Intro to CompSci (This was written in a *loose* tone to make it easier to read.)
By - TAs Moshe Rosensweig and Aaron Shakibpanah
How to write Objects - coding style

[] Introduction

The following is broken into 2 separate documents. The 1st is strictly about how to format a class. The second is about objects and important ideas related to objects. (The decision to split it into two was inspired by the acclaimed torah scholar the *Chafetz Chaim* who notes in his introduction that he also broke his work into 2 sections - not because the second section is less important, but because if it were put together, no one would read it.)

[] How is writing an Object different than what I've been doing until now?

Until now, you've just been writing all your code in one spot - in one method. That meant that it was difficult to really describe what your only method, your "main" method was doing. If I were to ask you what does this method do, and pointed to a method on some of your old hw's - you would spend 5 minutes telling me all the things it does. Organizing the class into methods, etc adds levels of clarity to your programming.

Objects are a structured way to write code. The point of this article though, is not to focus on why we like Objects, rather its about the coding style behind writing them.

There are 3 parts to objects: (1) fields (2) constructors and (3) methods. You need to choose how you will organize your objects and STICK to it. In other words, choose a convention. Why? Same reason as before, so your code will be easy to read. Much like an English essay has an introduction, supporting paragraphs, and a conclusion, the same is true for objects (the parallel is similar only in that they both have structure).

The following is the convention BlueJ taught - and it is a good convention to follow (I think Dr. Leff has a different style himself, but the way you do it is up to your personal preference). Put your fields on top, then your constructors, then your methods. Meaning the basic layout is:

```
import java.util.*;

public class Book{

     /*****************
      * Fields Go here
      *****************/

      /**********************
      * Constructors Go here
      ***********************/
```

```
        /*****************
        * Methods Go here
        ******************/
}
```

[1] Fields

Fields are the list of traits that define the class. There are 2 types of fields for a class, (1) Instance variables (2) Class variables (see other document for more). Put all your fields in one spot - in my convention it would be at the top.

[2] Constructors

The constructor is what gives birth to the object. You can have multiple constructors for a class (see other document for more). Put all your constructors in one spot - in my convention, they would go in the middle.

[3] Methods

You will have many methods. Group those all together - in my convention, they all go on the bottom.

When you put it all together it looks like - see bottom for an example.

[a] Method names

As you would expect, methods should have understandable names. Beyond that, there are general conventions about how to name certain types of methods. First, getters and setters. Most, if not all of your fields should be private (see other document). However, often it will be important to allow others to see the state of certain fields and even give them the ability to change the state. Therefore, you provide getters and setters. As you would expect, a getter, gets a value, and a setter sets a value. Getters start with the term “getX” and setters start with the term “setX”. See below for examples. (There are other conventions like this but I’m keeping things brief.)

[b] What goes inside a method? How long should a method be?

Good questions. The general answer is that every method should accomplish **ONE TASK**. This may not be obvious at first, but if you think about how to implement methods it might become clearer.

When you write a piece of code that has several steps to be done, you should write it with method names. For example, if I want to: (1) open a file and get all the relevant data inside the file (2) analyze the data (3) write a report (4) print out the data - I could just do it all in one method, or I could do this:

— code above omitted —

```
        /*
         * the 1st method below leads to a method called “openFileAndGetData” and returns an int array.
         * the 2nd method below leads to a method called “analyzeData” and returns an Analysis object.
         * the 3rd method below leads to a method called writeReport, which is a void method
         * the 4th method below leads to a method called printReport, which is a void method
        */
        private void  someMethod(){
                int[] releventData = openFileAndGetData(fileName);
                Analysis analysis = analyzeData(releventData);
                writeReport();
                printReport();
        }
— code below omitted —
```

This is very easy to read - because the names are very descriptive, and it’s so concise. When I run this code, the computer will look at the line that says “openFileAndGetData(fileName)” and jump to that method in your code and run that code. That method has a very clear and defined goal - in this case to open the file and get data from it. Each method now has an obvious task. Now when I ask you what the “openFileAndGetData” method does, you can sum it up for me in a sentence. That should be true for **all** of your methods.

(By the way, this concept is call **cohesion**. To have “high cohesion” means this method accomplishes one task, which is a good thing. Consequently, “low cohesion” is bad.)

This is helpful for another reason - reuse. We all love to recycle right? Well, if you have a method “addXAndY(int x, int y)” you can use that method many times in many different spots. This method accomplishes a goal that may be useful in other circumstances than the one you wrote it for. Since you wrote it in it’s own method, it can be reused. For example, you could write:

```
— code above omitted —
        int a = addXAndY(5, 4);
        int b = addXAndY(a, 7);
— code below omitted —
```

However, to ensure methods are as reusable as possible, as much as you can, try to make them as independent as possible. What we mean by that, is best explained by an example. The “addXAndY” is clearly independent - all it works with is the two ints you give it. Other methods might change the state of a field. For example, if I had a person object with a field called “weight”, and I made the following method:

```
        private void modifyWeight(int foodWeight){ weight = (weight + foodWeight); }
```

Even though this method also adds two numbers together, it changes the state of the “weight” variable. You don’t want to do that every time you add two numbers together. Instead, you could use the original method we made “addXAndY” as follows: weight = addXAndY(weight, foodWeight); This is a good re-usage of our method.

(This is the concept of **coupling**. Coupling means how much this method depends / is tied to other stuff - look up the term for a proper definition, but we have discussed the general idea above.)

Example:

```
import java.util.*;

public class Book{

     /*****************
      * Fields Go here
      *****************/

      private int numberOfPages;
      private String description = "No Desc was provided";
      private String title;
      private String author;

      /**********************
      * Constructors Go here
      ***********************/

      /**
       * Set all the fields with parameters
       */
      public Book(int numPages, String theTitle, String theAuthor, 
String desc){
           numberOfPages = numPages;
           title = theTitle;
           author = theAuthor;
           description = desc;
      }

      /**
       * Constructor without a description
       */
      public Book(int numPages, String theTitle, String theAuthor){
           numberOfPages = numPages;
           title = theTitle;
           author = theAuthor;
           description = "No Desc was provided";
      }
```

```
        /*****************
        * Methods Go here
        ******************/
        /* Getters */

        public int getNumberOfPages() {
              return numberOfPages;
        }

        public String getAuthor() {
              return author;
        }

        /* Setters */

        private void setNumberOfPages(int newNumOfPages) {
              numberOfPages = newNumOfPages;
        }

        private void setDescription(String desc) {
              description = desc;
        }

}
```