det är i datumet som klassen representerar. Du skall använda en switch sats. Antag att månaden lagras som ett heltal.

b) I förra uppgiften behövde du en metod som avgör om ett år är ett skottår. Skriv ~~(klass-)~~metoden `public boolean isLeapYear()`. Information om skottår finns nedan. För poäng får du bara använda test på likhet, logiska operatorer samt "%". (samt return såklart).

c) Börja nu skriva på själva klassen `Datum`. Metoderna ovan antas finnas i klassen nu. Du skall här bara ha med *tre* konstruktorer, en med ett datum som parameter och en med tre heltal för år, månad och dag och en med ett heltal på formen yyyymmdd som alla skapar ett nytt datum. Den sista skall använda "/" och "%", för att dela upp sin parameter i år, månad och dag.

   Eventuella variabler som behövs skall också vara med, lagra som heltal. Du skall också skriva (och använda) metoden

   ```
   private boolean isDateOk() throws IllegalArgumentException
   ```

   som kontrollerar att det lagrade datumet är "ok" enligt $0<=$år$<=3000$, $1<=$månad$<=12$ och $1<=$dag$<=$antalet dagar i månaden. Om det inte är uppfyllt kastas en exception med lämplig text tex (för första hittade felet) enligt "`date not ok: year < 0  or year > 3000`".

d) Dags för metoden `public Datum tomorrow()` som skapar ett datum som representerar morgondagens datum.

e) För att kunna skriva ut ett månadsnamn behöver man kunna översätta från ett månadsnummer till en sträng dvs tex månad 3 -> "Mars". Skriv metoden

   ```
   public String monthNbr2month().
   ```

   Här vill jag att du *inte* använder if eller switch utan ett fält för att lösa problemet.

f) Skriv en equals metod som är arvs-säker.

g) Det är kanske bäst att objektet själv formaterar utskriften. Eftersom man vill kunna skriva ut på flera sätt så kan vi ha en parameter till toString enligt,

`public String toString(int toStringFormat).`

Om `toStringFormat`=0 returneras strängen på formen ”18 oktober 2012”, om den är 1 på formen ”18/10 2012” och alla andra värden ger formen ”2012 10 18”

h) Det är olämpligt att skriva just `toString` på det här viset med parameter, varför? Skriv en som ser ut som den bör se ut och som returnerar på formen ”18 oktober 2012”. Kan bägge existera samtidigt i vår klass? Motivera!

i) Sista uppgiften här är att skriva en main metod som använder modellen ovan. Det skall läsa in ett heltal från användaren, skapa ett datum, samt anropa `tomorrow` på lämpligt sätt för att åstadkomma utskriften ovan. Utskriften skall naturligtvis ske med `toString()`.

Några fakta bra att ha:

Antal dagar i månaderna: Månaderna 1, 3, 5, 7, 8, 10 och 12 har 31 dagar, 4, 6, 9 och 11 har 30 dagar och februari har 28 utom under skottår då den har 29.

Skottår: År 2000 är tex ett skottår. Skottår behövs eftersom ett år egentligen är 365.242 dagar långt, inte 365. För att kompensera för detta lägger man in en extra dag, 29 februari, på de år som är jämt delbara med 4. (365*4= 1460, (1460+1)/4 = 365.25) Eftersom detta ger ett år som är 365.25 dagar så blir det inte helt rätt heller, man behöver dra bort lite.

Här är de kompletta reglerna för skottår i den gregorianska kalendern: Ett år är skottår om det är jämnt delbart med 4. Sekelskiftesår (dvs delbart med 100) är dock endast skottår om det dessutom är jämnt delbart med 400. År 1800, 1900, 2100, 2200, 2300 och 2500 är alltså inte skottår, medan år 2000 och 2400 är det. Dvs

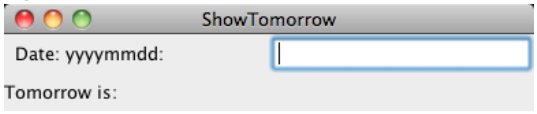
- år som är delbara med 4 är skottår men
  - år som dessutom är delbara med 100 är det inte men
    - år som dessutom är delbara med 400 är det.

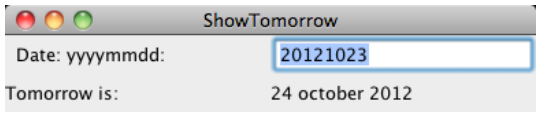
(20p)

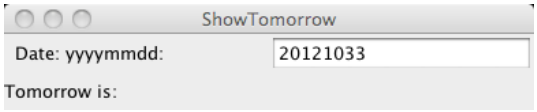
Problem 3. Testar: swing, händelsehantering

Vi skall också göra utskriften i förra uppgiften lite trevligare.

När användaren skriver in ett datum skall du använda klassen Datum från ovan för att skriva morgondagens datum enl.figurer nedan.
Använd en gridlayout, fånga exceptions vid fel (som sista inmatningen nedan) och gör en felruta.







(10p)

Problem 4. *Testar: läsa API*

När vi nu ändå håller på med datum så kan vi ju även titta på klassen Calendar som finns i Javas API (förkortad javadoc i slutet). Använd Calendar för att skriva ut tiden ”just nu” på formatet ”11:16:02 Thursday 25 september 2012”. Observera att timmar, minuter och sekunder < 10 skall skrivas som tex ”07” inte ”7”.

Du kan anta att det finns klassmetoder tillgängliga för (se motsvarande instansmetoder i förra uppgiften för beskrivning)

```
static String monthNbr2month(int monthNumber)
static String dayNbr2day(int dayNumber)
```

(6p)

Problem 5. *Testar: rekursion* Skriv en *rekursiv* funktion

```
public static int[] fromTo(int a, int b)
```

som givet två heltal a och b returnerar en vektor med innehåll (a, a+1, a+2, …, b). Om a>b returneras en tom vektor.

Den här typen av rekursiva funktioner är ganska knöliga att skriva i Java, du behöver en wrapper.

(7p)

Problem 6. *Testar: exceptions, enkla kontrollstrukturer, slumptal*

Uppgiften nedan gavs på en tidigare tenta: (något förkortad text här)

En A/D omvandlare (Analog/Digital omvanlare) är kopplat till någon form av analog signal tex en voltmätare, och omvandlar den analoga signalen till digitala värden tex i form av en double.

Antag att en A/D-omvandlare representeras med ett interface enligt

```
/**
* Skapar en AD konverterare som kastar en exception om
* värdena som läses ligger utanför lower..upper.
* Lower och upper ges som indata till konstruktorn.*/
public interface ADConverter {
  public double read() throws UnderVoltage, OverVoltage;
}
```

Metoden `read()` läser A/D-omvandlarens värden och returnerar dem. Om värdena ligger utanför intervallet `lower..upper` kastas en exception, `UnderVoltage` eller `OverVoltage`. Man kan alltså bestämma mellan vilka värden man accepterar att mätvärdena skall vara vid konstruktionen av objektet.

Du skall skriva ….. (förra uppgiften är inte så intressant här)

Antag nu att du vill provköra ditt program men utrustningen för AD-omvandlaren har inte kommit ännu. Du beslutar dig då för att simulera AD-omvandlaren dvs att skriva en klass som implementerar interfacet ovan. Metoden `read` skall skapa slumptal istället för att läsa den analoga signalen. Om talen är mindre än `lower` kastas exception `UnderVoltage` och om dom är större än `upper` kastas `OverVoltage`. Annars ges ett slumptal mellan `lower` och `upper` *med 2 decimaler* i retur. Du kan anta att lower och upper är positiva. Fastna inte i detaljer med skalningen.

Skriv också klassen `UnderVoltage`.

(6p)