CS 106A: Programming Methodology
Java Programming Patterns and Control Statements

based on a similar handout written by Eric Roberts, Mehran Sahami, and Keith Schwarz

This handout summarizes some of the most common programming patterns that you will encounter in Java. Each of these patterns is covered in more detail in Chapters 2, 3, or 4 of the textbook, but it's good to have them in a reference as well.

The first set of patterns involves getting data in and out of the computer, which provide the necessary support for the input and output of a typical programming task. The patterns you use depend on the type of value, as shown in the following table:

Type	Declaration	Input pattern
Integer	`int var = value;`	`var = readInt("prompt");`
Real Number	`double var = value;`	`var = readDouble("prompt");`
String	`String var = value;`	`var = readLine("prompt");`

The following patterns are useful in calculations:

English	Java (long form)	Java (shorthand form)
Add *y* to *x*.	`x = x + y;`	`x += y;`
Subtract *y* from *x*.	`x = x - y;`	`x -= y;`
Add 1 to *x* (increment *x*).	`x = x + 1;`	`x++;`
Subtract 1 from *x* (decrement *x*).	`x = x - 1;`	`x--;`

The most helpful patterns, however, encompass programming operations on a larger scale and help you establish the overall strategy of a program. The most important ones are described in the next few sections.

The repeat-N-times pattern: (page 101)

This pattern is the same as it was in Karel and is used for the same purpose.

```
for (int i = 0; i < N; i++) {
    statements to be repeated
}
```

In Java, however, you are allowed to use the value of the index variable **`i`** in the body of the loop.

The read-until-sentinel pattern: (page 102)

This pattern is useful whenever you need to read in values until the user enters a particular value to signal the end of input. Such values are called **sentinels**.

```
while (true) {
    prompt user and read in a value
    if (value == sentinel) break;
    rest of body
}
```

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 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 = 0; i < 10; i++) {          ———— Control line
    statements                          ———— Body
}
```

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.