



ASSIGNMENT 6

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.

WARM-UP AND SUGGESTIONS FOR FURTHER EXERCISES

As a warm-up I suggest the review questions from Chapter 6 on Objects: Data Abstraction (pp. 227/228 of my copy of the textbook).

If you want to do further programming exercises I suggest the following ones from the book (pp. 228–230):

- Exercises 4–6 modifying the `Counter` class used in class.
- Exercises 13 and 14 about implementing a class representing a clock.
- Exercise 16 which asks you to implement a data type to represent data about a person.
- Exercise 15 (mathematical) implementing a data type to represent the complex numbers.
- Exercises 21–23 (mathematical) that ask about representing polynomials and perform certain simple operations on them.

It is up to you whether or not you want to work on these exercises. Those exercises will not be marked.

QUESTIONS

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

LEARNING OBJECTIVES: Implementing a simple class; implementing a class which is used by others; sticking to the specification.

Despite its apparent length this question is easy and not as long as it may look.

You are member of a development team that wrote a small and primitive calculator. You can download the compiled files `Calculator.class` and `Accumulator.class` from the homepage and run the calculator using the command `java Calculator`. The calculator displays the current value of the accumulator (a `double`) value, and allows the user to input commands like the following:

- `sin` (enter return)
which will calculate the sine of the current value hold in the accumulator;
- `sin(4)` (enter return), or `sin 4` (enter return)
which will store the value `sin(4)` in the accumulator;
- `random` (enter return)
generating a random number between 0 and 1 and storing it in the accumulator;
- `log` (enter return)
which will calculate the natural logarithm of the current accumulator value and stores it in the accumulator; if the current accumulator is less or equal than 0 (and the logarithm is undefined) the program prints an error message and leaves the accumulator unchanged
- `add 4` (enter return)
adding 4 to the current value hold in the accumulator.

Currently, the following commands are supported with their obvious meaning:

```
reset          quit
add number    sub number  mult number  div number
sqrt          log          exp          random
sqrt number   log number   exp number   random number
sin           cos
sin number    cos number
```

A possible session could look as follows:

```
$ java Calculator
0.0
random
0.7855214951293579
add(4)
4.785521495129358
sqrt
2.1875834830079874
exp 1
2.7182818284590455
log
1.0
sin
0.8414709848078965
cos num
Error! You didn't specify a number as argument
0.8414709848078965
div 0
Error! Division by zero.
0.8414709848078965
div 2
0.42073549240394825
```

```
quit
0.42073549240394825
$
```

- (a) Download the files `Calculator.class` and `Accumulator.class` from the homepage and run the calculator using the command `java Calculator`. Play around with the calculator to get a feel for it.
- (b) You are supposed to re-implement the class `Accumulator.java`. The UML class diagram showing all constructors and methods is as follows:

Accumulator
-acc: double
Accumulator() Accumulator(double) getValue(): double reset(): void add(double): void sub(double): void mult(double): void div(double): boolean sqrt(): boolean sqrt(double): boolean log(): boolean log(double): boolean exp(): void exp(double): void sin(): void sin(double): void cos(): void cos(double): void random(): void random(double): void

Implement *at least* the two constructors, the methods `getValue`, `reset`, `add`, `sub`, `mult`, `div` and both `log` methods. You may want to implement more of the methods.

You may have to consult the description of the class `Math` in the online Java documentation (access through the course homepage) to find out how to calculate the natural logarithm, or the sine or cosine.

Below again the details what the functions are supposed to do.

- `Accumulator()`
Sets `acc` to 0.
- `Accumulator(double x)`
Sets `acc` to `x`.
- `double getValue()`
Returns the value of `acc`.
- `void reset()`
Sets the accumulator back to 0.
- `void add(double x)`
Adds the value `x` to the accumulator.
- `void sub(double x)`
Subtracts `x` from the accumulator.
- `void mult(double x)`
Multiplies the accumulator by `x`.
- `boolean div(double x)`
If `x` equals zero the function returns `false` and leaves the accumulator unchanged. Otherwise the function divides the accumulator by `x` and returns `true`.

- `boolean sqrt()`
Returns `false` if the accumulator is negative. Otherwise the function calculates the square root of the accumulator, stores the new value there, and returns `true`.
 - `boolean sqrt(double x)`
Returns `false` if the `x` is negative. Otherwise the function calculates the square root `x`, stores the value in the accumulator, and returns `true`.
 - `boolean log()`
Returns `false` if the accumulator is zero or negative. Otherwise the function calculates the natural logarithm of the accumulator, stores the value there, and returns `true`.
 - `boolean log(double x)`
Returns `false` if `x` is zero or negative. Otherwise the function calculates the natural logarithm of `x`, stores the value in the accumulator, and returns `true`.
 - `void exp()`
Takes the the mathematical constant e to the power of the accumulator and stores the value there.
 - `void exp(double x)`
Sets the accumulator to e^x .
 - `void sin()`
Calculates the sine of the accumulator, stores the value there.
 - `void sin(double x)`
Stores $\sin(x)$ in the accumulator.
 - `void cos()`
Calculates the cosine of the accumulator, stores the value there.
 - `void cos(double x)`
Stores $\cos(x)$ in the accumulator.
 - `void random()`
Stores a random number between 0 (including) and 1 (excluding) in the accumulator.
 - `void random(double x)`
Stores a random number between 0 (including) and x (excluding) in the accumulator.
- (c) Compile your file `Accumulator.java` to check for syntax errors, and then run the program `Calculator` which will now use *your* implementation of the accumulator. Of course, you can now use only those function that you implemented.

WHAT TO SUBMIT: Submit the code of your class `Accumulator.java`, and a sample printout showing the program `Calculator` running with your class.

2. (*Theatre*)

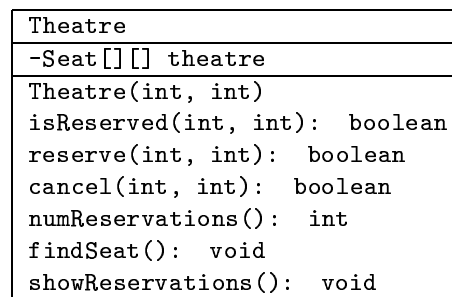
LEARNING OBJECTIVES: Implementing and testing a simple class; use of a given class; understanding the code of a given class.

In this question we will implement a class `Theatre` which will represent a movie theatre and which can be used in a booking system.

A theatre will consist of an array of seats. The functions that we want to use are checking whether a particular seat is booked or not, find the next empty seat, or book a seat. Also, we want to print a theatre, showing booked seats as * and free seats as -.

- Download the file `Seat.java` which implements a seat in the movie theatre. It contains one attribute, a private boolean variable which is true if the seat is occupied, and false otherwise. The class `Seat` comes with certain constructors and methods that operate on seats.
- Draw the UML class diagram of the class `Seat`.
- Describe in your own words what the meaning of the methods are, i.e., what those methods are supposed to do.

In the next couple of questions we will implement class `Theatre` with the following UML class diagram:



In the following you are supposed to implement that class.

- `private Seat[][] theatre`
The private attribute is a pointer pointing to a two dimensional array of `Seats`. To be precise, it is a pointer pointing to a two-dimensional array of *pointers* to `Seats`.
- `Theatre(int row, int seat)`
The constructor is to create the two dimensional array of (pointers of) `Seats`, with `row` many rows and `seat` many seats in each row.
Don't forget that since the two-dimensional array contains only pointers that *all* the `Seats` have to be created as well. So you need two nested loops to access all cells of the two-dimensional array, and create each seat using something like `theatre[i][j] = new Seat();`
- `boolean isReserved(int row, int seat)`
The function returns `true` if that seat is already reserved (or occupied), and `false` otherwise. Note that rows and seats are numbered from 1, but the array cells start at 0. Note also that for example `theatre[i][j]` is an object of class `Seat`, so you can apply methods of class `Seat` to this object.
- `boolean reserve(int row, int seat)`
The method returns `true` if the particular `seat` in row `row` can be booked, `false` otherwise. In case the seat can be booked it is booked (i.e., occupied).
- `boolean cancel(int row, int seat)`
The method returns `true` if the particular `seat` in row `row` was reserved (occupied), `false` if the seat was not reserved. In case the seat was reserved before it releases that seat.
- `int numReservations()`
The method returns the number of reserved seats in the theatre.
You simply have to run through all seats in the theatre using two nested loops and increment a counter each time you find a seat booked (occupied).

(j) `void findSeat()`

The method should print out a statement on the screen about the next empty seat. If there is no empty seat there should be a message saying so.

This method is a little tricky and you should think first about the algorithms solving the problem. As a hint, you should again search through all seats using nested loops. If you find an un-occupied seat you should abandon the loops and print out that seat's row and seat number. If you work through all seats and do not find an un-occupied seat then you should print that there are no more seats available.

(k) `void showReservations()`

The method prints the row numbers, some extra spaces, and then for each seat in the row either - or * depending on whether the seat is un-occupied or not. A possible print-out from this message could be the following:

```
8      -----***-----
7      ---**-----**---
6      -----***-----
5      -----
4      -----
3      -----***
2      --**-----
1      -----
```

Test your methods using the program `TestTheatre.java` which you can download from the homepage. In case you test your class before implementing all methods you have to comment out some of the commands.

WHAT TO SUBMIT: Submit the file containing your class `Theatre`, and the printout showing the running of `TestTheatre.java` using your class `Theatre`.

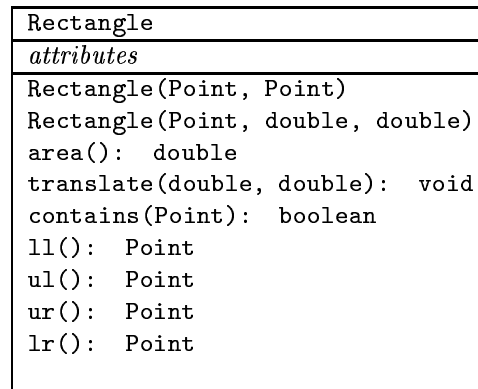
3. (Different implementations of one class)

This question is optional.

LEARNING OBJECTIVES: Implementing a simple class in two different ways; different implementations not visible to the user of the class.

In this exercise we will implement a class representing a rectangle in the plane in two different ways.

- (a) Download the file `Point.java` and study it. The class represents a point in the plane. Draw the UML class diagram of the class and submit it.
- (b) Download the compiled file `Rectangle.class` and the file `TestRectangle.java`. Compile `TestRectangle` and run it. It is a short program to test the implementation of a class `Rectangle` which has the following UML class diagram:



The meaning of those methods is as follows:

- `Rectangle(Point ll, Point ur)`
Creates a rectangle with `ll` as lower left corner and `ur` as upper right corner.
 - `Rectangle(Point ll, double width, double height)`
Creates a rectangle with `ll` as lower left corner and width `width` and height `height`.
 - `void double area()`
Returns the area of the rectangle.
 - `void translate(double deltaX, double deltaY)`
Translates (moves) the rectangle in the direction of the x-axis by `deltaX`, and in the direction of the y-axis by `deltaY`.
 - `boolean contains(Point p)`
Returns `true` if the point `p` is in the rectangle, `false` otherwise.
 - `Point ll()`
This and the remaining 3 methods return the point of the lower-left (upper-left, upper-right, lower-right respectively) corner.
- (c) Implement a class `Rectangle` which has the signature as described above, where the (internal) attributes are two points `ll` and `ur` representing the lower left and upper right corner of the rectangle. For the last four methods you may have to create a point within the method which you then can return.
To test your implementation use the program `TestRectangle.java`. Submit the code of the class.
 - (d) Rename the file that contains your implementation of the previous question (or move the file to a sub-folder).
Implement again the class `Rectangle` with the signature described above, where the (internal) attributes are this time a point `ll` representing the lower left corner of the rectangle, and two double values representing the width and height of the rectangle. To test your implementation use the program `TestRectangle.java`. Submit the code of the class.

WHAT TO SUBMIT: Submit the code of the two different implementations of the class `Rectangle`.