**🔷 What is Nashorn?**

**Nashorn** is a **JavaScript engine** introduced in **Java 8** (JDK 8) under the package jdk.nashorn.api.scripting. It allows developers to:

* Execute **JavaScript code** from within Java applications.
* Interact between **Java and JavaScript** seamlessly.
* Replace the older **Rhino** engine (from Java 6 and 7) with better performance (thanks to invokedynamic in Java 7).

📍 “Nashorn” is a German word meaning “rhinoceros,” and it's a nod to the earlier JavaScript engine Rhino.

**🚀 Key Features**

| **Feature** | **Description** |
| --- | --- |
| ✅ Fast Execution | Uses invokedynamic for optimized bytecode |
| ✅ ECMAScript 5.1 Compliant | Partial support for ES6 (limited) |
| ✅ Java ↔ JavaScript Interoperability | Call Java methods/classes from JavaScript |
| ✅ Lightweight | Can run JS scripts inside a small Java app |
| ❌ Deprecated (JDK 11), Removed (JDK 15) | But still used in legacy projects |

**🛠️ Basic Usage**

**1. Evaluate JavaScript code from Java**

import javax.script.ScriptEngine;

import javax.script.ScriptEngineManager;

public class NashornExample {

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

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

engine.eval("print('Hello from JavaScript');");

}

}

**Output:**

Hello from JavaScript

**🔁 Java <-> JavaScript Interoperability**

**2. Calling JavaScript functions from Java**

String jsCode = "function greet(name) { return 'Hello, ' + name; }";

engine.eval(jsCode);

Invocable invocable = (Invocable) engine;

Object result = invocable.invokeFunction("greet", "Avinash");

System.out.println(result); // Output: Hello, Avinash

**3. Calling Java from JavaScript**

String jsCode = "var list = new java.util.ArrayList(); list.add('One'); list.add('Two'); list";

Object result = engine.eval(jsCode);

System.out.println(result); // Output: [One, Two]

✅ You can access any public Java class directly from JS via java. namespace.

**📦 Importing Java Packages in JS**

var ArrayList = Java.type("java.util.ArrayList");

var list = new ArrayList();

list.add("Java");

list.add("Nashorn");

print(list); // [Java, Nashorn]

**🔃 Bind Java variables to JS**

Bindings bindings = engine.createBindings();

bindings.put("x", 10);

bindings.put("y", 20);

engine.eval("var sum = x + y; print('Sum is ' + sum);", bindings);

**⚠️ Nashorn Deprecation and Removal**

| **Version** | **Status** |
| --- | --- |
| JDK 8 | Introduced |
| JDK 11 | Deprecated |
| JDK 15 | Removed |

✅ **Alternative**: Use **GraalVM's JavaScript engine** or external JS engines like **Rhino**, **Duktape**, or **J2V8**.

**🔍 Use Cases**

* Embedding scripting into Java apps (customization/plugins).
* Writing DSLs or templates with JavaScript.
* Rapid prototyping using dynamic code.
* Processing JSON/JS data in Java (without full JS engine).

**✅ Pros and ❌ Cons**

| **Pros** | **Cons** |
| --- | --- |
| ✅ Seamless Java–JavaScript bridge | ❌ Deprecated and removed |
| ✅ Faster than Rhino | ❌ Limited ES6 support |
| ✅ Good for small scripting tasks | ❌ Not future-proof in newer JDKs |

**📌 Summary**

* **Nashorn** lets you run JavaScript code inside Java apps (Java 8+).
* It's useful for **embedding scripts, customization, and automation**.
* Offers **tight integration** with Java code.
* **Deprecated in JDK 11**, **removed in JDK 15** — consider **GraalVM** if you need JavaScript support in newer Java versions.