

Lesson 17: JShell



ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Objectives

After completing this lesson, you should be able to:

- Explain the REPL process and how it differs from writing code in an IDE
- Launch JShell
- Create JShell scratch variables and snippets
- Identify available JShell commands and other capabilities
- Identify how an IDE enhances the JShell user experience



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Topics

- Testing code and APIs
- JShell Basics
- JShell in an IDE

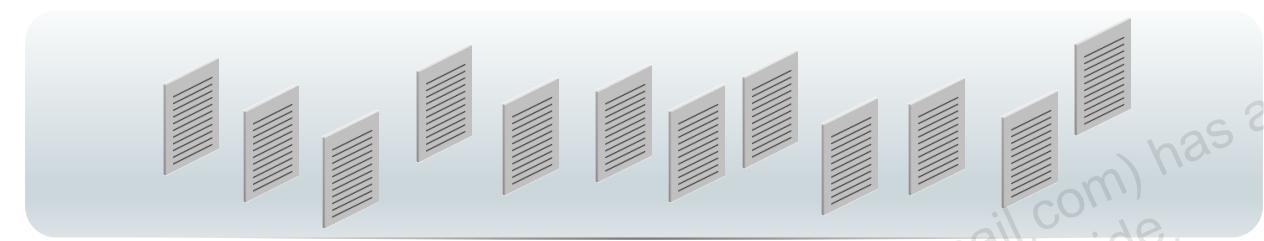


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



A Million Test Classes and Main Methods

- Production code is dedicated to properly launching and running an application.
 - We'd complicate it by adding throwaway code.
 - It's a dangerous place for experimentation.
 - We'd alternatively clutter the IDE by creating little main methods or test projects.
- Creating a new main method or project sometimes feels like an unnecessary ceremony.
 - We're not necessarily interested in creating or duplicating a program.
 - We're interested in testing a few lines of code.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

JShell Provides a Solution

- It's a command line interface.
- It avoids the ceremony of creating a new program and gets right into testing code.
- At any time you can:
 - Explore an API, language features, a class you wrote; do other experiments with logic, variables, or methods.
 - Prototype ideas and incrementally write more-complex code.
- You'll get instant feedback from the Read Evaluate Print Loop (REPL) process.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Topics

- Testing code and APIs
- **JShell Basics**
- JShell in an IDE



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Comparing Normal Execution with REPL

- Normal Execution:
 - You enter all your code ahead of time.
 - Compile your code.
 - The program runs once in its entirety.
 - If after the first run you realize you've made a mistake, you need to run the entire program again.
- JShell's REPL:
 - You enter one line of code at a time.
 - You get feedback on that one line.
 - If the feedback proved useful, you can use that information to alter your next line of code accordingly.
- We'll look at simple examples to illustrate this.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Getting Started with JShell and REPL

- To launch JShell:
 - Open a terminal.
 - Enter `jshell`.
- Start entering code, for example:
 - R. The expression `Math.pow(2, 7)` is **read** into JShell.
 - E. The expression is **evaluated**.
 - P. Its value is **printed**.
 - L. The state of JShell **loops** back to where it began.
 - Repeat the process and enter more expressions.



```
[oracle@edvmr1p0:~]$ jshell
| Welcome to JShell -- Version 9
| For an introduction type: /help intro

jshell> Math.pow(2,7);
$1 ==> 128.0

jshell>
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Scratch Variables

- `Math.pow(2, 7)` evaluates to 128.
- 128 is reported back as `$1`.
- `$1` is a JShell **scratch variable**.
- Like most other variables, a scratch variable can:
 - Store the result of a method call
 - Be referenced later
 - Have its value changes
 - Be primitives or Object types
- Names are auto generated.
- Great for testing unfamiliar methods or other short experiments.



A screenshot of a terminal window titled "oracle@edvmr1p0:~". The window shows the following JShell session:

```
[oracle@edvmr1p0 ~]$ jshell
| Welcome to JShell -- Version 9
| For an introduction type: /help intro
jshell> Math.pow(2,7);
$1 ==> 128.0

jshell> 2
$2 ==> 2

jshell> $1+$2
$3 ==> 130.0

jshell> "Hello World!"
$4 ==> "Hello World!"

jshell> 3.14
$5 ==> 3.14

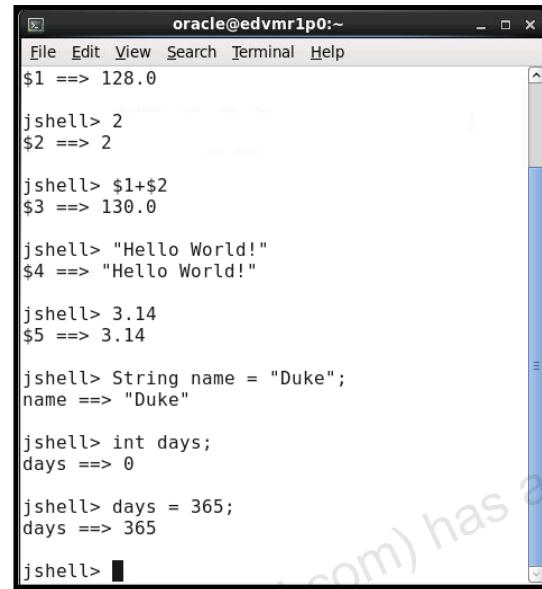
jshell> ■
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Declaring Traditional Variables

- Too many scratch variables lead to confusion.
 - You may create an unlimited number of scratch variables.
 - Their names aren't descriptive.
 - It becomes hard to remember the purpose of each one.
- Traditional variables have names which provides context for their purpose.
- JShell allows you to declare, reference, and manipulate variables as you normally would.



```
oracle@edvmr1p0:~$1 ==> 128.0
jshell> 2
$2 ==> 2
jshell> $1+$2
$3 ==> 130.0
jshell> "Hello World!"
$4 ==> "Hello World!"

jshell> 3.14
$5 ==> 3.14
jshell> String name = "Duke";
name ==> "Duke"
jshell> int days;
days ==> 0
jshell> days = 365;
days ==> 365
jshell>
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Code Snippets

The term **snippet** refers to the code you enter in a single JShell loop.

- Declarations

- String s = "hello"
- int twice(int x) {return x+x;}
- class Pair<T> {T a, b; Pair(...)}
- interface Reusable {}
- import java.nio.file.*

- Expressions

- Math.pow(2, 7)
- twice(12)
- new Pair<>("red", "blue")
- transactions.stream()
 - .filter(t->t.getType() ==trans.PIN)
 - .map(trans::getID)
 - .collect(toList())

- Statements

- while(mat.find()){...}
- if(x < 0){...}
- switch(val){
 - case FMT:
 - format();
 - break;...

- Not Allowed

- package foo; 
- Top-level access modifiers
 - static final
- Top-level statements of:
 - break continue return



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

A snippet can be one line. A snippet can be many lines including a loop, a method, a class... This slide shows examples of declarations, expressions, and statements. What's not allowed is syntax that would seem illogical.

A package statement doesn't make sense because your goal in JShell is to test code, not to develop libraries.

The break keyword wouldn't make sense unless there was a loop to break.

Completing a Code Snippet

- Some snippets are best written across many lines.

- Methods
- Classes
- for loop statements

- JShell waits for the snippet to be complete.
 - It detects the final closing curly brace.
 - Then it performs any evaluation.

A screenshot of a terminal window titled "oracle@edvmr1p0:~". The window has a menu bar with File, Edit, View, Search, Terminal, and Help. The main area shows the command "jshell> for(int i=0; i<5; i++){" followed by three ellipsis lines (...>) and a cursor. The scroll bar on the right is visible.

A screenshot of a terminal window titled "oracle@edvmr1p0:~". The window has a menu bar with File, Edit, View, Search, Terminal, and Help. The main area shows the completed code "jshell> for(int i=0; i<5; i++){...> System.out.print(i + " ");...>}" followed by the output "0 1 2 3 4" and the command "jshell>". The scroll bar on the right is visible.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Tab Completion and Tab Tips

Confused about your options?

- After the dot operator, press tab to see a list of available fields, variables, or classes.
- Press tab as you call a method to view possible signatures.
- Press tab again to see documentation.

The image shows two terminal windows side-by-side. The top window, titled 'oracle@edvmr1po:~', displays the Java shell (jshell) command 'jshell> System.' followed by a list of available methods and fields. The bottom window, also titled 'oracle@edvmr1po:~', shows the command 'jshell> System.exit()' followed by a 'Signatures:' section and a prompt to 'press tab again to see documentation'.

Top Window (jshell> System.):

```
jshell> System.  
Logger  
class  
currentTimeMillis()  
gc()  
getProperty()  
identityHashCode()  
lineSeparator()  
mapLibraryName()  
runFinalization()  
setIn()  
setProperty()  
LoggerFinder  
clearProperty()  
err  
getLogger()  
getSecurityManager()  
in  
load()  
nanoTime()  
runFinalizersOnExit()  
setOut()  
setSecurityManager()  
arraycopy()  
console()  
exit()  
getProperties()  
getenv()  
inheritedChannel()  
loadLibrary()  
out  
setErr()  
setProperties()
```

Bottom Window (jshell> System.exit()):

```
jshell> System.exit()  
Signatures:  
void System.exit(int status)  
<press tab again to see documentation>  
jshell> System.exit()
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

This is helpful if you forget or are curious about what options are available to you from a particular class.

Semicolons

A screenshot of a Java JShell terminal window titled "oracle@edvmr1p0:~". The terminal shows the following interactions:

```
jshell> Math.pow(2,7);
$1 ==> 128.0

jshell> Math.pow(2,7)
$2 ==> 128.0

jshell> for(int i=0; i<5; i++){
...>     System.out.print(i + " ");
...>
0 1 2 3 4
jshell> for(int i=0; i<5; i++){
...>     System.out.print(i + " ")
...>
| Error:
| ';' expected
| System.out.print(i + " ")
|
jshell> ■
```

The terminal window has a blue vertical scroll bar on the right. Three blue arrows point from the explanatory text on the right to specific lines in the terminal output: one arrow points to the first line starting with "jshell> Math.pow(2,7);", another points to the line "jshell> Math.pow(2,7)", and a third points to the line "System.out.print(i + " ")".

The semicolon which ends a snippet is optional.

All other semicolons are mandatory.

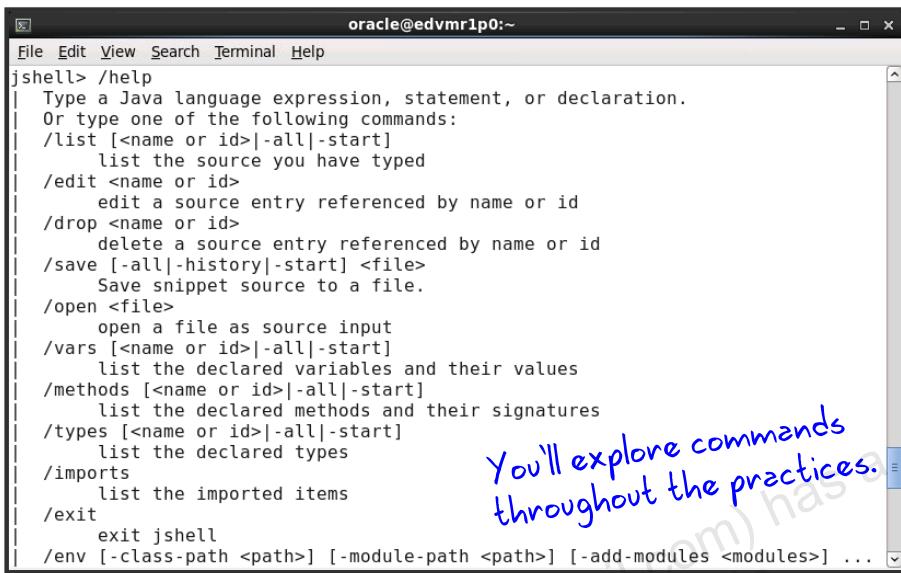


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Omitting the final semicolon in a snippet is helpful if your snippets are very short, which allows you to work more quickly.

JShell Commands

- Commands allow you to do many things. For example:
 - Get snippet information.
 - Edit a snippet.
 - Affect the JShell session.
 - Show history.
- They're distinguished by a leading slash /
- Enter the /help command to reveal a list of all commands.



The screenshot shows a terminal window titled "oracle@edvmrlp0:~". The command "/help" is entered, and the output lists various JShell commands:

```
jshell> /help
Type a Java language expression, statement, or declaration.
Or type one of the following commands:
/list [<name or id>|-all|-start]
    list the source you have typed
/edit <name or id>
    edit a source entry referenced by name or id
/drop <name or id>
    delete a source entry referenced by name or id
/save [-all|-history|-start] <file>
    Save snippet source to a file.
/open <file>
    open a file as source input
/vars [<name or id>|-all|-start]
    list the declared variables and their values
/methods [<name or id>|-all|-start]
    list the declared methods and their signatures
/types [<name or id>|-all|-start]
    list the declared types
/imports
    list the imported items
/exit
    exit jshell
/env [-class-path <path>] [-module-path <path>] [-add-modules <modules>] ...
```

A handwritten note in blue ink on the right side of the terminal window says: "You'll explore commands throughout the practices."



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Importing Packages

- Several packages are imported into JShell by default.
 - Type `/imports` to reveal the list.
- To test other APIs:
 - Write an import statement for the relevant packages.
 - Ensure the classpath is set appropriately.
 - JShell reports the classpath when it launches.
 - Use the `/classpath` command to set it manually.



A screenshot of a terminal window titled "oracle@edvmr1p0:~". The window has a menu bar with File, Edit, View, Search, Terminal, and Help. The terminal prompt is "jshell>". The user has run the command "/imports", which lists several Java package imports:
import java.io.*
import java.math.*
import java.net.*
import java.nio.file.*
import java.util.*
import java.util.concurrent.*
import java.util.function.*
import java.util.prefs.*
import java.util.regex.*
import java.util.stream.*



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Quiz 17-1

Do you need to make an import statement before creating an `ArrayList` in JShell?

- a. Yes
- b. No



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Answer: No

Creating an `ArrayList` in an IDE normally requires an import statement. JShell is different because this import is done as part of automatically importing the package `java.util`.

Students may want to enter the `/imports` command or try creating an `ArrayList` in JShell themselves to uncover the answer.

Topics

- Testing code and APIs
- JShell Basics
- JShell in an IDE



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Why Incorporate JShell in an IDE?

- IDEs perform a lot of work on behalf of developers.
- IDEs are designed to help developers with complex projects.
 - Precision code editing
 - Shortcuts (for example, `sout +Tab` for `System.out.println()`)
 - Auto-complete
 - Tips for fixing broken code
 - Java documentation integration
 - Matching curly braces
- Combine the benefits of two tools.
 - Quick feedback from JShell's REPL
 - Robust assistance from an IDE



```
12 | [1]-> for(int i=0; i<5; i++){
13 |     System.out.println(i++);
14 |
15 | [2]-> for(int i=0; i<5; i++){
16 |     System.out.println(i);
17 |
18 | [3]-> for(int i=0; i<10; i++){
19 |     System.out.println(i);
20 |
21 | }
```

Output - Java Shell - project Sol_11_02_JShell

```
0
1
2
3
4
5
0
1
2
3
4
```

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Use Cases

- Experiment with unfamiliar code:
 - A class your colleague wrote
 - A Java API
 - A third party library or module
- Bypass and preserve the existing program.
 - Run quick tests without breaking existing code.
 - Simulate a scenario.
- Test ideas on how to build out your program.
 - Start with simple tests.
 - Gradually build up complexity.
 - Eventually integrate a workable solution with the rest of your program.



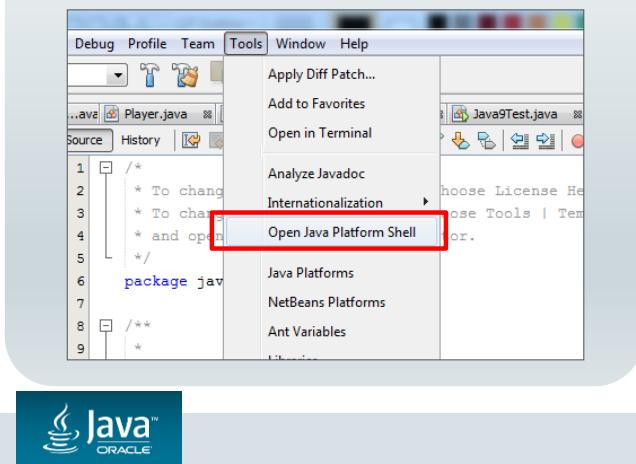
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Run quick tests. Your existing code is already dedicated to initializing other components of your program. You don't need to break it to run a quick test.

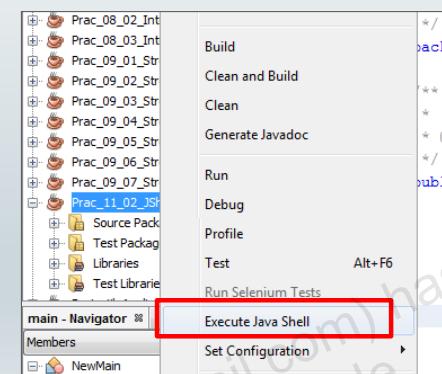
Simulate a scenario. For example, will your program need to run for a long time before it reaches a state where a particular class is instantiated and testable? Bypass this by instantiating the class and simulating the state you need.

Two Ways to Open JShell in NetBeans

1. Open a general JShell session.
 - Select **Tools**.
 - **Open Java Platform Shell**.



2. Open JShell on a specific project.
 - Right-click your project.
 - Select **Execute Java Shell**.
 - Make any necessary imports.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

When you open JShell on a project, you're specifying a classpath. JShell tells you the classpath when it launches. You'll need to manually write import statements for any packages within the project you wish to experiment with.

Summary

In this lesson, you should have learned how to:

- Explain the REPL process and how it differs from writing code in an IDE
- Launch JShell
- Create JShell scratch variables and snippets
- Identify available JShell commands and other capabilities
- Identify how an IDE enhances the JShell user experience



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Practices

- 17-1: Variables in JShell
- 17-2: Methods in JShell
- 17-3: Forward-Referencing



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2018, Oracle and/or its affiliates.

Moisés Ocampo Sámano (aos_moy78@hotmail.com) has a
non-transferable license to use this Student Guide.