This week’s lecture dealt with the subject of introducing an additional class `Rectangle` to structure the data in the “mouse clicks in windows” problem (class as a data container). Next a non-default constructor was added, the data fields (data members) were made private, and a boolean function `containsPoint` was added to abstract from the details of the representation of the rectangle and from the computation involved (data abstraction and procedural abstraction). This resulted in the following two Java-files (available from the web page) that will be the starting point for the following exercises.

```
import java.util.*; // provides Scanner class description

// employs a class Rectangle with a non-default constructor
// and a boolean function containsPoint
public class PointsInRectangles {
    // pre: the input consists of a number k ( k >= 0 ) followed by
    // k times the x and y coordinates of the lowerleft and upperright
    // corners of a rectangle followed by
    // a number l ( l >= 0 ) followed by
    // l times the x and y coordinates of a point as follows:
    //
    // k
    // x1_0 y1_0 xh_1 yh_0
    // ...  
    // x1_(k-1) y1_(k-1) xh_(k-1) yh_(k-1)
    // l
    // xp_0 yp_0
    // ... 
    // xp_(l-1) yp_(l-1)
    //
    // coordinate system:
    //
    // y
    // |
    // |
    // +------------------>
    // x
    //
    // post: the output contains for each point in the input either the number
    // of the first rectangle the point is contained in or if the point is
    // not contained in any rectangle the text "not contained in any rectangle"

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Rectangle[] rect; // to store the rectangles
        int nr; // to store the number of rectangles
        int np; // to store the number of points
        // read number of rectangles and store in nr
        nr = sc.nextInt();
        assert nr >= 0 :
            "Number of rectangles should be nonnegative. (input was: \$nr + ", ");
        // create required storage space in array rect
        // (this does NOT create the Rectangle objects themselves,
        // only the storage space to hold references to Rectangle objects)
        rect = new Rectangle[nr];
        // read x and y coordinates of lowerleft and upperright corners
        // of the rectangles, create corresponding Rectangle object,
```
```java
// and store coordinates within the created object
for (int i = 0; i != nr; i++) {
    // create a Rectangle object representing the rectangle
    // corresponding to the next 4 coordinates in the input
    // and store a reference to it in rect[i]
    rect[i] = new Rectangle(sc.nextInt(), sc.nextInt(),
                            sc.nextInt(), sc.nextInt());
}

// read number of points and store in np
np = sc.nextInt();
assert np >= 0:
    "Number of points should be nonnegative. (input was: " + np + ")"

// process the points one by one and per point print either the number
// of the rectangle it is contained in or, if not contained in any rectangle,
// the text "not contained in any rectangle"
int xp, yp; // to store the x and y coordinate of a point
for (int j = 0; j != np; j++) {
    // read coordinates of the next point
    xp = sc.nextInt();
    yp = sc.nextInt();

    // determine the number of the first rectangle containing point (xp,yp). if any
    int ir = 0;
    // invariant (linear search):
    // 0 <= ir <= nr &&
    // point (xp,yp) is not contained in rectangles numbered 0 to ir−1
    while (ir != nr && !rect[ir].containsPoint(xp, yp)) {
        ir = ir + 1;
    }
    // 0 <= i <= nr
    // && point (xp,yp) is not contained in rectangles numbered 0 to ir−1
    // && (ir == nr || (xp,yp) is contained in rectangle numbered ir )
    if (ir == nr) { // (xp,yp) not contained in any rectangle
        System.out.println("not contained in any rectangle");
    } else {
        System.out.println(ir);
    }
}
} // end main

Rectangle.java

// class Rectangle having a non-default constructor
// and a boolean function containsPoint:
// the data fields representing the coordinates of the lowerleft
// and upperright corners are private so that, given the current class interface,
// they cannot be accessed nor changed by any other object (inaccessible and immutable)
public class Rectangle {

    // x and y coordinates of lowerleft and upperright corner of rectangle
    private int fXL, fYL, fXH, fYH;
    // pre: aXL <= aXH && aYL <= aYH
    // post: the created Rectangle object representing a rectangle with lowerleft
    // and upperright corners with coordinates given by aXL, aYL, aXH, and aYH
    public Rectangle(int aXL, int aYL, int aXH, int aYH) {
        assert aXL <= aXH && aYL <= aYH :
            "Precondition of constructor Rectangle(aXL,aYL,aXH,aYH) not satisfied."
        fXL = aXL;
        fYL = aYL;
        fXH = aXH;
        fYH = aYH;
```
\( fYH = aYH \);
}

// pre: -
// return: (the value of the predicate) point \((aX, aY)\) is contained within this rectangle
public boolean containsPoint(int aX, int aY) {
    return fXL <= aX && aX <= fXH && fYL <= aY && aY <= fYH;
}

1. **Identifying subtasks in PointsInRectangles**

   The computation in class PointsInRectangles is done completely in method main(). Identify subtasks in this computation (at least 2) and introduce an appropriate (static) method for each subtask. This might entail changing a number of local variables of main() into global variables (static data members) of class PointsInRectangles. Change the implementation of main() to use these methods.

2. **Adding methods to class Rectangle**

   (a) The data fields of class Rectangle are declared private so that, given the current class interface, they cannot be accessed nor changed by any other object. A standard way to make a data field accessible is to introduce a so-called getter function. For data field \( fXL \) this would be a function with header

   ```java
   // pre: ...
   // return: ...
   public int getXL() { ... }
   ```

   Similarly changing a data field is made possible by introducing a so-called setter method. For data field \( fXL \) this would be a method with header

   ```java
   // pre: ...
   // post: ...
   public void setXL(int aXL) { ... }
   ```

   Extend class Rectangle with getters and setters for all of its data fields.

   (b) Extend class Rectangle with an integer function that returns the circumference (omtrek) of the represented rectangle

   ```java
   // pre: ...
   // return: ...
   public int circumference() { ... }
   ```

   (c) Extend class Rectangle with an integer function that returns the area of the represented rectangle

   ```java
   // pre: ...
   // return: ...
   public int area() { ... }
   ```

   (d) Extend class Rectangle with a boolean function that has a rectangle parameter and returns whether the represented rectangle intersects with the rectangle represented by the parameter

   ```java
   // pre: ...
   // return: ...
   public boolean intersectsRectangle(Rectangle r) { ... }
   ```

   Likewise for containment

   ```java
   // pre: ...
   // return: ...
   public boolean containsRectangle(Rectangle r) { ... }
   ```
(e) Extend class Rectangle with a method that has a rectangle parameter and changes the represented rectangle into the intersection of the rectangle that was represented before the call and the rectangle represented by the parameter

```java
public void intersectWith(Rectangle r) {
    // pre: . . . (exclude calls leading to an empty intersection )
    // post: . . .
}
```

Also introduce a function that returns a new rectangle (object) that represents the intersection of the represented rectangle and the rectangle represented by the parameter

```java
public Rectangle intersection(Rectangle r) {
    // pre: . . .
    // return: . . .
}
```

(f) Extend class Rectangle with a method that has a rectangle parameter and changes the represented rectangle into the smallest rectangle that contains both the rectangle represented before the call and the rectangle represented by the parameter

```java
public void expand(Rectangle r) {
    // pre: . . .
    // post: . . .
}
```

(g) Other extensions of class Rectangle are for instance a method that translates the rectangle along a given direction, a method that scales the rectangle, a method that rotates the rectangle, a method that prints a representation of the rectangle, a method that reads the coordinates of a rectangle and changes the represented rectangle to that rectangle, etc..

3. **Alternative internal representation of class Rectangle**
   The internal representation of class Rectangle consists of the coordinates of the lower left and upper right corner points of the rectangle. An alternative internal representation is one that consists of the coordinates of the lower left corner point, the length of a side that is parallel to the x-axis, and the length of a side that is parallel to the y-axis. Reimplement class Rectangle and all of its methods using this alternative internal implementation.

4. **Using additional methods of class Rectangle**
   Use the class Rectangle with all extensions introduced in the previous exercise to solve the following problems (usually the input to the program consists of a number \( n \) \(( n > 0) \) followed by the x- and y-coordinates of the lowerleft and upperright corners of \( n \) rectangles)
   (a) Determine the sum of the areas of the rectangles.
   (b) Determine the number of a rectangle with the largest area.
   (c) What is the smallest rectangle that contains all rectangles? (This rectangle need not be any of the rectangles in the input sequence!)
   (d) Is there a rectangle in the input sequence that contains all other rectangles in the input sequence?
   (e) Are all rectangles in the input sequence disjoint?

5. **Alternative internal representation of class TimeOfDay**
   The design and implementation of class TimeOfDay (representing a moment in time on a day) has been discussed at the lecture. An internal representation consisting of one single private field containing the number of seconds elapsed since midnight was chosen for the implementation. In this exercise you are asked to implement the class using the alternative internal representation mentioned consisting of three private fields containing the hours, the minutes, and the seconds respectively. (The file TimeOfDay.java on the web page contains the implementation presented at the lecture.)