Session: 18

Additional Features of Java 8

Objectives

- Explain the Nashorn Engine
- Describe the jjs tool and its use for scripting
- Explain the new mathematical functions in Java 8

Introduction 1/2

- ✓ Nashorn is a German term which means Rhinoceros.
- ✓ In Java, it refers to the newly included JavaScript engine.
- ✓ Java Nashorn is the replacement for existing JavaScript engine in versions up to Java 7 called the Rhino.
- ✓ Nashorn and Rhino implement JavaScript language for JVM.
- ✓ Slower functioning of Rhino caused the need for a new engine, Nashorn.
- ✓ JavaScript on JVM is friendly with Nashorn that helps in faster implementation of Java language.

Introduction 2/2

Main Goal of Nashorn:

- ✓ Provide a lightweight and high-performance JavaScript runtime in Java.

Nashorn:

- ✓ Compiles JavaScript into Java bytecode.
- ✓ Uses `InvokeDynamic` API of JVM specification that makes it faster than its predecessor.
- ✓ Also has a command line tool called jjs.
- ✓ Helps developers embed JavaScript in applications and also invoke Java methods and classes from JavaScript code.
- ✓ Helps developers extend functionality of applications and make them more versatile.

Primary Goals of Nashorn

- ✓ Should pass ECMAScript-262 compliance tests
- ✓ Will be based on ECMAScript-262 Edition 5.1 language specification
- ✓ Applications should perform better than Rhino and memory usage capabilities should also be better than Rhino
- ✓ Mutual support must be provided for Java and JavaScript code access between both the languages
- ✓ A new command line tool called `jjs` should validate JavaScript codes
- ✓ Should have support for JSR 223 (`javax.script` API)
- ✓ Must be safe from additional security risks
- ✓ Libraries provided with Nashorn should behave correctly under localization
- ✓ Error messages and documentation should be mapped to meet international standards

The `jjs` Tool 1/7

The `jjs` is a new command line tool for running JavaScript through Nashorn.

The bin folder of JDK 8 is bundled with newly included `jjs` tool, which is used to launch the Nashron command line interpreter.

The jjs Tool 2/7

Following Code Snippet shows how to execute a JavaScript program using Nashorn.

```
$jjs newfile.js
```

Following Code Snippet shows how to launch the jjs interpreter in interactive mode.

```
C:\>jjs
jjs> print ("hi everyone this is
Nashornjjs")
Hi everyone this is Nashornjjs
jjs>
```

To exit from the interactive mode of jjs, just type quit().

The jjs Tool 3/7

Consider an example where you want to quickly find the sum of some numbers.
Enter the code shown in Code Snippet and save it as test.js.

```
var data = [1,3,5,7,11]
var sum = data.reduce(function(x, y){ return x + y},0)
print(sum)
```

Then, give `jjs test.js` at the command prompt.
The output of the code when executed with `jjs` tool is 27.

Structure of the jjs Tool

```
jjs [options] [script-files] [-- arguments]
```

where,

options
One or more options of the `jjs` command that are separated by empty spaces controls the conditions in which scripts are interpreted by Nashorn.

Script files
The script files that are separated by spaces can be interpreted using Nashorn. In case there is no script file assigned, an interactive shell will be initiated.

arguments
Argument values are passed with a double hyphen prefix `(--)`. Accessing the values can be done with `arguments` property.

The jjs Tool 5/7

Code Snippet shows how to invoke Nashorn in interactive mode and assign `sampleValue` to the property named `sampleKey` and then retrieve it using `getProperty()`.

```
jjs -D sampleKey=sampleValue

jjs>java.lang.System.getProperty("sampleKey")

sampleValue

jjs>
```

The code can be repeated with appropriate modifications to set multiple properties.

Setting multiple properties

doe:dump-on-error
This option provides a full stack trace when an error occurs. By default, only a brief error message is displayed and when this option is used, one can obtain a detailed message.

fv:fullversion
This option displays the full Nashorn version string.

fx:
This option launches the script as a JavaFX application.

h: help
This option displays the `options` list with appropriate descriptions.

t=zone or timezone=zone
This option provides a particular time zone for script execution. It overrides the default time zone of the system OS.

v:version
Displays the Nashorn version string.

The jjs Tool 7/7

- ✓ Scripting is one of the options in `jjs`.
- ✓ Running `Scripting` opens up an interactive shell that allows user to type and evaluate the JavaScript in JVM.
- ✓ By using this option, user can also pass variables as strings.

```
var studentname = 'James';
print("Hello, ${studentname}!");
```

The result displays `Hello James`.

ScriptEngine 1/5

- ✓ The `ScriptEngine` interface and `ScriptEngineManager` class are used to run JavaScript in Java.
- ✓ These are defined in public API `javax.script` and must be imported into your application.
- ✓ When Oracle `Nashorn` is available, it can be accessed using the identifier, `nashorn`.
- ✓ Code Snippet here shows the process of importing `ScriptEngine` in a Java program.
- ✓ `ScriptException` class is imported to handle any script-related exceptions.

```
import javax.script.ScriptEngine;//to use ScriptEngine
import javax.script.ScriptEngineManager;//to use ScriptEngineManager
import javax.script.ScriptException; // to handle exceptions
```

ScriptEngine 2/5

`ScriptEngineManager` is used to initiate Nashorn engine

Following Code Snippet shows how to use `ScriptEngineManager` in Java

```
ScriptEngineManager newEngManager = new ScriptEngineManager();
ScriptEngine newEng = newEngManager.getEngineByName("nashorn");
```

ScriptEngine 3/5

- After the import of `ScriptEngine` and `ScriptEngineManager`, evaluation of JavaScript can be done anytime.
- Code Snippet shows the evaluation of JavaScript.
- Note that it is mandatory to enclose the code in a try-catch block that will handle the `ScriptException`, failing which, there will be compiler errors.

```
try{
    newEng.eval("function f(g) { print(g) }");//evaluation of JavaScript
    newEng.eval("f(' Hi this is through JavaScript being executed from within
    Java');"); //displays result
}catch(ScriptException e){…}
```

ScriptEngine 4/5

In the code shown earlier:

- A JavaScript function `f` is defined that takes one argument and prints its value.
- This function is then called in the next statement and a string argument is passed.
- As a result, when the code is executed, the string is displayed in the output.

A `FileReader` can also be passed as an input in the evaluation method `(eval)`.

Following Code Snippet demonstrates this. Ensure that appropriate exception handling is done while using this or similar code dealing with files.

```
newEng.eval(new FileReader('newfile.js'));//as input
```

ScriptEngine 5/5

In this code, `newEngManager` is an instance of `ScriptEngineManager`. It will be used to create an instance of `ScriptEngine` using the `getEngineByName()` method. Through `newEng`, the JavaScript `eval` method is applied to evaluate the variables `x`, `y`, and `z`.

```
import javax.script.ScriptEngineManager;//to use ScriptEngineManager
import javax.script.ScriptEngine;//to use ScriptEngine
import javax.script.*;
public class ScriptEngineUsage {
 public static void main(String args[]) throws ScriptException
 {
 ScriptEngineManager newEngManager = new ScriptEngineManager();
 ScriptEngine newEng = newEngManager.getEngineByName("javascript");
 newEng.eval("var x = 10;");
 newEng.eval("var y = 20;");
 newEng.eval("var z = x + y;");
 newEng.eval("print (z);");//displays result
 }
}
```

Output:

30

Importing Classes and Packages 1/2

Code Snippet shows importing of `java.util` and other packages. This code needs to be saved as a `.js` file and then, executed with `jjs` tool.

```
var imports = new JavaImporter(java.util,java.io,java.nio.file);
with(imports){//importing packages
    var samplePaths = new LinkedList();
    print(samplePaths instanceof LinkedList);//true
    samplePaths.add("newDoc1");
    samplePaths.add("newDoc2");
    samplePaths.add("newDoc3");
    print(samplePaths)// [newDoc1, newDoc2, newDoc3]
}//displays result
```

In this code:

1. Variable `samplePaths` is created as an instance of `LinkedList`, and using `add()` method, a list of nodes is created and displayed.

2. Code assumes there are three files, `newDoc1`, `newDoc2`, and `newDoc3` created in current folder.

Importing Classes and Packages 2/2

Following snippet shows the usage of `.write` method:

```
for(var x=0; x<samplePaths.size();x++)
FileSystems.getDefault().getPath(samplePaths.get(x))
) .write("test\n".getBytes())
```

Note that Nashorn allows importing existing Java classes, but creating new classes is also possible with this new JavaScript engine.

Extending Classes and Interfaces

`Java.type` and `Java.extend` functions are used to extend Java classes.

Following Code Snippet shows an example with `Callable` interface and `call` method implementation. This code needs to be saved as a `.js` file and executed with `jjs`.

```
var newConcur = new JavaImporter(java.util,java.util.concurrent);
//extending using Java.type
var newCall = Java.type("java.util.concurrent.Callable");
with(newConcur){
     var newExec=Executors.newCachedThreadPool();
     var newTasks=new LinkedHashSet();
     for(var x=1;x<200;x++){
     var SampleTask=Java.extend(newCall,{ call:function()
     { print("Result displayed as "+x)}})
     var newTaskA = new SampleTask();
     newTasks.add(newTaskA);
     newExec.submit(newTaskA);
    }
}//displays result
```

Here, `newConcur` is defined as a `JavaImporter` instance. `newCall` will represent the `Callable` type. A new `LinkedHashSet` is created and sub tasks are added to it. Using a cached thread pool, a list of numbers are then displayed as output.

Invocable Interface 1/2

An Invocable Interface initiation helps in invoking JavaScript directly from Java. For this, Java objects are passed as function arguments and the resultant data can be returned back to the Java method.

```
Invocable tryInvoke =(Invocable)engine;//to invoke engine
```

Invocable Interface 2/2

`invokeFunction()` method can be applied to invoke any user-defined function.

Code Snippet shows `invokeFunction()` method.

```
engine.eval("function f(g) { print(g) }");
tryInvoke.invokeFunction("f,"HI ");// invoke function usage
```

Mathematical Methods in Java 1/7

Basic mathematical operations such as logarithms, exponential square root, numeric calculations, and trigonometric methods are made easier with `java.lang` package.

Basic Math Methods:

- `Math.abs ()`
- `Math.ceil ()`
- `Math.floor ()`

Mathematical Methods in Java 2/7

`Math.abs ()`

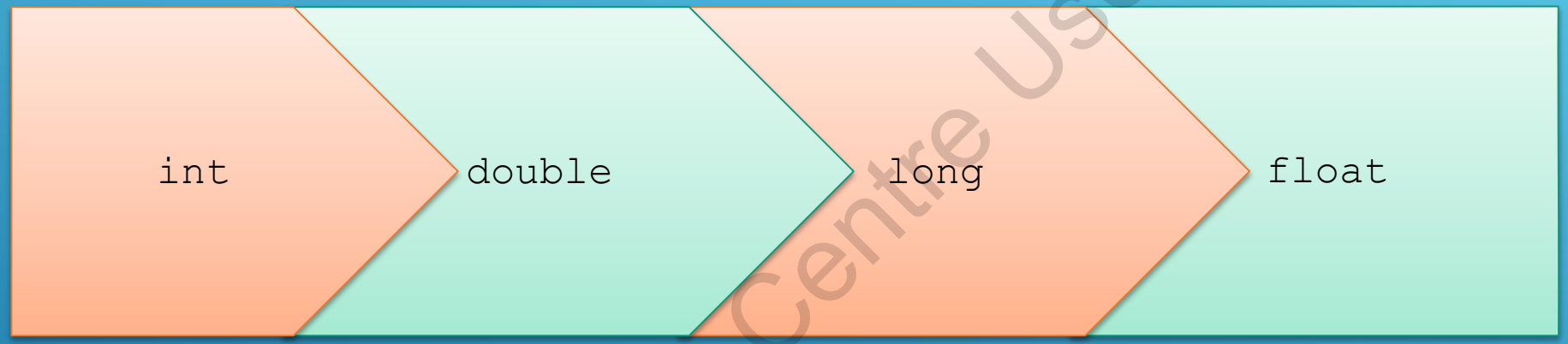
Returns only positive values even if given input contains negative values.

```
public class BasicMathDemo {
   public static void main(String args[]) {
      int abs1 = Math.abs(10); // abs1 = 10
      int abs2 = Math.abs(-20); // abs2 = 20
      System.out.println("Result A: " +abs1);
      System.out.println("Result B: " +abs2);
    }
}//displays result
```

Here, the code produces output as the same value as the input. However, the negative value (`-20`) is returned as positive value (`20`).

Mathematical Methods in Java 3/7

`Math.abs()` method can be overloaded by using one of these:



Usage of the overloaded methods may depend on the respective parameters and operations performed.

Mathematical Methods in Java 4/7

`Math.ceil ()`

Is used to round up a floating-point value into integer value.

```
public class BasicMathDemo {
    public static void main(String args[]) {
       double objCeil = Math.ceil(6.454);
       System.out.println("Result as " + objCeil);
    }
}//displays result
```

After executing this code, `objCeil` contains value 7.0.

Mathematical Methods in Java 5/7

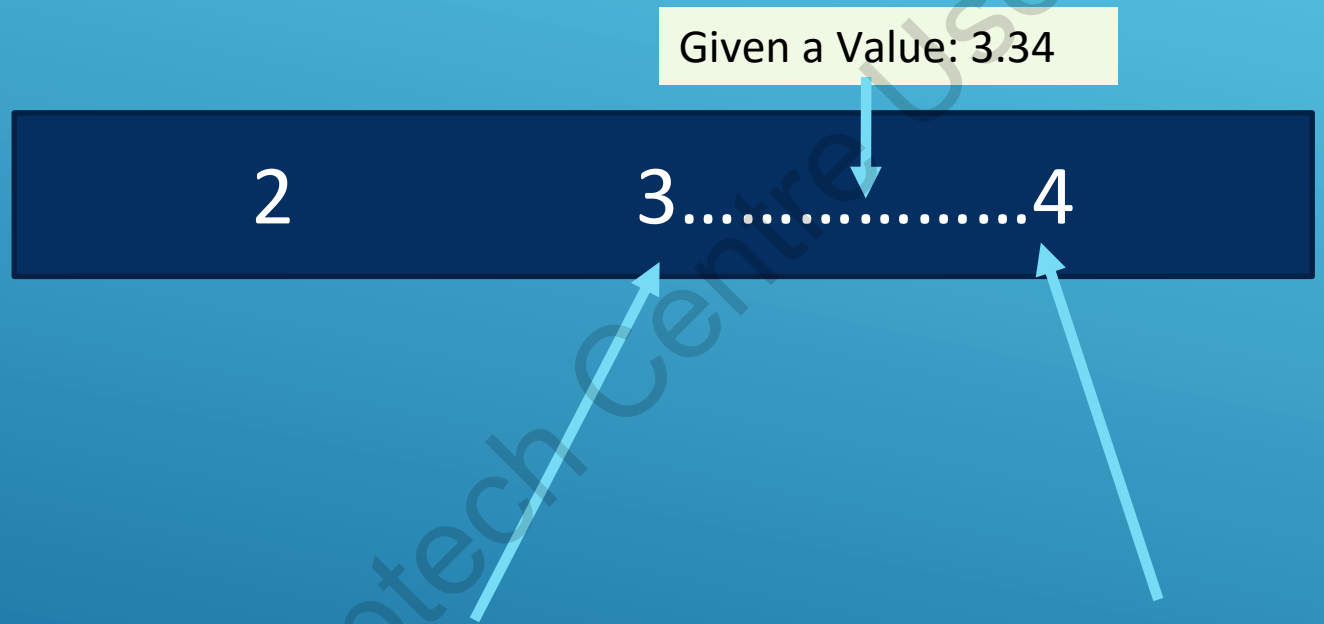
`Math.floor ()`

- This method rounds down a floating-point value down to the closest integer value.
- The rounded value is displayed as a double.
- Example given here demonstrates the `Math.floor()` method.

```
public class BasicMathFunctons {
public static void main(String args[]) {
double objFloor = Math.floor(6.454);
// will result in objFloor = 6.0
System.out.println("Result as " + objFloor);
 }
}//displays result
```

After executing this code, `objFloor` contains value 6.0.

`floor()` versus `ceil()`



Mathematical Methods in Java 7/7

Table lists a few more basic Math methods:

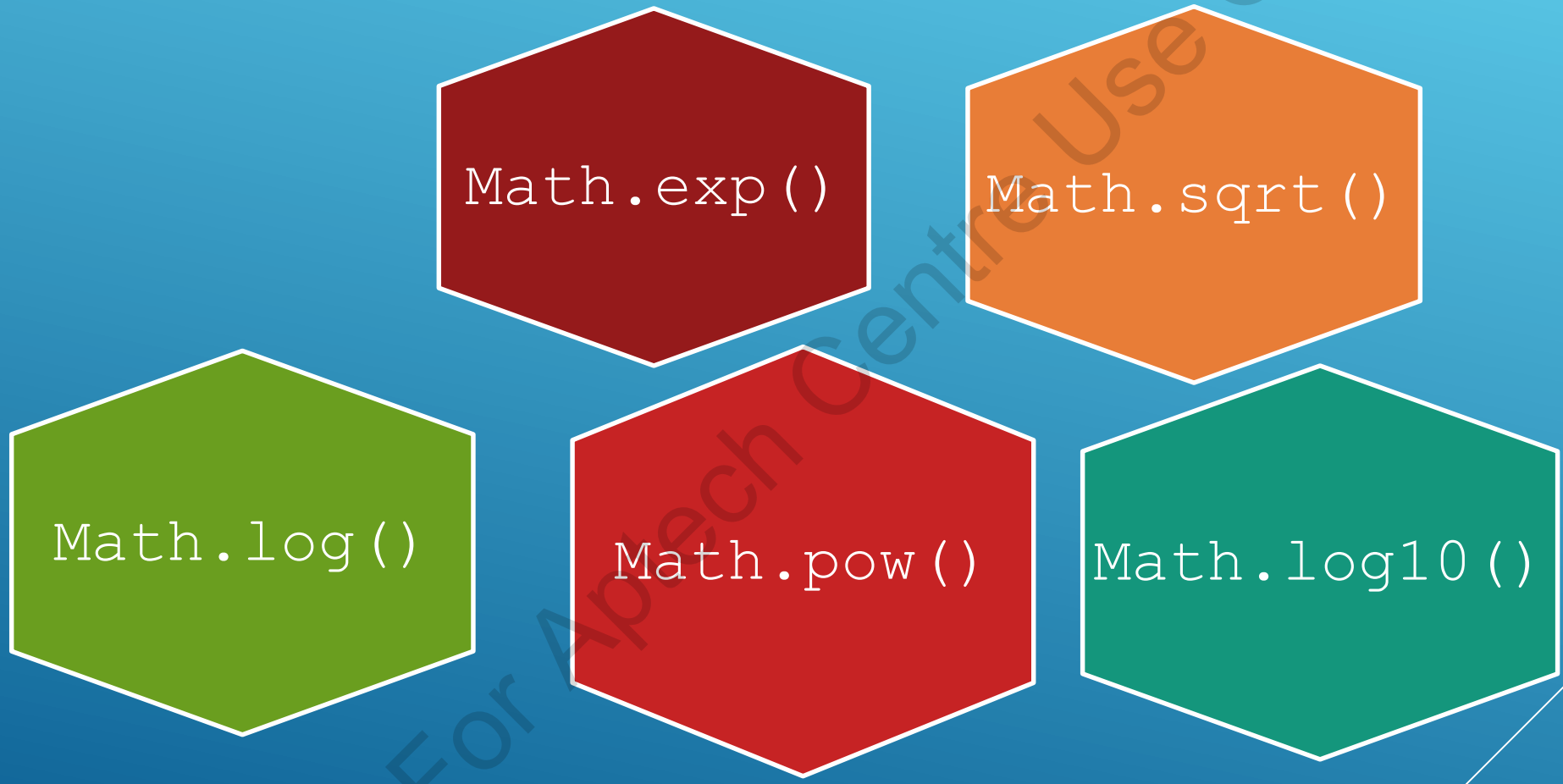
Method Name	Description	Example
`Math.floorDiv()`	Similar to `floor()`, but is combined with division. `Math.floorDiv()` divides an integer or long value by another one, and rounds resultant value to closest integer value.	`double output = Math.floorDiv(100,9);` Output: 11.0
`Math.min()`	Produces the smallest of given values passed as inputs.	`int newMin = Math.min(5,12);` Here, `newMin` variable produces output as 5
`Math.max()`	Similar to `Math.min()` with one difference that it produces the biggest value of the given inputs.	`int newMax = Math.max(5, 12);` Here, `newMax` variable produces output as 12.

`Math.round()`	Rounds a `float` or `double` to the closest integer.
	
	`Math.round()` method applies common mathematical rules for rounding the values.
		`double newDecrease=Math.round(44.324);`
		`double newIncrease =Math.round(44.654);`
		
		Here, `newDecrease` variable produces output as 44.0 and `newIncrease` variable produces 45 as output.

`Math.random()`	Returns a random floating point value between 0 and 1 by default. However, `Math.random` can be applied to get a random number between 0 and n (any number within 100).	`double newRand1 = Math.random();`
		`double newRand2 = Math.random() * 50D;`
		
		Output:
		`newRand1`:0.7463562032119188
		`newRand2`:26.360465065790073

Exponential and Logarithmic Math Methods 1/5

`Math` class contains methods for exponential and logarithmic calculations such as:



Exponential and Logarithmic Math Methods 2/5

`Math.exp()` produces e (Euler's number) increased to the power of the value given as a parameter.

Following example shows `Math.exp()` in use.

```
public class ExpoandLogMathFunctons {
   public static void main(String args[]) {
     double newExpA = Math.exp(4);
     System.out.println("OutputA = " + newExpA);
     double newExpB = Math.exp(5);
     System.out.println("OutputB = " + newExpB);
    }
}//displays result
```

Here, `newExpA` and `newExpB` variables produce the following output:

```
OutputA = 54.598150033144236
OutputB = 148.4131591025766
```

Exponential and Logarithmic Math Methods 3/5

Logarithm can be obtained using `Math.log()`. It also uses Euler's number e as the base. This method also performs the reverse function of `Math.exp()`.

Following example demonstrates `Math.log()` method.

```
public class ExpoandLogMathFunctons {
public static void main(String args[]) {
double newLogA = Math.log(2);
System.out.println("OutputA = " + newLogA);
double newLogB = Math.log(100);
System.out.println("OutputB = " + newLogB);
}
}//displays result
```

Here, `newLogA` and `newLogB` are variables that produce following result:

```
OutputA = 0.6931471805599453
OutputB = 4.605170185988092s
```

Exponential and Logarithmic Math Methods 4/5

- `Math.pow()` method takes two parameters and produces the value of the first parameter raised to the power of the second parameter.
- Following example demonstrates the `Math.pow()` method.

```
public class ExpoandLogMathFunctions {
public static void main(String args[]){
double newPowerA = Math.pow(2, 4);
System.out.println("OutputA as = " + newPowerA);
double newPowerB = Math.pow(2, 5);
System.out.println("OutputB as = " + newPowerB);
}
}
```

Here, `newPowerA` and `newPowerB` produce following results:

```
OutputA as = 16.0
OutputB as = 32.0
```

Exponential and Logarithmic Math Methods 5/5

- `Math.sqrt()`
- This method performs the square root operation for the given parameter.
- Example here demonstrates `Math.sqrt()` method.

```
public class ExpoandLogMathFunctions {
  public static void main(String args[]){
    double newSrootA = Math.sqrt(8);
    System.out.println("OutputA = " + newSrootA);
    double newSrootB = Math.sqrt(25);
    System.out.println("OutputB = " + newSrootB);
  }
}
```

Here, the `newSrootA` and `newSrootB` produce the following results:

```
OutputA = 2.828427124746l903
OutputB = 5.0
```

Trigonometric Math Methods 1/11

- The `Math` class also includes a set of trigonometric methods that can calculate values used in trigonometry such as sine, cosine, tan, and so on.
- `Math.PI` is a constant double that contains a value closest to PI.
- `Math.PI` is frequently used in trigonometric operations/calculations.

Trigonometric Math Methods 2/11

Method Name	Description	Example

`Math.sin()`	Performs the sine operation. It calculates the sine value of the given angle value in radians.	`double newSin = Math.sin(Math.PI);`
		
		`System.out.println("The value of sin = " + newSin);`
		
		Output:
		`The value of sin = 1.2246467991473532E-16`

Trigonometric Math Methods 3/11

Method Name	Description	Example

Math.cos()	Performs the cos operation. It calculates the cosine value of the given angle value in radians.	`double newTan = Math.tan(Math.PI);`
		
		`System.out.println("The value of tan = " + newTan);`
		
		Output:
		`The value of tan = 1.2246467991473532E-16`

Trigonometric Math Methods 4/11

Method Name	Description	Example

`Math.asin()`	Performs the arc sine value calculation of a value between `1` and `-1`.	`double newAsin = Math.asin(Math.PI);`
		
		`System.out.println("The value of Asin = " + newAsin);`
		
		Output:
		`The value of Asin = NaN`

Trigonometric Math Methods 5/11

Method Name	Description	Example

`Math.acos()`	Performs the arc cos value calculation of a value between `1` and `-1`.	`double newAcos = Math.acos(1.0);`
		
		`System.out.println("The value of acos = " + newAcos);`
		
		Output:
		`The value of acos = 0.0`

Method Name	Description	Example

Math.atan()	Performs the arc tangent value calculation of a value between 1 and -1.	`double newAtan = Math.atan(1.0);`
		
		`System.out.println("The value of Atan = " +newAtan);`
		
		Output:
		`The value of Atan = 0.7853981633974483`

Trigonometric Math Methods 7/11

Method Name	Description	Example

`Math.sinh()`	Performs the hyperbolic sine value calculation of a value between `1 and -1`.	`double newSinh = Math.sinh(1.0);`
		
		`System.out.println("The value of sinh = " + newSinh);`
		
		Output:
		`The value of sinh = 1.1752011936438014`

Trigonometric Math Methods 8/11

Method Name	Description	Example

Math.cosh()	Performs the hyperbolic cosine value calculation of a given value between 1 and -1.	`double newCosh = Math.cosh(1.0);`
		
		`System.out.println("The value of Cosh = " + newCosh);`
		
		Output:
		`The value of Cosh = 1.543080634815244`

Trigonometric Math Methods 9/11

Method Name	Description	Example

`Math.tanh()`	Performs the hyperbolic tangent value calculation of a given value between `1` and `-1`.	`double newTanh = Math.tanh(1.0);`
		
		`System.out.println("The value of tanh = " + newTanh);`
		
		Output:
		`The value of tanh = 0.7615941559557649`

Trigonometric Math Methods 10/11

Method Name	Description	Example

`Math.toDegrees()`	Performs the convert operation of an angle in radians to degrees.	`double newDegrees = Math.toDegrees(Math.PI);`
		
		`System.out.println("Output = " + newDegrees);`
		
		Output = 180.0

Trigonometric Math Methods 11/11

Method Name	Description	Example

`Math.toRadians()`	Performs an reverse operation of `Math.toDegrees()` method; it performs the convert operation of an angle in degrees to radians.	`double newRadians = Math.toRadians(180);`
		
		`System.out.println("Out put = " + newRadians);`
		
		Output = 3.141592653589793

Exact Numeric Operations 1/2

Following methods are used to perform exact numeric operations:

- `addExact`
- `subtractExact`
- `multiplyExact`
- `incrementExact`
- `decrementExact`
- `negateExact`
- `toIntExact`

Exact Numeric Operations 2/2

Example given here demonstrates the usage of `addExact()` method.

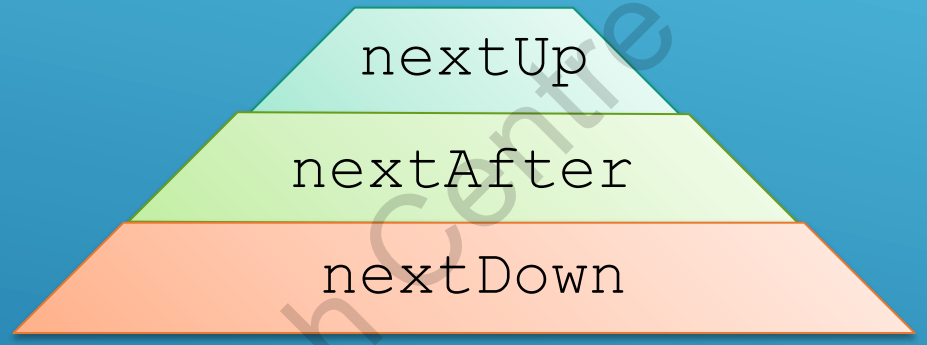
```
public class ExactMethodDemo {
  public static void main(String args[]) {
     int ex1 = 900000000; //
     int ex2 = 1250000000; //
     System.out.println(Math.addExact( ex1 , ex2 ));
  }
}//displays result
```

Ideally, if `ex1` and `ex2` were added using the + sign, the program will not show any errors instead, it will produce an inaccurate result as it exceeds the maximum limit for integers.

Hence, here using `addExact()` is recommended so that it would throw an exception and alert the user instead of displaying an incorrect output.

Next Numeric Operations 1/2

Methods that perform numeric operations where there is a need to display the closest value of a given number:



Next Numeric Operations 2/2

Example given here demonstrates the `nextDown()` method.

```
public class ExactMethodDemo {
   public static void main(String args[]) {
     int nextA = 1000; //
     System.out.println(Math.nextDown( nextA));
    }
}//displays result
```

Output:

```
999.99994
```

Summary

- Nashorn and Rhino implements a JavaScript engine to enable its use with JVM. Slower functioning of Rhino caused the need for Nashorn
- jjs is a command line tool to launch Nashorn
- Various jjs command options control the conditions in which scripts are interpreted by Nashorn
- Nashorn enables JavaScript functions to be invoked directly from Java. Also, Java objects can be passed as function arguments, and the resultant data can be returned back to the Java method
- Advanced mathematical operations can be performed with new methods in Math class