Control Statements

This handout offers some additional notes on Java’s control statements (described more fully in Chapter 4 of the textbook) that emphasize the important concepts. It also describes a programming problem making use of various control structures.

To write programs, you need to understand control statements 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

```java
for (int i = 0; i < 10; i++) {
    statements
}
```

the statements in the body will be repeated for each of the values of i from 0 to 9. It doesn’t matter at all what those statements are.

**Boolean data**

Another important topic is that of the data type boolean, which 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 (both covered on page 78 of textbook). The most important lessons to take from these sections are:

• Watch out for confusing = (assignment) with == (equality). This feature of several programming languages (including C, C++, and Java) 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 quarter.
Checkerboard 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” section of the CS 106A home page. Also, the methods in Figure 1 will help with the assignment.

Figure 1. Some useful methods in acm.graphics

<table>
<thead>
<tr>
<th>Constructors</th>
<th>Methods common to all graphical object</th>
<th>Methods available for GRect and GOval only</th>
<th>Methods available for GLabel only</th>
</tr>
</thead>
<tbody>
<tr>
<td>new GLabel(String text) or new GLabel(String text, double x, double y)</td>
<td>void setLocation(double x, double y)</td>
<td>void setFilled(boolean fill)</td>
<td>void setFont(String fontName)</td>
</tr>
<tr>
<td>Creates a new GLabel object; the second form sets its location as well.</td>
<td>Sets the location of this object to the specified coordinates.</td>
<td>Sets whether this object is filled (true means filled, false means outlined).</td>
<td>Sets the font, as described in Chapter 5.</td>
</tr>
<tr>
<td>new GRect(double x, double y, double width, double height)</td>
<td>void move(double dx, double dy)</td>
<td>boolean isFilled()</td>
<td>double getAscent()</td>
</tr>
<tr>
<td>Creates a new GRect object; the x and y parameters can be omitted and default to 0.</td>
<td>Moves the object using the displacements dx and dy.</td>
<td>Returns true if the object is filled.</td>
<td>Returns the height above the baseline.</td>
</tr>
<tr>
<td>new GOval(double x, double y, double width, double height)</td>
<td>double getWidth()</td>
<td>void setFillColor(Color c)</td>
<td></td>
</tr>
<tr>
<td>Creates a new GOval object; the x and y parameters can be omitted and default to 0.</td>
<td>Returns the width of the object.</td>
<td>Sets the color used to fill this object. If the color is null, filling uses the color of the object.</td>
<td></td>
</tr>
<tr>
<td>new GLine(double x1, double y1, double x2, double y2)</td>
<td>double getHeight()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Creates a new GLine object connecting (x1,y1) and (x2,y2).</td>
<td>Returns the height of the object.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Solution to the Checkerboard problem

```java
/*
 * File: Checkerboard.java
 * -----------------------
 * This program draws a checkerboard.
 */

import acm.graphics.*;
import acm.program.*;

/*
 * This class draws a checkerboard on the graphics window.
 * The size of the checkerboard is specified by the
 * constants NROWS and NCOLUMNS, and the checkerboard fills
 * the vertical space available.
 */

public class Checkerboard extends GraphicsProgram {

    /* Number of rows */
    private static final int NROWS = 8;

    /* Number of columns */
    private static final int NCOLUMNS = 8;

    /* Runs the program */
    public void run() {
        int sqSize = getHeight() / NROWS;
        for (int i = 0; i < NROWS; i++) {
            for (int j = 0; j < NCOLUMNS; j++) {
                int x = j * sqSize;
                int y = i * sqSize;
                GRect sq = new GRect(x, y, sqSize, sqSize);
                sq.setFilled(((i + j) % 2) != 0);
                add(sq);
            }
        }
    }
}
```