Lösningsförslag till tentamen

Kurs	**Objektorienterad programmering**
Tentamensdatum	**2013-12-17**
Program	**DAI2**
Läsår	**2013/2014, lp 2**
Examinator	**Uno Holmer**

Uppgift 1 (10 p)

Ingen lösning ges. Se kurslitteraturen.

Uppgift 2 (7+7 p)

a)

```
public static double compute(Collection<Double> collection,Computer comp)
throws UndefinedOperationException
{
    for ( double x : collection )
        comp.addValue(x);
    return comp.getValue();
}
```

b)

```
public class ArithmeticMean implements Computer {
    private int noOfValues;
    private double sum;

    public ArithmeticMean() {
        noOfValues = 0;
        sum = 0;
    }

    public void addValue(double value) {
        sum += value;
        noOfValues++;
    }

    public double getValue() throws UndefinedOperationException {
        if ( noOfValues == 0 )
            throw new UndefinedOperationException("Division by zero");
        return sum/noOfValues;
    }
}
```

Uppgift 3 (2+1+4+2+2+2+4 p)

a)
```
private Set<String> friends = new HashSet<String>();
```

b)
```
public Iterator<String> getFriends() {
    return friends.iterator();
}
```

c)
```
public boolean removeFriend(String id) {
    Iterator<String> it = friends.iterator();
    while ( it.hasNext() ) {
        String x = it.next();
        if ( id.equals(x) ) {
            it.remove();
            return true;
        }
    }
    return false;
}
```

d)
```
private HashMap<String,Membership> members = new HashMap<>();
```

e)
```
public boolean addMember(Membership m) {
    if ( members.containsKey(m.getEmail()))
        return false;
    else {
        members.put(m.getEmail(),m);
        return true;
    }
}
```

f)
```
public boolean connect(String id1,String id2) {
    if ( ! members.containsKey(id1) || ! members.containsKey(id2) )
        return false;
    else {
        members.get(id1).addFriend(id2);
        members.get(id2).addFriend(id1);
        return true;
    }
}
```

g)
```
public boolean removeMember(String id) {
    if ( ! members.containsKey(id) )
        return false;
    else {
        // For each of m's friends, remove friendship to m.
        Iterator<String> it = members.get(id).getFriends();
        while ( it.hasNext() )
            members.get(it.next()).removeFriend(id);
        // Finally remove id.
        members.remove(id);
        return true;
    }
}
```

Uppgift 4 (10 p)

```
public class RandomString extends Random {
    private char[] randChars;

    public RandomString(String randChars) {
        this.randChars = randChars.toCharArray();
    }

    public String nextString(int length) {
        StringBuilder buf = new StringBuilder(length);
        for ( int i = 0; i < length; i++ )
            buf.append(randChars[nextInt(randChars.length)]);
        return buf.toString();
    }
}
```

Uppgift 5 (4+1+4 p)

a)

```
@Override
public final boolean equals(Object other) {
    if ( this == other )
        return true;
    else if ( other instanceof Customer ) {
        return ((Customer)other).customerId == customerId;
    } else
        return false;
}
```

b)

```
@Override
public int hashCode() {
    return customerId;
}
```

c)

```
@Override
public Customer clone() {
    try {
        Customer copy = (Customer)super.clone();
        copy.contact = contact.clone();
        return copy;
    }
    catch (CloneNotSupportedException e) {
        throw new InternalError();
    }
}

@Override
public Contact clone() {
    try {
        return (Contact)super.clone();
    }
    catch (CloneNotSupportedException e) {
        throw new InternalError();
    }
}
```