10-06-2020

Working with Jshell Variables :—

There are two types of variables :—

① Explicit Variables (Declared by Programmer) eg.
```
Int x=50;
String s="Shershah";
```

② Implicit Variables or Scratch variables
Automatically created by jshell to temporarily hold values given by programmers.

eg:— jshell> 10+50 ↵

D:\Java> jshell -v ↵

jshell> 10+20 ↵

You can see the output which tells about implicit variables $1, $2, $3.--- are created by the Compiler.

Suppose, we have $3 to hold the result of 10+20
then jshell> $3+50 ↵
will give 80 and it is assigned to $4

Let:— jshell> int x=50; ↵

jshell> String x="Shershah"; ↵

int x created earlier is overridden by String x and the last declaration is preserved (earlier creation will be destroyed)

Hence, in jshell two variables with the same name is not allowed, but if it is declared then last declaration will be preserved and previous will be replaced by the new one.

jshell> String s1="10"; ↵

jshell> /vars → list of all variables
/var or /va or /v

We can get help about any command by

jshell> /help vars ↵

jshell> /vars ↵ (list of all active variables)

jshell> /vars -all ↵ (list of all active and non-active variables with values

To drop a variable :-

jshell> /drop x ↵ (deletes a variable x)

jshell> /drop $4 $5 ↵ (deletes variables $4 and $5)

Create a list of String

jshell> List<String> fruits = List.of("Mango", "Banana");

jshell> List<String> vegs = List.of("Potato", "tomato", "Brinjal");

jshell> List<String> language = List.of("Java", "C", "C++", "Python"); ↵

jshell> List<List<String>> l = List.of(fruits, vegs, language); ↵

Difference between println/print and printf functions

System.out.println("Hello") ↵ prints "Hello" but did not return anything since its return type is void.	System.out.printf("Hello") ↵ prints "Hello" but returns PrintStream type hence creates scratch variable of PrintStream type

jshell> System.out.println("Hello"); ↵ (void) returns

jshell> System.out.printf("Hello"); ↵ (PrintStream) returns

jshell> methods :— inside jshell

declare methods ; — use editor with jshell to make big methods.

```
jshell> public void m1()
   > {
   > System.out.println("Hello from Jshell");
   > }
```

calling method:—

```
jshell> m1() ↵
jshell> public void wishing(String nm)
        {
System.out.println("Hello" + nm + " Nice to meet")
        }
jshell> wishing("RRSirs") ↵
  Hello RRSirs Nice to meet
jshell> public void m2()
        {
        }
        public void m2(int i)
        {
        }
```

Overloading is also possible

List all methods :—

```
jshell> /methods ↵
```

Help about any command :—

```
jshell> /help methods ↵
```

```
jshell> public void m1(int i) { }
jshell> public int m1(int i) { }
```

Same Signature

The first method m1(int i) is overridden (Replaced) by m1(int i) of int return type

Using any number of arguments in function

```
jshell> public void sum(int... x)
        { int total=0;
          for(int x1 : x)
          {
          total = total + x1;
          }
        System.out.println("The Sum:" + total);
        }
```

In Jshell, we can use un-declared variable inside method :-

```
jshell> public void m4()
   ...> {
   ...> System.out.println(x);
   ...> }
```

It cannot be invoked until variable x is declared.

```
jshell> int x=50;
jshell> m4() ↵
      50
```

```
jshell> public void m5()
   ...> {
   ...>    m6();
   ...> }
```

declared without m6()

→ After declaring m6() only we will be able to call m5() method.

```
jshell> public void m6()
   ...> {
   ...> System.out.println("This is m6 method called from m5");
   ...> }
```

| created method m6()
| update modified method m5() } message printed as out put

```
jshell> m5(); ↵
   This is m6 method called from m5
```

jshell> /drop command is used to delete a method

```
jshell> /methods ↵
jshell>  /drop wish ↵   will delete wish() method.
jshell>  /drop m2 ↵   will give ambiguity error if
                      more than two methods with different
                      signature are available.
```

Snippet ID (sid)

Jshell> /drop 4 ↵ by using method Id we can delete whatever is required.

Working with external editors :—

D:\Java> jshell -v ↵ [It has in-built editor also.]

jshell> /edit ↵ → To execute Jshell inbuilt-editor. with three options [Cancel] [Accept] [Exit].

jshell> /list ↵ → Shows the method which is accepted by in-built editor.

jshell> methodname() ↵ methodname() executed

In-built editor provided by oracle corporation is not so good because there is no any syntax error indicator. Jshell allows us to make our own editor for the jshell. The command is used for this purpose is —

jshell> /set editor "C:\\Windows\\....\notepad.exe" ↵

(available only for current session)

Complete path of editor being used.

eg:- jshell> /set editor "C:\\windows\\system32\\notepad.exe" ↵

jshell> /edit ↵ Notepad will be opened

Once we exited from jshell then again jshell inbuilt editor will invoked.

Now, we want to set editor permanently then we have to do—

jshell> /set editor "C:\\windows\\system32\\notepad.exe" ↵

jshell> /set editor -retain ↵ Remember this

Now Notepad is permanently our own default editor for Jshell.

Jshell> /set editor -default ↵

Jshell> /set editor -retain ↵ } setting again default editor as permanent editor of Jshell.

Jshell> /set editor "C:\\Program Files(x86)\\Editplus\\editplus.exe" ↵

Jshell> /edit ↵

Jshell> /set editor -retain ↵

/list ↵ command displays the functions with body.

IDE's are made for developing software applications. So, better to use IDE's if you are expert. Jshell is available for beginners only or testing purpose for developers. IDE's are not recommended to use as editor to Jshell.

— END —

Thats all today.

In next class we will use classes, interfaces and enums in Jshell.

jshell> /types shows the available types of class, interfaces and enums

jshell> /list → shows the class body, interface body or enum body.