Quick Guide to java.lang.System

Last modified: April 12, 2018

by baeldung (http://www.baeldung.com/author/baeldung/)

Java (http://www.baeldung.com/category/java/) +

1. Overview

In this tutorial, we'll take a quick look at the *java.lang.System* class and its features and core functionality.

2. IO

System is a part of *java.lang*, and one of its main features is to give us access to the standard I/O streams.

Simply put, it exposes three fields, one for each stream:

- *out*
- *err*
- *in*

2.1. *System.out*

System.out points to the standard output stream, exposing it as a *PrintStream*, and we can use it to print text to the console:

```
System.out.print("some inline message");
```

An advanced usage of *System* is to call *System.setOut*, which we can use to customize the location to which *System.out* will write:

```
// Redirect to a text file
System.setOut(new PrintStream("filename.txt"));
```

2.2. *System.err*

System.err is a lot like *System.out*. Both fields are instances of *PrintStream,* and both are for printing messages to the console.

But *System.err* represents standard error and we use that specifically to output error messages:

```
System.err.print("some inline error message");
```

Consoles will often render the error stream differently than the output stream.

For more information, check the *PrintStream* (https://docs.oracle.com/javase/8/docs/api/java/io/PrintStream.html) documentation.

2.3. *System.in*

System.in points to the standard in, exposing it as an *InputStream,* and we can use it for reading input from the console.

And while a bit more involved, we can still manage:

```
public String readUsername(int length) throws IOException {
    byte[] name = new byte[length];
    System.in.read(name, 0, length); // by default, from the console
    return new String(name);
}
```

By calling *System.in.read*, the application stops and waits for input from the standard in. Whatever the next *length* bytes are will be read from the stream and stored in the byte array.

Anything else typed by the user stays in the stream, waiting for another call to *read.*

Of course, operating at that low of a level can be challenging and error-prone, so we can clean it up a bit with *BufferedReader*.

```
public String readUsername() throws IOException {
    BufferedReader reader = new BufferedReader(
      new InputStreamReader(System.in));
    return reader.readLine();
}
```

With the above arrangement, *readLine* will read from *System.in* until the user hits return, which is a bit closer to what we might expect.

Note that we purposely don't close the stream in this case. **Closing the standard *in* means that it cannot be read again for the lifecycle of the program!**

And finally, an advanced usage of *System.in* is to call *System.setIn* to redirect it to a different *InputStream*.

3. Utility Methods

System provides us with numerous methods to help us with things like:

- Accessing the console
- Copying arrays

- Observing date and time
- Exiting the JRE
- Accessing runtime properties
- Accessing environment variables, and
- Administering garbage collection

3.1. Accessing the Console

Java 1.6 introduced another way of interacting with the console than simply using *System.out* and *in* directly.

We can access it by calling *System.console*:

```
public String readUsername() {
    Console console = System.console();

    return console == null ? null :
      console.readLine("%s", "Enter your name: ");
}
```

Note that depending upon the underlying operating system and how we launch Java to run the current program, ***console* might return *null*, so always make sure to check before using**.

Check out the Console (https://docs.oracle.com/javase/7/docs/api/java/io/Console.html) documentation for more uses.

3.2. Copying Arrays

System.arraycopy is an old C-style way of copying one array into another.

Mostly, *arraycopy* is intended to copy one complete array into another array:

```
int[] a = {34, 22, 44, 2, 55, 3};
int[] b = new int[a.length];

System.arraycopy(a, 0, b, 0, a.length);
assertArrayEquals(a, b);
```

However, we can specify the starting position for both arrays, as well as how many elements to copy.

For example, let's say we want to copy 2 elements from *a*, starting at *a[1]* to *b*, starting at *b[3]*:

```
System.arraycopy(a, 1, b, 3, 2);
assertArrayEquals(new int[] {0, 0, 0, 22, 44, 0}, b);
```

Also, remember that *arraycopy* will throw:

- *NullPointerException* if either array is *null*
- *IndexOutOfBoundsException* if the copy references either array beyond its range
- *ArrayStoreException* if the copy results in a type mismatch

3.3. Observing Date and Time

There're two methods related to time in *System*. One is *currentTimeMillis* and the other is *nanoTime*.

currentTimeMillis returns the number of milliseconds passed since the Unix Epoch, which is January 1, 1970 12:00 AM UTC:

```
public long nowPlusOneHour() {
    return System.currentTimeMillis() + 3600 * 1000L;
}

public String nowPrettyPrinted() {
    return new Date(System.currentTimeMillis()).toString();
}
```

nanoTime returns the time relative to JVM startup. We can call it multiple times to mark the passage of time in the application:

```
long startTime = System.nanoTime();
// ...
long endTime = System.nanoTime();

assertTrue(endTime - startTime < 10000);
```

Note that since *nanoTime* is so fine-grained, **it's safer to do *endTime – startTime < 10000* than *endTime < startTime* due to the possibility of numerical overflow (https://docs.oracle.com/javase/8/docs/api/java/lang/System.html#nanoTime)**.

3.4. Exiting the Program

If we want to programmatically exit the currently executed program, *System.exit* will do the trick.

To invoke *exit*, we need to specify an exit code, which will get sent to the console or shell that launched the program.

By convention in Unix, a status of 0 means a normal exit, while non-zero means some error occurred:

```
if (error) {
    System.exit(1);
} else {
    System.exit(0);
}
```

Note that for most programs nowadays, it'd be strange to need to call this. **When called in a web server application, for example, it may take down the entire site!**

3.5. Accessing Runtime Properties

System provides access to runtime properties with *getProperty*.

And we can manage them with *setProperty* and *clearProperty*:

```java
public String getJavaVMVendor() {
    System.getProperty("java.vm.vendor");
}

System.setProperty("abckey", "abcvaluefoo");
assertEquals("abcvaluefoo", System.getProperty("abckey"));

System.clearProperty("abckey");
assertNull(System.getProperty("abckey"));
```

Properties specified via *-D* are accessible via *getProperty*.

We can also provide a default:

```java
System.clearProperty("dbHost");
String myKey = System.getProperty("dbHost", "db.host.com");
assertEquals("db.host.com", myKey);
```

And *System.getProperties* provides a collection of all system properties:

```java
Properties properties = System.getProperties();
```

From which we can do any *Properties* operations:

```java
public void clearAllProperties() {
    System.getProperties().clear();
}
```

3.6. Accessing Environment Variables

System also provides read-only access to environment variables with *getenv*.

If we want to access the *PATH* environment variable, for example, we can do:

```java
public String getPath() {
    return System.getenv("PATH");
}
```

3.7. Administering Garbage Collection

Typically, garbage collection efforts are opaque to our programs. On occasion, though, we may want to make a direct suggestion to the JVM.

System.runFinalization is a method that allows us to suggest that the JVM run its finalize routine.

System.gc is a method that allows us to suggest that the JVM run its garbage collection routine.

Since contracts of these two methods don't guarantee that finalization or garbage collection will run, their usefulness is narrow.

However, they could be exercised as an optimization, say invoking *gc* when a desktop app gets minimized:

```
public void windowStateChanged(WindowEvent event) {
    if ( event == WindowEvent.WINDOW_DEACTIVATED ) {
        System.gc(); // if it ends up running, great!
    }
}
```

For more on finalization, check out our finalize guide (http://www.baeldung.com/java-finalize).

4. Conclusion

In this article, we got to see some of the fields and methods *System* provides. The complete list can be found in the official System documentation (https://docs.oracle.com/javase/7/docs/api/java/lang/System.html).

Also, check out all the examples in this article over on Github (https://github.com/eugenp/tutorials/tree/master/core-java).

Build your API
with SPRING

Leave a Reply

Be the First to Comment!

Start the discussion

✉ Subscribe ▾

CATEGORIES

SPRING (HTTP://WWW.BAELDUNG.COM/CATEGORY/SPRING/)

REST (HTTP://WWW.BAELDUNG.COM/CATEGORY/REST/)

JAVA (HTTP://WWW.BAELDUNG.COM/CATEGORY/JAVA/)

SECURITY (HTTP://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)

PERSISTENCE (HTTP://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)

JACKSON (HTTP://WWW.BAELDUNG.COM/CATEGORY/JACKSON/)

HTTPCLIENT (HTTP://WWW.BAELDUNG.COM/CATEGORY/HTTP/)

KOTLIN (HTTP://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIES

JAVA “BACK TO BASICS” TUTORIAL (HTTP://WWW.BAELDUNG.COM/JAVA-TUTORIAL)

JACKSON JSON TUTORIAL (HTTP://WWW.BAELDUNG.COM/JACKSON)

HTTPCLIENT 4 TUTORIAL (HTTP://WWW.BAELDUNG.COM/HTTPCLIENT-GUIDE)

REST WITH SPRING TUTORIAL (HTTP://WWW.BAELDUNG.COM/REST-WITH-SPRING-SERIES/)

SPRING PERSISTENCE TUTORIAL (HTTP://WWW.BAELDUNG.COM/PERSISTENCE-WITH-SPRING-SERIES/)
SECURITY WITH SPRING (HTTP://WWW.BAELDUNG.COM/SECURITY-SPRING)

ABOUT

ABOUT BAELDUNG (HTTP://WWW.BAELDUNG.COM/ABOUT/)
THE COURSES (HTTP://COURSES.BAELDUNG.COM)
CONSULTING WORK (HTTP://WWW.BAELDUNG.COM/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
THE FULL ARCHIVE (HTTP://WWW.BAELDUNG.COM/FULL_ARCHIVE)
WRITE FOR BAELDUNG (HTTP://WWW.BAELDUNG.COM/CONTRIBUTION-GUIDELINES)
CONTACT (HTTP://WWW.BAELDUNG.COM/CONTACT)
COMPANY INFO (HTTP://WWW.BAELDUNG.COM/BAELDUNG-COMPANY-INFO)
TERMS OF SERVICE (HTTP://WWW.BAELDUNG.COM/TERMS-OF-SERVICE)
PRIVACY POLICY (HTTP://WWW.BAELDUNG.COM/PRIVACY-POLICY)
EDITORS (HTTP://WWW.BAELDUNG.COM/EDITORS)
MEDIA KIT (PDF) (HTTPS://S3.AMAZONAWS.COM/BAELDUNG.COM/BAELDUNG+-+MEDIA+KIT.PDF)