Institutionen för
Data- och informationsteknik
JSk

TENTAMEN

OBJEKTORIENTERAD PROGRAMVARUUTVECKLING

Övningstentamen

OBS! Det kan finnas kurser med samma eller liknande namn på olika utbildningslinjer. Denna tentamen gäller *endast* för den eller de utbildningslinjer som anges ovan. Kontrollera därför noga att denna tentamen gäller för den utbildningslinje du själv går på.

TID: 4 timmar

Ansvarig: Jan Skansholm

Förfrågningar:

Betygsgränser: Sammanlagt maximalt 60 poäng.
På tentamen ges graderade betyg:
CTH: 3:a 24 poäng, 4:a 36 poäng, 5:a 48 poäng
GU: G 24 poäng, VG 48 poäng

Hjälpmedel: Skansholm, *Java direkt med Swing*, valfri upplaga, Studentlitteratur.
(Understrykningar och mindre anteckningar i boken är tillåtna.)

Inga kalkylatorer är tillåtna.

Tänk på:

- att skriva tydligt och disponera papperet på ett lämpligt sätt.
- att börja varje ny uppgift på nytt blad. Skriv endast på en sida av papperet
- Skriv den (anonyma) kod du fått av tentamensvakten på *alla* blad.

De råd och anvisningar som givits under kursen skall följas vid programkonstruktionerna. Det innebär bl.a. att onödigt komplicerade, långa och/eller ostrukturerade lösningar i värsta fall ej bedöms.

Uppgift 1)

a) Vad blir det för utskrift när detta program exekveras?

```
public class C {
  public static void main(String[] argv) {
    int k = 1;
    int[] x = {0, 1, 2, 3};
    int[] y = x;
    lurig(x, y, k);
    System.out.println(x[0] + k + y[0]);
  }

  public static void lurig(int[] p, int[] q, int r) {
    p[0] = 1;
    q[0] = 2;
    r = 3;
  }
}
```

(2 p)

b) Vad skrivs ut när följande program exekveras?

```
public class EnKlass {
  private int x, y;
  private static int z;

  public EnKlass(int a, int b, int c) {
    x = a;
    y = b;
    z = c;
  }

  public String toString() {
    return x + " " + y + " " + z;
  }
}

public class StatTest {
  public static void main(String[] s) {
    EnKlass a = new EnKlass(1, 2, 3);
    EnKlass b = new EnKlass(4, 5, 6);
    System.out.println(a);
    System.out.println(b);
  }
}
```

(2 p)

c) Ange för var och en av följande rader om den är korrekt eller inte. Om en rad är felaktig så ange vad felet är.

```
/* rad 1 */ List<Integer> l1 = new List<Integer>();
/* rad 2 */ List<double>  l2 = new ArrayList<double>();
/* rad 3 */ LinkedList<String> l3 = new LinkedList<String>();
/* rad 4 */ List<l3> l4 = new LinkedList<l3>();
/* rad 5 */ LinkedList<String> l5 = new ArrayList<String>();
/* rad 6 */ List<String> l6 = new LinkedList<String>(l3);
```

(3 p)

Uppgift 2) Konstruera en klassmetod med namnet `tidsintervall` vilken får två tidpunkter `t1` och `t2` som parametrar och som räknar ut hur lång tid det är från `t1` till `t2`. Placera metoden i en klass med namnet `Tidklass`. Parametrarna och resultatet skall vara av typen `String` och ha formen `tt:mm:ss`. Observera att timmar, minuter och sekunder *alltid* anges med två siffror, t.ex. 08:17:05. Detta skall gälla även i resultatet. Metoden skall fungera även om `t2` anger en tid som är mindre än `t1`. I så fall skall man anta att `t2` har inträffar dygnet efter `t1`. Tips: Räkna om tiderna till sekunder.

(10 p)

Uppgift 3) För att kunna hålla ordning på alla deltagare i en löpartävling har man samlat information om dem i en textfil. Raderna i filen kan t.ex. se ut så här:

```
15:03:00 17:45:07 37 Anders Olsson 1972 Elitlöparna
15:08:00 16:59:59 4503 Nils Persson 1959 IK Spring
15:03:00 17:39:01 3506 Brandman Larsson 1965
```

Först kommer starttiden skriven enligt `tt:mm:ss`. (Eftersom det är så många deltagare startar inte alla samtidigt). Därefter kommer sluttiden (också enligt `tt:mm:ss`). För dem som inte kom i mål har sluttiden angivits som `00:00:00`. Resten av raden består av startnumret (varje deltagare har ett unikt startnummer), namn, födelseår och eventuell klubbtillhörighet. Mellan varje uppgift finns ett eller flera blanka tecken. Du kan anta att alla rader i filen följer detta mönster. Däremot vet du inte exakt hur många rader filen innehåller.

Man vill bestämma deltagarnas placering och behöver därför beräkna tiden det tog för varje deltagare att springa runt banan och sortera alla deltagarna efter denna tid. *Skriv ett program som läser innehållet i filen och som skapar en ny fil där personerna är sorterade efter hur fort de sprungit* (den som sprang fortast först). I den nya filen ska det först på varje rad stå tiden det tog att springa runt banan, sedan startnummer, namnet på personen, födelseår och eventuell klubbtillhörighet. Det kan t.ex. se ut på följande sätt

```
01:51:59 4503 Nils Persson 1959 IK Spring
02:36:01 3506 Brandman Larsson 1965
02:42:07 37 Anders Olsson 1972 Elitlöparna
```

Endast de löpare som kommit i mål skall finnas med i den nya filen. Namnen på den ursprungliga filen och den nya filen skall läsas in från dialogrutor.

Tips1: Du får gärna använda dig av metoden `tidsintervall` från uppgift 2 (även om du inte löst den uppgiften.)
Tips2: Objekt av klassen `String` är naturligt jämförbara. Man kan därför, innan man skriver ut raderna till den nya filen, lägga dem i en objektsamling som man antingen sorterar när alla rader lagts dit eller som har egenskapen att raderna sorteras automatiskt när de läggs in. (Använd en standardklass.)

(10 p)

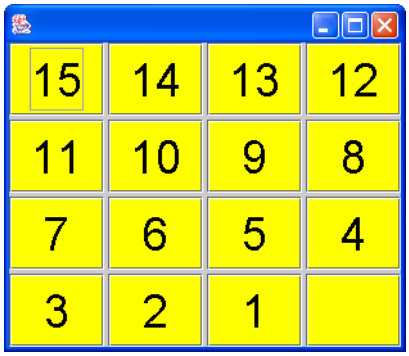
Uppgift 4) Om man inte hade haft tillgång till standardklassen `Math` hade man kunnat använda s.k. Maclaurin-serier för att beräkna värdet av vissa vanliga funktioner. Funktionen *sin* kan t.ex. beräknas med följande oändliga serie:

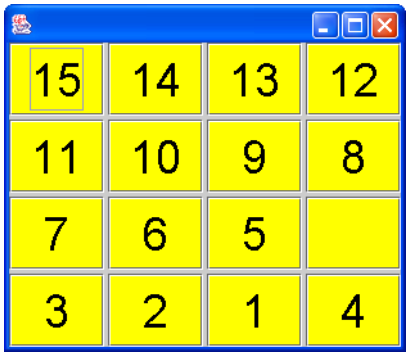
$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \ldots$$

Skriv ett klass `C` som innehåller en klassmetod `sin`. Metoden skall ha en parameter x av typen `double` och beräkna och returnera värdet *sin(x)*. Beräkningen skall ske med hjälp av serien ovan. Börja med att beräkna den första termen och beräkna sedan nästa term utgående från den förra. Upprepa detta tills den senaste termen som beräknats till beloppet blir mindre än 10^{-5}.

(10 p)

Uppgift 5) Skriv ett program som låter oss spela det s k 15-spelet. När programmet startas skall det komma upp ett fönster med knappar enligt vänstra figuren. Knappen längst ned till höger skall vara blank, medan övriga är försedda med nummer enligt figuren.





Man skall sedan oupphörligt kunna trycka på knapparna. Om man trycker på en knapp som har den blanka knappen som granne (dvs rakt ovanför eller nedanför sig eller till vänster eller höger om sig), skall texten på knapparna byta plats. Om man t.ex. trycker på knappen med texten "4" i den vänstra figuren så skall fönstret ändras så att det kommer att se ut som i den högra figuren. Om man trycker på den blanka knappen eller på en knapp som inte har blank granne skall programmet generera ett pipande ljud.

Tips: Låt varje ruta representeras av ett objekt av en egen klass `Ruta`, vilken skall vara en subklass till `JButton`. Om du låter varje ruta själv hålla reda på sin placering på spelplanen (sin rad och sin kolumn) så blir lösningen enkel. Om du dessutom hela tiden ser till att ha en referens till den tomma rutan så blir det ännu enklare.

(12 p)

Uppgift 6) Nedanstående kod kan tänkas vara hämtad från ett dataspel. Klassen `Varelse` beskriver de olika varelser som kan ingå i spelet och klassen `Gäng` en grupp av sådana varelser. Om man stöter på en varelse i spelet kan denna orsaka skada. Hur stor skadan blir beror på vilken typ av varelse det är och hur mycket energi denna varelse har. Stöter man på en grupp av varelser adderas de skador som gruppmedlemmarna orsakar. Metoden `gemensamSkada` beräknar den sammanlagda skadan en grupp orsakar.

```
public class Varelse {
  public static final int ORM = 0;
  public static final int TROLL = 1;
  public static final int SPINDEL = 2;

  public int typ;            // innehåller en av typerna ovan
  public int energi = 100;   // 0=död, 100=full styrka
  public String namn;

  public Varelse(String namn, int typ) {
    this.namn = namn;
    this.typ = typ;
  }

}

class Gäng {
  private Varelse[] medlemmar;

  public Gäng(int storlek) {
    medlemmar = new Varelse[storlek];
  }

  public void läggTill(Varelse v, int index) {
    medlemmar[index] = v;
  }

  public int gemensamSkada() {
    int tot = 0;
    for (int i=0; i < medlemmar.length; i++)
      if (medlemmar[i] != null)
        if (medlemmar[i].typ == Varelse.ORM)
          tot += 10*medlemmar[i].energi;
        else if (medlemmar[i].typ == Varelse.TROLL)
          tot += 5*medlemmar[i].energi;
        else if (medlemmar[i].typ == Varelse.SPINDEL)
          if (medlemmar[i].energi > 0)
            tot += 100;
          else
            tot += 0;
    return tot;
  }
}
```

I koden används klasser men den är ändå inte objektorienterad. Den strider mot grundläggande objektorienterade principer:

- Data och de operationer som utförs på dessa data har inte samlats till ett enda ställe i koden. Beräkningar görs på fel ställe.
- Data har inte kapslats in. En varelses namn skall bestämmas när varelsen skapas men skall sedan inte kunna ändras, bara avläsas. En varelses energi skall visserligen både kunna ändras och avläsas, men endast värden i intervallet 0 till 100 är tillåtna.

Din uppgift är nu att skriva om koden så att den bli objektorienterad. I din lösning får inte variabeln `typ` finnas med. Du skall istället använda dig av de objektorienterade "verktyg" som ingått i kursen (arv, dynamisk bindning, abstrakta klasser etc.)

(11 p)