Lösningsförslag till tentamen

Kurs	**Objektorienterad programmering**
Tentamensdatum	**2014-04-25**
Program	**DAI 2**
Läsår	**2013/2014, lp 2**
Examinator	**Uno Holmer**

Uppgift 1 (10 p)

Ingen lösning ges. Se kurslitteraturen.

Uppgift 2 (2+6+4 p)

a)
```
private Map<String,Set<String>> users;
private Map<String,Set<String>> mailingLists;

public MailingLists() {
    users = new TreeMap<String,Set<String>>();
    mailingLists = new TreeMap<String,Set<String>>();
}
```

b)
```
public void add(String userName,String listCsv)
    throws IllegalArgumentException
{
    if ( userName == null )
        userName = "";
    userName = userName.trim().toLowerCase();
    if ( userName.length() == 0 )
        throw new
              IllegalArgumentException("invalid user name passed to add");
    if ( ! users.containsKey(userName) )
        users.put(userName,new TreeSet<String>());

    for ( String listName : parseCsv(listCsv) ) {
        if ( ! mailingLists.containsKey(listName) )
            mailingLists.put(listName,new TreeSet<String>());
        users.get(userName).add(listName);
        mailingLists.get(listName).add(userName);
    }
}
```

c)
```
private String[] parseCsv(String csv) {
    String[] arr = csv.trim().toLowerCase().split(",");
    ArrayList<String> al = new ArrayList<String>();
    for ( String s : arr ) {
        s = s.trim();
        if ( s.length() > 0 )
            al.add(s);
    }
    return al.toArray(new String[al.size()]);
}
```

alt.
```
private String[] parseCsv(String csv) {
    return csv.trim().split(" *[, *]+ *");
}
```

Uppgift 3 (4+6 p)

a)
Fel, vid överskuggning måste den överskuggande metoden ha minst lika hög synlighet som den överskuggade metoden i basklassen, vilket inte är uppfyllt här:

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

Fel, denna metod överskuggar ingen metod i basklassen eftersom basklassmetoden har en parameter av typ `double`:

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

Korrekt, detta är en helt ny metod (namnet `setPrice` överlagras). Den är inte en överskuggning av `setPrice` i basklassen och gör heller ej anspråk på att vara det (det finns inget `@Override`-direktiv innan):

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

Detta är en korrekt överlagring av `getPrice`. Synligheten hos den överskuggande metoden får vara högre än hos den överskuggade:

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

b) 280

Uppgift 4 (7+7 p)

a)

```
public class AmPmTime extends Time {
    // Detta blir ingen överskuggning!
    public void setHour(int hour,boolean isPm)
            throws IllegalArgumentException {
        if ( hour < 1 || hour > 12 )
            throw new IllegalArgumentException("Hour out of range");

        setHour(hour%12 + (isPm ? 12 : 0));
    }

    @Override
    public int getHour() {
        int h = super.getHour();
        return (h%12 == 0) ? 12 : h%12;    }

    public boolean isPm() {
        return super.getHour() >= 12;
    }
}
```

b)
```
public static int[] rollTwoDice(Die d1,Die d2,int n) {
    int arraySize = d1.getSize() + d2.getSize() + 1;
    int[] dieSums = new int[arraySize];
    for ( int i = 0; i < arraySize; i++ )
        dieSums[i] = 0;
    int i = 0;
    while ( i++ < n ) {
        d1.roll();
        d2.roll();
        dieSums[d1.getValue()+d2.getValue()]++;
    }
    return dieSums;
}
```

Uppgift 5 (7+7 p)

a)
```
private Vector diff (Point p1,Point p2) {
    return new Vector(p1.getXposition()-p2.getXposition(),
                      p1.getYposition()-p2.getYposition());
}

private ArrayList<Vector> walk(ArrayList<Point> points) {
    ArrayList<Vector> w = new ArrayList<Vector>();
    for ( int i = 0; i < points.size() - 1; i++ )
        w.add(diff(points.get(i+1),points.get(i)));
    return w;
}
```

b)
```
public boolean equals(Object other) {
    if ( this == other )
        return true;
    else if ( other != null && getClass() == other.getClass() ){
        Group o = (Group)other;
        return subPictures.equals(o.subPictures) &&
               walk(subPictures).equals(walk(o.subPictures));
    }
    return false;
}
```