Partner Work 2 Reference Sheet: Variables, Math, Methods, Objects

By Duke Hutchings for CSC 130

Variables

In pretty much all modern programming languages, variables are used to retain information for computation. In Java, every variable has a data type (like int or double), a name, and a value. For example, in the following code, we create an integer called `x` with an initial value of 5.

```
int x = 5;
```

If we later want to change the value of `x`, then we use an assignment statement (seen above with use of the equal sign), but we no longer use the data type. Once defined, a variable always has the same data type, even though its value may change.

```
x = 10;
```

Sometimes we will want to adjust the value of a variable based on its current value. We do so by referring to the variable in the assignment statement. For example, if we want to increase `x` by 1:

```
x = x + 1;
```

We can refer to as many variables as we would like on the right hand side of the assignment (the equal sign). However, only one variable may appear on the left-hand side of the assignment (we can change only one variable at a time).

For more on variables, see Chapter 2, pp. 36-43.

For more on data types, see Chapter 4, pp. 132-139.

Math

So far we have learned five main math operators in Java.

```
int x = 5;
int y = 7;
int z = 0;

z = x - y; // subtract: z gets a value of -2
z = x + y; // add: z gets a value of 12
z = x * y; // multiply: z gets a value of 35
z = x / y; // integer division: z gets a value of 0
z = x % y; // mod (remainder after integer division): z gets a value of 5
z = y / x; // integer division: z gets a value of 1
z = y % x; // mod: z gets a value of 2
```

Division is the trickiest because of data types. Remember, once defined, a variable always holds data of its defined type. Even though 7 divided by 5 is usually expressed as 1.4 on a calculator, the integer result is 1 and the remainder is 2. We need to use both typecasting and the double data type to get 1.4 as a result stored in a variable:

```
double d = 0;
d = x / (double)y;

// above, cast one of the integers as a double
// an integer divided by a double results in a double
// store the result in a variable with type double
```

Try to avoid this common error:

```
double d = 0;
d = (double)(x / y);

// above, since x and y are integers, the result is an integer
// the casting to double happens too late; we already lost the precise value
```

But this is OK:

```
int i = (int)(7.4 / 2.2);

// above, since 7.4 and 2.2 are doubles, the result is a double
// the casting to int happens after the precise division takes place
```

For more on math and casting, see Chapter 4, pp. 139-146.

Methods

In Java, a method is a sequence of code that accomplishes a defined task. In Processing for example, the two main methods are `setup()` and `draw()`, which by now should be familiar to you. We can add our own methods to any Java program.

Many methods use *parameters* to allow input data to affect what the method does. For example, if we were drawing a face based on a common point, we might have a method like this:

```
public void drawFace(int x, int y, int size) {
  ellipseMode(CENTER);
  ellipse(x, y, size, size);
  int q = (size / 4) - 1;
  ellipse(x - q, y - q, q, q); // left eye
  ellipse(x + q, y - q, q, q); // right eye
}
```

This method defines three parameters to allow drawing to take place near location (`x`, `y`) and to draw faces of various sizes defined by the `size` parameter. The parameters are special types of variables since they are defined in the parentheses of the method definition, not inside the code for the method itself. Otherwise, they are treated like variables. Inside of the Processing `draw()` method, we might see code like this:

```
public void draw() {
  drawFace(width/2, height/2, 100); // draw a big face in middle of screen
  drawFace(20, 20, 15); // draw a small face near the upper-left corner
}
```

The first example is the **method <u>definition</u>** and the second example **<u>calls</u>** the `drawFace()` method.

Methods – Page 2

The methods on the previous page are **void methods** because they do something but don't provide any results. Other methods are more purely computation in manner: they compute some result based on inputs. For example, if we wanted to calculate the average of three numbers:

```
public double avgThree(double d1, double d2, double d3) {
  double a = 0;
  a = (d1 + d2 + d3) / 3;
  return a;
}
```

The call to this method would look like this:

```
public void draw() {
  double avg = 0;
  avg = avgThree(255, 223, 223); // get the average value of a pink color
  fill(avg, avg, avg); // change the shape fill to a gray version of pink
  // more drawing code would go here
}
```

In the textbook, guidance on methods assumes you know more about objects than you currently know. You can try pp. 43-48 in Chapter 2 or pp. 95-101 in Chapter 3 but you might not find those to be very helpful.

Consider referring back to class notes posted on the Class Web site or your own notes.

Objects

In a sense, objects tie together variables and methods:

- objects typically model and store many variables
- objects use methods to manage those variables

Take for example a *Color* object in Java. It stores three integer variables (one value for each of red, green, and blue) and it allows a programmer to get a copy of the red, green, or blue values individually through its methods. It also uses methods to ensure that its values are legal (remember that red, green, and blue values must be between 0 and 255).

To create an object, the code looks very similar to creating a variable.

```
Color c = new Color();
```

We can also create a new color object and at the same time set its initial values;

```
Color almostBlack = new Color(9, 9, 9); // very, very dark gray
```

If we need to determine one of the values of a Color parameter...

```
public Color computeColor(Color startColor) {
  int r = startColor.getRed();
  int g = startColor.getGreen();
  int b = startColor.getBlue();
  // do some computation on r, g, b
  Color endColor = new Color(r, g, b);
  return endColor;
}
```

Help in the textbook on objects:

- **(Color specifically) pp. 68-70 in Chapter 2**
- **pp. 43-51 in Chapter 2**
- **all of Chapter 3**