NAME

jshell − interactively evaluate declarations, statements, and expressions of the Java programming language in a read−eval−print loop (REPL)

SYNOPSIS

`jshell` [*options*] [*load−files*]

options Command−line options, separated by spaces. See **Options for jshell**.

load−files

One or more scripts to run when the tool is started. Scripts can contain any valid code snippets or JShell commands.

The script can be a local file or one of following predefined scripts:

`DEFAULT`
Loads the default entries, which are commonly used as imports.

`JAVASE`
Imports all Java SE packages.

`PRINTING`
Defines `print`, `println`, and `printf` as `jshell` methods for use within the tool.

`TOOLING`
Defines `javac`, `jar`, and other methods for running JDK tools via their command−line interface within the `jshell` tool.

For more than one script, use a space to separate the names. Scripts are run in the order in which they're entered on the command line. Command−line scripts are run after startup scripts. To run a script after JShell is started, use the `/open` command.

To accept input from standard input and suppress the interactive I/O, enter a hyphen (−) for *load−files*. This option enables the use of the `jshell` tool in pipe chains.

DESCRIPTION

JShell provides a way to interactively evaluate declarations, statements, and expressions of the Java programming language, making it easier to learn the language, explore unfamiliar code and APIs, and prototype complex code. Java statements, variable definitions, method definitions, class definitions, import statements, and expressions are accepted. The bits of code entered are called snippets.

As snippets are entered, they're evaluated, and feedback is provided. Feedback varies from the results and explanations of actions to nothing, depending on the snippet entered and the feedback mode chosen. Errors are described regardless of the feedback mode. Start with the verbose mode to get the most feedback while learning the tool.

Command−line options are available for configuring the initial environment when JShell is started. Within JShell, commands are available for modifying the environment as needed.

Existing snippets can be loaded from a file to initialize a JShell session, or at any time within a session. Snippets can be modified within the session to try out different variations and make corrections. To keep snippets for later use, save them to a file.

OPTIONS FOR JSHELL

`--add-exports` *module*/*package*
Specifies a package to be considered as exported from its defining module.

`--add-modules` *module*[`,` *module*...]
Specifies the root modules to resolve in addition to the initial module.

`-C`*flag* passes *flag* to the Java compiler inside JShell. For example, `-C-Xlint` enables all the recommended lint warnings, and `-C--release=<N>` compiles for Java SE N, as if −−release N was specified.

`--class-path` *path*
: Specifies the directories and archives that are searched to locate class files. This option overrides the path in the `CLASSPATH` environment variable. If the environment variable isn't set and this option isn't used, then the current directory is searched. For Linux and macOS, use a colon (:) to separate items in the path. For Windows, use a semicolon (;) to separate items.

`--enable-preview`
: Allows code to depend on the preview features of this release.

`--execution` *specification*
: Specifies an alternate execution engine, where *specification* is an ExecutionControl spec. See the documentation of the package jdk.jshell.spi for the syntax of the spec.

`--feedback` *mode*
: Sets the initial level of feedback provided in response to what's entered. The initial level can be overridden within a session by using the `/set feedback` *mode* command. The default is `normal`.

  The following values are valid for *mode*:

  `verbose`
  : Provides detailed feedback for entries. Additional information about the action performed is displayed after the result of the action. The next prompt is separated from the feedback by a blank line.

  `normal`
  : Provides an average amount of feedback. The next prompt is separated from the feedback by a blank line.

  `concise`
  : Provides minimal feedback. The next prompt immediately follows the code snippet or feedback.

  `silent`
  : Provides no feedback. The next prompt immediately follows the code snippet.

  custom
  : Provides custom feedback based on how the mode is defined. Custom feedback modes are created within JShell by using the `/set mode` command.

`--help` or `-h` or `-?`
: Prints a summary of standard options and exits the tool.

`--help-extra` or `-X`
: Prints a summary of nonstandard options and exits the tool. Nonstandard options are subject to change without notice.

`-J`*flag*
: passes *flag* to the runtime system, but has no effect on the execution of code snippets. To specify flags that affect the execution of code snippets, use `-R`*flag*. Alternatively, use `-J`*flag* with `--execution local`.

`--module-path` *modulepath*
: Specifies where to find application modules. For Linux and macOS, use a colon (:) to separate items in the path. For Windows, use a semicolon (;) to separate items.

`--no-startup`
: Prevents startup scripts from running when JShell starts. Use this option to run only the scripts entered on the command line when JShell is started, or to start JShell without any preloaded information if no scripts are entered. This option can't be used if the `--startup` option is used.

`-q`
: Sets the feedback mode to `concise`, which is the same as entering `--feedback concise`.

`-R`*flag*
: passes *flag* to the runtime system only when code snippets are executed. For example, `-R-Dfoo=bar` means that execution of the snippet `System.getProperty("foo")` will return `"bar"`.

`-s` Sets the feedback mode to `silent`, which is the same as entering `--feedback silent`.

`--show-version`
Prints version information and enters the tool.

`--startup` *file*
Overrides the default startup script for this session. The script can contain any valid code snippets or commands.

The script can be a local file or one of the following predefined scripts:

`DEFAULT`
Loads the default entries, which are commonly used as imports.

`JAVASE`
Imports all Java SE packages.

`PRINTING`
Defines `print`, `println`, and `printf` as `jshell` methods for use within the tool.

`TOOLING`
Defines `javac`, `jar`, and other methods for running JDK tools via their command−line interface within the `jshell` tool.

For more than one script, provide a separate instance of this option for each script. Startup scripts are run when JShell is first started and when the session is restarted with the `/reset`, `/reload`, or `/env` command. Startup scripts are run in the order in which they're entered on the command line.

This option can't be used if the `--no-startup` option is used.

`-v` Sets the feedback mode to `verbose`, which is the same as entering `--feedback verbose`.

`--version`
Prints version information and exits the tool.

JSHELL COMMANDS

Within the `jshell` tool, commands are used to modify the environment and manage code snippets.

`/drop` {*name*|*id*|*startID−endID*} [{*name*|*id*|*startID−endID*}...]
Drops snippets identified by name, ID, or ID range, making them inactive. For a range of IDs, provide the starting ID and ending ID separated with a hyphen. To provide a list, separate the items in the list with a space. Use the `/list` command to see the IDs of code snippets.

`/edit` [*option*]
Opens an editor. If no option is entered, then the editor opens with the active snippets.

The following options are valid:

{*name*|*id*|*startID−endID*} [{*name*|*id*|*startID−endID*}...]
Opens the editor with the snippets identified by name, ID, or ID range. For a range of IDs, provide the starting ID and ending ID separated with a hyphen. To provide a list, separate the items in the list with a space. Use the `/list` command to see the IDs of code snippets.

`-all` Opens the editor with all snippets, including startup snippets and snippets that failed, were overwritten, or were dropped.

`-start`
Opens the editor with startup snippets that were evaluated when JShell was started.

To exit edit mode, close the editor window, or respond to the prompt provided if the `-wait` option was used when the editor was set.

Use the `/set editor` command to specify the editor to use. If no editor is set, then the following environment variables are checked in order: `JSHELLEDITOR`, `VISUAL`, and `EDITOR`. If no

editor is set in JShell and none of the editor environment variables is set, then a simple default editor is used.

`/env` [*options*]
: Displays the environment settings, or updates the environment settings and restarts the session. If no option is entered, then the current environment settings are displayed. If one or more options are entered, then the session is restarted as follows:

  - Updates the environment settings with the provided options.
  - Resets the execution state.
  - Runs the startup scripts.
  - Silently replays the history in the order entered. The history includes all valid snippets or `/drop` commands entered at the `jshell` prompt, in scripts entered on the command line, or scripts entered with the `/open` command.

  Environment settings entered on the command line or provided with a previous `/reset`, `/env`, or `/reload` command are maintained unless an *option* is entered that overwrites the setting.

  The following options are valid:

  `--add-modules` *module*[`,` *module*...]
  : Specifies the root modules to resolve in addition to the initial module.

  `--add-exports` *source−module*`/`*package*`=`*target−module*[`,` *target−module*]*
  : Adds an export of *package* from *source−module* to *target−module*.

  `--class-path` *path*
  : Specifies the directories and archives that are searched to locate class files. This option overrides the path in the `CLASSPATH` environment variable. If the environment variable isn't set and this option isn't used, then the current directory is searched. For Linux and macOS, use a colon (`:`) to separate items in the path. For Windows, use a semicolon (`;`) to separate items.

  `--module-path` *modulepath*
  : Specifies where to find application modules. For Linux and macOS, use a colon (`:`) to separate items in the path. For Windows, use a semicolon (`;`) to separate items.

`/exit` [*integer−expression−snippet*]
: Exits the tool. If no snippet is entered, the exit status is zero. If a snippet is entered and the result of the snippet is an integer, the result is used as the exit status. If an error occurs, or the result of the snippet is not an integer, an error is displayed and the tool remains active.

`/history`
: Displays what was entered in this session.

`/help` [*command*|*subject*]
: Displays information about commands and subjects. If no options are entered, then a summary of information for all commands and a list of available subjects are displayed. If a valid command is provided, then expanded information for that command is displayed. If a valid subject is entered, then information about that subject is displayed.

  The following values for *subject* are valid:

  `context`
  : Describes the options that are available for configuring the environment.

  `intro` Provides an introduction to the tool.

  `shortcuts`
  : Describes keystrokes for completing commands and snippets. See **Input Shortcuts**.

`/imports`
: Displays the current active imports, including those from the startup scripts and scripts that were entered on the command line when JShell was started.

`/list` [*option*]
: Displays a list of snippets and their IDs. If no option is entered, then all active snippets are displayed, but startup snippets aren't.

  The following options are valid:

  {*name*|*id*|*startID*−*endID*} [{*name*|*id*|*startID*−*endID*}...]
  : Displays the snippets identified by name, ID, or ID range. For a range of IDs, provide the starting ID and ending ID separated with a hyphen. To provide a list, separate the items in the list with a space.

  `−all` Displays all snippets, including startup snippets and snippets that failed, were overwritten, or were dropped. IDs that begin with `s` are startup snippets. IDs that begin with `e` are snippets that failed.

  `−start`
  : Displays startup snippets that were evaluated when JShell was started.

`/methods` [*option*]
: Displays information about the methods that were entered. If no option is entered, then the name, parameter types, and return type of all active methods are displayed.

  The following options are valid:

  {*name*|*id*|*startID*−*endID*} [{*name*|*id*|*startID*−*endID*}...]
  : Displays information for methods identified by name, ID, or ID range. For a range of IDs, provide the starting ID and ending ID separated with a hyphen. To provide a list, separate the items in the list with a space. Use the `/list` command to see the IDs of code snippets.

  `−all` Displays information for all methods, including those added when JShell was started, and methods that failed, were overwritten, or were dropped.

  `−start`
  : Displays information for startup methods that were added when JShell was started.

`/open` *file*
: Opens the script specified and reads the snippets into the tool. The script can be a local file or one of the following predefined scripts:

  `DEFAULT`
  : Loads the default entries, which are commonly used as imports.

  `JAVASE`
  : Imports all Java SE packages.

  `PRINTING`
  : Defines `print`, `println`, and `printf` as `jshell` methods for use within the tool.

  `TOOLING`
  : Defines `javac`, `jar`, and other methods for running JDK tools via their command−line interface within the `jshell` tool.

`/reload` [*options*]
: Restarts the session as follows:

  - Updates the environment settings with the provided options, if any.
  - Resets the execution state.

- Runs the startup scripts.
- Replays the history in the order entered. The history includes all valid snippets or `/drop` commands entered at the `jshell` prompt, in scripts entered on the command line, or scripts entered with the `/open` command.

Environment settings entered on the command line or provided with a previous `/reset`, `/env`, or `/reload` command are maintained unless an *option* is entered that overwrites the setting.

The following options are valid:

`--add-modules` *module*[`,` *module*...]
: Specifies the root modules to resolve in addition to the initial module.

`--add-exports` *source−module*`/`*package*`=`*target−module*[`,` *target−module*]*
: Adds an export of *package* from *source−module* to *target−module*.

`--class-path` *path*
: Specifies the directories and archives that are searched to locate class files. This option overrides the path in the `CLASSPATH` environment variable. If the environment variable isn't set and this option isn't used, then the current directory is searched. For Linux and macOS, use a colon (`:`) to separate items in the path. For Windows, use a semicolon (`;`) to separate items.

`--module-path` *modulepath*
: Specifies where to find application modules. For Linux and macOS, use a colon (`:`) to separate items in the path. For Windows, use a semicolon (`;`) to separate items.

`-quiet`
: Replays the valid history without displaying it. Errors are displayed.

`-restore`
: Resets the environment to the state at the start of the previous run of the tool or to the last time a `/reset`, `/reload`, or `/env` command was executed in the previous run. The valid history since that point is replayed. Use this option to restore a previous JShell session.

`/reset` [*options*]
: Discards all entered snippets and restarts the session as follows:

  - Updates the environment settings with the provided options, if any.
  - Resets the execution state.
  - Runs the startup scripts.

  History is not replayed. All code that was entered is lost.

  Environment settings entered on the command line or provided with a previous `/reset`, `/env`, or `/reload` command are maintained unless an *option* is entered that overwrites the setting.

  The following options are valid:

  `--add-modules` *module*[`,` *module*...]
  : Specifies the root modules to resolve in addition to the initial module.

  `--add-exports` *source−module*`/`*package*`=`*target−module*[`,` *target−module*]*
  : Adds an export of *package* from *source−module* to *target−module*.

  `--class-path` *path*
  : Specifies the directories and archives that are searched to locate class files. This option overrides the path in the `CLASSPATH` environment variable. If the environment variable isn't set and this option isn't used, then the current directory is searched. For Linux and macOS, use a colon (`:`) to separate items in the path. For Windows, use a semicolon (`;`) to separate items.

`--module-path` *modulepath*
: Specifies where to find application modules. For Linux and macOS, use a colon (`:`) to separate items in the path. For Windows, use a semicolon (`;`) to separate items.

`/save` [*options*] *file*
: Saves snippets and commands to the file specified. If no options are entered, then active snippets are saved.

  The following options are valid:

  {*name*|*id*|*startID*−*endID*} [{*name*|*id*|*startID*−*endID*}...]
  : Saves the snippets and commands identified by name, ID, or ID range. For a range of IDs, provide the starting ID and ending ID separated with a hyphen. To provide a list, separate the items in the list with a space. Use the `/list` command to see the IDs of the code snippets.

  `-all` Saves all snippets, including startup snippets and snippets that were overwritten or failed.

  `-history`
  : Saves the sequential history of all commands and snippets entered in the current session.

  `-start`
  : Saves the current startup settings. If no startup scripts were provided, then an empty file is saved.

`/set` [*setting*]
: Sets configuration information, including the external editor, startup settings, and feedback mode. This command is also used to create a custom feedback mode with customized prompt, format, and truncation values. If no setting is entered, then the current setting for the editor, startup settings, and feedback mode are displayed.

  The following values are valid for `setting`:

  `editor` [*options*] [*command*]
  : Sets the command used to start an external editor when the `/edit` command is entered. The command can include command arguments separated by spaces. If no command or options are entered, then the current setting is displayed.

    The following options are valid:

    `-default`
    : Sets the editor to the default editor provided with JShell. This option can't be used if a command for starting an editor is entered.

    `-delete`
    : Sets the editor to the one in effect when the session started. If used with the `-retain` option, then the retained editor setting is deleted and the editor is set to the first of the following environment variables found: `JSHELLEDITOR`, `VISUAL`, or `EDITOR`. If none of the editor environment variables are set, then this option sets the editor to the default editor.

      This option can't be used if a command for starting an editor is entered.

    `-retain`
    : Saves the editor setting across sessions. If no other option or a command is entered, then the current setting is saved.

    `-wait` Prompts the user to indicate when editing is complete. Otherwise control returns to JShell when the editor exits. Use this option if the editor being used exits immediately, for example, when an edit window already exists. This option is valid only when a command for starting an editor is entered.

`feedback` [*mode*]
: Sets the feedback mode used to respond to input. If no mode is entered, then the current mode is displayed.

  The following modes are valid: `concise`, `normal`, `silent`, `verbose`, and any custom mode created with the `/set mode` command.

`format` *mode field* "*format−string*" *selector*
: Sets the format of the feedback provided in response to input. If no mode is entered, then the current formats for all fields for all feedback modes are displayed. If only a mode is entered, then the current formats for that mode are displayed. If only a mode and field are entered, then the current formats for that field are displayed.

  To define a format, the following arguments are required:

  mode Specifies a feedback mode to which the response format is applied. Only custom modes created with the `/set mode` command can be modified.

  field Specifies a context−specific field to which the response format is applied. The fields are described in the online help, which is accessed from JShell using the `/help /set format` command.

  "*format−string*"
  : Specifies the string to use as the response format for the specified field and selector. The structure of the format string is described in the online help, which is accessed from JShell using the `/help /set format` command.

  selector
  : Specifies the context in which the response format is applied. The selectors are described in the online help, which is accessed from JShell using the `/help /set format` command.

`mode` [*mode−name*] [*existing−mode*] [*options*]
: Creates a custom feedback mode with the mode name provided. If no mode name is entered, then the settings for all modes are displayed, which includes the mode, prompt, format, and truncation settings. If the name of an existing mode is provided, then the settings from the existing mode are copied to the mode being created.

  The following options are valid:

  `−command`|`−quiet`
  : Specifies the level of feedback displayed for commands when using the mode. This option is required when creating a feedback mode. Use `−command` to show information and verification feedback for commands. Use `−quiet` to show only essential feedback for commands, such as error messages.

  `−delete`
  : Deletes the named feedback mode for this session. The name of the mode to delete is required. To permanently delete a retained mode, use the `−retain` option with this option. Predefined modes can't be deleted.

  `−retain`
  : Saves the named feedback mode across sessions. The name of the mode to retain is required.

  Configure the new feedback mode using the `/set prompt`, `/set format`, and `/set truncation` commands.

  To start using the new mode, use the `/set feedback` command.

`prompt` *mode* "*prompt−string*" "*continuation−prompt−string*"
: Sets the prompts for input within JShell. If no mode is entered, then the current prompts for all feedback modes are displayed. If only a mode is entered, then the current prompts

for that mode are displayed.

To define a prompt, the following arguments are required:

mode Specifies the feedback mode to which the prompts are applied. Only custom modes created with the `/set mode` command can be modified.

"*prompt−string*"
Specifies the string to use as the prompt for the first line of input.

"*continuation−prompt−string*"
Specifies the string to use as the prompt for the additional input lines needed to complete a snippet.

`start` [`−retain`] [*file* [*file*...]|*option*]
Sets the names of the startup scripts used when the next `/reset`, `/reload`, or `/env` command is entered. If more than one script is entered, then the scripts are run in the order entered. If no scripts or options are entered, then the current startup settings are displayed.

The scripts can be local files or one of the following predefined scripts:

`DEFAULT`
Loads the default entries, which are commonly used as imports.

`JAVASE`
Imports all Java SE packages.

`PRINTING`
Defines `print`, `println`, and `printf` as `jshell` methods for use within the tool.

`TOOLING`
Defines `javac`, `jar`, and other methods for running JDK tools via their command−line interface within the `jshell` tool.

The following options are valid:

`−default`
Sets the startup settings to the default settings.

`−none` Specifies that no startup settings are used.

Use the `−retain` option to save the start setting across sessions.

`truncation` *mode length selector*
Sets the maximum length of a displayed value. If no mode is entered, then the current truncation values for all feedback modes are displayed. If only a mode is entered, then the current truncation values for that mode are displayed.

To define truncation values, the following arguments are required:

mode Specifies the feedback mode to which the truncation value is applied. Only custom modes created with the `/set mode` command can be modified.

length Specifies the unsigned integer to use as the maximum length for the specified selector.

selector
Specifies the context in which the truncation value is applied. The selectors are described in the online help, which is accessed from JShell using the `/help /set truncation` command.

`/types` [*option*]
Displays classes, interfaces, and enums that were entered. If no option is entered, then all current active classes, interfaces, and enums are displayed.

The following options are valid:

{*name*|*id*|*startID*−*endID*} [{*name*|*id*|*startID*−*endID*}...]
: Displays information for classes, interfaces, and enums identified by name, ID, or ID range. For a range of IDs, provide the starting ID and ending ID separated with a hyphen. To provide a list, separate the items in the list with a space. Use the `/list` command to see the IDs of the code snippets.

`-all` Displays information for all classes, interfaces, and enums, including those added when JShell was started, and classes, interfaces, and enums that failed, were overwritten, or were dropped.

`-start`
: Displays information for startup classes, interfaces, and enums that were added when JShell was started.

`/vars` [*option*]
: Displays the name, type, and value of variables that were entered. If no option is entered, then all current active variables are displayed.

The following options are valid:

{*name*|*id*|*startID*−*endID*} [{*name*|*id*|*startID*−*endID*}...]
: Displays information for variables identified by name, ID, or ID range. For a range of IDs, provide the starting ID and ending ID separated with a hyphen. To provide a list, separate the items in the list with a space. Use the `/list` command to see the IDs of the code snippets.

`-all` Displays information for all variables, including those added when JShell was started, and variables that failed, were overwritten, or were dropped.

`-start`
: Displays information for startup variables that were added when JShell was started.

`/?` Same as the `/help` command.

`/!` Reruns the last snippet.

/{*name*|*id*|*startID*−*endID*} [{*name*|*id*|*startID*−*endID*}...]
: Reruns the snippets identified by ID, range of IDs, or name. For a range of IDs, provide the starting ID and ending ID separated with a hyphen. To provide a list, separate the items in the list with a space. The first item in the list must be an ID or ID range. Use the `/list` command to see the IDs of the code snippets.

`/-`*n* Reruns the −*n*th previous snippet. For example, if 15 code snippets were entered, then `/-4` runs the 11th snippet. Commands aren't included in the count.

INPUT SHORTCUTS

The following shortcuts are available for entering commands and snippets in JShell.

Tab completion

<tab> When entering snippets, commands, subcommands, command arguments, or command options, use the Tab key to automatically complete the item. If the item can't be determined from what was entered, then possible options are provided.

When entering a method call, use the Tab key after the method call's opening parenthesis to see the parameters for the method. If the method has more than one signature, then all signatures are displayed. Pressing the Tab key a second time displays the description of the method and the parameters for the first signature. Continue pressing the Tab key for a description of any additional signatures.

Shift+<Tab> V
: After entering a complete expression, use this key sequence to convert the expression to a variable declaration of a type determined by the type of the expression.

Shift+<Tab> M
After entering a complete expression or statement, use this key sequence to convert the expression or statement to a method declaration. If an expression is entered, the return type is based on the type of the expression.

Shift+<Tab> I
When an identifier is entered that can't be resolved, use this key sequence to show possible imports that resolve the identifier based on the content of the specified class path.

Command abbreviations
An abbreviation of a command is accepted if the abbreviation uniquely identifies a command. For example, `/l` is recognized as the `/list` command. However, `/s` isn't a valid abbreviation because it can't be determined if the `/set` or `/save` command is meant. Use `/se` for the `/set` command or `/sa` for the `/save` command.

Abbreviations are also accepted for subcommands, command arguments, and command options. For example, use `/m -a` to display all methods.

History navigation
A history of what was entered is maintained across sessions. Use the up and down arrows to scroll through commands and snippets from the current and past sessions. Use the Ctrl key with the up and down arrows to skip all but the first line of multiline snippets.

History search
Use the Ctrl+R key combination to search the history for the string entered. The prompt changes to show the string and the match. Ctrl+R searches backwards from the current location in the history through earlier entries. Ctrl+S searches forward from the current location in the history though later entries.

INPUT EDITING

The editing capabilities of JShell are similar to that of other common shells. Keyboard keys and key combinations provide line editing shortcuts. The Ctrl key and Meta key are used in key combinations. If your keyboard doesn't have a Meta key, then the Alt key is often mapped to provide Meta key functionality.

Line Editing Shortcuts

Key or Key Combination	Action
Return	Enter the current line.
Left arrow	Move the cursor to the left one character.
Right arrow	Move the cursor to the right one character.
Ctrl+A	Move the cursor to the beginning of the line.
Ctrl+E	Move the cursor to the end of the line.
Meta+B	Move the cursor to the left one word.
Meta+F	Move the cursor to the right one word.
Delete	Delete the character under the cursor.
Backspace	Delete the character before the cursor.
Ctrl+K	Delete the text from the cursor to the end of the line.
Meta+D	Delete the text from the cursor to the end of the word.
Ctrl+W	Delete the text from the cursor to the previous white space.
Ctrl+Y	Paste the most recently deleted text into the line.
Meta+Y	After Ctrl+Y, press to cycle through the previously deleted text.

EXAMPLE OF STARTING AND STOPPING A JSHELL SESSION

JShell is provided with the JDK. To start a session, enter `jshell` on the command line. A welcome message is printed, and a prompt for entering commands and snippets is provided.

```
% jshell
|  Welcome to JShell -- Version 9
|  For an introduction type: /help intro

jshell>
```

To see which snippets were automatically loaded when JShell started, use the `/list -start` command. The default startup snippets are import statements for common packages. The ID for each snippet begins with the letter *s*, which indicates it's a startup snippet.

```
jshell> /list -start

  s1 : import java.io.*;
  s2 : import java.math.*;
  s3 : import java.net.*;
  s4 : import java.nio.file.*;
  s5 : import java.util.*;
  s6 : import java.util.concurrent.*;
  s7 : import java.util.function.*;
  s8 : import java.util.prefs.*;
  s9 : import java.util.regex.*;
 s10 : import java.util.stream.*;

jshell>
```

To end the session, use the `/exit` command.

```
jshell> /exit
|  Goodbye

%
```

EXAMPLE OF ENTERING SNIPPETS

Snippets are Java statements, variable definitions, method definitions, class definitions, import statements, and expressions. Terminating semicolons are automatically added to the end of a completed snippet if they're missing.

The following example shows two variables and a method being defined, and the method being run. Note that a scratch variable is automatically created to hold the result because no variable was provided.

```
jshell> int a=4
a ==> 4

jshell> int b=8
b ==> 8

jshell> int square(int i1) {
   ...> return i1 * i1;
   ...> }
|  created method square(int)

jshell> square(b)
$5 ==> 64
```

EXAMPLE OF CHANGING SNIPPETS

Change the definition of a variable, method, or class by entering it again.

The following examples shows a method being defined and the method run:

```
jshell> String grade(int testScore) {
   ...>     if (testScore >= 90) {
   ...>         return "Pass";
   ...>     }
   ...>     return "Fail";
   ...> }
|  created method grade(int)

jshell> grade(88)
$3 ==> "Fail"
```

To change the method `grade` to allow more students to pass, enter the method definition again and change the pass score to `80`. Use the up arrow key to retrieve the previous entries to avoid having to reenter them and make the change in the `if` statement. The following example shows the new definition and reruns the method to show the new result:

```
jshell> String grade(int testScore) {
   ...>     if (testScore >= 80) {
   ...>         return "Pass";
   ...>     }
   ...>     return "Fail";
   ...> }
|  modified method grade(int)

jshell> grade(88)
$5 ==> "Pass"
```

For snippets that are more than a few lines long, or to make more than a few changes, use the `/edit` command to open the snippet in an editor. After the changes are complete, close the edit window to return control to the JShell session. The following example shows the command and the feedback provided when the edit window is closed. The `/list` command is used to show that the pass score was changed to `85`.

```
jshell> /edit grade
|  modified method grade(int)
jshell> /list grade

   6 : String grade(int testScore) {
           if (testScore >= 85) {
               return "Pass";
           }
           return "Fail";
       }
```

EXAMPLE OF CREATING A CUSTOM FEEDBACK MODE

The feedback mode determines the prompt that's displayed, the feedback messages that are provided as snippets are entered, and the maximum length of a displayed value. Predefined feedback modes are provided. Commands for creating custom feedback modes are also provided.

Use the `/set mode` command to create a new feedback mode. In the following example, the new mode `mymode`, is based on the predefined feedback mode, `normal`, and verifying command feedback is displayed:

```
jshell> /set mode mymode normal -command
|  Created new feedback mode: mymode
```

Because the new mode is based on the `normal` mode, the prompts are the same. The following example shows how to see what prompts are used and then changes the prompts to custom strings. The first string represents the standard JShell prompt. The second string represents the prompt for additional lines in multiline snippets.

```
jshell> /set prompt mymode
|  /set prompt mymode "\njshell> " "   ...> "

jshell> /set prompt mymode "\nprompt$ " "   continue$ "
```

The maximum length of a displayed value is controlled by the truncation setting. Different types of values can have different lengths. The following example sets an overall truncation value of 72, and a truncation value of 500 for variable value expressions:

```
jshell> /set truncation mymode 72

jshell> /set truncation mymode 500 varvalue
```

The feedback displayed after snippets are entered is controlled by the format setting and is based on the type of snippet entered and the action taken for that snippet. In the predefined mode `normal`, the string `created` is displayed when a method is created. The following example shows how to change that string to `defined`:

```
jshell> /set format mymode action "defined" added-primary
```

Use the `/set feedback` command to start using the feedback mode that was just created. The following example shows the custom mode in use:

```
jshell> /set feedback mymode
|  Feedback mode: mymode

prompt$ int square (int num1){
   continue$ return num1*num1;
   continue$ }
|  defined method square(int)

prompt$
```