# Introduction to Nashorn

## Agenda

1. Nashorn Engine
2. Features of Nashorn
3. Invoking Nashorn from the Command Line
4. Evaluating a simple JS statement from Java code
5. Evaluating a script file from Java code
6. Invoking a Script Function from Java code
7. Invoking a Script Function which returns a value
8. Using Java in Scripts

# Nashorn Engine

## Nashorn Engine

- **Nashorn JavaScript Engine** is introduced in Java 8.
- It is used to interpret the JavaScript code in a Java application or from the command line.
- It is an implementation of the ECMAScript Edition 5.1 Language Specification.
- It was fully developed in **Java language by Oracle**.
- It is included in the Java SE Development Kit (JDK) version 8 or above.

## Features of Nashorn

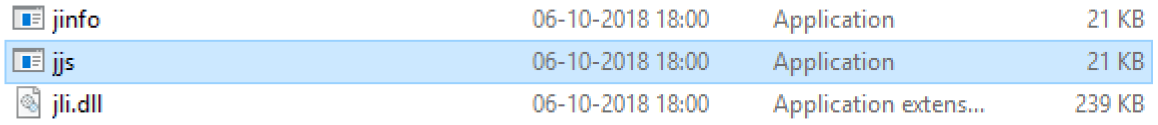
## Features of Nashorn

- It provides **interoperability** between the Java and JavaScript worlds. That means your Java code can call JavaScript code, and vice versa.
- It provides global objects to **access and instantiate Java classes** from JavaScript. Their members can be accessed using the familiar '**.**' notation as in Java.
- All the Java built-in functions can be accessed from JavaScript code and vice versa.
- Java collection classes are supported and interpreted as arrays.
- It comes with a number of small extensions to make it easier to use JavaScript for shell scripting.
- We can write **JavaFX** applications entirely in JavaScript using Nashorn.

## Invoking Nashorn from the Command Line

## Invoking Nashorn from the Command Line

- **jjs** is the recommended command line tool, created specifically for invoking the Nashorn engine.
- We can find this application in the **bin** folder of the JDK(8 or above) installation directory.

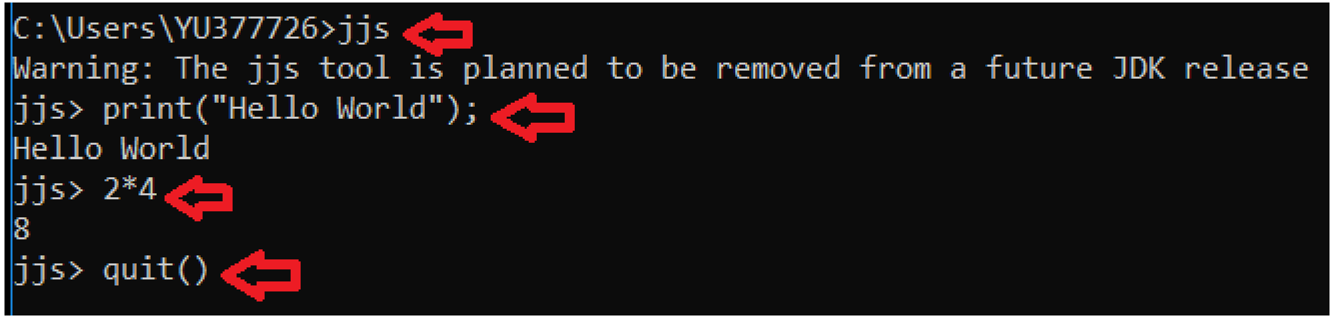


```
jinfo     06-10-2018 18:00    Application              21 KB
jjs       06-10-2018 18:00    Application              21 KB
jli.dll   06-10-2018 18:00    Application extens...   239 KB
```

# Invoking Nashorn from the Command Line contd..

## jjs in Interactive Mode

- Open command prompt and issue **jjs** command to enter into the interactive mode.
- Execute the below statements one by one.

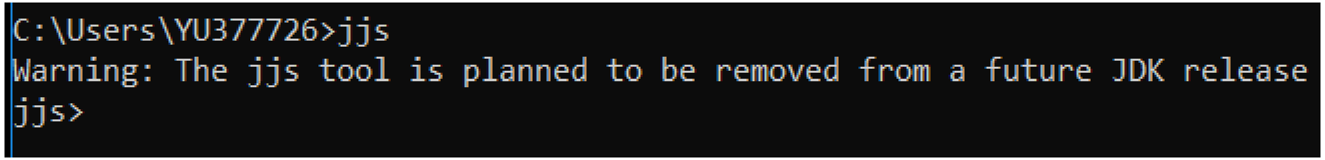


```
C:\Users\YU377726>jjs
Warning: The jjs tool is planned to be removed from a future JDK release
jjs> print("Hello World");
Hello World
jjs> 2*4
8
jjs> quit()
```

- Use **quit( )** function to exit the interactive mode.

## Invoking Nashorn from the Command Line contd..

**Why do we get this warning?**

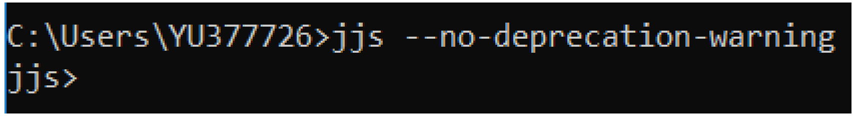


```
C:\Users\YU377726>jjs
Warning: The jjs tool is planned to be removed from a future JDK release
jjs>
```

- With the release of Java 11, Nashorn is deprecated, and will likely be removed from the JDK at a later time. The GraalVM was suggested as a replacement.

**Is it possible to ignore this warning?**

- Yes. It can be ignored by adding `--no-deprecation-warning` to the jjs command.

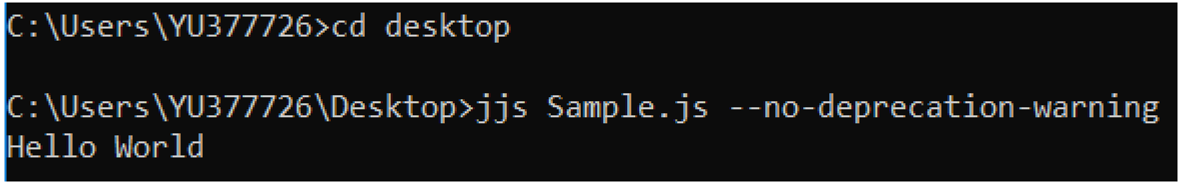


```
C:\Users\YU377726>jjs --no-deprecation-warning
jjs>
```

## Invoking Nashorn from the Command Line contd..

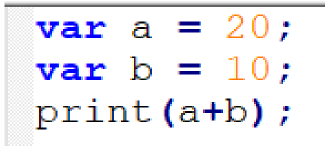
### Interpreting js File

- Create a **Sample.js** file which contains **print("Hello World");**
- Navigate to the location where this file is present and run the script with **jjs Sample.js**

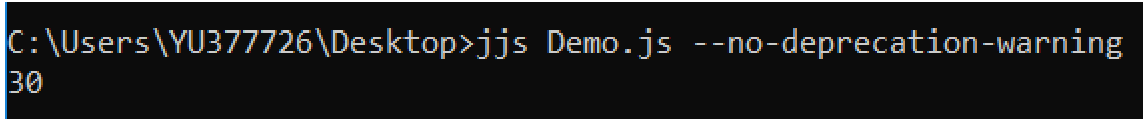


```
C:\Users\YU377726>cd desktop

C:\Users\YU377726\Desktop>jjs Sample.js --no-deprecation-warning
Hello World
```

- Create a **Demo.js** file with the below content and run the script.



```
var a = 20;
var b = 10;
print(a+b);
```



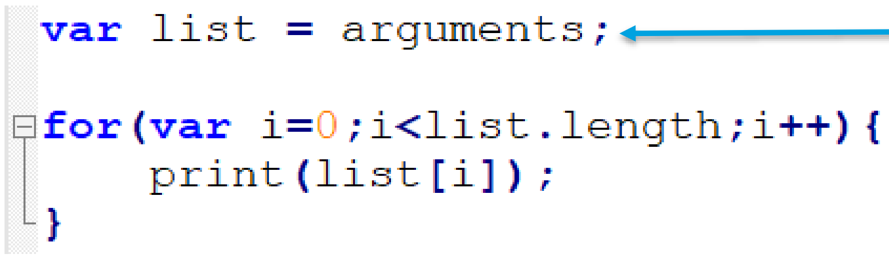
```
C:\Users\YU377726\Desktop>jjs Demo.js --no-deprecation-warning
30
```

## Invoking Nashorn from the Command Line contd..

### Passing Arguments

- We can pass arguments as inputs to our script.

**Demo.js**



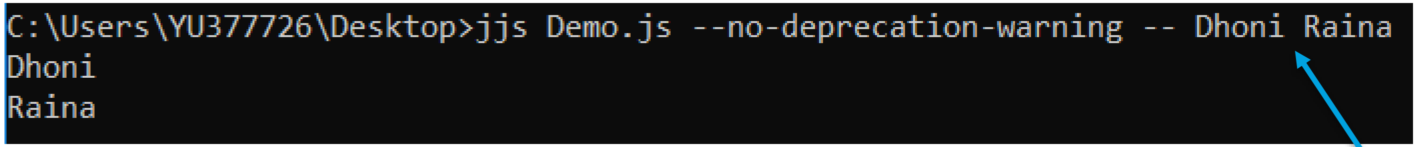
```
var list = arguments;

for(var i=0;i<list.length;i++){
    print(list[i]);
}
```

**arguments** is an Array-like object that contains the values of the arguments passed to that script.

## Invoking Nashorn from the Command Line contd..

**Passing Arguments continued..**

**Output**



```
C:\Users\YU377726\Desktop>jjs Demo.js --no-deprecation-warning -- Dhoni Raina
Dhoni
Raina
```

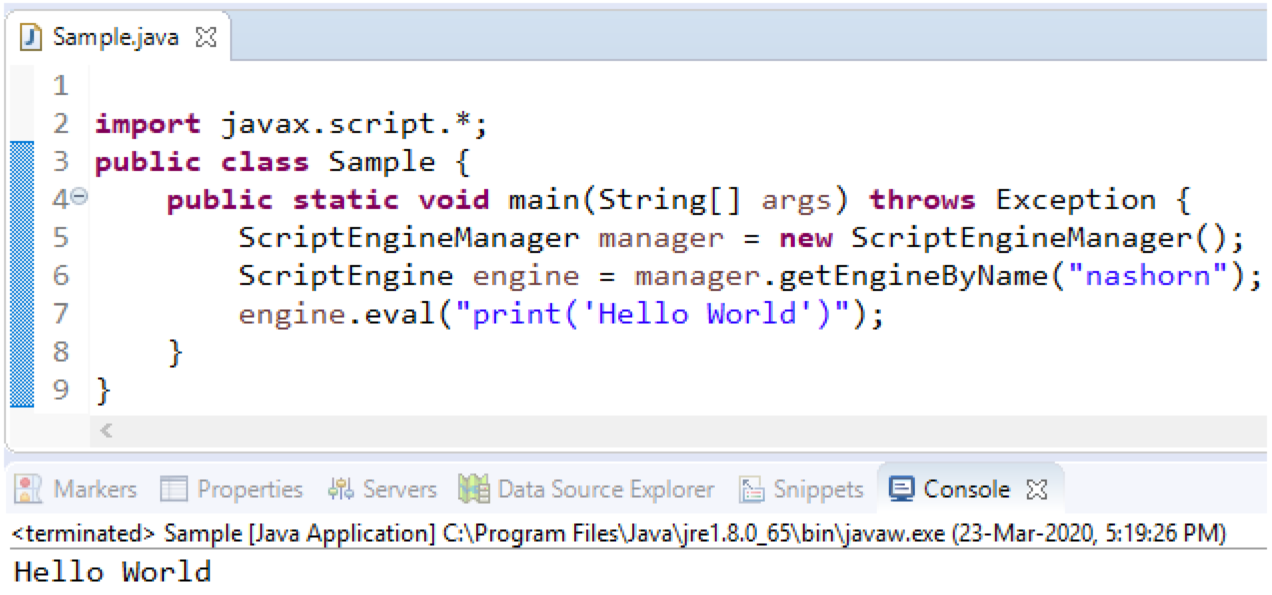
arguments passed to the script.

- Arguments are passed through the jjs command: `jjs file-name -- arguments`
- All values after the **double hyphen** marker (--) are passed to the script as arguments.
- These values can be accessed by using the **arguments** property.

## Evaluating a simple JS statement from Java code

## Evaluating a simple JS statement from Java code

- The Java Scripting API is composed of classes and interfaces present in this **javax.script** package.



Sample.java

```java

import javax.script.*;
public class Sample {
    public static void main(String[] args) throws Exception {
        ScriptEngineManager manager = new ScriptEngineManager();
        ScriptEngine engine = manager.getEngineByName("nashorn");
        engine.eval("print('Hello World')");
    }
}
```

```
<terminated> Sample [Java Application] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (23-Mar-2020, 5:19:26 PM)
Hello World
```

## Evaluating a simple JS statement from Java code contd..

**Example explained:**

1. Importing the java package which contains the Java Scripting API.

```
import javax.script.*;
```

2. The **getEngineByName**(String name) method of **ScriptEngineManager** class looks up and creates a **ScriptEngine** instance for the given name.

```
ScriptEngineManager manager = new ScriptEngineManager();
ScriptEngine engine = manager.getEngineByName("nashorn");
```

3. The **eval**(String script) method of **ScriptEngine** class executes the specified script.

```
engine.eval("print('Hello World')");
```

## Evaluating a simple JS statement from Java code contd..

4. The **eval** method throws **ScriptException** if the specified script has any errors.

```
public static void main(String[] args) throws Exception
```

**Example for ScriptException:**

```
ScriptEngineManager manager = new ScriptEngineManager();
ScriptEngine engine = manager.getEngineByName("nashorn");
int x = 10, y = 20;
engine.eval("print(x+y)");
```

**Reason:** x and y are Java variables, they cannot be accessed in the JavaScript statement.

**Fix:** `engine.eval("print("+(x+y)+")");`

**Output:** 30

## Predict the output

**Program: 1**

```
ScriptEngineManager manager = new ScriptEngineManager();

ScriptEngine engine = manager.getEngineByName("nashorn");

String name = "Arun";

engine.eval("print('Welcome " + name + "')");
```

**Program: 2**

```
ScriptEngineManager manager = new ScriptEngineManager();

ScriptEngine engine = manager.getEngineByName("nashorn");

engine.eval("print(Math.pow(4, 3))");
```

## Predict the output

**Program: 3**

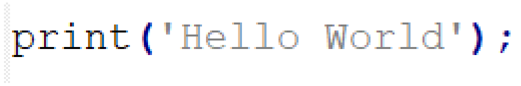
```
ScriptEngineManager manager = new ScriptEngineManager();

ScriptEngine engine = manager.getEngineByName("nashorn");

int sum = 0;

sum = (int) engine.eval("10 + 30");

System.out.println(sum);
```

## Evaluating a script file from Java code

## Evaluating a script file from Java code

- In this example, the **eval( )** method takes in a **FileReader** object that reads JavaScript code from the file Sample.js.
- By wrapping various input stream objects as readers, it is possible to execute scripts from files, URLs, and other resources.

**Sample.js**



```
print('Hello World');
```

## Evaluating a script file from Java code contd..

**Java code**

```
ScriptEngineManager manager = new ScriptEngineManager();

ScriptEngine engine = manager.getEngineByName("nashorn");

FileReader fr = new FileReader("C:\\Users\\Desktop\\Sample.js");

engine.eval(fr);
```

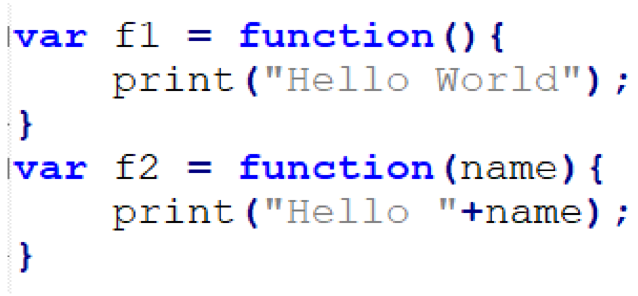
**Output**

```
Hello World
```

## Invoking a Script Function from Java code

## Invoking a Script Function from Java code

**Sample.js**



```
var f1 = function(){
    print("Hello World");
}
var f2 = function(name){
    print("Hello "+name);
}
```

**Java code**

```
ScriptEngineManager manager = new ScriptEngineManager();

ScriptEngine engine = manager.getEngineByName("nashorn");

FileReader fr = new FileReader("C:\\Users\\Desktop\\Sample.js");

engine.eval(fr);     //continued..
```

## Invoking a Script Function from Java code contd..

// **Invocable:** The optional interface implemented by ScriptEngines whose methods allow the invocation of procedures in scripts that have previously been executed.

```
Invocable invocable = (Invocable)engine;
```

// **invokeFunction( ):** Used to call the functions defined in scripts.

```
invocable.invokeFunction("f1");

invocable.invokeFunction("f2","Arun");
```

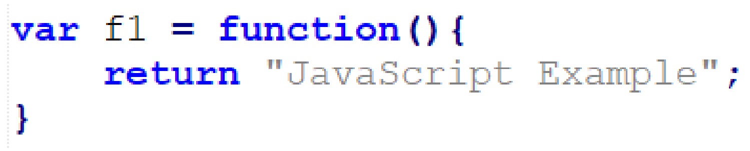
**Output**

```
Hello World
Hello Arun
```

## Invoking a Script Function which returns a value

## Invoking a Script Function which returns a value

**Sample.js**



```javascript
var f1 = function(){
    return "JavaScript Example";
}
```

**Java code**

```java
ScriptEngineManager manager = new ScriptEngineManager();

ScriptEngine engine = manager.getEngineByName("nashorn");

FileReader fr = new FileReader("C:\\Users\\Desktop\\Sample.js");

engine.eval(fr);        //continued..
```

## Invoking a Script Function which returns a value contd..

```
Invocable invocable = (Invocable)engine;

String msg = invocable.invokeFunction("f1").toString();

System.out.println(msg);
```

**Output**

```
JavaScript Example
```

# Using Java in Scripts

## Using Java in Scripts

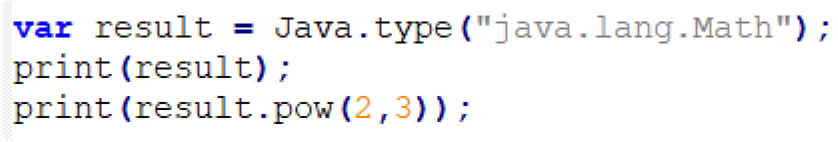
- So far we have discussed how to run JS code and files from a Java program. Now let’s understand how to use Java classes and arrays in JavaScript code.

**Using Java predefined classes and their functions**

- Nashorn interprets Java classes as JavaClass function objects.
- Call the **Java.type( )** function, which returns a type object that corresponds to the full name of the class passed to it as a string.

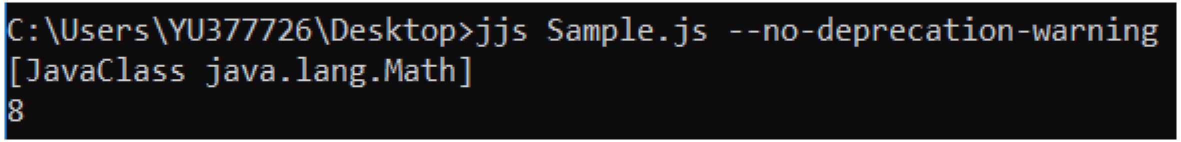
## Using Java in Scripts contd..

**Example: 1**



```
var result = Java.type("java.lang.Math");
print(result);
print(result.pow(2,3));
```

**Output**

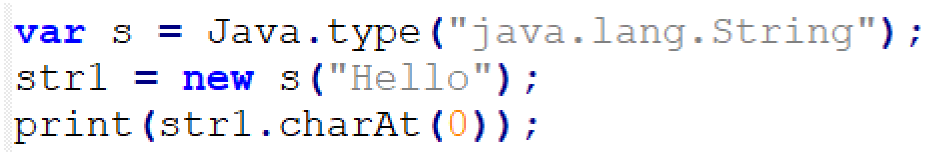


```
C:\Users\YU377726\Desktop>jjs Sample.js --no-deprecation-warning
[JavaClass java.lang.Math]
8
```

## Using Java in Scripts contd..

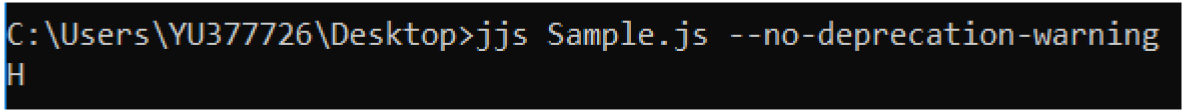
- JavaClass function objects can be used as constructors for creating objects.

**Example: 2**



```
var s = Java.type("java.lang.String");
str1 = new s("Hello");
print(str1.charAt(0));
```

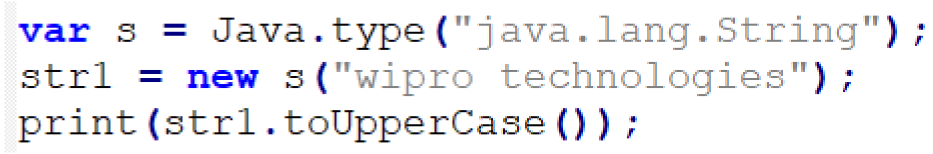
**Output**



```
C:\Users\YU377726\Desktop>jjs Sample.js --no-deprecation-warning
H
```

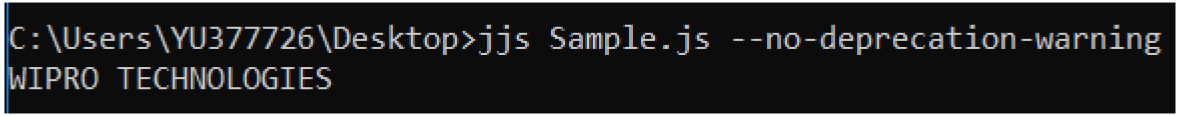
## Using Java in Scripts contd..

**Example: 3**



```
var s = Java.type("java.lang.String");
str1 = new s("wipro technologies");
print(str1.toUpperCase());
```

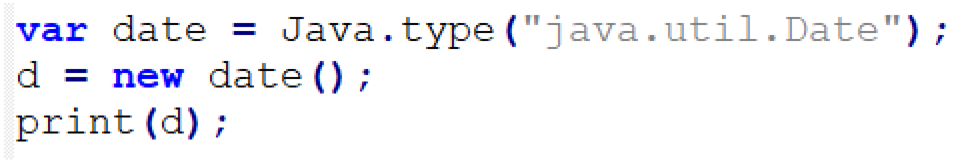
**Output**



```
C:\Users\YU377726\Desktop>jjs Sample.js --no-deprecation-warning
WIPRO TECHNOLOGIES
```

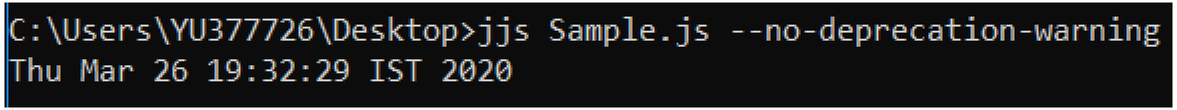
## Using Java in Scripts contd..

**Example: 4**



```
var date = Java.type("java.util.Date");
d = new date();
print(d);
```

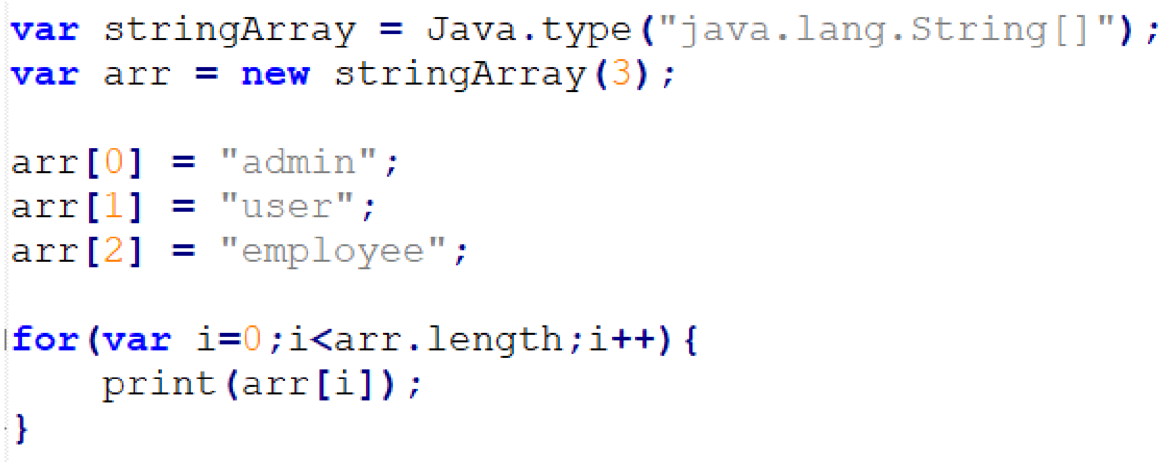
**Output**



```
C:\Users\YU377726\Desktop>jjs Sample.js --no-deprecation-warning
Thu Mar 26 19:32:29 IST 2020
```

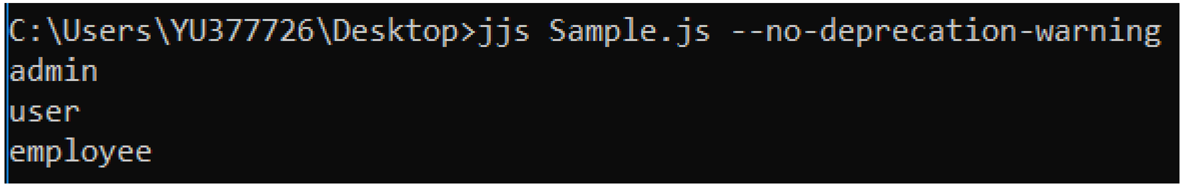
## Using Java in Scripts contd..

**Example: 5**



```
var stringArray = Java.type("java.lang.String[]");
var arr = new stringArray(3);

arr[0] = "admin";
arr[1] = "user";
arr[2] = "employee";

for(var i=0;i<arr.length;i++){
    print(arr[i]);
}
```

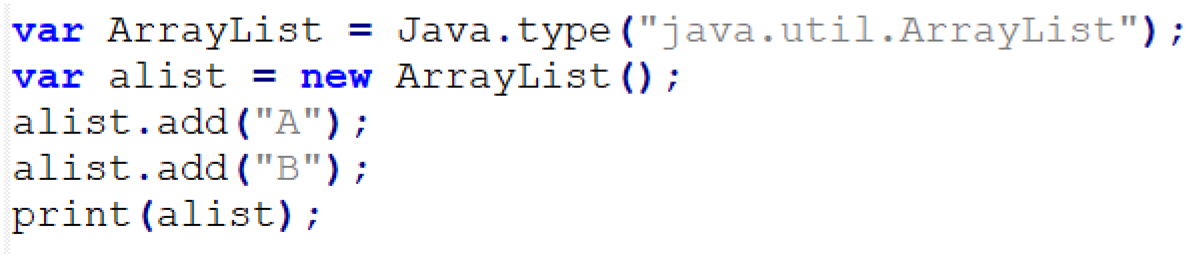
**Output**



```
C:\Users\YU377726\Desktop>jjs Sample.js --no-deprecation-warning
admin
user
employee
```

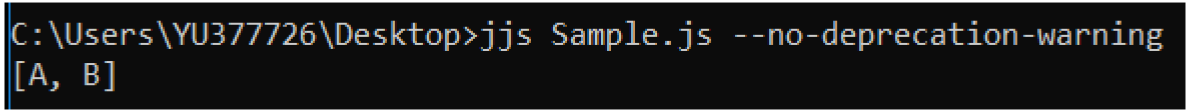
## Using Java in Scripts contd..

**Example: 6**



```
var ArrayList = Java.type("java.util.ArrayList");
var alist = new ArrayList();
alist.add("A");
alist.add("B");
print(alist);
```

**Output**



```
C:\Users\YU377726\Desktop>jjs Sample.js --no-deprecation-warning
[A, B]
```

## Thank you