Lösningsförslag till tentamen

Kurs	**Objektorienterad programmering**
Tentamensdatum	**2009-12-14**
Program	**DAI 2**
Läsår	**2010/2011, lp 2**
Examinator	**Uno Holmer**

Uppgift 1 (10 p)

Ingen lösning ges. Se kurslitteraturen.

Uppgift 2 (2+5+7 p)

a)
```
private boolean validField(String s)
{
    return s != null && s.equals(s.trim());
}
```

b)
```
private void filterItems(List<MailItem> items)
{
    Iterator<MailItem> it = items.iterator();
    while ( it.hasNext() )
        if ( ! validField(it.next().getTo()) )
            it.remove();
}
```

b)
```
public MailServer(List<MailItem> items)
{
    mailboxes = new HashMap<String,LinkedList<MailItem>>();
    filterItems(items);
    for ( MailItem item : items )
    {
        String user = item.getTo();
        if ( ! mailboxes.containsKey(user) )
            mailboxes.put( user, new LinkedList<MailItem>() );

        mailboxes.get(user).add(item);
    }
}
```

Uppgift 3 (6+6 p)

a) Om variabeln har typ `A` är endast metoder i `A` kända, anropet `a.f()` är därför korrekt, men inte det två andra. Det hjälper alltså inte att `C` implementerar `g` och `h`. Av analoga skäl kan endast `g` anropas via en variabel av typ `B`.

b) Utskriften blir 13. Observera att varje subklassobjekt har sin egen basklassdel och därför sin egen `x`-variabel. Ett `Sub1`-objekt har därför två `x`-variabler, en i basklassdelen (osynlig i `Sub1`), och en i subklassdelen.

Uppgift 4 (6+6 p)

a)

```
public class EnhancedDate extends Date {

    public EnhancedDate(int y,int m,int d)
        throws IllegalArgumentException
    {
        super(y,m,d);
    }

    public void setDate(String date) throws IllegalArgumentException
    {
        if ( date == null )
            throw new IllegalArgumentException();
        String[] ymd = date.trim().split("-");
        if ( ymd.length != 3 )
            throw new IllegalArgumentException();
        try {
            int y = Integer.parseInt(ymd[0]);
            int m = Integer.parseInt(ymd[1]);
            int d = Integer.parseInt(ymd[2]);
            setDate(y,m,d);
        }
        catch (NumberFormatException e) {
            throw new IllegalArgumentException();
        }
    }

    public String toString() {
        return getYear() + "-" +
               getMonth() + "-" +
               getDay();
    }
}
```

b)

```
// med map
public class MapMultiSet implements MultiSet {
    private HashMap<Integer,Integer> theMap;

    public MapMultiSet() {
        theMap = new HashMap<Integer,Integer>();
    }

    public void add(int x) {
        if ( ! theMap.containsKey(x) )
            theMap.put(x,1);
        else {
            int count = theMap.get(x);
            theMap.put(x,count+1);
        }
    }
    public boolean contains(int x) {
        return count(x) > 0;
    }
    public int count(int x) {
        if ( ! theMap.containsKey(x) )
            return 0;
        else
            return theMap.get(x);
    }
}
```

```
// med lista
public class ListMultiSet implements MultiSet {
    private ArrayList<Integer> theList;

    public ListMultiSet() {
        theList = new ArrayList<Integer>();
    }

    public void add(int x) {
        theList.add(x);
    }

    public boolean contains(int x) {
        return theList.contains(x);
    }

    public int count(int x) {
        int n = 0;
        for ( int e : theList )
            if ( e == x )
                n++;
        return n;
    }
}
```

Uppgift 5 (7+5 p)

a)

```
public boolean equals(Object other)
{
    if ( other == this )
       return true;
    if ( other instanceof Rational )
    {
        Rational tmp = (Rational)other;
        return p*tmp.q == q*tmp.p;
    }
    return false;
}
```

b)

Uppgiften visade sig bli svårare än avsett. Följande metod är inte helt korrekt men den, eller en likvärdig konstruktion, ger ändå full poäng.

```
public int hashCode() {
    int code =  123;
    code = 37*code + p;
    code = 37*code + q;
    return code;
}
```

En närmare förklaring ges på nästa sida.

Metoden ovan bryter mot regeln som säger att två lika objekt skall ha samma hashkod. T.ex. är 3/4 och 6/8 lika men de får givetvis olika hashkod med ovanstående algoritm. En lösning är att *normalisera* alla `Rational`-objekt till sin *kanoniska form*. Det vill säga, förkorta bråket så långt det går. Enklast är att använda Euklides algoritm som bestämmer den största gemensamma delaren till två heltal (GCD = Greatest Common Divisor).[1] Här följer två varianter i Java, först en iterativ:

```
private int gcd(int a,int b) {
    while ( b != 0 ) {
        int rest = a % b;
        a = b;
        b = rest;
    }
    return Math.abs(a);
}
```

och här en rekursiv:

```
private int gcd(int a,int b) {
    if ( b == 0 )
        return Math.abs(a);
    else
        return gcd(b,a % b);
}
```

Nu kan vi skriva metoden

```
private void normalize() {
    int d = gcd(p,q);
    p = p/d;
    q = q/d;
}
```

Om `normalize` anropas i konstruktorn kommer alla tal att lagras i normaliserad form:

```
public Rational(int p,int q) {
    this.p = p;
    this.q = q;
    normalize();
}
```

Så både `new Rational(6,8)` och `new Rational(3,4)` ger objekt som lagrar talet som ¾ internt. Efter dessa tillägg ger `hashCode` ett korrekt resultat. *Tack till Jesper Sjövall för påpekandet!*

[1] Euklides algoritm är en av historiens äldsta kända algoritmer, se t.ex. Wikipedia.