**JShell (Java 9 New Feature)**

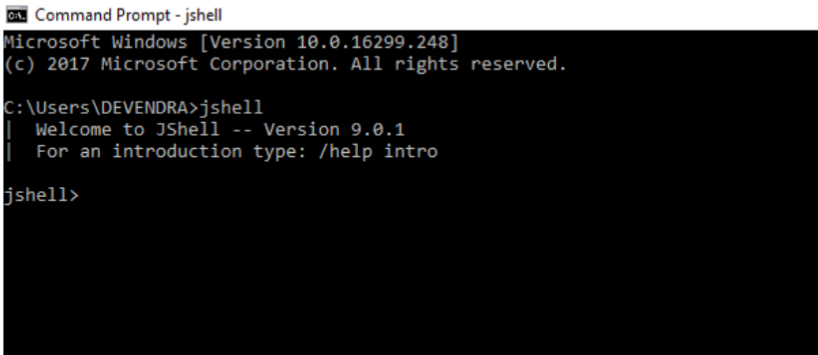
**Introduction to JShell**   
The Java Shell tool (JShell) is an interactive tool for learning the Java programming language and prototyping Java code. JShell is a Read-Evaluate-Print Loop (REPL), which evaluates declarations, statements, and expressions as they are entered and immediately shows the results. The tool is run from the command line. JShell was introduced in java 9 and hence can not be used in java 8.

**Why Use JShell ?**   
Jshell has reduced all the efforts that are required to run a Java program and test a business logic. If we don’t use Jshell, creating of Java program involves the following steps.

1. Open editor and write program
2. Save the program
3. Compile the program
4. Edit if any compile time error
5. Run the program
6. Edit if any runtime error
7. Repeat the process

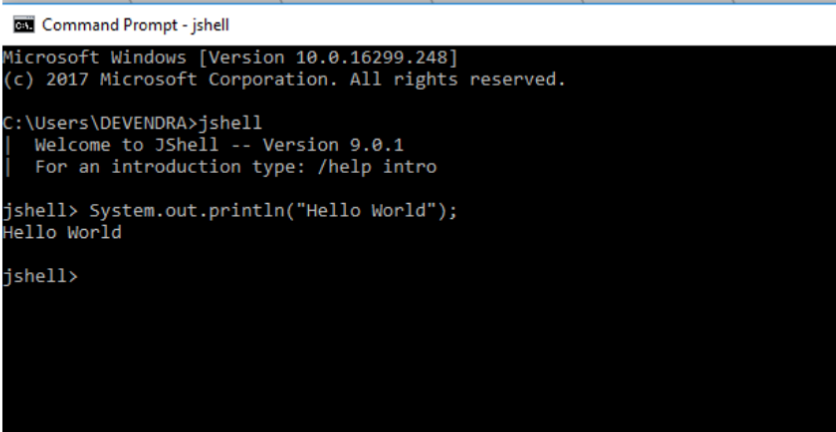
Jshell does not require above steps. We can evaluate statements, methods and classes, even can write hello program without creating class.   
JShell helps you try out code and easily explore options as you develop your program. You can test individual statements, try out different variations of a method, and experiment with unfamiliar APIs within the JShell session. JShell doesn’t replace an IDE. As you develop your program, paste code into JShell to try it out, and then paste working code from JShell into your program editor or IDE.

**Starting and Stopping JShell**   
To start Jshell, first we must have installed Java 9 then open terminal in Linux or command prompt in windows and type jshell. It will start jshell session and displays a welcome message to the console. 



**Hello Java Message**

To display a simple “Hello Java” message, write print command without creating class and hit enter. 



To exit JShell, enter command:

jshell> /exit

| Goodbye

**Snippets**   
JShell accepts Java statements; variable, method, and class definitions; imports; and expressions. These pieces of Java code are referred to as snippets.   
Enter the following sample statement at the prompt, and review the output that is shown: 

jshell> int x = 11

x ==> 11

| created variable x : int

First, the result is shown. Read this as: the variable x has the value 45. Because you are in verbose mode, a description of what occurred is also shown. Informative messages start with a vertical bar. Notice that both the name and the type of the created variable are shown.   
When an expression is entered that doesn’t have a named variable, a scratch variable is created so that the value can be referenced later. The following example shows scratch values for an expression and for the results of a method. The example also shows the continuation prompt (…>) that is used when a snippet requires more than one line of input to complete:

jshell> 8 + 8

$3 ==> 16

| created scratch variable $3 : int

jshell> String double(String s) {

...> return s + s;

...> }

| created method double(String)

jshell> double("Geeks")

$5 ==> "GeeksGeeks"

| created scratch variable $5 : String

**Exceptions**   
In an exception backtrace, feedback identifies the snippet and the location within the snippet where the exception occurred.

The location within the code entered into JShell is displayed as #ID:line-number, where snippet ID is the number displayed by the /list command, and line-number is the line number within the snippet. In the following example, the exception occurs in snippet 1, which is the divide() method, on the second line of the method:

jshell> int half(int x, int y) {

...> return x / y;

...> }

| created method divide(int,int)

jshell> divide(5, 0)

| java.lang.ArithmeticException thrown: / by zero

| at divide (#1:2)

| at (#2:1)

jshell> /list

1 : int divide(int x, int y) {

return x / y;

}

2 : divide(5, 0)

**Introduction to Commands**   
JShell commands control the environment and display information within a session.

Commands are distinguished from snippets by a leading forward slash (/). For information about the current variables, methods, and types, use the /vars, /methods, and /types commands. For a list of entered snippets, use the /list command. The following example shows these commands based on the examples 

jshell> /vars

| int x = 8

| int $3 = 16

| String $5 = "GeeksGeeks"

jshell> /methods

| double(String)String

jshell> /list

1 : System.out.println("Hi");

2 : int x = 8;

3 : 8 + 8

4 : String double(String s) {

return s + s;

}

5 : double("Ocean")

Notice that the types and values of variables and the type signature of methods are displayed.

JShell has a default startup script that is silently and automatically executed before JShell starts, so that you can get to work quickly. Entries from the startup script aren’t listed unless you request them with the /list -start or /list -all command: 

jshell> /list -all

s1 : import java.util.\*;

s2 : import java.io.\*;

s3 : import java.math.\*;

s4 : import java.net.\*;

s5 : import java.util.concurrent.\*;

s6 : import java.util.prefs.\*;

s7 : import java.util.regex.\*;

1 : System.out.println("Hi");

2 : int x = 8;

3 : 8 + 8

4 : String double(String s) {

return s + s;

}

5 : double("GeeksGeeks")

The default startup script consists of several common imports. You can personalize your startup entries with the /set start command. For information about this command, enter /help /set start. The /save -start command saves the current startup script as a starting point for your own startup script.

Other important commands include /exit to leave JShell, /save to save your snippets, and /open to enter snippets from a file.

Enter /help for a list of the JShell commands.

References – <https://docs.oracle.com/javase/9/tools/jshell.htm>