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 kommenterad.
- 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 direkt i tesen!*

1. Ett av villkoren bör vara falskt, vilket?

```
SomeClass o1 = new SomeClass(),o2 = o1.clone();
```

   a. `o1.equals(o2)`
   b. `o1 == o2`
   c. `o1 != o2`

2. Vilket påstående är korrekt om `Register` på basis av vad som visas nedan?

```
public class Register
{
    private LinkedList<Person> list;
    private String name;
     ...

    public Register(String name) { this.name = name; }

    public void add(Person p) { list.add(p); }
     ...
}
```

   a. klassen går ej att kompilera
   b. det går ej att skapa objekt av klassen
   c. exekveringen riskerar att avbrytas om objekt av klassen används
   d. klassen är felfri

3. Vad bör man eftersträva vid objektorienterad design?

   a. Klasser med hög kohesion och hög koppling till andra klasser.
   b. Klasser med låg kohesion och hög koppling till andra klasser.
   c. Klasser med låg kohesion och låg koppling till andra klasser.
   d. Klasser med hög kohesion och låg koppling till andra klasser.

4. I vilken ordning bör aktiviteterna utföras vid utbyggnad av ett program med ny funktionalitet?
   a. regressionstesta, bygg ut, refaktorera, regressionstesta
   b. regressionstesta, bygg ut, regressionstesta, refaktorera, regressionstesta
   c. regressionstesta, refaktorera, regressionstesta, bygg ut, regressionstesta
   d. regressionstesta, refaktorera, bygg ut, regressionstesta

5. Vilken av satserna a - d ger ett kompileringsfel?

```
public class A
{
    protected int x = 0;
    public int y = 0;
}

public class B extends A
{
    public void f() {
        x++;                        // a
        y = 2*x;                    // b
    }
}

...
B obj = new B();
obj.x = 100;                        // c
obj.f();
System.out.print(obj.y);            //d
```

(10 p)

Uppgift 2

I en av kursens laborationer arbetade vi med en enkel e-post-server. Uppgiften går ut på att lägga till en konstruktor i klassen `MailServer` som bygger upp en brevlådestruktur med utgångspunkt från en lista av brevobjekt. Brevens mottagarfält skall användas för att konstruera brevlådor för motsvarande användare. Brev med tomma mottagarfält skall tas bort ur listan och inte ge upphov till några brevlådor. Vi antar för enkelhets skull att mottagarfältet i ett brev bara kan innehålla en mottagare. De för uppgiften relevanta delarna av klasserna `MailItem` och `MailServer`:

```
public class MailItem {
    public String getTo() { ... }
    ...
}
public class MailServer {
    private HashMap<String,LinkedList<MailItem>> mailboxes;
    ...
}
```

a) Skriv en metod som returnerar **`true`** om parametersträngen är ett giltigt användarnamn. En sträng är giltig om den inte är **`null`**, samt inleds och avslutas med ”ickevita” tecken.

```
private boolean validField(String s)
```

(2 p)

b) Skriv en metod som tar bort alla brevobjekt med ogiltigt mottagarfält från en lista av brevobjekt. Metoden skall alltså mutera listan som ges som argument. Utnyttja metoden i a.

```
private void filterItems(List<MailItem> items)
```

(5 p)

c) Skriv en konstruktor som tar en lista av brevobjekt och bygger upp en brevlådestruktur i `mailboxes`. Alla brev i listan som är adresserade till en viss mottagare skall läggas i dennes brevlåda i den nya strukturen.

```
public MailServer(List<MailItem> items)
```

Utnyttja metoden i b. Lösningen får ej baseras på andra metoder som finns i laborationens version av klassen.

(7 p)

Uppgift 3

a)
Vilka anrop är typkorrekta och vilka är det inte? Motivera svaret!

```
public interface A {
    void f();
}
public abstract class B {
    public abstract void g();
}
public class C implements A extends B {
    public void g() { ... }
    public void f() { ... }
    public void h() { ... }
}

A a = new C();
a.f();
a.g();
a.h();

B b = new C();
b.f();
b.g();
b.h();
```

(6 p)

b)
Vad skrivs ut ? Motivera svaret!

```
public abstract class Base {
    private int x = 0;
    public int f() {
        x++;
        return x + g();
    }
    public abstract int g();
}
```

```
public class Sub1 extends Base {
    private int x = 1;
    public int g() {
        x = 2*x;
        return x;
    }
}
```

```
ArrayList<Base> list =
    new ArrayList<Base>();
list.add(new Sub1());
list.add(new Sub2());
int i = 0;
for ( Base obj : list )
    i = i + obj.f();

System.out.println(i);
```

```
public class Sub2 extends Base {
    private int y = 10;
    public int g() {
        y--;
       return y;
    }
}
```

(6p)

Uppgift 4

a)
Klassen `Date` kan användas för att representera datum. Klassen har en konstruktor

```
public Date(int y,int m,int d) throws IllegalArgumentException
```

samt metoderna

```
public void setDate(int y,int m,int d)
       throws IllegalArgumentException
public int getYear()
public int getMonth()
public int getDay()
```

Konstruktorn och `setDate` sätter ett datum och kastar `IllegalArgumentException` om datumet inte är korrekt, t.ex. om man anger 2011,2,29. Det skulle vara praktiskt om man också kunde ange datumet som en sträng, t.ex. `"2010-12-14"`. Definiera därför en *subklass* till `Date` kallad `EnhancedDate`. Subklassen skall ha två metoder:

```
public void setDate(String date) throws IllegalArgumentException
public String toString()
```

Basklassen saknar ju en `toString`-metod så vi gör en i subklasen istället. Följande visar några exempel på hur klassen kan användas (undantagshantering utelämnad):

```
EnhancedDate d = new EnhancedDate(2010,12,14);
System.out.println(d.getYear());          2010
System.out.println(d.getDay());           14
d.setDate("2010-12-04");
System.out.println(d);                    2010-12-4
```

Metoden `setDate` i subklassen skall kasta `IllegalArgumentException` i följande fall:

- Strängen är `null`.
- Strängen innehåller ej tre siffergrupper, t.ex. `"2010-12"`.
- En eller flera siffergrupper innehåller otillåtna tecken, t.ex. `"2010-xx-@#!"`.
- Datumet är rätt formaterat men inkorrekt, t.ex. 2011,2,29.

Exempel på tillåtna datumsträngar: `"1987-3-22"`, `"  2013-03-04  "`.
I `toString` behöver du inte stoppa in inledande nollor framför ensiffriga tal. Subklassen får ej ha några egna instans- eller klassvariabler. Tips: utnyttja lämpliga metoder i strängklassen och `Integer` för att analysera strängen (se bilagan).

(6 p)

b)
En multimängd är en oordnad samling som, till skillnad från en vanlig mängd, kan innehålla flera förekomster av varje element. Ex. {1,2,3,1,5,2,7,1,22,-8}. Följande gränssnitt beskriver en sådan mängdtyp i Java för heltalselement:

```
public interface MultiSet {
    void add(int x);          // add x to this set
    boolean contains(int x); // true if this set contains x, false ow
    int count(int x);         // return the number of occurrences
                              // of x in this set
}
```

Multimängder kan implementeras på olika sätt. Välj själv mellan att använda en map, eller en lista. Kalla din klass `MapMultiSet` eller `ListMultiSet` beroende på vilken lagringsform du använder. Man skall t.ex. kunna använda klassen så här:

```
MultiSet ms = new ListMultiSet();    // eller MapMultiSet()
ms.add(2);
ms.add(5);
ms.add(2);
ms.add(7);
ms.add(5);
ms.add(2);
ms.count(5);          2
ms.count(7);          1
ms.count(2);          3
ms.contains(5);       true
ms.contains(4);       false
```

(6 p)

Uppgift 5

Ett rationellt tal $\frac{p}{q}$ representerar en kvot mellan två heltal p och q, t.ex. 1/2, 3/4, 5/6, -1/7, 12/9.

Några lagar: $\frac{p}{q} \cdot \frac{r}{s} = \frac{pr}{qs}$, $\frac{p}{q} + \frac{r}{s} = \frac{ps+qr}{qs}$.

Två rationella tal $\frac{p}{q}$ och $\frac{r}{s}$ är lika om och endast om $ps = qr$.

Man kan representera ett rationellt tal som ett objekt. Klassen `Rational` innehåller några metoder för att manipulera rationella tal.

```
public final class Rational {
    private int p,q;

    public Rational(int p,int q) {
        this.p = p;
        this.q = q;
    }
    public int getDenominator() { return p; }
    public int getNumerator() { return q; }

    public Rational plus(Rational other) {
        return new Rational(p*other.q+q*other.p,q*other.q);
    }
    public Rational mult(Rational other) {
        return new Rational(p*other.p,q*other.q);
    }

    ...
}
```

a) Lägg till metoden

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

till `Rational`. Om objekten `a` och `b` båda är av typen `Rational` så skall `a.equals(b)` returnera **`true`** om de rationella talen som `a` och `b` representerar är lika enligt lagen ovan, annars skall `equals` returnera **`false`**.

(7 p)

b) Lägg till metoden

```
public int hashCode()
```

till `Rational`.

(5 p)