Problem 1: NRooks

NQueens is the problem of how to put N chess queens on an N board such that no queen can see the others according to standard chess rules. However, for simplicity lets assume that the board is filled with Rooks. A rook can see another rook if it shares the same x coordinate or y coordinate In this question we will represent a 2D board that can validate.

```
public static class NRooks {
    public static void main(String args[]) {
        boolean board[][] = createBoard(8);
        set(board, 3,3);

        boolean seesNothing = visibility(board, 3,3);
        System.out.println(isValid(board) && isComplete(board));
        board.set(0,1);
    }
```

(1) Implement a function that creates a square board filled with false to represent there are no rooks in any positions

(2) Implement a function that sets a single value in the current board to indicate there is a rook there called set.

(3) Implement a function that checks whether a rook can see another rook called visibility .

(4) Implement a function that checks whether the board has enough rooks.

(5) Implement a function that checks whether none of the rooks on the board can see one another

```
}
```

Problem 2: Bookstore

Implement a class that represents some basic details about a book in a bookstore: its title, author, price, and number of pages.

```
class Book {
```

(1) Create members to represent the aforementioned details

(2) Implement a constructor that lets the user pass in everything

(3) Implement a constructor that doesn't require passing in the price, but allows for a default price of 9.99.

(4) Implement getters or setters to indicate which properties can or cannot be changed (only the price can be changed).

}

The bookstore decides it also wants to be a library. Make a new class called BorrowableBook. A borrowable book always has a price of 0, and needs to store by some day and who borrowed it. Assume the day is just an integer (say, number of days since January 1 1970).

```
public class BorrowableBook
```

(5) Create members to represent the aforementioned details

(6) Implement a constructor that passes all of the forward data to the Book constructor.

(7) Implement getters and setters for the aforementioned data.

(8) Implement a function that, when passed in the current date, produces a brief email detailing how late the book is.

```
// System.out.println(borrowableBook.getLateEmail(currentDate));
// Says: "Hi BorrowerName, you borrowed ThisBookName, which was due OnThisDate.
//You are ThisLate." where BorrowerName/ThisBookName/OnThisDate/ThisLate
// are filled out properly
```

}

Now lets represent the entire inventory of books in a store

```
public class Inventory {
```

(9) Create members to represent the books in the store and the money in the cash register.

(10) Implement a method for adding a book to the inventory.

(11) Implement a method for buying a book to the inventory where the cost of the book is its price. To make the bookstore financially viable the book should cost 1 penny more when ti is sold.

(12) Implement a function for selling a book from its title. If the book is borrowable then it is merely borrowed, so this function needs to know the name of the person buying the book and the current date, which should both be passed as a parameter.

```
}
```

Problem 3: Spreadsheet

We saw in an assignment that we can store a spreadsheet as a String [][] , but accessing elements was rather painful. In this question we'll implement a spreadsheet that supports several types of data. The most basic type will be a Cell and we will represent several basic types like NumericCell, StringCell, and LogicalCell.

These derived types of cells will each store some data and pairs of Cells will be able to do some basic operations with one another:

- Add two cells together. Each type of cell can only be added with another cell of the same type.
- Generate a string representation that can be printed to a screen.
- An overload of Object.toString() for debug visualization.

(1) Implement the class Cell, which represents an empty cell. It should have a single function called add which takes another cell and returns a new Cell instance.

```
public class Cell {




}
```

(2) Implement the other types of cell and their data. Make sure that they know how to add with one another using overloading.:

(3) Implement a spreadsheet class that initializes a grid of Cell objects and has a function add that takes in three cell coordinates

//sheet.add(x0,y0,x1,y1,x2,y2) // -> adds cell at [x0,y0] with cell at [x1,y1] and writes the result to [x2,y2]

Problem 4: RectangleDrawing

In this problem we will let users draw rectangles on a screen by a dragging motion using Processing

We begin by defining a Rectangle class:

```
public class Polygon {
```

(1) Define the members to store the dimensions of the rectangle

(2) As rectangles will only be created by a dragging operation we can assume that they will always start with position but no width or height. Write a constructor that takes this into account.

(3) Once a rectangle is created the position is constant but its dimensions can change. Produce setters accordingly.

(4) Define a function that draws the rectangle by taking in a PApplet as a parameter.

Now lets define the Processing application

(1) Declare a class called App for the Processing application

```
public static void main(String[] args) { PApplet.main("App"); }
```

(2) Add data members to the application to represent the rectangle being created and a list of rectangles that have been completed.

(3) Implement a processing function to initialize the scene and create the first polygon being created.

(4) Implement the mouse interactions so every time the mouse is pressed a new rectangle starts being created, every time the mouse is dragged the rectangle's dimensions are changed appropriately, and when the mouse is released the rectangle stops being created. Be mindful that for the latter two operations the rectangle-in-progress might have been cancelled.

(5) Implement a keyboard interaction so when the user presses space (i.e. ' '), if there is a rectangle in progress, the rectangle ceases to be in progress.

(6) Implement a draw function that draws all of the finished rectangles with a black outline (i.e stroke, color 0,0,0) and blue interior (color 0,0,255), whereas the current polygon with a red outline (color 255,0,0).