Mr G's Java Jive

#3: Getting Input and Using It

With this handout you'll write a program to converse with the user. We'll call it "Converse."

What We Know So Far

We know how to create a **project**. We know how to create a **class**. We know how to create a **main** method, and we know how to **output** text to the screen from that main method. But what good is a program that does output with no input? Not very. So now we'll write a program that lets us do both.

Getting Started

Create a new class called **Converse** and enter the following code:

```
//Converse
//A program to converse with the user
//today's date your name

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

Organizing Longer Programs

Yo was a very short program. It only really had one line to it. As a result, there wasn't much organization that needed to be done with it. However, most other programs aren't quite that short and simple. Organizing them into nice, logical sections before you get down to work helps you to get things in the right order and keep things straight.

In **Converse** we're going to ask for the user's name and then repeat it back. In order to get the name, we'll need a place to put it. That place is a **variable**. With that in mind, now we know that we need to do three things:

1. Create variables
2. Get input from the user
3. Give a response to the user

This means that we'll need three main sections to our program. Or, rather, three main sections to our **main** method. Check out the **gray** additions to the code shown below, and then add them to your own.

```
public static void main()
   {//start main
      //variables

      //get input

      //give response

   }//end main
```

Variables

For now we're going to need two variables, one called **myScanner** and one called **name**. Look at the example below to see how to **define** them, and then I'll explain what we just did.

```
public static void main()
   {//start main
      //variables
        Scanner myScanner = new Scanner(System.in);
        String name;

      //get input
```

What we've just defined are two **objects** (there's that word again). One is from the **Scanner** class and one is from the **String** class. You can always tell an object by the fact that its type begins with a capital letter. We'll talk about other types shortly.

We created a new Scanner object called **myScanner** and said that it equaled **new Scanner(System.in)**. You might have been able to figure out that what this means is that **myScanner** is a new **Scanner** object based on what happens in **System.in**. You might even have figured out that if **System.out** has to do with sending information **out** of the program, then **System.in** must have something to do with getting it **in**. You'd be right.

The other object we created was a String object called **name**. This is where the user's name will be stored after it gets typed in.

By the way, both of these lines are **statements**, and so need **semicolons** at the end.

Get Input

OK, we have our variables, but the program still doesn't do anything. It's time to write some code to get this conversation started. Check out the example below:

```
public static void main()
   {//start main
      //variables
        Scanner myScanner = new Scanner(System.in);
        String name;

      //get input
        System.out.print("Hi! What's your name? ");
        name=myScanner.nextLine();

      //give response
```

We've added two new statements here. The first one uses a **print** instead of a **println**. What's the differnce? A **println** puts what you want on the screen and then drops down to the beginning of the **next line**. A **print**, on the other hand, puts what you want on the screen, and then stays right there on that line, waiting for more. We used a **print** for the first statement because we wanted the user to type in her name right at the end of that line. By the way, did you notice that there's a space before the final quotation mark? That's so that the user's name and the question don't run together on screen.

The second statement says that **name** will equal the result of **myScanner.nextLine()**. What's going on here? A Scanner object has a bunch of different **methods** it can use to get information in from the

keyboard. **NextLine()** gets the next **whole line** of text, **nextInt()** gets the next **integer**, **nextDouble()** gets the next **double**, etc. But we're getting a little ahead of ourselves here, since we haven't talked about **int**, **double**, or anything else yet. Anyway, **myScanner.nextLine()** pulls in the next whole line of text from the keyboard. And that's exactly what we want.

Give Response

We've greeted the user and asked for their name. We've sucked the name into the program. Now we need to do something with it. Let's respond! Check out the code below:

```
public static void main()
   {//start main
      //variables
        Scanner myScanner = new Scanner(System.in);
        String name;

      //get input
        System.out.print("Hi! What's your name? ");
        name=myScanner.nextLine();

      //give response
        System.out.println("Nice to meet you, "+name+".");

   }//end main
```

Pay careful attention to what just happened here, because there are three things going on in that **println** statement. The first is that we output a normal **literal** string of text inside of a set of quotation marks (**Nice to meet you,**). But after that, we did something new. We used the **plus sign** to add the contents of our String object **name** to the output. There's a fancy word for this, and it's **concatenation**. It means to add one string of text to another. Then, after we added **name** to everything in the first literal, we used the **plus sign** again to add another literal with just a **period** in it (well, you do have to end a sentence properly, you know).

When this whole thing is compiled and run, it should nicely greet the user, ask for their name, and then respond nicely too. Let's check it out!

Cannot Find Symbol

Whoa! What happened here. We wrote all the code down perfectly, but when we went to compile we got a message at the bottom of the window that said **cannot find symbol**. Then when we looked at the actual code, one line was highlighted as being the problem. It was the line that said **Scanner myScanner = new Scanner(System.in)**. What's up with that?

Actually, I set you up. I knew it wouldn't work. At least not right off the bat. You see, the **Scanner** class is part of another fileset of Java that you'll have to **import** if you want to use it. It's in **java.util.Scanner**. The example below shows you how to import it.

```
import java.util.Scanner;

public class Converse
{//start class
```

Now this should compile and run quite nicely. We'll do more with this in handout #5.

This page intentionally left **almost** blank.