CHALMERS TEKNISKA HÖGSKOLA
Datavetenskap TDA547

Tentamen i Objektorienterad programmering

Fredagen 13 januari 2012, 14.00 – 18.00.
Jourhavande lärare: Björn von Sydow, tel 0722/391401.

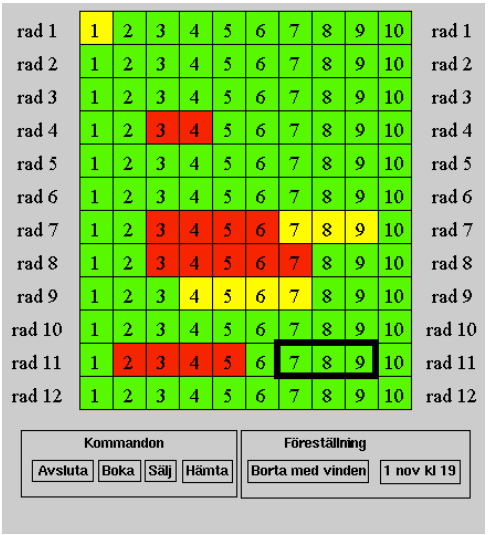
Inga hjälpmedel.

Lösningar till uppgifterna behöver ej kommenteras. Ej heller behöver nödvändiga importer anges. Triviala syntaxfel tolereras utan poängavdrag vid rättningen. Skriv läsligt!

Skrivningen kan ge maximalt 40 p; 20 p ger med säkerhet godkänt. 27 p ger betyget 4 och 33 p betyget 5.

Uppgifterna i denna tentamen har alla anknytning till ett tänkt platsbokningssystem av den typ som ofta används på biografer och teatrar. Observera dock att uppgifterna kan lösas oberoende av varandra.

Ett sådant program presenterar ett användargränssnitt med vidstående principiella utseende:



Huvuddelen av gränssnittet visar en biografsalong där de olika platserna kan ha tre färger: gröna platser är lediga, gula platser är bokade men ännu ej sålda och röda platser är sålda.

Dessutom kan en eller flera platser vara *markerade*, här genom en svart inramning. Markering görs av användaren genom att klicka med musen i en ledig plats samt dra musen med knappen nedtryckt längs en rad lediga platser och slutligen släppa musen. Endast intilliggande lediga platser på samma rad kan markeras.

Under detta finns till vänster fyra knappar:

- Avsluta-knappen, som avslutar programmet efter att dessförinnan ha sparat bokningsläget.
- Boka-knappen, som öppnar en dialogruta där användaren anger ett lösenord (normalt köparens telefonnummer); systemet lagrar lösenordet tillsammans med platsinformationen, noterar att dessa platser är bokade, ändrar färg på de markerade platserna till gul och tar bort markeringen.
- Sälj-knappen, som säljer de markerade platserna, dvs noterar att platserna är sålda, ändrar deras färg till rött samt tar bort markeringen. Dessutom skrivs biljetter ut.
- Hämta-knappen, som används då bokade biljetter ska avhämtas. En dialogruta öppnas och användaren ger ett lösenord, som fås från kunden. De bokade platser som lagrats tillsammans med lösenordet markeras (och kan sedan säljas med Sälj-knappen).

Till höger nedtill anges föreställning, dvs film och tidpunkt. Bägge dessa är valbara, dvs tryck med musen på en av dessa öppnar en meny med möjlighet att välja andra filmer resp andra tider.

Ovanstående beskrivning tjänar som bakgrund till de följande uppgifterna.

1. En plats i salongen representeras med ett objekt av klassen

```
public class Place {
  private int row, seat;

  public Place(int row, int seat) {
      ...
  }

  public int getRow() {
      ...
  }

  public int getSeat() {
      ...
  }
}
```

   (a) Komplettera klassdefinitionen genom att fylla i koden på de tre ställen som markerats med .... (3 p)

   (b) I en klass i programmet återfinns deklarationen

   ```
   Place p1, p2;
   ```

   Man vill i den kod som följer jämföra om `p1` och `p2` refererar till samma plats i salongen, dvs samma rad och säte. Förklara varför det inte är lämpligt att göra detta med kod av typen

   ```
   if (p1==p2) ...  ;
   ```

   (2 p)

   (c) Utvidga klassen i (a) genom att på lämpligt sätt definiera metoden `equals` så att man i stället kan testa på likhet med

   ```
   if (p1.equals(p2)) ...;
   ```

   (3 p)

2. En markering representeras av ett objekt av klassen

```
public class Selection {
  private Place firstPlace;
  private int nrPlaces;

  public Selection(Place firstPlace, int nrPlaces) {...}

  public Place getFirstPlace() {...}

  public int getNrPlaces() {...}
}
```

   Tillståndsvariabeln `firstPlace` är den vänstraste platsen i markeringen; `nrPlaces` är antalet markerade platser. Denna klass kompletteras på analogt sätt som i uppgift 1, men vi hoppar över detta här.

(a) Komplettera högerledet i satsen

```
Selection sel = ...;
```

så att `sel` kommer att referera till den markering som syns i bilden på första sidan. Inga objekt finns skapade sedan tidigare. (2 p)

(b) I programmet finns en klass `TicketPrinter`, som bland annat erbjuder metoderna

```
public static void printTicket(ShowData d, Place p);
public static void printTickets(ShowData d, Selection sel);
```

vilka används för att skriva ut biljetter på en speciell skrivare. Typen `ShowData` innehåller information om föreställningen, dvs film, datum och tid, som också ska finnas på biljetterna. Vi bryr oss inte om exakt hur den är definierad. Din uppgift är att definiera den andra av dessa metoder (dvs `printTickets`), som skriver ut flera biljetter, genom att använda dig av den första (dvs `printTicket`). Du ska *inte* definiera den första metoden eller klassen i övrigt. (4 p)

3. För att representera om en viss plats är ledig, bokad eller såld används en uppräkningstyp

```
public enum State {FREE, BOOKED, SOLD}
```

Bokningsläget för en viss föreställning representeras av ett objekt ur modellklassen

```
public class Bookings {

   private State[][] places;
   private ShowData data;

   public Bookings(ShowData data, int nrRows, int nrSeats) {...}

   public void book(Selection sel) {...}

   public void sell(Selection sel) {...}

   public int getNrRows() {...}

   public int getNrSeats() {...}

   public State getState(int row, int seat) {...}
}
```

De två sista parametrarna till konstrueraren anger antalet rader och antalet platser per rad.

(a) Ge den kod som behövs i konstrueraren så att bokningsläget initieras med att alla platser är lediga. (4 p)

(b) Ge också koden för de tre sista metoderna, med namn som börjar på `get`. (3 p)

(c) Ge koden för `sell`, som ska ändra tillståndet för de angivna platserna. Man behöver inte kontrollera att platserna är lediga innan anropet. (3 p)

4. Bokade platser lagras tillsammans med ett lösenord i en bokningslista, som är en instans av en klass som implementerar interfacet

```
public interface Reservations {
  public void reserve(String password, Selection res);
  public Selection getReservation(String password);
}
```

Man lagrar en bokning (en `Selection`) tillsammans med ett lösenord, t ex kundens telefonnummer med metoden `reserve`. Senare kan man söka upp bokningen med metoden `getReservation`; om ingen bokning finns med detta lösenord returneras `null`.

Definiera en klass som implementerar detta interface. Du får ignorera problemet att det redan kan finnas en bokning med det givna lösenordet när `reserve` anropas. Inga krav finns på att metoderna ska vara effektiva. (8 p)

5. Det grafiska uppritandet av bokningsläget sker med hjälp av klassen

```
public class BookingsView extends JPanel {
  private static final int SIZE = 20;
  private Bookings model;
  private Selection sel;

  public BookingsView(Bookings model) {
    this.model = model;
    sel = null;
    setPreferredSize(new Dimension(model.getSeats()*SIZE,
                                   model.getRows()*SIZE));
    addMouseListener(new MyListener());
  }


  private class MyListener extends MouseAdapter {
    public void mouseClicked(MouseEvent e) {...}
  }

  public void paintComponent(Graphics g) {
    // kod för att rita röda, gröna och gula platser samt markeringen.
  }

  public Selection getSelection() {return sel;}

  public void setSelection(Selection sel) {this.sel = sel;}
  ...
}
```

I en första version av programmet implementeras klassen så att en markering bara kan bestå av en enda plats; när man klickar med musen på en ledig plats sätts tillståndsvariabeln `sel`

till denna enda plats. Komplettera klassen ovan med den kod som behövs för att åstadkomma detta. Du ska bara åstadkomma detta; du behöver inte implementera uppritandet av markeringen i `paintComponents`. Däremot måste du kontrollera att platsen är ledig; annars sätts `sel` till `null`. (4 p)

6. Knapparna återfinns i en klass `GUI` med följande struktur

```
public class GUI extends JPanel {
  private Bookings model;
  private ShowData data;
  private BookingsView bv;

  private JButton sellButton;
  ...

  public class GUI(...) {
    ...
    sellButton = new JButton("Sälj");
    sellButton.addActionListener(new SellListener());
  }

  private class SellListener implements ActionListener {
     public void actionPerformed(ActionEvent e) {
       ...
     }
}
```

Implementera metoden `actionPerformed` i `SellListener`. (4 p)