Hemtentamen TDDC77

December 19, 2013

- Utlämnas Torsdag: 2013-12-19 kl 8:00
- Inlämnas senast Fredag: 2013-12-20 kl 9:00.
- Jourhavande lärare: Ahmed Rezine `ahmed.rezine@liu.se`, tillgänglig via mail under kontorstid

- Tentamen innehåller 4 uppgifter. Totalt kan erhållas 30p.
- Ungefärliga betygsgränser är 3: 15p, 4: 20p, 5: 25p.
- Tentamen skall redovisas skriftlig i digitalt format (ordbehandlingsprogram eller vanlig textfil) med en tydlig rubrik för varje uppgift/delfråga. Ni ska lämna in exakt en dokument, inte mer, inte mindre.
- Inlämning sker genom att svaren mailas till `ahmed.rezine.liu@analys.urkund.se`. När inlämningen är korrekt genomförd så får du ett bekräftelsemail tillbaka.
- Kursen eller tentamen får inte diskuteras med någon annan under tentamenstiden.
- Det är tillåtet att hämta information från böcker/internet/andra källor, men du måste själv formulera dina svar. Man får alltså inte kopiera text.
- **Tentamen skall vid inlämningen vara försedd med tentandens namn och personnummer, både i filnamnet och i mailets ämnesrad**
- **Mailets ämnesrad skall även innehålla texten "[TDDC77]"**
- Sent inlämnade tentor blir automatiskt underkända

1 Teori HT1 (5p)

Förklara noggrant följande begrepp. Om du har gjort brevet-uppgiften så har du redan full pott på denna deluppgift, skriv bara "brevet" som svar på denna fråga för att indikera detta. (1p/begrepp)

- array
- global variabel
- do-while-slinga
- void
- två variabler är alias

2 Teori HT2 (5p)

Förklara noggrant följande begrepp. (1p/begrepp)

- överlagring
- privat variabel
- abstrakt klass
- instansiering (instantiation)
- gränssnitt

3 Design av klasshierarkier (10p)

Nedan följer ett antal faktapunkter, din uppgift är att skriva ett kodskelett som bygger motsvarande klasshierarki. Ange så pass mycket av alla klass, gränssnitt, variabel- och metoddeklarationer att allt specificerat i uppgiften implementeras. Lösningen behöver inte vara komplett, körbar kod. Ange alla antaganden du gör för eventuella problem/tvetydigheter.

- En resenär har ett namn
- En biljett har en resenär
- En biljett har ett flygnummer
- En ekonomiklass biljett är en biljett
- En affärsklass biljett är en biljett
- Ett kundkort har en resenär
- Ett kundkort är en lounge access granter
- En affärsklass biljett är en lounge access granter
- En lounge access granter har en metod för att öpnna loungens dörr
- En resenär kan köpa flera biljetter

4 Praktiskt kodskrivande (10p)

Vi ska implementera ett naivt program som generera ett färdig löst Sudoku puzzle. Komplettera den givna koden enligt anvisningarna i kommentarerna märkta "TODO: ..." (7p). Koden skall skrivas enligt gängse kodkonventioner (2p) och kommenteras (1p). Koden ska kunna köras. Lägg till variabler och "import ..." instruktioner om det behövs.

```
import java.util.Random;
import java.util.Set;
import java.util.HashSet;


public class Sudoku {

    //TODO: add variables if needed
    private int dim;
    private int squareDim;

    private int[][] board;


    public Sudoku(int _dim){
     /* TODO: Write a constructor to initialize dim with _dim,
      * squareDim with (dim x dim), and to instantiate board
      * with an array of arrays of integers of size squareDim x squareDim.
      * The constructor will initialize the new board using the
      * bruteForceRandomInitialization() defined below
      */
    }


    public void bruteForceRandomInitialization(){
     /* TODO: use the Random class and its
      * nextInt(int) method to fill the board with
      * squareDim x squareDim numbers between 0
      * (inclusive) and size (exclusive).
      */
    }


    public String toString(){
     String result = "";

     for(int i=0; i<squareDim; i++){
         for(int j=0; j<squareDim; j++){
          result += " " + board[i][j];
         }
         result += "\n";
     }

     return result;
    }


    private boolean validRow(int row){
     /* TODO: write a method to check whether all
      * numbers on the Row row in the board are pairwise different
      * (i.e. there are no two numbers on Row row that
      * are equal). Tips: you can use HashSet if you want.
      */
    }

    private boolean validColumn(int col){
     /* TODO: write a method to check whether all
      * numbers on the Column col in the board are pairwise different
```

```
      * (i.e. there are no two numbers on Column col that
      * are equal). Tips: you can use HashSet if you want.
      */
    }

    private boolean validSquare(int row, int col){
     /* TODO: write a method to check whether all
      * numbers in the board in the square between Rows row
      * and row+dim (both inclusive) and columns col and col+dim
      * (also both inclusive) are pairwise different.
      * Tips: you can use HashSet if you want.
      */
    }

    public boolean isValid(){
     /* TODO: write a method that returns true exactly when
      * the current assignment to board is a Sudoku solution.
      * An assignment is a Sudoku solution if there are no
      * two cells in the board that are equal and
      * that belong to some:  row, column,
      * or (dim x dim) square delimited by the rows
      * x and x+dim and by the columns y and y+dim with x equal to
      * 0, dim, 2xdim, ... and y equal to 0, dim, 2xdim, ...
      */
    }

    public static void main(String[] args){

     Sudoku puzzle = new Sudoku(2);
     while(!puzzle.isValid()){
         puzzle.bruteForceRandomFilling();
     }
     System.out.println(puzzle);
    }

}
```