PROG2 Tenta 2013-06-07

Gäller SP:PROG2, DSK2:PROG2, FK:PROG2, FK:OOP, DSV1:P2 och ITK: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 nästa 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.

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)

Denna uppgift går ut på att komplettera ett program för en självbetjäningskassa i en liten livsmedelsbutik. Kassan visar följande fönster för en kund (bilden till vänster):

Meningen är att en kund ska kunna mata in ett antal[1] i textfältet nedtill och klicka i rutan som anger varan kunden har köpt. Rutorna representeras av knappar (objekt av klassen `JButton`). Programmet räknar ut beloppet (multiplicerar antalet med varans pris (se nedan)), skriver ut varan, antalet och beloppet i en rad i textarean och adderar beloppet totalbeloppet för kunden (se bilden ovan till höger).

Om textfältet med antal inte är ifyllt med numerisk information när användaren klickat i en ruta ska ett felmeddelande visas:

Knappen ”Ny kund” överst ska bara nollställa totalbeloppet och blankställa textarean.

Varornas priser är fasta och ska vara hårdkodade enligt följande:

Gurka	Paprika	Tomat	Mjölk	Yoghurt	Grädde	Bröd	Müsli	Kakor
11.70	46.30	27.35	7.50	15.50	13.72	23.20	27.30	13.00

På nästa sida finns koden till en klass `Kassa` som skapar ett sådant fönster. Tyvärr saknas funktionaliteten – inget händer när man klickar i rutorna o.s.v. Prisuppgifterna saknas också. Din uppgift är att komplettera klassen `Kassa` så att den fungerar enligt ovanstående.

Fortsättning på nästa sida

[1] I den här butiken säljs allting styckevist

Här följer koden till klassen `Kassa` (radnumreringen ingår ej i koden, den är till för att du enkelt ska kunna ange var ni gör tillägg/ändringar):

```
1    import javax.swing.*;
2    import java.awt.*;
3
4    class Kassa extends JFrame{
5        final String[] VAROR = {"Gurka", "Paprika", "Tomat",
6                        "Mjölk", "Yoghurt", "Grädde",
7                        "Bröd", "Müsli", "Kakor"};
8
9        public Kassa(){
10        super("Kassa");
11        JPanel north = new JPanel();
12        add(north, BorderLayout.NORTH);
13
14        JButton nyKundKnapp = new JButton("Ny kund");
15        north.add(nyKundKnapp);
16
17        JPanel center = new JPanel();
18        center.setLayout(new GridLayout(2,1));
19        add(center, BorderLayout.CENTER);
20        JPanel knappar = new JPanel();
21        center.add(knappar);
22        knappar.setLayout(new GridLayout(3,3,10,10));
23        for(int i = 0; i < VAROR.length; i++){
24            JButton b = new JButton(VAROR[i]);
25            knappar.add(b);
26        }
27        JTextArea display = new JTextArea();
28        center.add(new JScrollPane(display));
29
30        JPanel south = new JPanel();
31        add(south, BorderLayout.SOUTH);
32        south.add(new JLabel("Antal: "));
33        south.add(new JTextField(5));
34        south.add(new JLabel("Totalbelopp = "));
35        south.add(new JLabel("0"));
36
37        setDefaultCloseOperation(EXIT_ON_CLOSE);
38        setSize(300, 300);
39        setVisible(true);
40       }
41
42       public static void main(String[] args){
43        new Kassa();
44       }
45   }
```

Uppgift 2 (13 poäng)

Detta handlar om en stor flygplats där olika flyg mot flera destinationer kan starta vid samma klockslag. I ett program för hantering av detta representeras destionationer med strängar (namn på staden) medan klockslag representeras med objekt av följande klass:

```
class Klocka{
      private int timme, minuter;
      public Klocka(int timme, int minuter){
         this.timme = timme;
         this.minuter = minuter;
      }
      public String toString(){
         return timme+":"+minuter;
      }
}
```

Klockslag för avgångar mot de olika destinationerna samlas i en Map, t.ex.
`Map<String, List<Klocka>> departures = new HashMap<String, List<Klocka>>();`

En utskrift av informationen i denna Map skulle kunna se ut så här:

```
Wien  7:20 9:50 11:45 16:40
London 7:20 10:35 11:45 12:20 15:40 16:40
Krakow 10:35
Oslo 7:20 8:35 9:50 10:35 12:20 16:40
```

Nu behöver man en metod som tar en sådan Map som argument och "vänder på den": metoden ska skapa och returnera en ny Map där man för varje klockslag med avgångar kan se vart flygen går.

Metoden ska ha följande signatur:
`static Map<Klocka,List<String>> vändMap(Map<String,List<Klocka>> orgMap);`

En utskrift av informationen i den nya Mapen skulle kunna se ut så här:

```
7:20  Wien London Oslo
8:35  Oslo
9:50  Wien Oslo
10:35 London Krakow Oslo
11:45 Wien London
12:20 London Oslo
15:40 London
16:40 Wien London Oslo
```

Obs! dock att din metod **inte** ska göra någon utskrift utan skapa och returnera en ny Map.

Vid en genomgång av den nya Map:en ska informationen vara sorterad efter klockslagen. Ordningen mellan destinationer för ett visst klockslag är oväsentlig.

Ur den nya Mapen ska man som vanligt kunna få fram en lista med destinationer för ett visst klockslag, t.ex.:
`List<String> destinationer = nyaMapen.get(new Klocka(10, 35));`
och få en lista med strängarna "London", "Krakow", "Oslo".

Du får göra tillägg till klassen `Klocka` om du behöver det.

Uppgift 3 (13 poäng)

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

Bakgrund: en HashMap är mycket snabb på att lagra ett nyckel/värde-par och på att söka upp ett värde med ledning av en nyckel, men när man går igenom alla nycklar eller värden så kommer dessa i en konstig ordning[2]. Ofta är snabbheten är viktig när man söker efter värden, men inte lika viktig vid andra operationer, som genomgång av alla nycklar eller värden, eller lagring av nytt nyckel/värde-par.
Man kan alltså göra en datastrukturklass som internt använder en HashMap för sökningar, men kompletterar andra operationer på något sätt så att nycklar och värden presenteras i sorteringsordning efter nycklarna vid genomgång.

Skriv klassen `SortedHashMap<K,V>` som ska implementera följande förenklade Map-gränssnitt:

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

Metoderna i `SortedHashMap` ska bete sig på följande sätt:

- `get` – tar emot en nyckel och returnerar motsvarande värde eller null om nyckeln inte finns lagrad. Denna metod måste vara lika effektiv som motsvarande metod i HashMap
- `put` – tar emot en nyckel och ett värde och lagrar dessa. Om denna nyckel redan finns så ersätts det gamla värdet med det nya. Metoden returnerar det gamla värdet eller null om nyckeln inte fanns lagrad. Denna metod behöver inte vara lika effektiv som motsvarande metod i HashMap
- `keySet` – returnerar en mängd med alla nycklar, mängden ska vara sorterad i nycklarnas naturliga ordning. Denna metod får vara mindre effektiv än motsvarande metod i HashMap.
- `values` – returnerar en samling (egentligen en lista) med alla värden, värdena i samlingen ska vara i samma ordning som motsvarande nycklar har i ett keySet. Metoden får vara mindre effektiv än motsvarande metod i HashMap.

Av de generiska typparametrarna för `SortedHashMap` står `K` för nyckeltypen och `V` för värdetypen.
Obs att nycklarna måste kunna jämföras med varandra (annars kan man ju inte prata om sorteringsordning). Detta måste man ange som krav på nyckeltypen `K` när man deklarerar klassen `SortedHashMap`.

[2] Ordningen bestäms som bekant av nycklars hashCode och HashMap:ens kapacitet

Lösningsförslag PROG2 VT13 tenta 2013-06-07

Uppgift 1

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

class Kassa extends JFrame{
    private static final String[] VAROR = {"Gurka", "Paprika", "Tomat",
                                           "Mjölk", "Yoghurt", "Grädde",
                                           "Bröd", "Müsli", "Kakor"};
    private static final double[] PRISER = {11.7,46.3,27.3,7.5,15.0,
                                            13.7,23.2,27.3,13.0};
     private JTextField antalFält;
     private JLabel beloppLabel;
     private double totalt = 0.0;
     private Map<String,Double> prisLista = new HashMap<>();
     private JTextArea area = new JTextArea();

    Kassa(){
          super("Kassa");

          for(int i = 0; i < VAROR.length; i++)
              prisLista.put(VAROR[i], PRISER[i]);

          JPanel north = new JPanel();
          add(north, BorderLayout.NORTH);

          JButton nyKundKnapp = new JButton("Ny kund");
          north.add(nyKundKnapp);
          nyKundKnapp.addActionListener(new NyKundLyss());

          JPanel center = new JPanel();
          add(center, BorderLayout.CENTER);
          center.setLayout(new GridLayout(2,1));
          JPanel knappar = new JPanel();
          center.add(knappar);
          knappar.setLayout(new GridLayout(3,3,10,10));

          VaraLyss varaLyss = new VaraLyss();
          for(int i=0; i<VAROR.length; i++){
              JButton b = new JButton(VAROR[i]);
              knappar.add(b);
              b.addActionListener(varaLyss);
          }

          center.add(new JScrollPane(area));
          JPanel south = new JPanel();
          add(south, BorderLayout.SOUTH);
          south.add(new JLabel("Antal: "));
          antalFält = new JTextField(5);
          south.add(antalFält);
          south.add(new JLabel("Totalbelopp = "));
          beloppLabel = new JLabel(""+totalt);
          south.add(beloppLabel);

          setDefaultCloseOperation(EXIT_ON_CLOSE);
          setSize(300, 300);
          setVisible(true);
    } // Kosntruktorn
```

```
    class NyKundLyss implements ActionListener{
          public void actionPerformed(ActionEvent ave){
              totalt = 0;
              beloppLabel.setText(""+totalt);
              antalFält.setText("");
              area.setText("");
          } // actionPerformed
     } // NyKundLyss

     class VaraLyss implements ActionListener{
          public void actionPerformed(ActionEvent ave){
              try{
                   JButton b = (JButton)ave.getSource();
                   String vara = b.getText();
                   int antal = Integer.parseInt(antalFält.getText());
                   double belopp = antal * prisLista.get(vara);
                   area.append(vara + " " + antal + " " +
                               String.format("%.2f", belopp) + "\n");
                   totalt += belopp;
                   beloppLabel.setText(String.format("%.2f",totalt));
              }catch(NumberFormatException nfe){
                   JOptionPane.showMessageDialog(null, "Fel antal!");
              } // catch
          } // actionPerformed
     } // VaraLyss

    public static void main(String[] args){
          new Kassa();
    } // main

} // Kassa
```

Uppgift 2

```
class Klocka implements Comparable<Klocka>{
    private int timme, minuter;
    public Klocka(int timme, int minuter){
          this.timme = timme;
          this.minuter = minuter;
    }
    public String toString(){
          return timme+":"+minuter;
    }

    public int compareTo(Klocka other){
          int cmp = timme - other.timme;
          if (cmp != 0)
              return cmp;
          else
              return minuter - other.minuter;
    }
```

```
static Map<Klocka,List<String>> vändMap(Map<String,List<Klocka>> orgMap){
   Map<Klocka, List<String>> resultat = new TreeMap<>();
   for(Map.Entry<String, List<Klocka>> me : orgMap.entrySet()){
      String dest = me.getKey();
      List<Klocka> tider = me.getValue();
      for(Klocka kl : tider){
         List<String> dests = resultat.get(kl);
         if (dests == null){
            dests = new ArrayList<String>();
            resultat.put(kl, dests);
         }
         dests.add(dest);
      } // for Klocka
   } // for Map.Entry
   return resultat;
}
```

Uppgift 3

```
import java.util.*;

class SortedHashMap<K extends Comparable<K>, V> implements Map<K,V>{
    private Map<K, V> hashMap = new HashMap<K, V>();
    private Map<K,V> treeMap = new TreeMap<K, V>();

    public V get(K key){
         return hashMap.get(key);
    }

    public V put(K key, V value){
         V oldValue = hashMap.put(key, value);
         treeMap.put(key, value);
         return oldValue;
    }

    public Set<K> keySet(){
         return new TreeSet<K>(treeMap.keySet());
    }

    public Collection<V> values(){
         return new ArrayList<V>(treeMap.values());
    }
}
```