Mr G's Java Jive

#2: Yo! Our First Program

With this handout you'll write your first program, which we'll call "Yo."

Programs, Classes, and Objects, Oh My!

People regularly refer to Java as a language that lets you write **Object Oriented Programs** (abbreviated **OOP** – I'm not kidding). This is only partially correct. Java does make it really easy to create **objects** (more on those much later), but if you do it properly, those objects are really parts of **classes**.

OK, so what's a **class**? Basically speaking, everything in Java is a class, and that includes what we would normally call a **program**.

Two Main Types of Classes

A program is a particular type of class that I'll call a **standalone class**. I call it this because it can, well, stand alone. It can actually do things. It can run all by itself, although it often gets help from other classes.

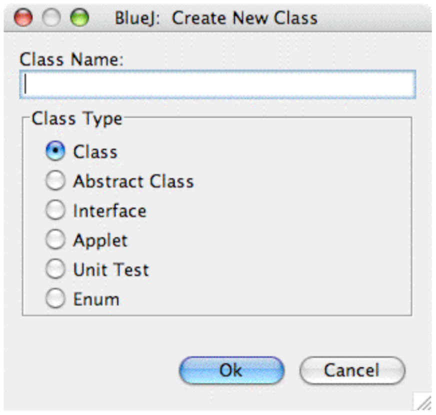
And this brings us to our second type of class. I'll call this a **helper class** because, well, it helps the standalone class.

So if I ever ask you what the two main types of classes are, the answer I'm looking for is **standalone** and **helper**.

Now let's get to work.

Creating Your Class

On the left-hand side of your project window, you'll see a little button that says **New Class**. Click on it, and you'll get the dialog box shown below.



The name of our class will be **Yo**. Type that in where it says **Class Name**. Don't hit the **OK** button yet.

Six Types of Classes?

OK, I said that there were two main types of classes, and now not only are you looking at a list of six, but the two I mentioned aren't even in the list. No problem. The two main types of classes I'm talking about are part of the **Class** class (How's that for confusing?). The other five classes are types that we're not even going to deal with in the class – I mean course.

For our purposes, whether a class is standalone or helper depends entirely on what's in it, and we'll get to that in a minute. But still, don't hit the **OK** button yet. There's one more thing to talk about.

Case Sensitivity

Java is a **case-sensitive** language. What I mean by this is that:

```
        this != This != THIS != ThIs
```

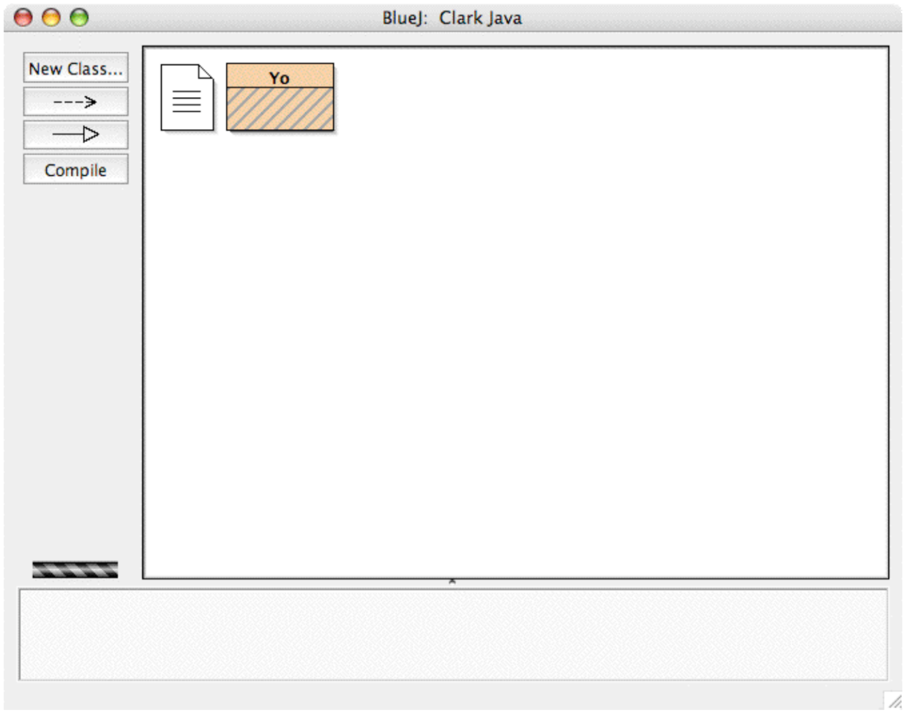
OK, so what was all that about? Well, let's start off with the **!=** thing, especially since that'll come up later. **!=** is Java for **not equal to**. So what I said was that **this** (all lowercase) is not the same as **This** (with a capital T), which is not the same as **THIS** (all lowercase), which is not the same as **ThIs** (crazy mixed-up case).

In Java you could have two variables named **num** and **NUM** and they would both be entirely different things.

So why am I telling you this? Because how you type your code matters. Your case has to match exactly, otherwise your program won't work.

What does this have to do with anything we've been doing so far? I said to call the new class **Yo**. That's uppercase Y and lowercase o.

Once you're sure that you've properly named your class, you can hit the **OK** button. When you do that, a new item appears in your project window, and it's called **Yo**. Check out the example below.



Writing Our Program

Double-click on **Yo**, and it opens up to show you what looks like an already written program. It is. BlueJ assumes that there are certain things that you want to start off with. It's wrong here. There's probably a way to turn this off, but I don't know it at the moment. So instead, just select **everything** in the file and **delete** it. Now we can start from scratch.

Comments

The very first thing we'll need are comments. The computer doesn't care one way or the other about comments. Comments are for the programmer, the programmer's teacher, and anyone else who comes along later on to try to figure out what was going on in the programmer's head when she wrote that horrible piece of code in the first place.

Since the computer doesn't care about the comments, we need a way to tell it to ignore them. This is done by putting in two slash marks (**//**). This tells the computer to ignore what comes after them.

So, now that you understand that, it's time to enter your first three lines of code. They'll be comments telling the **name** of the program, **what it does**, the **date** it was written, and **who** wrote it. The example below shows what Clark's program looks like so far:

```
//Yo
//Our first program
//7.1.06 Clark Kent
```

The Class Code

True, we've created the class already, but now we need to write the code for it. In the example below (and all examples from now on), the code in **gray highlight** is the new stuff to be added to what you already have.

```
//Yo
//Our first program
//7.1.06 Clark Kent

public class Yo
{//start class
}//end class
```

The first line (**public class Yo**) is the **class header**. It tells the name of the class and that it's **public** (don't worry about public and private right now. The next two lines are the **beginning** and **end** of the class. The symbol for **beginning** a section of code is the **left curly bracket**, and the symbol for **ending** it is the **right curly bracket**. There are comments after each one so that you can keep track of what's beginning and ending. When you write a long enough program, you'll want to keep these straight.

When you put in the ending curly bracket, BlueJ may automatically indent it. No problem. Just back it up to where we want it to be, which is right underneath the beginning bracket.

So after all this you have an empty class that does absolutely nothing. For now.

It's a good idea that anytime you're going to be doing beginning and ending curly brackets, to do them as a pair right from the start. That way you know that you always have a matched set. You can enter the information that goes between them later. And that's exactly what's going to happen now.

The Main Method

The main thing that distinguishes a standalone class from a helper class is the **main** method. A standalone class **must** have **one** and only **one** main method (that is a method called **main**). A helper class, on the other hand, will **never** have a main method. Got that straight? Good, then let's get to work entering that main method.

The example below shows what the code should look like now:

```
//Yo
//Our first program
//7.1.06 Clark Kent

public class Yo
{//start class
   public static void main()
   {//start main
   }//end main
}//end class
```

What does all this stuff mean? Again, don't worry about the **public** or the **static** for the moment. **Void** means that the method doesn't **return** anything (and don't worry about that either). **Main** is the name of the method. The two **parentheses** are there because all methods have to end with a set of parentheses. You'll see why later.

Did you notice that all we did here with **main** was to give the **method declaration**, the **beginning**, and the **ending**, just like we did with the class? If you did, then you probably figured out that we're going to fill the main method in later on, and that we just wanted to be sure that we had a matched set of brackets.

Formatting

Formatting is very important. Not for the computer, it doesn't care, but for the humans who have to read your code. A Java program should read like an **outline** (does anyone write those anymore?). Since **main** is part of **Yo**, we've indented it three spaces in. In a few moments, when we write the rest of **main**, we'll indent that three spaces inside of that.

The Rest of Main

OK, so now, after all this effort, the actual guts of the program will just be one line, inserted between **{//start main** and **}//end main**. Check out the example below, and notice how it's indented within the **main** method.

```
//Yo
//Our first program
//7.1.06 Clark Kent

public class Yo
{//start class
   public static void main()
   {//start main
      System.out.println("Yo! This is the program.");
   }//end main
}//end class
```

The Println Command

You've probably figured out that **println** (pronounced print**line**) is what prints stuff to the screen. But what's with all that **System.out** stuff before it?

Simply put, **println** (and its cousin, **print**, which you'll meet later on) is a command inside of the helper class **System.out**. There may be tons of other **printlns** out there. There may be a **Fred.println**, a **Sue.println**, and an **Amy.println** out there, but the one **println** that just about everyone wants to use is the one in **System.out**, so when you want to use that one, you need to call it by its **full name**.

By the way, did you notice that the name of the helper class is **System.out** with a capital S and a lowercase O? Attention to detail is very important here, because if you typed **system.out.println**, nothing would happen.

The **println** command writes a **string** of text to the screen. In this case, the string of text is what's between the quotation marks inside the parentheses. Anything inside the quotation marks of a **println** or **print** command goes on screen exactly the way it looks.

Semicolons

OK, now about that **semicolon**. All Java **statements** must end with a semicolon. That's how Java knows that it's the end of that statement.

But why does only one line in the entire program have a semicolon after it? Because, as hard as it may be to believe, only one line in the entire program is a **statement**.

Take a good look at the code. The first three lines are **comments**. They get ignored anyway. The line after that is the **class header**, that just tells us the name and type of the class. Not a statement. That's followed by the beginning curly bracket. Not a statement. This is followed by the **method header** (similar to the **class header**). Not a statement. Finally, after the beginning curly bracket for the method, we have **one** statement, and this gets a semicolon at the end of it.

Yeah, it seems like a lot of work for one statement, but not to worry, future programs will be much longer than this.

Compiling Your Program.

OK, if everything's all set, it's time to **compile** your program.

Compiling is a fancy word for how BlueJ (or any other Java environment) translates what you wrote in Java into the 1s and 0s that the computer needs in order to run the program. To compile your code, simply click on the **Compile** button at the top of your class window. If you like keyboard commands, **Apple-K** will do the job quite nicely (**Ctrl-K** if you're on a Windows machine).

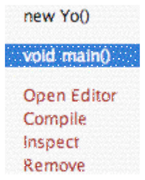
As BlueJ compiles your code, it looks for errors. Any time it finds an error, it will highlight it in yellow and tell you, at the bottom of the screen, what it **thinks** the problem is. I say it will tell you what it **thinks** the problem is because it doesn't always get it right. Sometimes you'll have done something so incredibly weird that it doesn't quite know what to make of it, and so it makes its best guess as to what the problem is.

OK, so if it's smart enough to know that you've made a mistake, why doesn't it just fix it for you? Because maybe what it thinks the problem is isn't the real problem. This is especially the case when the compiler finds a lot of errors. One mistake early on in the program can make everything that comes after it look bad. This is why you always fix the **first** mistake and then recompile before you try fixing any others. They might all disappear after you've fixed that first one.

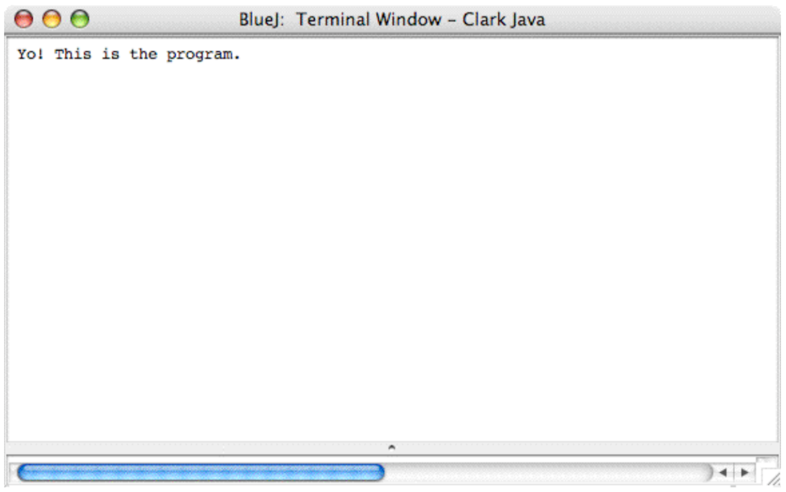
After you've successfully compiled your program, the bottom of the screen should say **Class compiled – no syntax errors**. Now it's time to run the program.

Running Your Program

To run your program, go back to the project window and **right-click** (**Ctrl-click** if you don't have a two-button mouse) on the **Yo** item. That'll bring up a little menu like the one on the right.



Select **void main()** from the list and now the **terminal** window comes up and displays the results of your **println** command. Check out the example below.



```
Yo! This is the program.
```

Enough for Now

OK, so with all that done, you've successfully written, compiled, and run your first program. Give yourself a hand.

The next handout will deal with getting **input** for this program. After all, what good is a program that doesn't let you give it information?