CHALMERS

Institutionen för data- och informationsteknik

TENTAMEN

KURSNAMN	**Objektorienterad programmering, 7.5p**
PROGRAM:	**DAI 2 (TIDAL-2)** **2013/2014, lp 2**
KURSBETECKNING	**DAT050**
EXAMINATOR	**Uno Holmer**
TID FÖR TENTAMEN	**Fredagen den 25/4 2014, 8.30-12.30**
HJÄLPMEDEL	**Java API** (bifogas tesen)
ANSV LÄRARE	**Uno Holmer** **tel. 772 5730** **besöker tentamen ca kl. 9.30 samt ca 11.15**
DATUM FÖR ANSLAG	**Senast den 12/5 2014** **datum för visning meddelas senare**
ÖVRIG INFORM.	**Betygsgränser: 3 - 24p, 4 - 36p, 5 – 48p. (max 60p)**

TENTAMEN: Objektorienterad programmering

Läs detta!

- *Uppgifterna är inte ordnade efter svårighetsgrad.*
- Börja varje hel uppgift på ett nytt blad.
- Ordna bladen i uppgiftsordning.
- Skriv din tentamenskod på varje blad (så att vi inte slarvar bort dem).
- **Skriv rent dina svar. Oläsliga svar r ä t t a s e j!**
- Programkod som finns i tentamenstesen behöver ej upprepas.
- Programkod skall skrivas i Java 5, eller senare version, och vara indenterad och renskriven.
- Onödigt komplicerade lösningar ger poängavdrag.
- Givna deklarationer, parameterlistor etc. får ej ändras.
- Läs igenom tentamenstesen och förbered ev. frågor.

I en uppgift som består av flera delar får du använda dig av funktioner klasser etc. från tidigare deluppgifter, även om du inte löst dessa.

Lycka till!

Uppgift 1

Välj **ett** alternativ för varje fråga! Garderingar ger noll poäng. Inga motiveringar krävs. Varje korrekt svar ger två poäng. Besvara delfrågorna 1-5 på ett blad.

1. Vilken av `clone`-metoderna kopierar `B` på korrekt sätt?

```
public class A {
   ...
   public A clone()  { ... }
   ...
}
public class B {
   private int x;
   private A a;
   ...
   public B clone() { se nedan }
   ...
}
```

a)
```
public B clone() {
   return (B)super.clone();
}
```

b)
```
public B clone() {
   B tmp = (B)super.clone();
   tmp.a = a.clone();
   return tmp;
}
```

c)
```
public B clone() {
   B tmp = new B();
   tmp.x = x;
   tmp.a = a.clone();
   return tmp;
}
```

d)
```
public B clone() {
   B tmp = this.clone();
   tmp.a = a.clone();
   return tmp;
}
```

2. Vilket påstående om klasserna nedan stämmer bäst?

```
public class Person {
    private String name;
    private ContactInfo contact;
     ...
}
public class ContactInfo {
    private String address;
    private String phone;
    private String email;
    private List<Person> friends;
     ...
    public int getNoOfFriends() { ... }
    public Person getFriend(int i) { ... }
     ...
}
```

a. Klasserna har hög kohesion och kopplingen mellan klasserna är hög.
b. Klasserna har låg kohesion och kopplingen mellan klasserna är låg.
c. Klasserna har hög kohesion och kopplingen mellan klasserna är låg.
d. Klasserna har låg kohesion och kopplingen mellan klasserna är hög.

3. En av deklarationerna ger ett typfel, vilken?
   a. `List<LinkedList> l = new ArrayList<LinkedList>();`
   b. `ArrayList<List> l = new ArrayList<List>();`
   c. `ArrayList<List> l = new ArrayList<LinkedList>();`
   d. `List<List> l = new ArrayList<List>();`

4. Givet klassen `Num`

```
public class Num {
    private int x;
    public Num() { x = 0; }
    public void set(int x) { this.x = x; }
    public int get() { return x; }
}
```

vad skrivs ut om detta exekveras:

```
ArrayList<Num> list = new ArrayList<Num>();
Num n = new Num();

for (int j = 0; j < 3; j++)
    list.add(n);

int k = 1;
for (Num x : list)
    x.set(k++);

for (Num x : list)
    System.out.println(x.get());
```

   a. `0 0 0`
   b. `3 3 3`
   c. `1 2 3`
   d. `3 2 1`

5. Betrakta följande javapaket och klasser:

```
package module1;

public class C1 {
    private int x;
    protected int y;
    public int z;
}

public class C2 extends C1 { ... }

public class C3 { ... }
```

```
package module2;
import module1.*;

public class C4 extends C1 { ... }

public class C5 { ... }
```

Låt C_i (v_1, … ,v_n) beteckna att variablerna v_1, … ,v_n är synliga i klassen C_i.
Vilket av alternativen beskriver synligheten hos variablerna x, y och z i klasserna ovan?

a. C2(y,z), C3(z), C4(y,z), C5(z)
b. C2(y,z), C3(z), C4(z), C5(z)
c. C2(y,z), C3(y,z), C4(y,z), C5(z)
d. C2(y,z), C3(y,z), C4(z), C5(z)

(10 p)

Uppgift 2

Innan dagens sociala medier fanns var det vanligt att man använde epostlistor för att sprida information, och de används väl fortfarande en del. Listorna innehåller medlemmar som får alla nya brev adresserade till listan. En användare kan också anmäla sig till flera listor. Uppgiften är att konstruera klassen `MailingLists` som håller reda på ett antal användarnamn och ett antal epostlistor. Klassen skall ha följande metoder

```
public void add(String user,String listCsv)
throws IllegalArgumentException
```

Lägger till listnamnen i `listCsv` till `user`.[1] Om `user` saknas i databasen skall en ny användare med det namnet läggas till. Om ett listnamn i `listCsv` saknas i databasen skall det läggas till. Undantaget `IllegalArgumentException` kastas om `user` är **`null`**, en tom sträng, eller en sträng med endast "vita" tecken.

```
public Set<String> getmailingLists(String user)
```

Returnerar en mängd av namnen på listorna som `user` är registrerad på.

```
public Set<String> getMembers(String list)
```

Returnerar en mängd av användarnamn som är registrerade på `list`.

```
private String[] parseCsv(String csv)
```

Returnerar ett fält av de kommaseparerade strängarna i `csv`.

a) Definiera lämpliga instansvariabler för att lagra informationen om användarnas epostlistor och epostlistornas medlemmar. Använd två avbildningstabeller (mappar). Namnen i en samling användarnamn eller en samling listnamn skall vara unika. Lagra därför dessa samlingar i mängder (set). Skriv också en konstruktor som skapar strukturerna.

(2 p)

b) Implementera metoden `add`. Se spec. ovan.

(6 p)

c) Implementera metoden `parseCsv`.
Ex. Om inparametern är strängen `"  mat ,, cyklar,  Husdjur  "` skall fältet `{"mat","cyklar","husdjur"}` returneras.

(4 p)

[1] CSV = Comma Separated Values. Ex. `"mat,bilar,resor"`.

Uppgift 3

a)
Vilka metoddeklarationer i `VatPrice` är typkorrekta och vilka ger kompileringsfel? Motivera svaret!

```
public class NetPrice {

    private String partNumber;
    private double price = 0;

    public void setPrice(double price) {
        this.price = price;
    }

    protected double getPrice() {
        return price;
    }
    ...
}

public class VatPrice extends NetPrice {

    protected void setPrice(double price) {
        super.setPrice(price/1.25);
    }

    @Override
    public void setPrice(int price) {
        super.setPrice(price/1.25);
    }

    public void setPrice(double price,double vat) {
        super.setPrice(price/vat);
    }

    @Override
    public double getPrice() {
        return 1.25*super.getPrice();
    }
}
```

(4 p)

b)
Studera följande klasser:

```
public interface Int {
    int h(int x);
}

public class Base {
    private int x = 100;

    public void f(int y) {
        g(x-y);
    }

    public void g(int a) {
        x = x + a;
    }

    public int getX() {
        return x;
    }
}

public class Sub extends Base implements Int {
    private int y = 200;

    public void g(int b) {
        super.g(b);
        y = b + getX();
    }

    public int h(int x) {
        f(x);
        return y;
    }
}
```

Vad skrivs ut om följande exekveras? Motivera svaret!

```
    Int obj = new Sub();
    System.out.println(obj.h(10));
```

(6p)

Uppgift 4

a)
Klassen `Time` kan användas för att representera klockslag med 24-timmarsvisning. Klassen har en konstruktor

```
public Time()
```

som sätter tiden till 00:00, samt metoderna

```
public void setHour(int hour) throws IllegalArgumentException
public void setMinute(int minute) throws IllegalArgumentException
public int getHour()
public int getMinute()
```

De två första metoderna kastar `IllegalArgumentException` om inparametern har ett ogiltigt värde. Timmar skall ligga mellan 0 och 23, och minuter mellan 0 och 59. I t.ex. USA används tolvtimmarsvisning där klockslag under dygnets första hälft markeras med AM, och i den andra hälften med PM. Följande tabell visar sambandet mellan de två systemen för några utvalda klockslag:

00	01	11	12	13	23
12AM	1AM	11AM	12PM	1PM	11PM

Definiera en *subklass* till `Time` kallad `AmPmTime`. Subklassen skall ha metoder så att man även kan sätta tiden i 12-timmarsformatet, samt få tillbaks tiden i detta format:

```
public void setHour(int hour,boolean isPm)
throws IllegalArgumentException
```
Sätter tiden i detta objekt till `hour`.
`IllegalArgumentException` kastas om `hour` ligger utanför intervallet [1,12].

```
public int getHour()
```
Returnerar timtiden i detta objekt i intervallet [1,12].

```
public boolean isPm()
```
Returnerar `true` om timtiden i detta objekt anger en eftermiddagstid och `false` om den anger en förmiddagstid.

Följande visar några exempel på hur klassen kan användas (undantagshantering utelämnad):

```
AmPmTime t = new AmPmTime();
t.setHour(8,false);
t.setMinute(27);
System.out.println(t.getHour());          8
System.out.println(t.getMinute());        27
System.out.println(t.isPm());             false
t.setHour(19);
System.out.println(t.getHour());          7
System.out.println(t.isPm());             true
```

Basklassen `Time` är given och skall ej ändras. I subklassen får inga instansvariabler deklareras.

(7 p)

b)
Gränssnittet `Die` (=tärning) beskriver metoder för klasser som kan simulera en tärning:

```
public interface Die {
    void roll();      // kastar tärningen
    int getSize();    // returnerar antal sidor på tärningen
    int getValue();   // returnerar tärningens värde
}
```

Tärningens värde motsvarar det antal ögon som är vänt uppåt på en vanlig "fysisk" tärning.
Du skall inte implementera tärningsklasser här, men däremot skriva en metod

```
public static int[] rollTwoDice(Die d1,Die d2,int n)
```

som simulerar `n` kast med två tärningar och räknar alla tärningssummor vilka returneras i ett heltalsfält. Om summan s fås m gånger skall fältets s:te element vara m. De två första fältelementen skall vara noll eftersom den minsta tärningssumma man kan få är två.
Ex. Om den första tärningen har 6 sidor och den andra 8 skall det returnerade fältet ha 15 platser. Det blir ju 13 möjliga utfall med summor från 2 till 14 vilka räknas i position 2 till 14 i fältet.
Ex. En utskrift av ett sådant fält efter en million kast:

```
0
0
20846       summan 2
41439
62181
83174
104600
124915
125269
125068
104318
83399
62331
41743
20717       summan 14
```

I uppgiften ingår ej att skriva subklasser till `Die`.

(7 p)

Uppgift 5

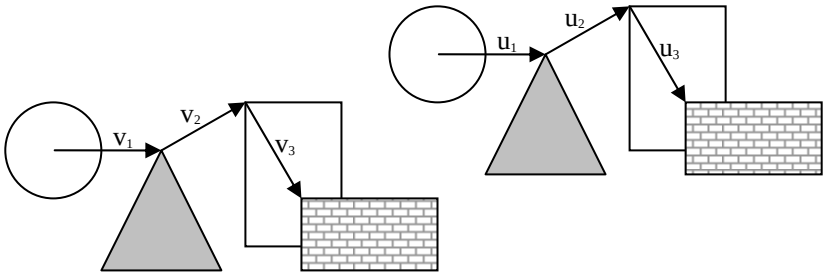
I en av kursens laborationer arbetade vi med olika klasser för geometriska figurer, däribland `Group` för att representera sammansatta figurer. Uppgiften går ut på att konstruera en likhetsmetod för sammansatta figurer. Basklass för alla figurklasserna är `Point`. Vi bortser från alla detaljer som saknar relevans för uppgiften.

```
public abstract class Point {
    ...
    public int getXposition() ...
    public int getYposition() ...
    ...
    public abstract boolean equals(Object other);     // Ny
}
```

Klassen `Group` har i korthet följande utseende:

```
public class Group extends Point {
    private ArrayList<Point> subPictures;
    ...
    public boolean equals(Object other) { ... }       // Ny
    ...
}
```

Vi skall tillämpa principen att om `a.equals(b)` returnerar **`true`** för figurerna `a` och `b` så skall de se likadana ut när de ritas upp. Att de är lika innebär dock inte att de måste finnas på samma plats. Enkla figurer, t.ex. cirklar, kvadrater eller trianglar är lika om de har samma dimensioner och färg. För figurgrupper blir det lite mer komplicerat. Två grupper är lika om de har lika många delfigurer och dessa är parvis lika, men det räcker inte. Delfigurernas inbördes geometriska orientering måste också vara lika. I exemplet nedan innebär det att vektorerna $u_1 = v_1$, $u_2 = v_2$ och $u_3 = v_3$. En följd av sådana vektorer skall vi kalla en promenad (walk) genom gruppen. Ex. (v_1,v_2,v_3) och (u_1,u_2,u_3).



För representation av vektorer finns den färdiga klassen (detaljer utelämnade):

```
public class Vector {
    public Vector(int x,int y)
    public int getX()
    public int getY()
    public boolean equals(Object other)
}
```

a) Skriv en metod `walk` som givet en lista av punkter (`Point`) returnerar en lista av vektorer. Tips: Skriv en privat hjälpmetod `diff` som returnerar skillnadsvektorn mellan två punkter.

(7 p)

forts.

b) Inför metoden

```
public boolean equals(Object other)
```

i `Group`. Om objekten `a` och `b` båda är av typen `Group` så skall `a.equals(b)` returnera **`true`** om grupperna som `a` och `b` representerar är lika enligt principerna ovan, annars **`false`**. Tips: Utnyttja att listklassen har en `equals`-metod.

(7 p)