Laboration 4: Huvudräkning

Syfte

Syftet med denna laboration är att utveckla ett något större program som är uppbyggt av ett flertal klasser och som använder grafik och händelsestyrd programmering.

Förberedelser

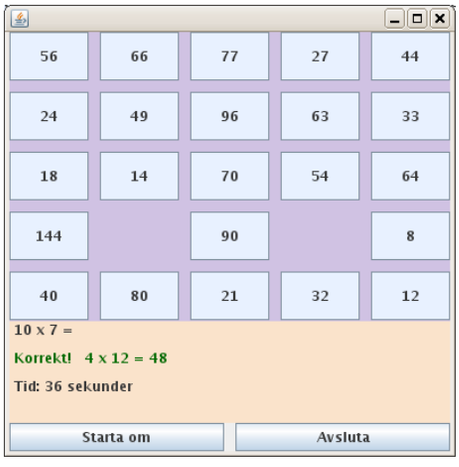
Läs noga igenom laborationstesen. Börja inte koda innan ni analyserat problemet. Följ de råd som ges angående programmets uppbyggnad.

Uppgift

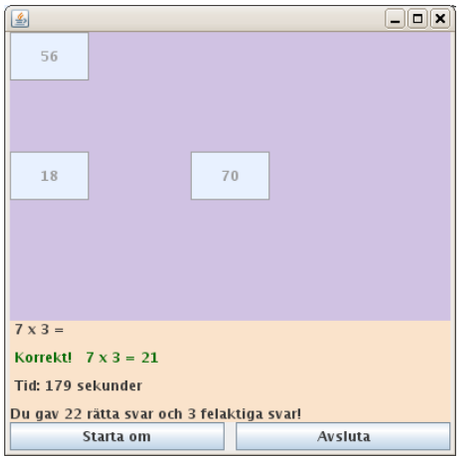
Uppgiften består i att göra ett grafiskt program för att träna huvudräkning. I exemplet nedan handlar det om multiplikation av två heltal mellan 2 och 12. I programmet du skriver kan du, om du så vill, välja andra räkneoperationer och/eller andra intervall av tal.

När programmet startas visas ett fönster enligt figuren nedan.

Fönstret består av tre delar. I den översta delen av fönstret finns ett antal knappar, i detta exempel 25 stycken, på vilka ett unikt tal är angivet. Programmet kommer att generera 25 slumpmässiga multiplikationer och det korrekta svaret på var och en av dessa multiplikationer finns på exakt en av dessa knappar. Vilken multiplikation som efterfrågas anges i den mittersta delen av fönstret. Uppgiften för användaren är att trycka på den knapp som anger det korrekta svaret på den efterfrågade multiplikation. Om ett korrekt svar ges försvinner knappen som användaren tryckte på. I den mittersta delen av fönstret anges också huruvida det senast avgivna svaret är korrekt eller inte, samt hur lång tid det passerat sedan programmet startades.



När programmet genererat och användaren besvarat samtliga multiplikationer skriver programmet ut hur många korrekta respektive felaktiga svar användaren avgivit. Vidare stoppas tidsräknaren, vilket innebär att användaren får reda på hur lång tid det tog för honom/henne att ange samtliga svar. Dessutom låses de knappar som återstår i den övre delen av fönstret, så att de inte längre går att trycka på. På de återstående knapparna återfinns de korrekta svaren på de multiplikationer för vilka användaren givit felaktigt svar.



I den nedersta delen av fönstret finns två knappar för att avsluta respektive återstarta programmet.

Testa hur programmet fungerar

För att kunna testa hur programmet fungerar finns en applet upplagd på kursens hemsida.

Programmets uppbyggnad

För att få en bra struktur i ett program eftersträvar man att dela in programmet i ett antal lämpliga klasser, vilka sinsemellan samarbetar för att lösa uppgiften. Trots att det program ni skall utveckla är relativt litet kan designen av programmet göras på många olika sätt, därför ges här ett förslag på hur ni bör bygga upp ert program för att få en överblickbar struktur och för att underlätta ert utvecklingsarbete.

Det slutliga programmet, dvs det grafiska fönstret i vilket man kan träna huvudräkning, implementeras i klassen MentalArithmetic. När vi betraktar fönstret ser vi att det är uppbyggt av tre huvudkomponenter:

- ytan på vilken vi har inmatningsknapparna med svarsalternativen,
- annonstavlan där bl.a. tiden och nästa uttryck vars svar efterfrågas anges,
- knappsatsen där användaren kan starta om eller avsluta programmet.

Implementera var och en av dessa komponenter i en egen klass. Kalla dessa klasser för InPutPanel, OutPutPanel respektive ButtonPanel.

För att få en bra uppbyggnad på programmet är det viktigt att användargränssnittet är åtskilt från programmets logik. I alla program kan man förändra användargränssnittet men aldrig programmets logik (då skulle vi ju få ett program som gör något annat). Förutom de grafiska komponenterna InPutPanel, OutPutPanel och ButtonPanel behövs således en klass som innehåller sådant som inte har med den grafiska presentationen att göra, t.ex. att generera de uttryck som användaren skall beräkna. Kalla denna klass för Model.

Klassen MentalArithmetic *har* alltså ett objekt av klassen InPutPanel, ett objekt av klassen OutPutPanel, ett objekt av klassen ButtonPanel och ett objekt av klassen Model.

De olika aktörerna i klassen MentalArithmetic - klasserna InPutPanel, OutPutPanel, ButtonPanel och Model - måste samverka med varandra vid lösandet av den givna uppgiften. Således måste aktörerna kopplas i hop med varandra, dvs vi måste bestämma vilka *relationer* som aktörerna har sinsemellan. Detta kan göras på flera olika sätt. För att få en bra och enkel struktur har vi i klassen MentalArithmetic specificerat följande relationer:

OutPutPanel: *känner till* Model.

InPutPanel: *känner till* Model och *känner till* OutPutPanel.

ButtonPanel: *känner till* Model, *känner till* InPutPanel och känner till OutPutPanel.

De olika klasserna

Klassen Model

Klassen Model håller reda på programmets logik och ansvarar för den bokföring som hör till detta. Det är alltså Model som bestämmer vilka uttryck som användaren skall besvara och i vilken ordning dessa skall presenteras för användaren. Model bokför också t.ex. hur många rätta respektive felaktiga svar användaren avgivit.

För att bestämma vilka uttryck som skall presenteras för användaren och i vilken ordning dessa skall presenteras har Model ett objekt av klassen Random.

Eftersom ett uttryck, t.ex. 12 * 9, består av flera komponenter är det lämpligt att skapa en särskild klass Expression för att avbilda uttryck, istället för att handa varje komponent i uttrycket i separata variabler.

Samspelet med InPutPanel

Klassen Model måste innehålla metoder så att klassen InPutPanel dels kan skriva ut svarsalternativen på på inmatningsknapparna, dels kan vidarebefordra de svar som användaren ger via inmatnings-knapparna.

Samspelet med OutPutPanel

Klassen Model måste tillhandahålla metoder så att OutPutPanel kan få tillgång till de uppgifter som skall skrivas ut, t.ex. uttrycket som skall besvaras, korrekt svar för uttrycket samt antal avgivna rätta respektive felaktiga svar.

Klassen InPutPanel

Klassen InPutPanel utgör det grafiska kommunikationsmedlet via vilket användaren anger sina svar på de efterfrågade uttrycken. Kommunikationsmedlet består av ett antal knapp-objekt som är organiserade i ett två-dimensionellt fält. Knapp-objekten är av klassen JButton. Klassen InPutPanel *har* alltså ett fält av JButton-objekt.

Uppgiften för klassen InPutPanel är dels att visa upp svarsalternativen för den uppsättning uttryck som klassen Model bestämmer att användaren skall besvara, dels att för varje uttryck som efterfrågas registrera svaret som användaren ger och rapportera detta svar vidare till klassen Model. Dessutom beordrar InPutPanel klassen OutPutPanel att skriva ut resultatet av användarens svar.

Klassen OutPutPanel

Klassen OutPutPanel är programmets annonstavla på vilken t.ex. anslårs de uttryck som användaren skall besvara samt huruvida användaren angav rätt svar eller ej. För att få reda på dessa uppgifter måste OutPutPanel kommunicera med klassen Model, eftersom det är Model som bokför alla dessa uppgifter.

På anslagstavlan anges också hur lång tid (i sekunder) det gått sedan svaret på det första uttrycket efterfrågades. Denna tidsuppräkning sköter klassen OutPutPanel om själv, genom att använda sig av ett objekt av klassen javax.swing.Timer.

Lämpligen implementeras OutPutPanel med hjälp av fyra JLabel- objekt.

Klassen ButtonPanel

Klassen ButtonPanel består av två knappar för att starta en ny spelomgång respektive avsluta spelet. När en ny spelomgång skall startas måste detta meddelas Model, InPutPanel och OutPutPanel så att dessa kan initiera ett nytt startläge. ButtonPanel måste således känna till samtliga dessa klasser.

Klassen MentalArithmetic

Det är i klassen MentalArithmetic som alla kopplingar mellan aktörerna skall göras. Således gäller att klassen MentalArithmetic *har* ett objekt av klassen InPutPanel, *har* ett objekt av klassen OutPutPanel, *har* ett objekt av klassen ButtonPanel och *har* ett objekt av klassen Model.

Klassen MentalArithmetic är redan implementerad. Koden visas nedan och kan kopieras från kursens hemsida.

```
import javax.swing.*;
import java.awt.*;
public class MentalArithmetic extends JFrame {
    public MentalArithmetic(int row, int col) {
        Model theModel = new Model(row, col);
        OutPutPanel outPanel = new OutPutPanel(theModel);
        InPutPanel inPanel = new  InPutPanel(theModel, outPanel, row, col);
        ButtonPanel buttPanel = new ButtonPanel(theModel, inPanel, outPanel);
        JPanel center = new JPanel();
        center.setLayout(new BorderLayout());
        center.add("Center", inPanel);
        center.add("South", outPanel);
        setLayout(new BorderLayout());
        add("Center", center);
        add("South", buttPanel);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400,400);
        setVisible(true);
    }//konstruktor
    public static void main(String[] args) {
        MentalArithmetic win = new MentalArithmetic(5, 5);
    }//main
}//MentalArithmetic
```

Studera koden och *förvissa dej om att du förstår hur de olika objekten kopplas samman*, då detta har påverkan på de klasser ni själva skall implementera.

Tips på ordningsföljd i implementationsarbetet

1. Utveckla de grafiska delarna i klassen InPutPanel.
2. Utveckla de grafiska delarna i klassen OutPutPanel.
3. Utveckla de grafiska delarna i klassen ButtonPanel.
4. Nu finns de grafiska delarna i alla klasser som klassen MentalArithmetic använder. Skriv ett skelett till klassen Model, så blir det möjligt att köra huvudprogrammet i klassen MentalArithmetic.
5. Utveckla och testa de delar av klassen Model som erfordras i samspelet med InPutPanel. Till en början är det lämpligt att t.ex. bortse från att uttrycken skall genereras på så sätt att alla svarsalternativ skall bli unika. Lägg till denna egenskap senare i utvecklingsarbetet.
6. Utveckla och testa de delar av klassen Model som erfordras i samspelet med OutPutPanel.
7. Utveckla klassen ButtonPanel och de delar av klassen Model som erfordras.

Redovisning

Laborationen skall dels demonstreras och godkännas av en handledare, dels redovisas skriftligt enligt de direktiv som finns på kursens hemsida.

Sista redovisningsdag och sista inlämningsdag för dokumentationen är fredag 2/3.