JShell Java

Ibrahim Patel 8-920-710

**Goals**

The JShell API and tool will provide a way to interactively evaluate declarations, statements, and expressions of the Java programming language within the JShell state. The JShell state includes an evolving code and execution state. To facilitate rapid investigation and coding, statements and expressions need not occur within a method, expressions need not have side-effects, variables need not occur within a class, and methods need not occur within a class or interface.

The jshell tool will be a command-line tool with features to ease interaction including: a history with editing, tab-completion, automatic addition of needed terminal semicolons, and configurable predefined imports and definitions.

**Non-Goals**

A new interactive language is not the goal: All accepted input must match grammar productions in the Java Language Specification (JLS). Further, within an appropriate surrounding context, all accepted input must be valid Java code (JShell will automatically provide that surrounding context -- the "wrapping"). That is, if Xis an input that JShell accepts (as opposed to rejects with error) then there is an Aand B such that AXB is a valid program in the Java programming language.

Out of scope are graphical interfaces, debugger support, and IDE-like functionality.

**Motivation**

Immediate feedback is important when learning a programming language. The number one reason schools cite for moving away from Java as a teaching language is that other languages have a "REPL" and have far lower bars to an initial "Hello, world!" program. A Read-Eval-Print Loop (REPL) is an interactive programming tool which loops, continually reading user input, evaluating the input, and printing the value of the input or a description of the state change the input caused. Scala, Ruby, JavaScript, Haskell, Clojure, and Python all have REPLs and all allow small initial programs. JShell adds REPL functionality to the Java platform.

Exploration of coding options is also important for developers prototyping code or investigating a new API. Interactive evaluation is vastly more efficient in this regard than edit/compile/execute and System.out.println.

Without the ceremony of class Foo { public static void main(String[] args) { ... } }, learning and exploration is streamlined.

**Description**

**Functionality**

The JShell API will provide all of JShell's evaluation functionality. The code fragments that are input to the API are referred to as "snippets". The jshell tool will also use the JShell completion API to determine when input is incomplete (and the user must be prompted for more) or would be complete if a semicolon were added (in which case the tool will append the semicolon). The tool will have a set of commands for query, saving and restoring work, and configuration. Commands will be distinguished from snippets by a leading slash.

**Snippets**

A snippet must correspond to one of the following Java Language Specification (JLS) syntax productions:

* *ImportDeclaration*
* *ClassDeclaration*
* *InterfaceDeclaration*
* *MethodDeclaration*
* *FieldDeclaration*
* *Statement*
* *Primary*

In a top-level declaration, the access modifiers (public, protected, and private), and the modifiers final and static are not allowed and are ignored with a warning; the modifiers default and synchronized are not allowed, and the modifier abstract is allowed only on classes.

The break, continue, and return statements have no appropriate context at the top-level, and are not allowed.

The above modifiers and statements are, of course, allowed within a nested context in which they are valid. For example:

class C { public static final int m() { return 43; } }

Note that a *PackageDeclaration* is not allowed. All JShell code is placed in the transient jshell package.