Lesson 17: JShell

17

ORACLE

Objectives

After completing this lesson, you should be able to:

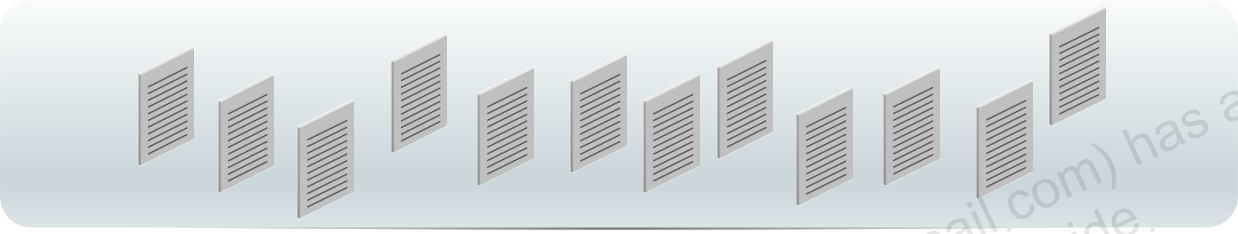
- Explain the REPL process and how it differs from writing code in an IDE
- Launch JShell
- Create JShell scratch variables and snippets
- Identify available JShell commands and other capabilities
- Identify how an IDE enhances the JShell user experience

Topics

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

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.



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.

This can be a great resource for both new and experienced developers who are looking to get a feel for language features.

Topics

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

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.

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.

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.

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.

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`

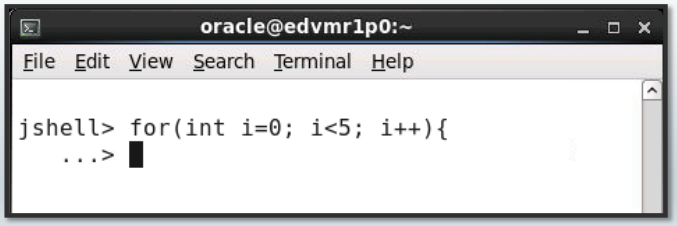
A snippet can be one line. A snippet can be many lines including a loop, a method, a class… This slide shows examples of declarations, expressions, and statements. What's not allowed is syntax that would seem illogical.

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

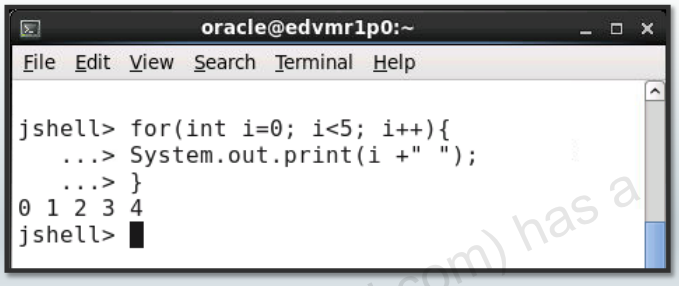
The break keyword wouldn't make sense unless there was a loop to break.

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.

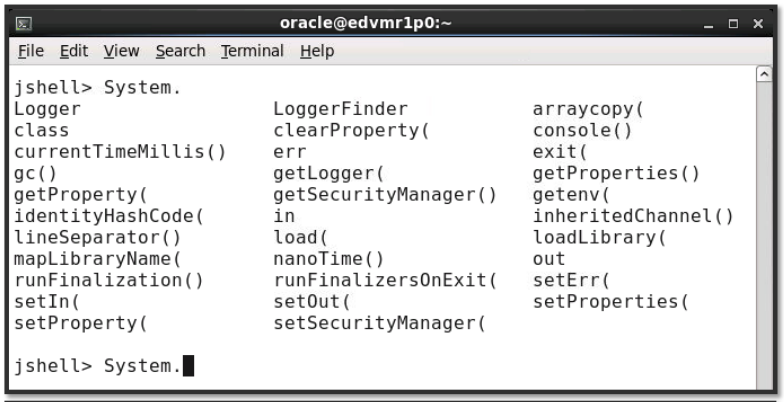


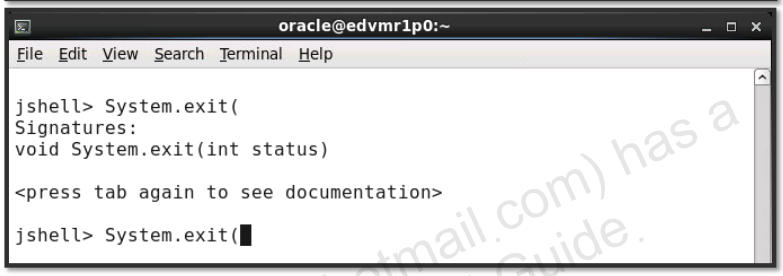
Tab Completion and Tab Tips

Confused about your options?

- After the dot operator, press tab to see a list of available fields, variables, or classes.

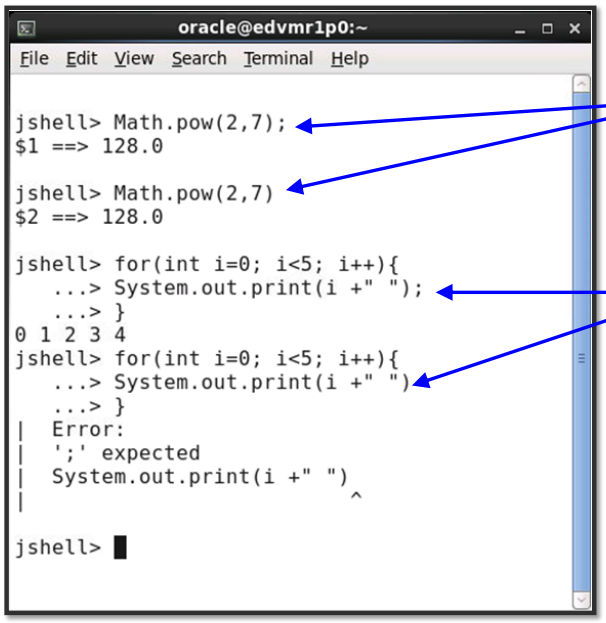
- Press tab as you call a method to view possible signatures.
- Press tab again to see documentation.





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

Semicolons



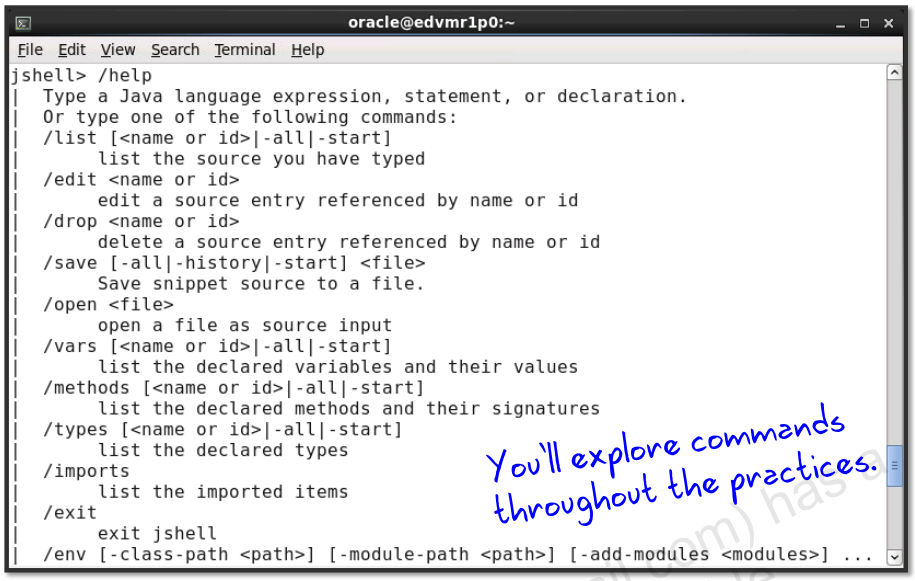
The semicolon which ends a snippet is optional.

All other semicolons are mandatory.

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.



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.

Quiz 17-1

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

a. Yes
b. No

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.

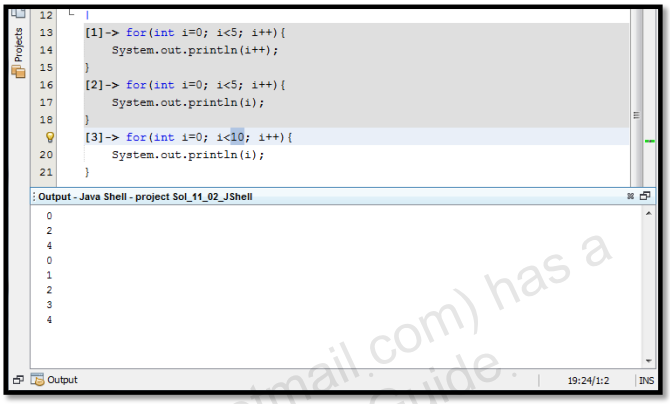
Students may want to enter the /imports command or try creating an ArrayList in JShell themselves to uncover the answer.

Topics

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

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



Use Cases

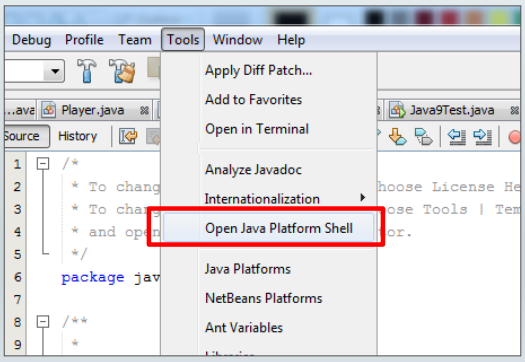
- Experiment with unfamiliar code:
  - A class your colleague wrote
  - A Java API
  - A third party library or module
- Bypass and preserve the existing program.
  - Run quick tests without breaking existing code.
  - Simulate a scenario.
- Test ideas on how to build out your program.
  - Start with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

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.

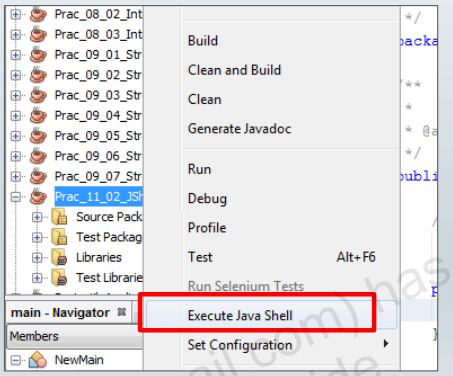
Simulate a scenario. For example, will your program need to run for a long time before it reaches a state where a particular class is instantiated and testable? Bypass this by instantiating the class and simulating the state you need.

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.



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

Practices

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