CSC 102 Lecture Notes Week 1
Introduction to the Course
Introduction to Java

I. **Relevant reading.**

A. Horstmann chapters 1 - 6

B. Writeups for Labs 1 and 2

C. Various cited material in writeups

II. **Go over the syllabus.**

III. **Go over Lab 1.**

IV. **Go over Lab 2.**

V. **Introducing Java to C programmers.**

A. The fundamental aspects of the two languages are very similar.

1. Syntactically, Java and C look very similar.
2. There is not a lot of initial "culture shock" in terms of the basic programming constructs you learned in CSC 101.
3. Stylistically, Java is generally more verbose than C.

B. Many of the basics in C and Java that are much the same.

1. Primitive data types of `int`, `double`, `char`.
2. The way primitive variables and parameters are declared, e.g.,

```
int i;
double x,y,z;
char c1, c2;
```

3. Most of conditional and loop statements -- `if`, `while`, `for`, `switch`.
   a. There are a couple extended forms of `for` loops, but everything that's in C is in Java.
   b. Java has the concept of *iterators*, about which we'll be learning a bit later in the quarter.
   c. For now, you can use what you know about conditionals and loops straight from C.

C. The core fundamentals of C functions and Java methods are the same.

1. As we'll see shortly, there are some important and significant differences between C and Java in the area of function/method definition and invocation.
2. For starters, what you know about C functions carries over into Java.

D. Both C and Java use a `main` function (method) as the starting point for a program.

VI. **Beyond the basics, Java has some fundamentally new features not found in C, most significantly:**

A. ***classes*** -- for organizing programs

B. ***access control*** -- to support data abstraction

C. ***exceptions*** -- for better error handling

D. ***inheritance*** -- to make things "object-oriented"

E. ***GUI support*** -- for better user interaction

VII. **Java methods compared to C functions.**

A. Methods and functions do fundamentally the same things:
   1. They're called to perform a computation.
   2. They can be sent parameters.
   3. They can return a value.
   4. They can use local variables in their computation.

B. The major difference between a method and a function is that the method *belongs to* the class in which it is defined.

C. This "belonging to" affects that way Java methods are invoked, how the methods access data, and how the methods return their results.

D. To illustrate these effects, consider the following C and Java implementations of the rectangle data structure discussed in chapter 2 of the book.

Rectangle.java:

```
/****
 *
 * A simple Java program that defines a rectangle data structure and two methods
 * that operate on rectangles.
 *
 */

public class Rectangle {
    int x;
    int y;
    int width;
    int height;

    Rectangle(int x, int y, int width, int height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }

    void move(int x_increment, int y_increment) {
        x = x + x_increment;
        y = y + y_increment;
    }

    boolean equals(Rectangle r) {
        return x == r.x &&
               y == r.y &&
               width == r.width &&
               height == r.height;
    }

    public static void main(String[] args) {
        Rectangle r1 = new Rectangle(10, 20, 100, 200);
        Rectangle r2 = new Rectangle(20, 30, 100, 200);
        boolean eq;

        eq = r1.equals(r2);
        if (eq == false) {
            System.out.println("r1 not = r2");
```

```
        }
        else {
            System.out.println("r1 = r2");
        }

        r1.move(10, 10);
        eq = r1.equals(r2);
        if (eq == false) {
            System.out.println("r1 not = r2");
        }
        else {
            System.out.println("r1 = r2");
        }
    }

}
```

rectangle.c:

```
/****
 *
 * A simple C program that defines a rectangle data structure and two functions
 * that operate on rectangles.
 *
 */

#include <stdio.h>

struct Rectangle {
    int x;
    int y;
    int width;
    int height;
};

    struct Rectangle move(struct Rectangle r, int x_increment, int y_increment) {
        r.x = r.x + x_increment;
        r.y = r.y + y_increment;
        return r;
    }

    unsigned char equals(struct Rectangle r1, struct Rectangle r2) {
        return r1.x == r2.x &&
               r1.y == r2.y &&
               r1.width == r2.width &&
               r1.height == r2.height;
    }

    int main() {
        struct Rectangle r1 = {10, 20, 100, 200};
        struct Rectangle r2 = {20, 30, 100, 200};
        unsigned char eq;

        eq = equals(r1, r2);
        if (eq == 0) {
            printf("r1 not = r20);
        }
        else {
            printf("r1 = r20);
```

```
    }

    r1 = move(r1, 10, 10);
    eq = equals(r1, r2);
    if (eq == 0) {
        printf("r1 not = r20);
    }
    else {
        printf("r1 = r20);
    }

}
```

E. **The following are particularly noteworthy differences between the two programs:**

1. The root name of the Java file and the class it contains must be spelled exactly the same; no similar restriction exists for C programs.
2. In terms of data structuring, the instance variables (aka, data members) of a Java class are the same as the data fields of a C struct.
3. Java class values are created and initialized with *constructor* methods; e.g., compare

```
Rectangle r1 = new Rectangle(10, 20, 100, 200);
```

with

```
struct Rectangle r1 = {10, 20, 100, 200};
```

4. Since Java methods *belong* to their classes, the class itself is available as an implicit input and output to every method; e.g., compare

```
void move(int x_increment, int y_increment) {
    x = x + x_increment;
    y = y + y_increment;
}
```

with

```
struct Rectangle move(struct Rectangle r, int x_increment, int y_increment) {
    r.x = r.x + x_increment;
    r.y = r.y + y_increment;
    return r;
}
```

and compare

```
boolean equals(Rectangle r) {
    return x == r.x &&
           y == r.y &&
           width == r.width &&
           height == r.height;
}
```

with

```
unsigned char equals(struct Rectangle r1, struct Rectangle r2) {
    return r1.x == r2.x &&
           r1.y == r2.y &&
           r1.width == r2.width &&
           r1.height == r2.height;
}
```

F. In general, a method *m* from a class *c* in Java is invoked like this

`c.m(...)`

where the comparable invocation in C is a function *f* called with a struct parameter *s*, like this

`f(s, ...)`

G. A Java method will frequently compute its result by modifying class instance variables, where in C the computed value needs to be returned from the function, or the function needs to modify global variables.

H. When necessary, the Java keyword `this` is used to refer explicitly to the class object.

1. Unless there is a naming conflict, `this` is an *implicit* parameter to all methods in a class, and there is no need to mention it explicitly.
2. For example, following definition of the `Rectangle.move` method is equivalent to one above

```
void move(int x_increment, int y_increment) {
    this.x = this.x + x_increment;
    this.y = this.y + y_increment;
}
```

3. The explicit use of `this` is only necessary when the name a class variable is the same as a method parameter or local variable, as in the definition of the `Rectangle` initializing constructor:

```
Rectangle(int x, int y, int width, int height) {
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
}
```

where `this` is necessary because the names of the constructor parameters are the same as the names of the class data fields.

4. An alternate way to deal with this naming issue would be to give the constructor parameters different names, as in

```
Rectangle(int x_val, int y_val, int width_val, int height_val) {
    x = x_val;
    y = y_val;
    width = width_val;
    height = height_val;
}
```

VIII. **Summary of chapters 1-6 of the text, including key topics for 102, andnon-topics for 102.**

A. Chapter 1 is a review of material from CSC 101, plus a very high-level introduction to Java.

1. Sections 1.1 through 1.3 are the 101 review material; you can skim through these.
2. Sections 1.4 through 1.6 are the high-level introduction to Java; the work you did in Lab 1 covered this material; the book's coverage is worth a read.
3. Sections 1.7 and 1.8 are more review of 101 material; they are a pretty lucid discussion of program errors and algorithms that are worth a quick read.

B. Chapter 2 is a good introductory treatment of Java classes and objects.

1. Sections 2.1 through 2.10 are directly relevant to Lab 2 and Program 1.
2. Sections 2.11 through 2.13 cover graphics concepts that are relevant to Program 2; you can skip them for now, and we'll come back to them in a week or so.

C. Chapter 3 is also a good introduction to class implementation and testing.

1. Sections 3.1 through 3.8 are directly relevant to Labs 2 and, as well as Program 1.

2. Section 3.9 covers graphics concepts that are not immediately relevant to 102; we will not be doing graphical user interfaces (GUIs) until program 4, so you can skip for now this and other sections of the book that discuss GUI concepts.

D. Chapter 4 covers fundamental data types in Java.
   1. These include numbers, constants, and strings.
   2. It also has an introduction to the `Scanner` class, which is relevant to lab 2.

E. Chapters 5 and 6 cover Java conditionals (`if`s) and loops (`while`s and `for`s).
   1. These features are quite similar to what's available in C, with some fancy looping bits that we'll cover later in the quarter.
   2. Conditionals and loops are basic to just about any program, so this material is relevant all just about all of the labs and programs, starting with Lab 2 and Program 1.

IX. Some initial discussion of programming assignment 1.

A. To aid in your C-to-Java transition, there is C version of the `Fraction.java` class declaration in the 102 class directory `programs/1`, in a file named `fraction.h`.

B. This example provides a comparative C definition of `Fraction.java` that you may find helpful.

C. The work you will do in Lab 2 is also designed to help with understanding th structure of Java programs like the one you're implementing in programming assignment 1.