Basic JShell

JShell is a command-line tool for exploring parts of Java, helping you to learn the language, and also for prototyping code. It is called a *Read-Eval-Print Loop* (REPL) tool: In a command line terminal, you type a line of code and press the return/enter key; the line is read, evaluated, and any results are printed.

To the right, we show the first use of it. On line 1, we typed "jshell" and hit the return key, thus starting the tool. The tool printed a welcome. On line 2, we typed "5" and hit the return key. The tool printed "$1 ==> 5", showing us that the value of the expression we typed was 5 and that it has been assigned to a variable $1, so that we can later access the value later if we want. On line 3, we typed "/exit" to exit from the tool —it's important to know how to do that! The tool printed "Goodbye".

```
Davids-MacBook-Pro-3:~ davidgries$ jshell 1
|  Welcome to JShell -- Version 11.0.5
|  For an introduction type: /help intro

jshell> 5 2
$1 ==> 5

jshell> /exit 3
|  Goodbye
Davids-MacBook-Pro-3:~ davidgries$
```

Declarations and assignments

Line 1 to the right shows that, in Java, a variable must be declared before it is used.

Line 2 shows a declaration of **int** variable j along with an assignment of a value to it. JShell indicates that j has been assigned the value y.

In line 3, expression 2*3 is evaluated; JShell shows the value, 6, and also indicates that it has been assigned to variable $2.

Line 4 illustrates two points. First, note that the assignment is missing a semicolon. However, if a semicolon is missing at the end of a statement, JShell inserts it for you. Second, note that variable $2 is used in the righthand expression of the assignment.

```
jshell> j= 5 + 2; 1
|  Error:
|  cannot find symbol
|    symbol:   variable j
|  j= 5 + 2;
|  ^

jshell> int j= 5+2; 2
j ==> 7

jshell> 2*3 3
$2 ==> 6

jshell> j= j + $2 4
j ==> 13
```

if and if-else statements

You can write simple if statements on one line, as shown to the right. Don't hit the return key in the middle of it, because that action typically means the end of the input and what has been written will be processed by JShell. We show you below how to handle multiline input.

```
jshell> if (j < 10) j= 20; else j= 30;

jshell> j
j ==> 30
```

Multiline input

If you type an opening parenthesis and part of expression, as shown to the right, JShell recognizes that more text needs to be typed. It therefore types an indented "arrow" and waits for more input. We show an example, with "4)" being typed and then the return key being pressed. One can, of course have several lines; as long as the closing parenthesis hasn't been typed, JShell assumes more input is coming.

```
jshell> (5 +
   ...>
```

```
jshell> (5 +
   ...> 4)
$1 ==> 9

jshell>
```

How to stop when a mistake has been made

Sometimes, you made a mistake while typing a multiline snippet of code. You can have it deleted and start over by typing control-c. You see that happen in the image to the right. After typing "sf what?" and then the return key, we typed control-c to delete what we had typed and start over.

```
jshell> if (j
   ...>  sf what?
   ...>

jshell>
```

multiline if-else

To the right, is an if-else statement typed into JShell. As you can imagine, the appearance of an opening brace "{" without the balancing closing brace "}" causes JShell to assume more input is coming when the return key is pressed. In the image to the right we put in the indentation to show the structure. JShell does not do that for us.

```
jshell> if (b) {
   ...>    j= 6;
   ...>    j= j+1;
   ...> } else {
   ...>   j= 2;
   ...> }

jshell> j
j ==> 7
```