Math 121: Introduction to Computing Handout #8

# Control Structures

This handout offers some notes about Java's control structures, emphasizing the important concepts, and then describes a problem whose solution is available in the folder for programming assignment #2.

To write programs, you need to understand control structures from two perspectives: you must have a holistic sense of when to use them and why, but you must also learn to understand the reductionistic details. For this big-picture perspective, you can rely to a large extent on your experience from Karel:

- If you want to test a condition that requires an **`if`** statement in Karel, you need the **`if`** statement in Java.
- If you would use the **`while`** or **`for`** statement in Karel, you will presumably use the same statement form in Java.

The other holistic point that is essential about control statements is that the control line is conceptually independent from the body. Thus, if you see a construct like

```
for (int i = 1; i <= 10; i++) {          ———— Control line
    statements                           ———— Body
}
```

the statements in the body will be repeated for each of the values of **`i`** from 1 to 10. It doesn't matter at all what those statements are.

## Boolean data

The data type **`boolean`** is the means by which Java programs ask questions. In Karel, the counterparts to **`boolean`** are the conditions such as **`frontIsClear()`** or **`beepersPresent()`**. In Java, the range of available conditions is much richer and involves the relational operators and the logical operators. The most important lessons to take from these sections are:

- Watch out for confusing **`=`** (assignment) with **`==`** (equality). This feature of the C-class of languages (C, C++, Java, C#) has probably caused more bugs than any other.
- Be careful to understand both the interpretation and the evaluation order of the logical operators **`&&`** (and), **`||`** (or), and **`!`** (not).

The time you put into making sure you understand **`boolean`** data now will pay for itself many times over when the programs get more complicated later in the term.

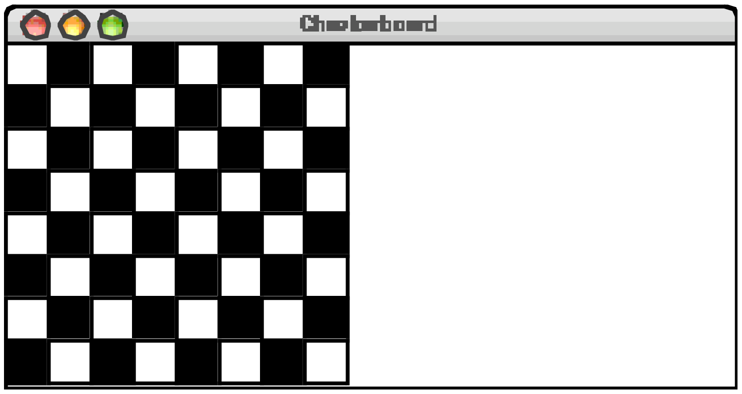
## Control statement paradigms

Make sure that you understand the basic structure of the four main control statements in Java: **`if`**, **`switch`**, **`while`**, and **`for`**. Note that the **`break`** statement occurs in two different contexts: as the last line in **`case`** clauses within a **`switch`** statement and to solve the loop-

and-a-half problem in **`while`** loops. The section on the loop-and-a-half problem is also worth special attention.

## Problem

Create a **`GraphicsProgram`** subclass that draws a checkerboard in the graphics window. The number of rows and columns are given by the named constants **`NROWS`** and **`NCOLUMNS`**, and the squares should be sized so that they fill the vertical space. For example, if **`NROWS`** and **`NCOLUMNS`** are both 8, running this program should produce the following output:



## Graphics library documentation

The **`javadoc`** documentation for the ACM libraries is available under the "Links" page for this course, on my home page. Also, the methods in Figure 1 will help with the assignment.

**Figure 1. Some useful methods in acm.graphics**

**Constructors**

| |
|---|
| **`new GLabel(String text)`** *or* **`new GLabel(String text, double x, double y)`**<br>Creates a new **`GLabel`** object; the second form sets its location as well. |
| **`new GRect(double x, double y, double width, double height)`**<br>Creates a new **`GRect`** object; the **`x`** and **`y`** parameters can be omitted and default to 0. |
| **`new GOval(double x, double y, double width, double height)`**<br>Creates a new **`GOval`** object; the **`x`** and **`y`** parameters can be omitted and default to 0. |
| **`new GLine(double x1, double y1, double x2, double y2)`**<br>Creates a new **`GLine`** object connecting (**`x1`**, **`y1`**) and (**`x2`**, **`y2`**). |

**Methods common to all graphical object**

| |
|---|
| **`void setLocation(double x, double y)`**<br>Sets the location of this object to the specified coordinates. |
| **`void move(double dx, double dy)`**<br>Moves the object using the displacements **`dx`** and **`dy`**. |
| **`double getWidth()`**<br>Returns the width of the object. |
| **`double getHeight()`**<br>Returns the height of the object. |
| **`void setColor(Color c)`**<br>Sets the color of the object. |

**Methods available for GRect and GOval only**

| |
|---|
| **`void setFilled(boolean fill)`**<br>Sets whether this object is filled (**`true`** means filled, **`false`** means outlined). |
| **`boolean isFilled()`**<br>Returns **`true`** if the object is filled. |
| **`void setFillColor(Color c)`**<br>Sets the color used to fill this object. If the color is **`null`**, filling uses the color of the object. |

**Methods available for GLabel only**

| |
|---|
| **`void setFont(String fontName)`**<br>Sets the font, as described in Chapter 5. |
| **`double getAscent()`**<br>Returns the height above the baseline. |