



ASSIGNMENT 6 – SOLUTION

GENERAL INFORMATION

This assignment is made public on Friday, March 14, 2003. The assignment is due on

Friday, March 21, 1 PM.

Hand in your assignment to the teaching assistant running your lab session.

The first page of your (written) assignment has to contain at least the following information:

- the course name (Introduction to Programming - Concepts and Tools)
- your name and your student number
- name and student number of the fellow student if you submit in pairs
- assignment number

Please staple your assignment!

You will get back the graded assignment one week after submission deadline.

QUESTIONS

1. (*Calculator, Math-class.*)

SOLUTION

```
class Accumulator{

    private double acc;

    Accumulator(){acc = 0;}

    Accumulator(double a){acc = a;}

    public double getValue(){return acc;}

    public void reset(){acc = 0;}

    public void add(double x){acc+=x;}

    public void sub(double x){acc-=x;}

    public void mult(double x){acc*=x;}

    public boolean div(double x){
        if(x == 0)
            return false;
        else
        {
            acc/=x;
            return true;
        }
    }
}
```

```

/* Unary functions */
public boolean sqrt(){
    if(acc < 0)
        return false;
    else
    {
        acc = Math.sqrt(acc);
        return true;
    }
}

public boolean log(){
    if(acc <= 0)
        return false;
    else
    {
        acc = Math.log(acc);
        return true;
    }
}

public void sin(){acc=Math.sin(acc);}
public void cos(){acc=Math.cos(acc);}
public void exp(){acc=Math.exp(acc);}

public void random(){acc=Math.random();}

/* Their binary pendant */

public boolean sqrt(double x){
    if(x < 0)
        return false;
    else
    {
        acc = Math.sqrt(x);
        return true;
    }
}

public boolean log(double x){
    if(x <= 0)
        return false;
    else
    {
        acc = Math.log(x);
        return true;
    }
}

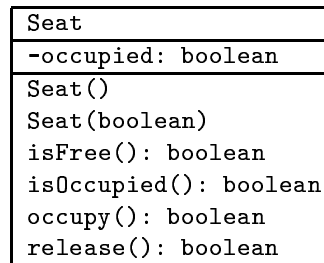
public void sin(double x){acc=Math.sin(x);}
public void cos(double x){acc=Math.cos(x);}
public void exp(double x){acc=Math.exp(x);}
public void random(double x){acc=Math.random()*x;}
}

```

2. (Theatre)

SOLUTION

(a) UML class diagram for class `Seat`:



- (b)
- `Seat()`
The constructor does nothing, though the boolean variable is automatically initialised to `false`.
 - `Seat(boolean)`
The constructor sets the private attribute to the value passed to the constructor.
 - `boolean isFree()`
The method returns `true` if the seat is un-occupied, otherwise it returns `false`.
 - `boolean isOccupied()`
This method does just the opposite of the one above, and returns `true` exactly if the seat is occupied.
 - `boolean occupy()`
The method occupies a seat if possible, i.e., sets the internal private attribute to `true`. If the seat was already occupied the method returns `false` to indicate that this seat cannot be occupied, otherwise the method returns `true` that the seat was successfully occupied.
 - `boolean release()`
The method returns `false` if the seat wasn't occupied. Otherwise the seat is released and the method returns `true` to indicate this.

(c)

```
/*
 * Class to hold an array of Seats to mimic a movie theatre.
 *
 * Carsten Butz, September 2002
 */

class Theatre{

    private Seat[] [] theatre;

    /* Constructors
    */
    Theatre(int numRows, int numSeats){
        /* Theatre constructor makes a new movie theatre with numRows rows and numSeats
        * seats (chairs) in each row. All seats are unreserved (unoccupied)
        * in the beginning.
        */
        /* Konstruktoren Theater laver en ny biografstal med numRows rækker og numSeats
        * stole i hver række. Alle sæder skal være ureserverede i begyndelsen.
        */
        theatre = new Seat[numRows][numSeats];
        for(int row = 0; row < theatre.length; row++)
            for(int col = 0; col < theatre[row].length; col++)
                theatre[row][col] = new Seat();
    }
}
```

```

/* Instance Methods
*/
boolean isReserved(int row, int seat){
    /* The method returns true if the seat at location (row, seat) is reserved.
    * The method returns false in all other cases.
    * Be careful that row numbers run from 1 to numRows, and seat numbers from
    * 1 to numSeats.
    */
    /* Returnerer true, hvis sædet (row, seat), dvs. sædet med rækkenummer row og
    * stolnummer seat, er reserveret. Returnerer false i alle andre tilfælde.
    * NB: Tilladelige
    * rækkenumre er 1..numRows, og tilladelige stolenumre er 1..numSeats.
    */
    return theatre[row-1][seat-1].isOccupied();
}

boolean reserve(int row, int seat){
    /* Books the seat at location (row, seat) and returns true if that
    * seat is available.
    * Returns false if that seat is already reserved.
    */
    /* Reserverer sæde (row, chair) og returnerer true, hvis det ikke
    * allerede er reserveret.
    * Returnerer false, hvis sædet allerede er reserveret.
    */
    return theatre[row-1][seat-1].occupy();
}

boolean cancel(int row, int seat){
    /* Cancels a seat reservation at location (row, seat) is that seat was booked,
    * and returns true in that case.
    * The method returns false if that seat had not been reserved.
    */
    /* Afbestiller sædereservering af (row, seat) hvis det er reserveret,
    * og returnerer true i dette tilfælde.
    * Returnerer false, hvis sædet ikke er reserveret i forvejen.
    */
    return theatre[row-1][seat-1].release();
}

int numReservations(){
    /* Returns the number of reserved seats.
    */
    /* Returnerer antallet af reserverede sæder.
    */
    int count = 0;
    for(int i = 0; i<theatre.length; i++)
        for(int j = 0; j<theatre[i].length; j++)
            if(theatre[i][j].isOccupied())
                count++;
    return count;
}

void findSeat(){
    /* Prints (with System.out.println) a sentence about the next available seat,
    * for example "Next free seat: Row 4, seat 8."
    */
    /* Printer (med System.out.println) en sætning om, hvor der findes et ledigt sæde;

```

```

    * f.eks. "Næste ledige plads: Række 4, stol 8."
    */
boolean found = false;
int i=0;
int j =0;

while(i<theatre.length && !found){
    j=0;
    while(j<theatre[i].length && !found){
        if(theatre[i][j].isFree())
            found = true;
        j++;
    }
    i++;
}
if(!found)
    System.out.println("Sorry, there are no more free seats available.");
else
    System.out.println("Next free seat: Row " + i + ", Seat " + j);

return;
}

void showReservations(){
    /* Prints an overview over all reservations. Reserved seats are shown as "*",
    * available seats as "-".
    * For each row the row number is shown, then a couple of blanks, and then the
    * reservations. An example is
    *
    * 8      -----
    * 7      ---**-----
    * 6      ----***-----
    * 5      -----
    * 4      -----
    * 3      -----***
    * 2      ---**-----
    * 1      -----
    */
    /* Printer en oversigt over alle reserveringer, hvor reserverede pladser vises
    * med "*" og ledige pladser med "-".
    * Hver række vises med rækkenummer efterfulgt af et antal "blanks" og så
    * rækkens reservationer. F.eks.
    *
    * 8      -----
    * 7      ---**-----
    * 6      ----***-----
    * 5      -----
    * 4      -----
    * 3      -----***
    * 2      ---**-----
    * 1      -----
    */

for(int row = theatre.length-1; row>=0; row--){
    {
        System.out.print((row+1) + "\t");

        for(int j = 0; j<theatre[row].length; j++)
            if(theatre[row][j].isFree())

```

```
        System.out.print("-");  
    else  
        System.out.print("*");  
  
    System.out.println();  
    }  
    return;  
    }  
}
```

3. (Different implementations of one class)

SOLUTION

private double first private double second
Point(double, double) double first() double second() double distanct() double translate(double, double) String toString()

```
class Rectangle{

    private Point ll;
    private Point ur;

    Rectangle(Point ll, Point ur){
        this.ll = ll;
        this.ur = ur;
    }

    Rectangle(Point ll, double width, double height){
        this.ll = ll;
        this.ur = new Point(ll.first()+width,ll.second()+height);
    }

    public double area(){
        double a;
        a = Math.pow(ll.first()-ur.first(),2);
        a+= Math.pow(ll.second()-ur.second(),2);
        return Math.sqrt(a);
    }

    public void translate(double deltaX, double deltaY){
        ll.translate(deltaX,deltaY);
        ur.translate(deltaX,deltaY);
    }

    public boolean contains(Point p){
        return ll.first()<=p.first() && p.first()<=ur.first()
            && ll.second()<=p.second() && p.second()<=ur.second();
    }

    public Point ll(){return ll;}

    public Point ur(){return ur;}

    public Point ul(){return new Point(ll.first(),ur.second());}

    public Point lr(){return new Point(ur.first(),ll.second());}

}
```

```

class Rectangle{
    /*
     * Point ll is the lower left corner of the rectangle.
     */

    private Point ll;
    private double width;
    private double height;

    Rectangle(Point one, Point two){
        this.one = one;
        this.width = two.second()-one.second();
        this.height = two.first()-one.first();
    }

    Rectangle(Point one, double width, double height){
        this.ll = one;
        this.width = width;
        this.height = height;
    }

    public double area(){
        return width*height;
    }

    public void translate(double deltaX, double deltaY){
        ll.translate(deltaX,deltaY);
    }

    public boolean contains(Point p){
        boolean firstCoord = ll.first()<=p.first() && p.first()<=ll.first()+width;
        boolean secondCoord = ll.second()<=p.second() && p.second()<=ll.second()+height;

        return firstCoord && secondCoord;
    }

    public Point ll(){return ll;}

    public Point ur(){return new Point(ll.first()+width,ll.second()+height);}

    public Point ul(){return new Point(ll.first(),ll.second()+height);}

    public Point lr(){return new Point(ll.first()+width,ll.second());}

}

```