PROG2 Tenta 2012-06-01

Gäller SP:PROG2, DSK2:PROG2, FK:PROG2, FK:OOP och DSV1:P2

Tentan består av tre uppgifter. Max poäng är 38.
För betyget E (godkänd) krävs minst 23 poäng och **minst en poäng på varje uppgift.**
Betygen A-D ges enligt betygskriteria i Daisy och på omstående sida.

Hjälpmedel:

Tillåtna hjälpmedel är medhavda böcker om Java.

Anvisningar:

Skriv endast på ena sidan av bladen.
Påbörja varje ny uppgift på nytt blad.
Skriv tydligt - oläsbara svar beaktas inte.

Även icke fullständiga lösningar beaktas.
Kommentera gärna era lösningar.

Lösningsförslag kommer att presenteras i delkursens FC-konferens.

Lycka till!

Tentan betygsätts i A/B/C/D/E/Fx/F-skalan enligt följande kriteria:

A - Samtliga lösningar är felfria (förutom uppenbara små misstag), fullständiga, genomförda med användning av de på kursen presenterade Java-teknikerna och lämpliga klasser/metoder ur Javas standardbibliotek och med kod som är klar, effektiv och inte onödigt omständlig. Lösningarna tar hänsyn till alla i uppgiftstexten angivna situationer och innehåller alla i uppgiftstexten begärda felkontroller. Däremot behöver de inte innehålla andra felkontroller eller ta hänsyn till andra situationer än de som angetts i uppgiftstexten.

B - Som för betyget A utom att någon eller ett par lösningar kan innehålla något grövre misstag eller underlåta att ta hänsyn till någon enstaka angiven situation eller någon enstaka begärd felkontroll. Lösningarna kan vara något mer omständliga än nödvändigt.

C - Lösningarna är i princip korrekta men kan vara behäftade med fel och/eller underlåta att ta hänsyn till någon enstaka angiven situation eller begärd felkontroll. Lösningarna kan vara mer omständliga än nödvändigt (t.ex. skrivna med egen kod där standardklasser/-metoder kunde ha använts).

D - Som för betyget C utom att någon eller ett par lösningar kan innehålla grövre fel av principiell karaktär. Lösningarna kan även innehålla grövre syntaktiska misstag (t.ex. sammanblandning med andra språk med liknande syntax).

E - Som för betyget D utom att de flesta eller alla lösningar är behäftade med grövre fel av principiell karaktär. Icke desto mindre måste lösningarna visa grundläggande förståelse för problemet och åtminstone en ansats till korrekt lösning.

Fx - En lösning är helt felaktig eller saknas medan de övriga lösningar är i princip korrekta, men kan vara behäftade med fel och/eller underlåta att ta hänsyn till någon enstaka angiven situation eller begärd felkontroll.

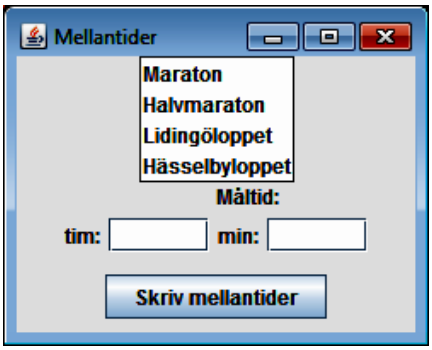
F - Flera lösningar är helt felaktiga eller saknas.

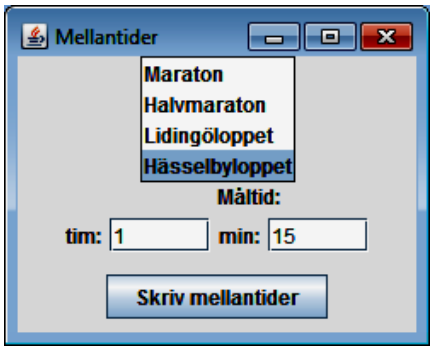
Betyget Fx innebär att studenten kan komplettera examinationen med en extra inlämningsuppgift för att få godkänt på aktuell tentamen (E men ej högre betyg). Kompletteringsuppgiften måste lämnas in enligt angiven deadline och kan endast användas för att få betyget E på den aktuella tentan.

Uppgift 1 (12 poäng)

Det är ju populärt att springa långdistanslopp, t.ex. är drygt 21000 löpare anmälda till Stockholm Marathon imorgon lördag 2012-06-01.
När man springer långdistanslopp och vill klara av loppet på en viss tid kan det vara bra att ha med sig en remsa där man för varje kilometer ser vid vilken tid (sedan starten) man bör passera kilometer-markeringen – man antar då konstant fart genom hela loppet.
Denna uppgift handlar om ett program för skapande av en sådan remsa.

Programmet ska skapa ett fönster som ser ut så här:





I listan upptil anges fyra alternativa lopp som användaren ska välja ett utav: Maraton på 42.2 km, Halvmaraton på 21.1 km, Lidingöloppet på 30 km eller Hässelbyloppet på 10 km.

Sedan ska användaren mata in den tid hon vill springa loppet på uppdelat i timmar och minuter, se ovan till vänster där användaren vill springa Hässelbyloppet på 1:15.
Sedan kan användaren klicka på knappen "Skriv mellantider", varvid programmet ska visa en dialogruta med följande utseende:

Alltså för varje kilometer, vid vilken tid sedan starten man ska passera. För de andra loppen blir utskriften givetvis mycket längre.

Om inmatningsfälten innehåller något annat än siffror så ska en dialogruta med ett felmeddelande visas. Samma gäller om minutfältet innehåller ett värde som är större än 59.

Något annat behöver inte kontrolleras. Du behöver inte heller bry dig om formattering av tidsangivelserna.

Beräkningar av tidsangivelser görs enklast genom att räkna om tiden till totalminuter (timmar * 60 + minuter), räkna ut mellantider i minuter och omvandla tillbaka till timmar (totalminuter / 60) resp. minuter (totalminuter % 60).

På nästa sida finns koden för ett program som skapar ett fönster enligt ovan, men saknar all funktionalitet. Modifiera/komplettera detta program så att det beter sig enligt beskrivningen ovan.

Nedan finns koden till klassen `Mellantider` som du ska modifiera/komplettera (radnumreringen ingår inte i koden, den är till för att du enkelt skall kunna förklara var du gör tillägg/ändringar). Koden är skriven i Java 6 men du får skriva din lösning i Java 7 om du vill.

```
1   import java.awt.*;
2   import javax.swing.*;
3   import javax.swing.border.*;
4
5   class Mellantider extends JFrame{
6       static final String[] LOPPTYP = {"Maraton", "Halvmaraton",
7                                "Lidingöloppet", "Hässelbyloppet"};
8       JList loppLista = new JList(LOPPTYP);
9
10      Mellantider(){
11        super("Mellantider");
12        setLayout(new BoxLayout(getContentPane(),BoxLayout.Y_AXIS));
13        loppLista.setBorder(new LineBorder(Color.BLACK));
14        loppLista.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
15        add(loppLista);
16        add(new JLabel("Måltid:"));
17        JPanel fältPanel = new JPanel();
18        add(fältPanel);
19        fältPanel.add(new JLabel("tim:"));
20        fältPanel.add(new JTextField(5));
21        fältPanel.add(new JLabel("min:"));
22        fältPanel.add(new JTextField(5));
23        JPanel knappPanel = new JPanel();
24        add(knappPanel);
25        knappPanel.add(new JButton("Skriv mellantider"));
26        setDefaultCloseOperation(EXIT_ON_CLOSE);
27        setSize(250,200);
28        setVisible(true);
29      }
30
31      public static void main(String[] args){
32        new Mellantider();
33      }
34  }
```

Uppgift 2 (12 poäng)

I ett Java-program där man bl.a. behöver hålla reda på olika personers adresser används följande klass för att representera en adress:

```
class Adress{
   private String ort;
   private int postnr;
   private String gata;

   public Adress(String ort, int postnr, String gata){
      this.ort = ort;
      this.postnr = postnr;
      this.gata = gata;
   }
   public String getOrt() { return ort; }
   public int getPostnr() { return postnr; }
   public String getGata() { return gata; }
}
```

Objekt av denna klass hanteras i följande `TreeMap`, med personernas namn som nyckel:

```
Map<String, Adress> adressLista = new TreeMap<String, Adress>();
```

Nu visar det sig att man ofta behöver ta fram alla personer som bor på en viss adress och att man vill effektivisera denna operation genom att skapa en `Map` där adressen är nyckeln och samlingen av alla personer som har denna adress utgör värdet. Denna nya `Map` bör vara en `HashMap`, den ska skapas ur den ursprungliga `Map`:en `adressLista` med en metod `makeNamesPerAddressMap`.

Din uppgift är att

a) se till att klassen `Adress` ovan kan användas som nyckel i en `HashMap`. Givetvis ska olika `Adress`-objekt som innehåller samma värden betraktas som lika

b) skriva metoden `makeNamesPerAddressMap`. Metoden ska ta den ursprungliga `Map`:en `adressLista` som argument och ur den skapa och returnera den nya `Map`:en, där adresser är nycklar och samlingar av alla namn på denna adress är värden. Obs att olika namn i den ursprungliga `Map`:en `adressLista` kan ha samma adress!

   Metoden `makeNamesPerAddressMap` bör vara statisk, men du behöver inte bry dig om vilken klass den ligger i (skriv bara metoden).

Uppgift 3 (14 poäng)

Denna uppgift går ut på att skriva en generisk klass `ArrayMap<K,V>` som implementerar ett förenklat gränssnitt `Map<K,V>`.

I Javas standardbibliotek finns ju en Map-implementering med hjälp av hashtabell (`HashMap`) och en annan med hjälp av binärt sökträd (`TreeMap`), men det saknas en Map-implementering med hjälp av en array (eller `ArrayList`). Det finns dock inget som hindrar en sådan implementering (utom att den blir ganska ineffektiv jämfört med de andra).

Du ska skriva klassen `ArrayMap<K,V>`.
Den generiska parametern K står för nyckeltypen, medan V står för värdetypen.

Klassen ska ligga i ett paket med namnet `maps`.

`ArrayMap` ska implementera ett förenklat gränssnitt `Map<K,V>` som ser ut så här:

```
public interface Map<K, V>{
   V put(K key, V value);
   V get(K key);
   V remove(K key);
   Set<K> keySet();
   Collection<V> values();
}
```

Implementeringen ska göras så att `ArrayMap` internt har en inre klass (motsvarande `Map.Entry`, men den behöver inte heta så) vars objekt ska representera nyckel/värde-par (innehålla två referenser - en till nyckeln och en till värdet) samt en `ArrayList` av sådana par-objekt.

Metoden `put` ska söka sekventiellt igenom `ArrayList`:en för att se om ett par med denna nyckel redan finns. Om det finns så ersätts det gamla värdet med det nya, och det gamla värdet returneras. Om det inte finns så skapas ett nytt objekt av par-klassen med referenser till nyckeln och värdet, och det nya paret stoppas in sist i `ArrayListen` - `put` ska i sådana fall returnera `null`.

Att två nycklar är lika avgörs med hjälp av nyckelobjektens metod `equals`.

Metoden `get` ska söka sekventiellt i `ArrayListen` efter ett par med denna nyckel. Hittas nyckeln så returneras motsvarande värde, hittas inte nyckeln så returneras `null`.

Metoden `remove` ska söka efter paret med angiven nyckel och ta bort nyckel/värde-paret från `ArrayList`:en. Om nyckeln inte hittas ska undantaget `NoSuchElementException` kastas.

Metoderna `keySet()` resp. `values()` ska returnera en mängd resp. en samling med nyckel-objekten resp. värde-objekten.

Lösningsförslag PROG2 VT12 tenta 2012-06-01

Uppgift 1

```
class Mellantider extends JFrame{
    static final String[] LOPPTYP = {"Maraton", "Halvmaraton",
                                     "Lidingöloppet", "Hässelbyloppet"};
    JList loppLista = new JList(LOPPTYP);
    static final double[] DISTANCES = {42.2, 21.1, 30, 10};
    JTextField timmarFält, minuterFält;
    Mellantider(){
       super("Mellantider");
       setLayout(new BoxLayout(getContentPane(),BoxLayout.Y_AXIS));
       loppLista.setBorder(new LineBorder(Color.BLACK));
       loppLista.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
       add(loppLista);
       add(new JLabel("Måltid:"));
       JPanel fältPanel = new JPanel();
       add(fältPanel);
       fältPanel.add(new JLabel("tim:"));
       timmarFält = new JTextField(5);
       fältPanel.add(timmarFält);
       fältPanel.add(new JLabel("min:"));
       minuterFält = new JTextField(5);
       fältPanel.add(minuterFält);
       JPanel knappPanel = new JPanel();
       add(knappPanel);
       JButton b = new JButton("Skriv mellantider");
       knappPanel.add(b);
       b.addActionListener(new SkrivutLyss());
       setDefaultCloseOperation(EXIT_ON_CLOSE);
       setSize(250,200);
       setVisible(true);
    }
    class SkrivutLyss implements ActionListener{
       public void actionPerformed(ActionEvent ave){
          try{
             int tim = Integer.parseInt(timmarFält.getText());
             int min = Integer.parseInt(minuterFält.getText());
             if (min > 59){
                JOptionPane.showMessageDialog(null, "Fel minuter!");
                return;
             }
             int sek = tim * 60 + min;
             int loppIndex = loppLista.getSelectedIndex();
             double kmTid = sek / DISTANCES[loppIndex];
             String str = "";
             for(int km = 1; km < DISTANCES[loppIndex]; km++){
                int tid = (int)(km * kmTid)
                str += km + " - " + String.format("%01d",tid / 60)+":"+
                       String.format("%02d",tid % 60) + "\n"; ;
             }
             int tid = (int)(DISTANCES[loppIndex] * kmTid);
             str += DISTANCES[loppIndex] + " - " +
                    String.format("%01d",tid / 60)+":"+
                    String.format("%02d",tid % 60) + "\n"; ;
             JOptionPane.showMessageDialog(null, str);
          }catch(NumberFormatException e){
             JOptionPane.showMessageDialog(null, "Ej numeriskt!");
           }
       }
    }
    public static void main(String[] args){
       new Mellantider();
    }
}
```

Uppgift 2

a)

```
class Adress{
    private String ort;
    private int postnr;
    private String gata;

    public Adress(String ort, int postnr, String gata){
          this.ort = ort;
          this.postnr = postnr;
          this.gata = gata;
    }

    public String getOrt() {
          return ort;
    }
    public int getPostnr() {
          return postnr;
    }
    public String getGata() {
          return gata;
    }
    public String toString(){
          return gata + " " + postnr + " " + ort;
    }

    public boolean equals(Object other){
          if (other instanceof Adress){
              Adress a = (Adress)other;
              return ort.equals(a.ort) && postnr == a.postnr &&
                    gata.equals(a.gata);
          }
          else
              return false;
    }

    public int hashCode(){
          return ort.hashCode() * 100000 + postnr + gata.hashCode();
    }
}
```

b)

```
static Map<Adress, List<String>>
                makeNamesPerAddressMap(Map<String, Adress> adressLista){
   Map<Adress, List<String>> nyMap = new HashMap<Adress, List<String>>();

   for(Map.Entry<String, Adress> me : adressLista.entrySet()){
      String namn = me.getKey();
      Adress adress = me.getValue();
      List<String> namnLista = nyMap.get(adress);
      if (namnLista == null){
         namnLista = new ArrayList<String>();
         nyMap.put(adress, namnLista);
      } // if
      namnLista.add(namn);
   } // for
   return nyMap;
} // makeNamesPerAddressMap
```

Uppgift 3

```
package maps;
import java.util.*;

public class ArrayMap<K,V> implements Map<K,V>{
    private class Pair{
           K key;
           V value;
           Pair(K key, V value){
               this.key = key;
               this.value = value;
           }
    }

    private ArrayList<Pair> data = new ArrayList<Pair>();

    private int find(K sought){
           for(int i = 0; i < data.size(); i++)
               if (data.get(i).key.equals(sought))
                      return i;
           return -1;
    }
    public V put(K key, V value){
           int pos = find(key);
           V oldValue = null;
           if (pos == -1)
               data.add(new Pair(key, value));
           else {
               Pair p = data.get(pos);
               oldValue = p.value;
               p.value = value;
           }
           return oldValue;
    }
    public V get(K key){
           int pos = find(key);
           if (pos == -1)
               return null;
           else
               return data.get(pos).value;
    }
    public V remove(K key){
           int pos = find(key);
           if (pos == -1)
               throw new NoSuchElementException("Ej hittad vid remove");
           Pair p = data.remove(pos);
           return p.value;
    }
    public Set<K> keySet(){
           Set<K> kSet = new HashSet<K>();
           for(Pair p : data)
               kSet.add(p.key);
           return kSet;
    }
    public Collection<V> values(){
           List<V> vList = new ArrayList<V>();
           for(Pair p : data)
               vList.add(p.value);
           return vList;
    }
}
```