PROG2 Tenta 2013-03-09

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.

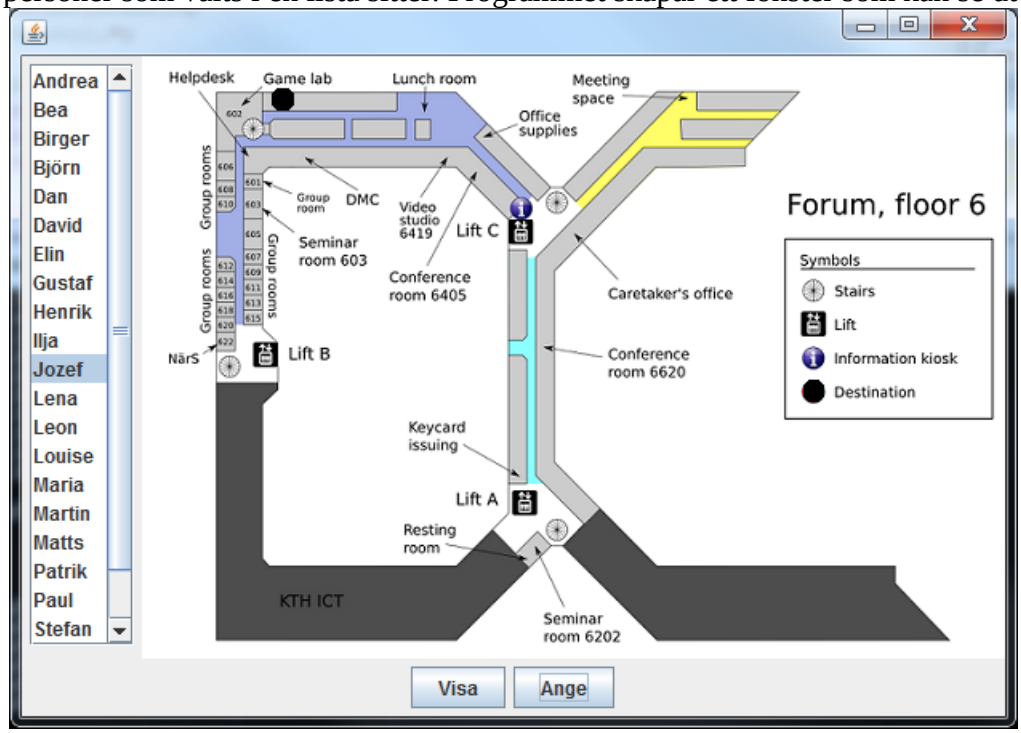
Fx - En lösning är helt felaktig eller saknas medan de övriga lösningar är i princip korrekta, men kan vara behäftade med fel och/eller underlåta att ta hänsyn till någon enstaka angiven situation eller begärd felkontroll.

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 (13 poäng)

Denna uppgift går ut på att komplettera ett program som visar en karta över ett kontor och kan visa var personer som valts i en lista sitter. Programmet skapar ett fönster som kan se ut så här:



Kartan är en bild som har lästs in från en fil - hårdkodad i programmet (se koden nästa sida). Till vänster finns en JList som innehåller namn på personer som har sina arbetsrum på kontoret – även dessa är hårdkodade i programmet.

Knappen ”Ange” är tänkt att användas av en administratör som kan välja ett namn i listan, klicka på knappen ”Ange” och sedan klicka på kartan, varvid programmet kommer ihåg positionen där den valda personen sitter. Programmet ritar också en röd cirkel kring denna position (syns som en svart cirkel ovan, uppe till vänster, just under texten ”Game lab”).

Klickar man på kartan utan att just ha klickat på knappen ”Ange” så ska ingenting hända.

Knappen ”Visa” används av besökare: de kan välja ett namn i listan och klicka på ”Visa”, och då visas cirkeln på kartan på den plats som sparats tidigare vid ”Ange”.

Tänk inte på några behörighetskontroller för knappen ”Ange”, vi låtsas som att ingen obehörig någonsin skulle komma på tanken att använda denna knapp.

Om ingen är vald i listan när man klickar på en knapp ska en meddelanderuta med ett felmeddelande visas. Om man klickar på knappen ”Visa” och den valda personen inte har fått någon plats ska meddelandet ”Ingen plats angiven!” visas i en meddelanderuta. Om man klickar på knappen ”Ange” och anger en ny plats för en person som redan har fått en plats så är det den nya platsen som gäller i fortsättningen (en person kan ha högst en plats).

På nästa sidan finns koden för klassen `Karta` som du ska komplettera. Det är kodat i Java 6 men du kan koda i Java 7 om du föredrar det. I Java 7 skulle rad 34 se ut så här:

```
JList<String> folkList = new JList<String>(namn);
```

Nedan finns koden till klassen `Karta` 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).

```
1     import javax.swing.*;
2     import java.awt.*;
3     import java.util.*;
4
5     class KartPanel extends JPanel{
6         private ImageIcon bild = new ImageIcon("plan6.png");
7         private int x, y;
8         private boolean visas = false;
9         protected void paintComponent(Graphics g){
10          super.paintComponent(g);
11          g.drawImage(bild.getImage(), 0,0, this);
12          if (visas){
13              g.setColor(Color.RED);
14              g.fillOval(x-10,y-10,15,15);
15          }
16        }
17        public void setXY(int x, int y){
18          this.x = x;
19          this.y = y;
20        }
21        public void setVisas(boolean b){
22          visas = b;
23          repaint();
24        }
25    }
26    class Karta extends JFrame{
27        String[] namn = {"Andrea", "Bea", "Birger", "Björn", "Dan", "David",
28                    "Elin", "Gustaf", "Henrik", "Ilja", "Jozef", "Lena",
29                    "Leon", "Louise", "Maria", "Martin", "Matts", "Patrik",
30                    "Paul", "Stefan", "Stewart", "Tessi"};
31        Karta(){
32          JPanel höger = new JPanel();
33          add(höger, BorderLayout.WEST);
34          JList folkList = new JList(namn);
35          folkList.setVisibleRowCount(20);
36          höger.add(new JScrollPane(folkList));
37          KartPanel kp = new KartPanel();
38          add(kp, BorderLayout.CENTER);
39          JPanel knappPanel = new JPanel();
40          add(knappPanel, BorderLayout.SOUTH);
41          knappPanel.add(new JButton("Visa"));
42          knappPanel.add(new JButton("Ange"));
43          setDefaultCloseOperation(EXIT_ON_CLOSE);
44          setSize(630, 450);
45          setVisible(true);
46        }
47        public static void main(String[] args){
48          new Karta();
49        }
50    }
```

Uppgift 2 (12 poäng)

I ett Java-program som ska hålla reda på passagerarlistor på flygplan har man följande klass för att representera en plats i flygplanet:

```
public class Plats{
    private int rad;
    private String stol;

    public Plats(int rad, String stol){
      if (rad < 1 || rad > 30 || !(stol.equals("A") || stol.equals("B")
           || stol.equals("C") || stol.equals("D") || stol.equals("E")))
          throw new IllegalArgumentException();
      this.rad = rad;
      this.stol = stol;
    }

    public int getRad(){
      return rad;
    }

    public String getStol(){
      return stol;
    }

    public String toString(){
      return "" + rad + stol;
    }
}
```

En plats identifieras alltså av radnummer och inom raden av en bokstavsbeteckning A till E.

a) Man vill kunna representera en passagerarlista med hjälp av en `TreeMap` med `Plats`-objekt som nycklar och passagerarnas namn (`String`) som värden.
Förbered klassen `Plats` för att dessa objekt ska kunna användas som nycklar i en `TreeMap`.

b) Skriv en metod som tar en sådan `TreeMap` som argument och skriver ut passagerarlistan (med plats och namn för varje passagerare) på `System.out` sorterad i platsnummerordning

c) Skriv en metod som tar en sådan `TreeMap` som argument och skriver ut passagerarlistan (med plats och namn för varje passagerare) på `System.out` sorterad i alfabetisk ordning efter passagerarnas namn

Uppgift 3 (13 poäng)

I många tillämpningar händer det att man har objekt som alternativt kan identifieras med två nycklar och där sökning efter båda nycklarna är lika viktig (t.ex. personer i ett kundregister som ska kunna sökas med hjälp av personnummer eller kundnummer).

Denna uppgift går ut på att skriva ett gränssnitt (*interface*) `DoubleKeyMap<K1,K2,V>` och en implementerande klass `DoubleKeyHashMap<K1, K2, V>` för sådana ändamål.

En `DoubleKeyMap` tänks ha följande metoder

- `put` som tar tre argument: första nyckeln, andra nyckeln och värdet. Metoden ska stoppa in nycklarna och värdet i datastrukturen. Om det redan finns ett värde för någon av nycklarna så ersätts det gamla värdet med det nya. Metoden returnerar ingenting.
- `getByFirstKey` - tar första nyckeln och returnerar värdet för denna nyckel eller `null` om det inte finns.
- `getBySecondKey` - tar andra nyckeln och returnerar värdet för denna nyckel eller `null` om det inte finns
- `firstKeySet` - returnerar mängden av alla första-nycklar
- `secondKeySet` - returnerar mängden av alla andra-nycklar
- `values` - returnerar samlingen av all värden

Den implementerande klassen `DoubleKeyHashMap` implementeras enklast genom att man internt har två `HashMap`:ar som håller reda på värdena identifierade med var sin nyckeln.

Man kan tänka sig att en tillämpning skulle kunna använda detta på följande sätt (tänk att man här har kundernas namn som första nyckel, kundnummer som andra nyckel och att värdena utgörs av kundernas saldo):

```
class KundReg{
    public static void main(String[] args){
      DoubleKeyMap<String, Integer, Integer> kunder =
          new DoubleKeyHashMap<String, Integer, Integer>();

      kunder.put("jozef", 1234, 300);
      kunder.put("stefan", 2345, 500);
      kunder.put("jonathan", 3456, 100);

      int saldo1 = kunder.getByFirstKey("jozef");
      int saldo2 = kunder.getBySecondKey(2345);

      int summa = 0;
      for(Integer ig : kunder.values())
          summa += ig;
      System.out.println("Totalt: " + summa);

    }
}
```

Lösningsförslag PROG2 HT12 tenta 2013-03-09

Uppgift 1

Kommentar: för att spara positionen på kartan för var personer sitter behöver man en Map där namnet är nyckeln och positionen värdet. Men positionen består av två värden – koordinaterna x och y. Dessa behöver förpackas på något sätt för att kunna läggas som värden i Map:en.
I lösningen nedan använder jag klassen Point från java.awt - man kan även konstruera en egen klass för detta eller använda någon lämplig datastruktur, eller "packa" koordinaterna i ett heltal genom någon aritmetisk operation m.fl. möjliga lösningar.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
class KartPanel extends JPanel{
    private ImageIcon bild = new ImageIcon("plan6.png");
    private int x, y;
    private boolean visas = false;
    protected void paintComponent(Graphics g){
          super.paintComponent(g);
          g.drawImage(bild.getImage(), 0,0, this);
          if (visas){
              g.setColor(Color.RED);
              g.fillOval(x-10,y-10,15,15);
          }
    }
    public void setXY(int x, int y){
          this.x = x;
          this.y = y;
    }
    public void setVisas(boolean b){
          visas = b;
          repaint();
    }
}
class Karta extends JFrame{
    String[] namn = {"Andrea","Bea","Birger","Björn","Dan","David","Elin",
                     "Gustaf","Henrik","Ilja","Jozef","Lena","Leon","Louise",
                     "Maria","Martin","Matts","Patrik","Paul","Stefan",
                     "Stewart","Tessi"};
    KartPanel kp = new KartPanel();
    JList folkList = new JList(namn);
    Map<String, Point> where = new HashMap<String,Point>();
    Karta(){
          JPanel höger = new JPanel();
          add(höger, BorderLayout.WEST);
          folkList.setVisibleRowCount(20);
          höger.add(new JScrollPane(folkList));

          add(kp, BorderLayout.CENTER);

          JPanel knappPanel = new JPanel();
          add(knappPanel, BorderLayout.SOUTH);
          JButton visaKnapp = new JButton("Visa");
          knappPanel.add(visaKnapp);
          visaKnapp.addActionListener(new VisaLyss());
          JButton angeKnapp = new JButton("Ange");
          knappPanel.add(angeKnapp);
          angeKnapp.addActionListener(new AngeLyss());

          setDefaultCloseOperation(EXIT_ON_CLOSE);
          setSize(630, 450);
          setVisible(true);
    }
```

```
    class AngeLyss implements ActionListener{
        public void actionPerformed(ActionEvent ave){
           kp.setVisas(false);
           kp.addMouseListener(new MusLyss());

        }
    }

    class MusLyss extends MouseAdapter{
        public void mouseClicked(MouseEvent mev){
           if (folkList.isSelectionEmpty()){
              JOptionPane.showMessageDialog(Karta.this, "Ingen vald!");
                 return;
          }

          String vem = (String)folkList.getSelectedValue();
          int x = mev.getX();
          int y = mev.getY();
          where.put(vem, new Point(x,y));
          kp.setXY(x, y);
          kp.setVisas(true);
          kp.removeMouseListener(this);
        }
    }

    class VisaLyss implements ActionListener{
        public void actionPerformed(ActionEvent ave){
           if (folkList.isSelectionEmpty()){
              JOptionPane.showMessageDialog(Karta.this, "Ingen vald!");
              return;
           }

           String vem = (String)folkList.getSelectedValue();
           Point p = where.get(vem);
           if (p == null){
              JOptionPane.showMessageDialog(Karta.this, "Ingen plats!");
              return;
           }

           kp.setXY(p.x, p.y);
           kp.setVisas(true);
        }
    }

    public static void main(String[] args){
          new Karta();
    }
}
```

Uppgift 2

a)

```
public class Plats implements Comparable<Plats>{
    private int rad;
    private String stol;

    public Plats(int rad, String stol){
          if (rad < 1 || rad > 30 || !(stol.equals("A") || stol.equals("B")
||
              stol.equals("C") || stol.equals("D")  || stol.equals("E")))
              throw new IllegalArgumentException();
          this.rad = rad;
          this.stol = stol;
    }
    public int getRad(){
          return rad;
    }
    public String getStol(){
          return stol;
    }
    public String toString(){
          return "" + rad + stol;
    }
    public int compareTo(Plats other){
          if (rad < other.rad)
              return -1;
          else if (rad > other.rad)
              return 1;
          else
              return stol.compareTo(other.stol);
    }
}
```

b)

```
 static void passagerarListaPerPlats(TreeMap<Plats, String> plista){
    for(Map.Entry<Plats, String> me : plista.entrySet())
       System.out.println(me.getKey() + "\t" + me.getValue());
 }
```

c)
Kommentar: en generell lösning (som ger ett mönster som skulle kunna användas till annat än utskrifter) går ut på att man tar ut platsbeteckningar och namnen ur Map:en och lägger dem i en lista, sorterar om listan och sedan går igenom den sorterade listan. Då måste platsen och namnet packas ihop i objekt av någon klass, t.ex. klassen PlatsOchNamn nedan. Om en lista med sådana objekt ska kunna sorteras så måste PlatsOchNamn göras Comparable eller så får man skapa en fristående Comparator för den. Jag väljer en fristående Comparator för det skulle kännas konstigt att göra en compareTo-metod som ignorerar ett av attributen i klassen.
Se längre ner för en enklare lösning.

```
class PlatsOchNamn{
      private Plats plats;
      private String namn;
      public PlatsOchNamn(Plats plats, String namn){
          this.plats = plats;
          this.namn = namn;
      }
      public String getNamn(){
          return namn;
      }
      public String toString(){
          return plats + ": " + namn;
      }
}

class NamnCMP implements Comparator<PlatsOchNamn>{
      public int compare(PlatsOchNamn pon1, PlatsOchNamn pon2){
          return pon1.getNamn().compareTo(pon2.getNamn());
      }
}

static void passagerarListaPerNamn(TreeMap<Plats,String> plista){
      ArrayList<PlatsOchNamn> lista = new ArrayList<PlatsOchNamn>();
      for(Map.Entry<Plats, String> me : plista.entrySet())
          lista.add(new PlatsOchNamn(me.getKey(), me.getValue()));

      Collections.sort(lista, new NamnCMP());
      for(PlatsOchNamn pon : lista)
          System.out.println(pon);
}
```

I tentauppgiften är det explicit angivet att metoden skulle användas för utskrift, så då kan man göra en mycket enklare lösning genom att vid genomgången av Map:en skapa strängar innehållande strängrepresentationen av platsbeteckning och namnet (helt enkelt de rader som ska skrivas ut) och sortera dessa strängar – ser man till att namnet står först så blir de sorterade i namnordning:

```
static void passagerarListaPerNamn(TreeMap<Plats,String> plista){
      ArrayList<String> lista = new ArrayList<String>();
      for(Map.Entry<Plats, String> me : plista.entrySet())
          lista.add("" + me.getValue() + "\t" + me.getKey());

      Collections.sort(lista);
      for(String rad : lista)
          System.out.println(rad);
}
```

Uppgift 3

```
import java.util.*;

public interface DoubleKeyMap<K1, K2, V>{
    void put(K1 firstKey, K2 secondKey, V value);
    V getByFirstKey(K1 key);
    V getBySecondKey(K2 key);
    Set<K1> firstKeySet();
    Set<K2> secondKeySet();
    Collection<V> values();
}
```

I implementeringen av metoden put nedan borde man egentligen kolla att `firstKey` och `secondKey` stämmer: de får antingen båda vara `null` (då skapas en ny post) eller så får ingen av dem vara null och deras värden måste vara samma som angavs tidigare (t.ex. kan man inte ha samma personnr med ett annat kundnr och vice versa). En sådan kontroll skulle kunna göras med två till interna `Map`:as, en `Map<K1, K2>` och en `Map<K2, K1>`. Det begärdes ingen sådan kontroll på tentan så jag gör den inte i lösningsförslaget.

```
import java.util.*;

public class DoubleKeyHashMap<K1, K2, V> implements DoubleKeyMap<K1,K2,V>{
    private Map<K1, V> firstMap = new HashMap<K1, V>();
    private Map<K2, V> secondMap = new HashMap<K2, V>();

    public void put(K1 firstKey, K2 secondKey, V value){
          firstMap.put(firstKey, value);
          secondMap.put(secondKey, value);
    }

    public V getByFirstKey(K1 key){
          return firstMap.get(key);
    }
    public V getBySecondKey(K2 key){
          return secondMap.get(key);
    }
    public Set<K1> firstKeySet(){
          return firstMap.keySet();
    }
    public Set<K2> secondKeySet(){
          return secondMap.keySet();
    }
    public Collection<V> values(){
          return firstMap.values();
    }
}
```