PROG2 Tenta 2011-05-21

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

Tentan består av tre uppgifter. Max poäng är 30.
För betyget E (godkänd) krävs minst 18 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. 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.

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 omständiga än nödvändigt (t.ex. skrivna med egen kod där standardklasser/-metoder kunde ha använts).

D - Som för betyget C utom att någon eller några av 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 - Som för E ovan utom att någon eller lösning är helt felaktig eller saknas.

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 här tentan.

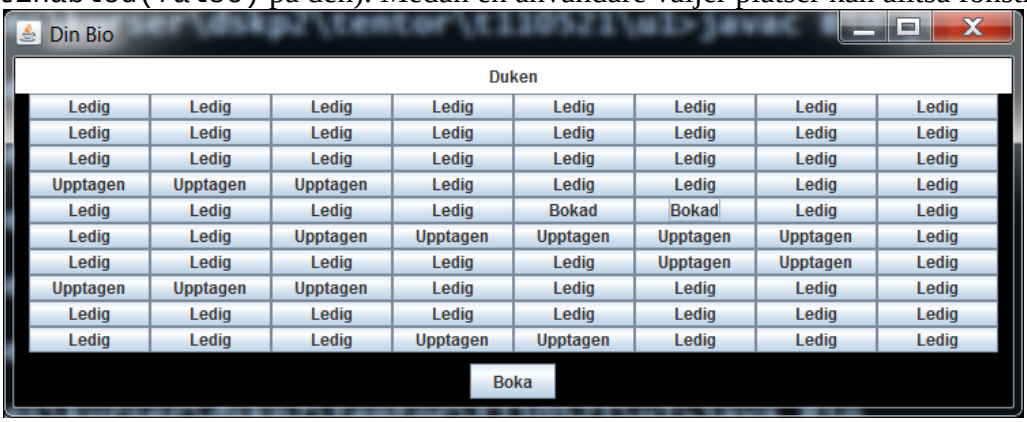
Uppgift 1 (9 poäng)

Uppgiften går ut på att komplettera ett litet program för bokning av biobiljetter (givetvis mycket förenklat). Programmet ska skapa ett fönster med följande utseende:

Rutorna representerar stolar i biosalongen (överst finns bara en JLabel med vit bakgrund för att visa var bioduken finns). Stolarna är ordnade i 10 rader med 8 stolar i varje rad. Stolarna är numrerade från 1 överst till vänster till 80 nederst till höger (radvis), men det syns inte på skärmen.

Användaren ska kunna klicka på rutor med texten "Ledig" och då ska texten ändras till "Bokad".
Om användaren klickar på en ruta där det står "Bokad" så ändras texten tillbaka till "Ledig".
När användaren har bestämt sig för vilka platser hon vill ha klickar hon på knappen "Boka" – då ändras de rutor som var markerade med "Bokad" till "Upptagen" och en meddelanderuta med de valda platsnumren visas:

Nu kan nästa användare boka platser, men då är givetvis de platser som bokats av tidigare användare upptagna (markerade med "Upptagen"). Rutor som är markerade med "Upptagen" ska inte reagera på användarens klickningar alls (kan ordnas genom att ta bort lyssnaren från en upptagen plats eller genom att göra `setEnabled(false)` på den). Medan en användare väljer platser kan alltså fönstret se ut så här:



där aktuell användare har valt men ännu inte bokat två platser på rad 5 (plats 36 och 37).
På nästa sida finns koden som du ska komplettera.

Nedan finns koden till klassen `Bio` som du ska 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
4     class Plats extends JButton{
5         private int nr;
6         public Plats(int nr){
7           this.nr = nr;
8         }
9         public int getNr(){
10          return nr;
11        }
12    }
13
14    class Bio extends JFrame{
15        Bio(){
16          super("Din Bio");
17          JPanel salongen = new JPanel();
18          salongen.setLayout(new GridLayout(10, 8));
19          for(int i=0; i < 80; i++){
20              Plats p = new Plats(i);
21              p.setText("Ledig");
22              salongen.add(p);
23          }
24          add(salongen);
25          JPanel north = new JPanel();
26          add(north, BorderLayout.NORTH);
27          north.setBackground(Color.WHITE);
28          north.setOpaque(true);
29          north.add(new JLabel("Duken"));
30          JPanel south = new JPanel();
31          add(south, BorderLayout.SOUTH);
32          south.add(new JButton("Boka"));
33          setDefaultCloseOperation(EXIT_ON_CLOSE);
34          setSize(750,300);
35          setVisible(true);
36        }
37
38        public static void main(String[] args){
39          new Bio();
40        }
41    }
```

Uppgift 2 (9 poäng)

Tidningar och andra periodiska publikationer identifieras med ett nummer som kallas ISSN (International Standard Serial Number). Det är ett åttasiffrigt nummer som brukar presenteras som två fyrsiffriga nummer med bindesträck emellan, t.ex. har Dagens Nyheter ISSN 1101-2447. Den sista siffran är en kontrollsiffra som dock kan vara 10 och då presenteras som ett X (t.ex. har Dagens Industri ISSN 0346-640X).

Nedan finns en klass som tänks representera ISSN (bortse från att det saknas felkontroller):

```
public class ISSN{
    private int part1, part2;
    private String checkDig;

    public ISSN(int part1, int part2, String checkDig){
        this.part1=part1; this.part2=part2; this.checkDig=checkDig;
    }

    public String toString(){
        return "ISSN " + part1 + "-" + part2 + checkDig;
    }
}
```

a) Objekt av ovanstående klass ska givetvis kunna användas som nycklar i olika Map:ar där två objekt med samma värden ska betraktas som lika. Komplettera klassen ISSN så att den kan användas som nyckeltyp i `HashMap` resp. `TreeMap`.

b) En annan klass representerar information om tidningar, klassen ser ut så här:

```
public class Tidning{
    private ISSN issn;
    private String namn;
    private int upplaga;

    Tidning(ISSN issn, String namn, int upplaga){
      this.issn = issn; this.namn = namn; this.upplaga = upplaga;
    }
    public ISSN getISSN(){ return issn; }
    public String getNamn() { return namn; }
    public int getUpplaga() { return upplaga; }
    public String toString() {
      return issn.toString()+" "+namn+" "+upplaga;
    }
}
```

Man behöver nu en metod

```
static void lista(Map<ISSN, Tidning> tidningar, int sortOrdning);
```

Metoden ska skriva ut alla tidningar i `Map:en` `tidningar` i den sorteringsordning som framgår av argumentet `sortOrdning`: om `sortOrdning` är 0 ska tidningarna i utskriften vara sorterade på ISSN, om den är 1 ska de vara sorterade efter namn och om den är 2 ska de vara sorterade efter upplagan (de med störst upplaga först).

Utskriften ska göras till `System.out`.

Uppgift 3 (12 poäng)

I litteraturen om datastrukturer beskrivs en enkel datastruktur med namnet *Bag* som inte finns i Javas standardbibliotek. En Bag är som en Set (mängd) utom att den inte filtrerar bort dubbletter - man kan addera samma (eller snarare lika) element flera gånger och de lagras i Bag:en lika många gånger som man adderat dem. En Bag brukar även kallas MultiSet.

Man skulle kunna implementera en Bag med hjälp av en `ArrayList` eller `LinkedList` (de tillåter ju dubletter) men då blir alla operationer långsamma – för att avgöra om ett element finns i Bag:en eller inte måste man söka sekvensiellt efter elementet.

Ett annat sätt (som du ska använda här) att implementera en Bag men inte offra effektivitet är att använda en snabb datastruktur (hashtabell eller träd) som inte tillåter dubletter, men att tillsammans med varje element lagra en uppgift om hur många gånger elementet finns i Bag:en. Utåt kan det verka som om det finns många lika element i Bag:en men internt finns varje element bara en gång, tillsammans med en räknare.

Denna uppgift går ut på att skriva ett **generiskt gränssnitt (interface) `Bag<E>`** som deklarerar funktionaliteten (metoderna) hos en Bag (se nedan för beskrivning av funktionaliteten) samt en **generisk klass `HashBag<E>`** som implementerar denna funktionlitet med hjälp av någon hashtabell-baserad samling (alltså `HashSet` eller `HashMap`). Den generiska parametern `E` är typen av objekt som hanteras i Bag:en.

Gränssnittet `Bag<E>` och klassen `HashBag<E>` ska ligga i ett **paket med namnet `bags`**.

Funktionalitet hos en Bag ska vara följande:

- `add` - tar som argument ett objekt av den typ som hanteras i Bag:en och adderar objektet till Bag:en. Returnerar ingenting.
- `contains` - tar som argument ett objekt av den typ som hanteras i Bag:en och returnerar `true` om det i Bag:en finns ett (eller flera) element som är lika med argumentobjektet, `false` annars.
- `count` - tar som argument ett objekt av den typ som hanteras i Bag:en och returnerar hur många element som är lika med argumentobjektet som finns i Bag:en (en `int`)
- `remove` - tar som argument ett objekt av den typ som hanteras i Bag:en och tar bort från Bag:en **ett** element som är lika med argumentobjektet. Om det fanns flera sådana element så tas alltså bara ett av dem bort, de övriga är kvar (internt minskas endast en räknare med 1 – det är bara när räknaren skulle ha blivit 0 som man verkligen tar bort elementet). Metoden returnerar ingenting. Om det inte fanns något element i Bag:en som var lika med argumentobjektet ska undantaget `NoSuchElementException` genereras

Lösningsförslag PROG2 VT11 Tenta 2011-05-21

Uppgift 1

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

class Plats extends JButton{
    private int nr;
    public Plats(int nr){
          this.nr = nr;
    }
    public int getNr(){
          return nr;
    }
}

class Bio extends JFrame{
    Set<Plats> bokadePlatser = new HashSet<Plats>();

    Bio(){
          super("Din Bio");
          JPanel salongen = new JPanel();
          salongen.setLayout(new GridLayout(10, 8));
          PlatsLyss pl = new PlatsLyss();
          for(int i=0; i < 80; i++){
              Plats p = new Plats(i);
              p.setText("Ledig");
              p.addActionListener(pl);
              salongen.add(p);
          }
          add(salongen);
          JPanel north = new JPanel();
          add(north, BorderLayout.NORTH);
          north.setBackground(Color.WHITE);
          north.setOpaque(true);
          north.add(new JLabel("Duken"));
          JPanel south = new JPanel();
          add(south, BorderLayout.SOUTH);
          JButton bokaButton = new JButton("Boka");
          south.add(bokaButton);
          bokaButton.addActionListener(new BokaLyss());
          setDefaultCloseOperation(EXIT_ON_CLOSE);
          setSize(750,300);
          setVisible(true);
    }

    class PlatsLyss implements ActionListener{
          public void actionPerformed(ActionEvent ave){
              Plats p = (Plats)ave.getSource();
              if (p.getText().equals("Bokad")){
                    p.setText("Ledig");
                    bokadePlatser.remove(p);
              }
              else {
                    p.setText("Bokad");
                    bokadePlatser.add(p);
              }
          }
    }
```

```
   class BokaLyss implements ActionListener{
          public void actionPerformed(ActionEvent ave){
              String str = "Du har bokat följande platser: ";
              for(Plats p : bokadePlatser){
                    p.setText("Upptagen");
                    p.setEnabled(false);
                    str += p.getNr() + " ";
              }
              bokadePlatser.clear();
              JOptionPane.showMessageDialog(null, str);
          }
    }

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

Uppgift 2

a)

```
public class ISSN implements Comparable<ISSN>{
    private int part1, part2;
    private String checkDig;

    .....
    public boolean equals(Object other){
          if (other instanceof ISSN){
              ISSN o = (ISSN)other;
              return part1 == o.part1 && part2 == o.part2 &&
                    checkDig.equals(o.checkDig);
          }
          else
              return false;
    }

    public int hashCode(){
          return part1*1000000 + part2*100 + checkDig.hashCode();
    }

    public int compareTo(ISSN other){
          int cmp = part1 - other.part1;
          if (cmp != 0)
              return cmp;
          cmp = part2 - other.part2;
          if (cmp != 0)
              return cmp;
          return checkDig.compareTo(other.checkDig);
    }
}
```

b)

```
import java.util.*;

class TidningISSNCmp implements Comparator<Tidning>{
    public int compare(Tidning t1, Tidning t2){
          return t1.getISSN().compareTo(t2.getISSN());
    }
}

class TidningNamnCmp implements Comparator<Tidning>{
    public int compare(Tidning t1, Tidning t2){
          return t1.getNamn().compareTo(t2.getNamn());
    }
}

class TidningUpplagaCmp implements Comparator<Tidning>{
    public int compare(Tidning t1, Tidning t2){
          return t2.getUpplaga() - t1.getUpplaga();
    }
}

static void lista(Map<ISSN, Tidning> tidningar, int sortOrdning){
   List<Tidning> tLista = newArrayList<Tidning>(tidningar.values());
   Comparator<Tidning> cmp = null;
   switch(sortOrdning){
      case 0: cmp = new TidningISSNCmp(); break;
      case 1: cmp = new TidningNamnCmp(); break;
      case 2: cmp = new TidningUpplagaCmp(); break;
   }
   Collections.sort(tLista, cmp);
   for(Tidning t : tLista)
      System.out.println(t);
}
```

Uppgift 3

```
package bags;

public interface Bag<E>{
    void add(E elem);
    boolean contains(E elem);
    int count(E elem);
    void remove(E elem);
}
```

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

public class HashBag<E> implements Bag<E>{
    private HashMap<E, Integer> data = new HashMap<E, Integer>();

    public void add(E elem){
          Integer counter = data.get(elem);
          if (counter == null)
              counter = new Integer(1);
          else
              counter = new Integer(counter + 1);
          data.put(elem, counter);
    }
```

```
    public boolean contains(E elem){
          return data.containsKey(elem);
    }

    public int count(E elem){
          Integer counter = data.get(elem);
          return (counter == null)? 0 : counter;
    }

    public void remove(E elem){
          Integer counter = data.get(elem);
          if (counter == null)
              throw new NoSuchElementException();
          if (counter == 1)
              data.remove(elem);
          else
              data.put(elem, counter - 1);
    }
} // HashBag
```