

# 25.7 Exploring a Class's Members and Viewing Documentation

[*Note:* This section may be read after studying [Chapter 6, Methods: A Deeper Look](#), and the preceding portions of [Chapter 25](#).]

The preceding section introduced basic auto-completion capabilities. When using JShell for experimentation and discovery, you'll often want to learn more about a class before using it. In this section, we'll show you how to:

- view the parameters required by a method so that you can call it correctly
- view the documentation for a method
- view the documentation for a field of a class
- view the documentation for a class, and
- view the list of overloads for a given method.

To demonstrate these features, let's explore class `Math`. Start a new JShell session or `/reset` the current one.

## 25.7.1 Listing Class `Math`'s

# static Members

As we discussed in [Chapter 6](#), class `Math` contains only **static** members—**static** methods for various mathematical calculations and the **static** constants `PI` and `E`. To view a complete list, type "`Math.`" then press *Tab*:

<code>jshell&gt; Math.</code>			
<code>E</code>	<code>IEEEremainder(</code>	<code>PI</code>	<code>abs(</code>
<code>acos(</code>	<code>addExact(</code>	<code>asin(</code>	<code>atan(</code>
<code>atan2(</code>	<code>cbrt(</code>	<code>ceil(</code>	<code>class</code>
<code>copySign(</code>	<code>cos(</code>	<code>cosh(</code>	<code>decrementExact(</code>
<code>exp(</code>	<code>expm1(</code>	<code>floor(</code>	<code>floorDiv(</code>
<code>floorMod(</code>	<code>fma(</code>	<code>getExponent(</code>	<code>hypot(</code>
<code>incrementExact(</code>	<code>log(</code>	<code>log10(</code>	<code>log1p(</code>
<code>max(</code>	<code>min(</code>	<code>multiplyExact(</code>	<code>multiplyFull(</code>
<code>multiplyHigh(</code>	<code>negateExact(</code>	<code>nextAfter(</code>	<code>nextDown(</code>
<code>nextUp(</code>	<code>pow(</code>	<code>random()</code>	<code>rint(</code>
<code>round(</code>	<code>scalb(</code>	<code>signum(</code>	<code>sin(</code>
<code>sinh(</code>	<code>sqrt(</code>	<code>subtractExact(</code>	<code>tan(</code>
<code>tanh(</code>	<code>toDegrees(</code>	<code>toIntExact(</code>	<code>toRadians(</code>
<code>ulp(</code>			
<code>jshell&gt; Math.</code>			

As you know, JShell auto-completion displays a list of

everything that can appear to the right of the dot ( . ). Here we typed a class name and a dot ( . ), so JShell shows only the class's **static** members. The names that are not followed by any parentheses (E and PI) are the class's **static** variables. All the other names are the class's **static** methods:

- Any method names followed by ( )—only `random` in this case—do not require any arguments.
- Any method names followed by only an opening left parenthesis, ( , require at least one argument or are overloaded.

You can easily view the value of the constants PI and E:

```
jshell> Math.PI  
$1 ==> 3.141592653589793
```

```
jshell> Math.E  
$2 ==> 2.718281828459045
```

```
jshell>
```

## 25.7.2 Viewing a Method's Parameters

Let's assume you wish to test `Math`'s `pow` method (introduced in [Section 5.4.2](#)), but you do not know the parameters it requires. You can type

---

```
Math.p
```



then press *Tab* to auto-complete the name `pow`:

---

```
jshell> Math.pow(
```



Since there are no other methods that begin with "pow", JShell also inserts the left parenthesis to indicate the beginning of a method call. Next, you can type *Tab* to view the method's parameters:

---

```
jshell> Math.pow(  
double Math.pow(double a, double b)  
  
<press tab again to see documentation>  
  
jshell> Math.pow(
```



JShell displays the method's return type, name and complete parameter list followed by the next `jshell>` prompt containing what you've typed so far. As you can see, the method requires two `double` parameters.

## 25.7.3 Viewing a Method's Documentation

JShell integrates the Java API documentation so you can view documentation conveniently in JShell, rather than requiring

you to use a separate web browser. Suppose you'd like to learn more about `pow` before completing your code snippet. You can press *Tab* again to view the method's Java documentation (known as its javadoc)—we cut out some of the documentation text and replaced it with a vertical ellipsis (...) to save space (try the steps in your own JShell session to see the complete text):

---

```
jshell> Math.pow(  
double Math.pow(double a, double b)  
Returns the value of the first argument raised to the  
second argument.Special cases:  
* If the second argument is positive or negative ze  
result is 1.0.  
...  
<press tab again to see next page>
```



For long documentation, JShell displays part of it, then shows the message

---

```
<press tab again to see next page>
```



You can press *Tab* to view the next page of documentation. The next `jshell>` prompt shows the portion of the snippet you've typed so far:

---

```
jshell> Math.pow(  
...)
```

## 25.7.4 Viewing a public Field's Documentation

You can use the *Tab* feature to learn more about a class's **public** fields. For example, if you enter `Math.PI` followed by *Tab*, JShell displays

```
jshell> Math.PI
PI

Signatures:
Math.PI:double

<press tab again to see documentation>
```

which shows `Math.PI`'s type and indicates that you can use *Tab* again to view the documentation. Doing so displays:

```
jshell> Math.PI
Math.PI:double
The double value that is closer than any other to pi,
the circumference of a circle to its diameter.
```

```
jshell> Math.PI
```

and the next `jshell>` prompt shows the portion of the snippet you've typed so far.

## 25.7.5 Viewing a Class's Documentation

You also can type a class name then *Tab* to view the class's fully qualified name. For example, typing `Math` then *Tab* shows:

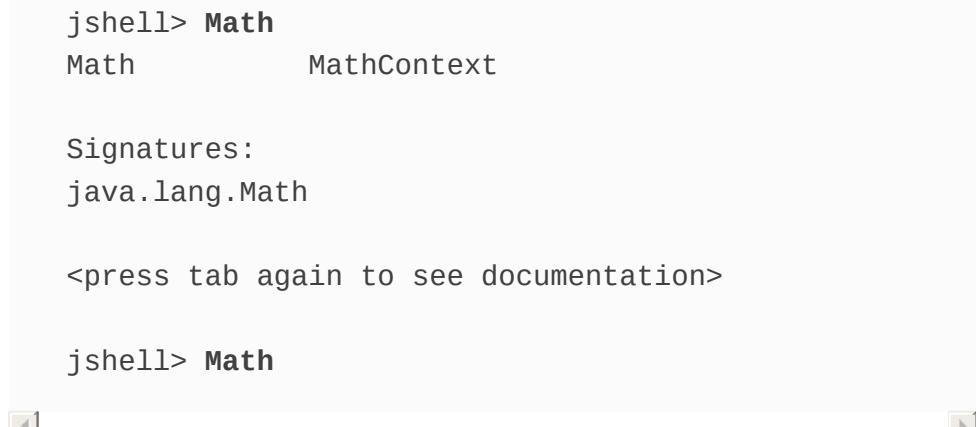
---

```
jshell> Math
Math           MathContext

Signatures:
java.lang.Math

<press tab again to see documentation>

jshell> Math
```

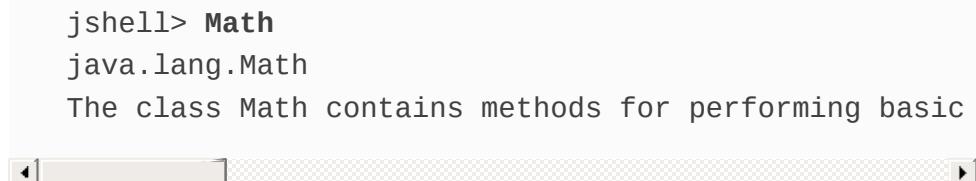


indicating that class `Math` is in the package `java.lang`.

Typing *Tab* again shows the beginning of the class's documentation:

---

```
jshell> Math
java.lang.Math
The class Math contains methods for performing basic
```



In this case, there is more documentation to view, so you can press *Tab* to view it. Whether or not you view the remaining documentation, the `jshell>` prompt shows the portion of the snippet you've typed so far:

---

```
jshell> Math
```



## 25.7.6 Viewing Method Overloads

Many classes have *overloaded* methods. When you press *Tab* to view an overloaded method's parameters, JShell displays the complete list of overloads, showing the parameters for every overload. For example, method `Math.abs` has four overloads:

---

```
jshell> Math.abs(  
$1 $2
```

Signatures:

```
int Math.abs(int a)  
long Math.abs(long a)  
float Math.abs(float a)  
double Math.abs(double a)
```

<press tab again to see documentation>

```
jshell> Math.abs(  
$1 $2
```



When you press *Tab* again to view the documentation, JShell shows you the *first* overload's documentation:

---

```
jshell> Math.abs(  
int Math.abs(int a)
```

Returns the absolute value of an int value. If the argument is negative, the argument is returned. If the argument is positive, the negation of the argument is returned.

...

<press tab again to see next page>



You can then press *Tab* to view the documentation for the next overload in the list. Again, whether or not you view the remaining documentation, the `jshell>` prompt shows the portion of the snippet you've typed so far.

## 25.7.7 Exploring Members of a Specific Object

The exploration features shown in [Sections 25.7.1–25.7.6](#) also apply to the members of a specific object. Let's create and explore a `String` object:

---

```
jshell> String dayName = "Monday"
dayName ==> "Monday"
```

```
jshell>
```



To view the methods you can call on the `dayName` object, type `"dayName . "` and press *Tab*:

```
jshell> dayName .
```

```
charAt(
```

```
chars()
```

```
codePointAt(
```

codePointBefore()	codePointCount()	codePoints()
compareTo()	compareToIgnoreCase()	concat()
contains()	contentEquals()	endsWith()
equals()	equalsIgnoreCase()	getBytes()
getChars()	getClass()	hashCode()
indexOf()	intern()	isEmpty()
lastIndexOf()	length()	matches()
notify()	notifyAll()	offsetByCodePoints()
regionMatches()	replace()	replaceAll()
replaceFirst()	split()	startsWith()
subSequence()	substring()	toCharArray()
toLowerCase()	toString()	toUpperCase()
trim()	wait()	
jshell> <b>dayName.</b>		

## Exploring `toUpperCase`

Let's investigate the `toUpperCase` method. Continue by typing "toU" and pressing *Tab* to auto-complete its name:

```
jshell> dayName.toUpperCase()
toUpperCase()
```

```
jshell> dayName.toUpperCase()
```





Then, type *Tab* to view its parameters:

---

```
jshell> dayName.toUpperCase()  
Signatures:  
String String.toUpperCase(Locale locale)  
String String.toUpperCase()
```

<press tab again to see documentation>

```
jshell> dayName.toUpperCase()
```



This method has two overloads. You can now use *Tab* to read about each overload, or simply choose the one you wish to use, by specifying the appropriate arguments (if any). In this case, we'll use the no-argument version to create a new **String** containing **MONDAY**, so we simply enter the closing right parenthesis of the method call and press *Enter*:

---

```
jshell> dayName.toUpperCase()  
$2 ==> "MONDAY"  
  
jshell>
```

## Exploring `substring`

Let's assume you want to create the new **String** **"DAY"**—a subset of the implicit variable **\$2**'s characters. For this purpose class **String** provides the overloaded method

`substring`. First type `"$2.subs"` and press *Tab* to auto-complete its the method's name:

---

```
jshell> $2.substring()  
substring()
```

```
jshell>
```

Next, use *Tab* to view the method's overloads:

---

```
jshell> $2.substring()  
Signatures:  
String String.substring(int beginIndex)  
String String.substring(int beginIndex, int endIndex)
```

```
<press tab again to see documentation>
```

```
jshell> $2.substring()
```

Next, use *Tab* again to view the first overload's documentation:

---

```
jshell> $2.substring()  
String String.substring(int beginIndex)  
Returns a string that is a substring of this string. It  
begins with the character at the specified index and  
ends at the end of this string.  
...  
<press tab again to see next page>
```

As you can see from the documentation, this overload of the

method enables you to obtain a substring starting from a specific character index (that is, position) and continuing through the end of the **String**. The first character in the **String** is at index 0. This is the version of the method we wish to use to obtain "DAY" from "MONDAY", so we can return to our code snippet at the `jshell>` prompt:

---

```
jshell> $2.substring()
```

A screenshot of a terminal window. The text "jshell> \$2.substring()" is visible on the screen. The window has scroll bars on the right and bottom.

Finally, we can complete our call to `substring` and press *Enter* to view the results:

---

```
jshell> $2.substring(3)  
$3 ==> "DAY"
```

```
jshell>
```

A screenshot of a terminal window. The text "jshell> \$2.substring(3)" is followed by the output "\$3 ==> \"DAY\"". The window has scroll bars on the right and bottom.