Integrando Java e JavaScript no NetBeans

5 Exemplos Práticos

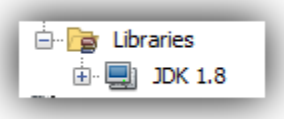
Neste artigo técnico, vamos explorar como criar uma classe Java no NetBeans capaz de executar funções JavaScript, utilizando a API ScriptEngine do Java. Essa integração permite incorporar scripts dinâmicos às aplicações Java de forma eficiente e flexível.

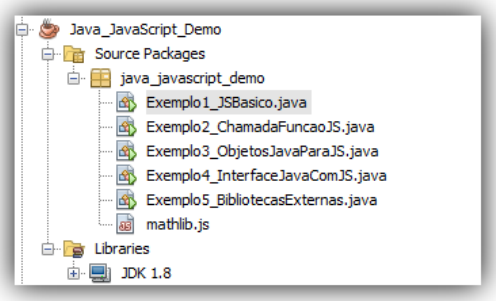
Pré-requisitos

- NetBeans 12 ou superior instalado
- JDK 8
- Conhecimento básico de Java e JavaScript

Configuração Inicial no NetBeans

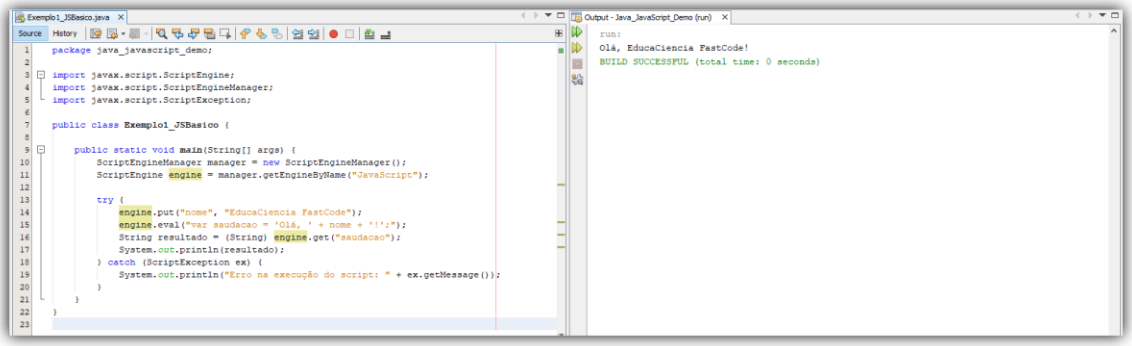
1. Crie um novo projeto Java:
   File → New Project → Java with Ant → Java Application
2. Nomeie o projeto como JavaJSDemo e clique em **Finish**.
3. O NetBeans criará automaticamente a classe principal JavaJSDemo.





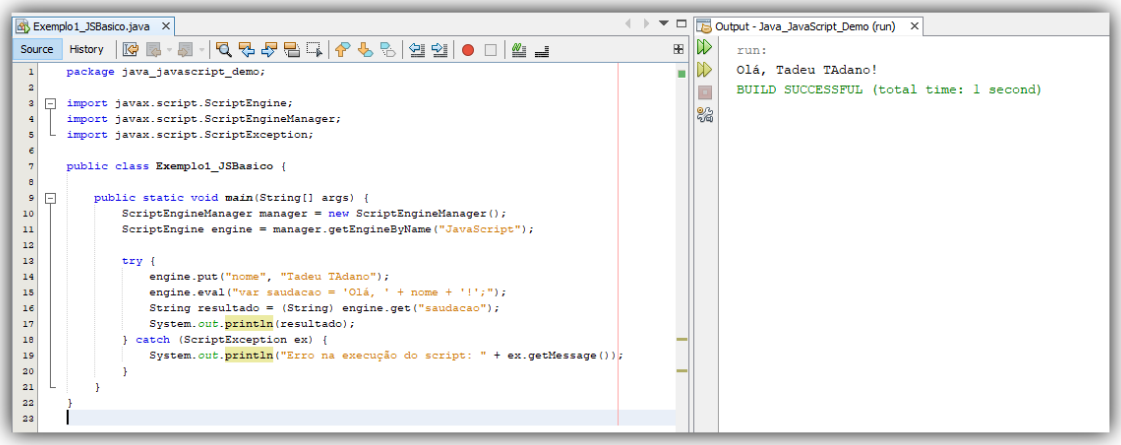
Exemplo 1: Executando um Script JavaScript Básico

```
package java_javascript_demo;

import javax.script.ScriptEngine;
import javax.script.ScriptEngineManager;
import javax.script.ScriptException;

public class Exemplo1_JSBasico {

    public static void main(String[] args) {
        ScriptEngineManager manager = new ScriptEngineManager();
        ScriptEngine engine = manager.getEngineByName("JavaScript");

        try {
            engine.put("nome", "EducaCiencia FastCode");
            engine.eval("var saudacao = 'Olá, ' + nome + '!';");
            String resultado = (String) engine.get("saudacao");
            System.out.println(resultado);
        } catch (ScriptException ex) {
            System.out.println("Erro na execução do script: " + ex.getMessage());
        }
    }
}
```



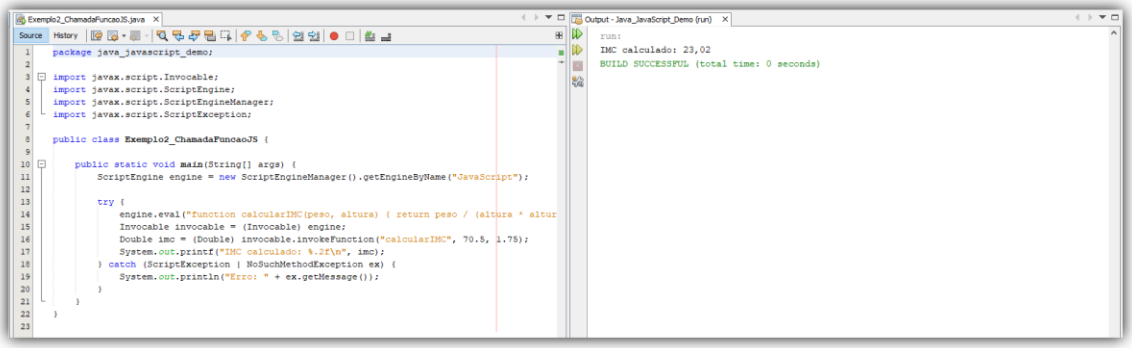
O que acontece:

- Criamos um gerenciador ScriptEngineManager e solicitamos a engine JavaScript.
- Definimos uma variável Java acessível no script JS.
- Executamos e recuperamos o valor da variável criada no script.



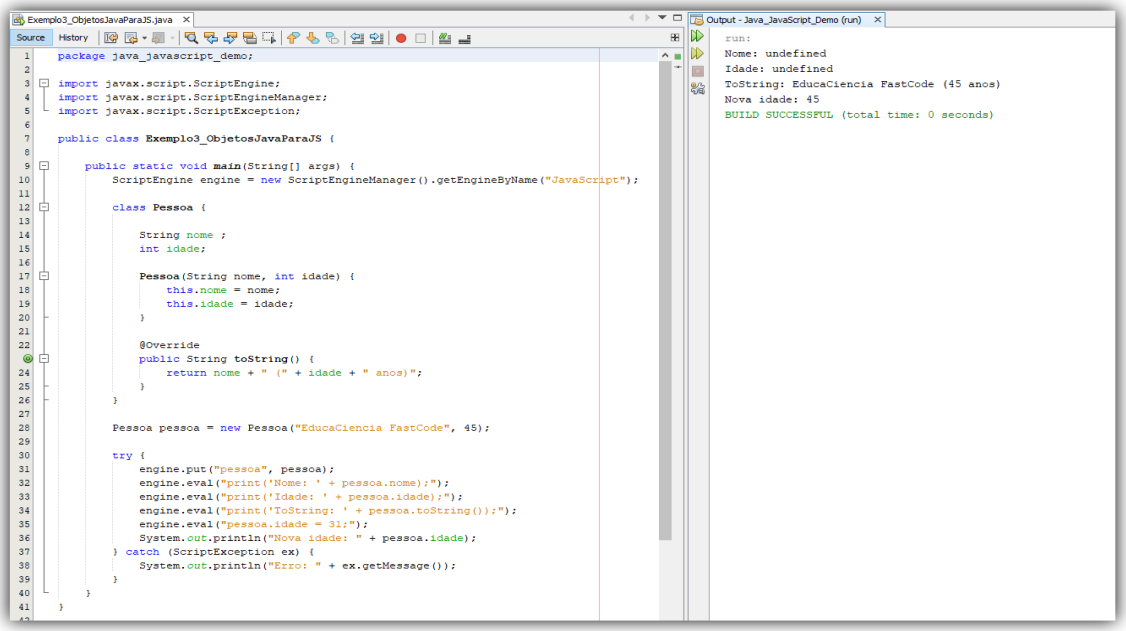
Exemplo 2: Chamando uma Função JavaScript a Partir do Java

```
import javax.script.Invocable;
import javax.script.ScriptEngine;
import javax.script.ScriptEngineManager;
import javax.script.ScriptException;

public class Exemplo2_ChamadaFuncaoJS {
    public static void main(String[] args) {
        ScriptEngine engine = new ScriptEngineManager().getEngineByName("JavaScript");

        try {
            engine.eval("function calcularIMC(peso, altura) { return peso / (altura * altura); }");
            Invocable invocable = (Invocable) engine;
            Double imc = (Double) invocable.invokeFunction("calcularIMC", 70.5, 1.75);
            System.out.printf("IMC calculado: %.2f\n", imc);
        } catch (ScriptException | NoSuchMethodException ex) {
            System.out.println("Erro: " + ex.getMessage());
        }
    }
}
```



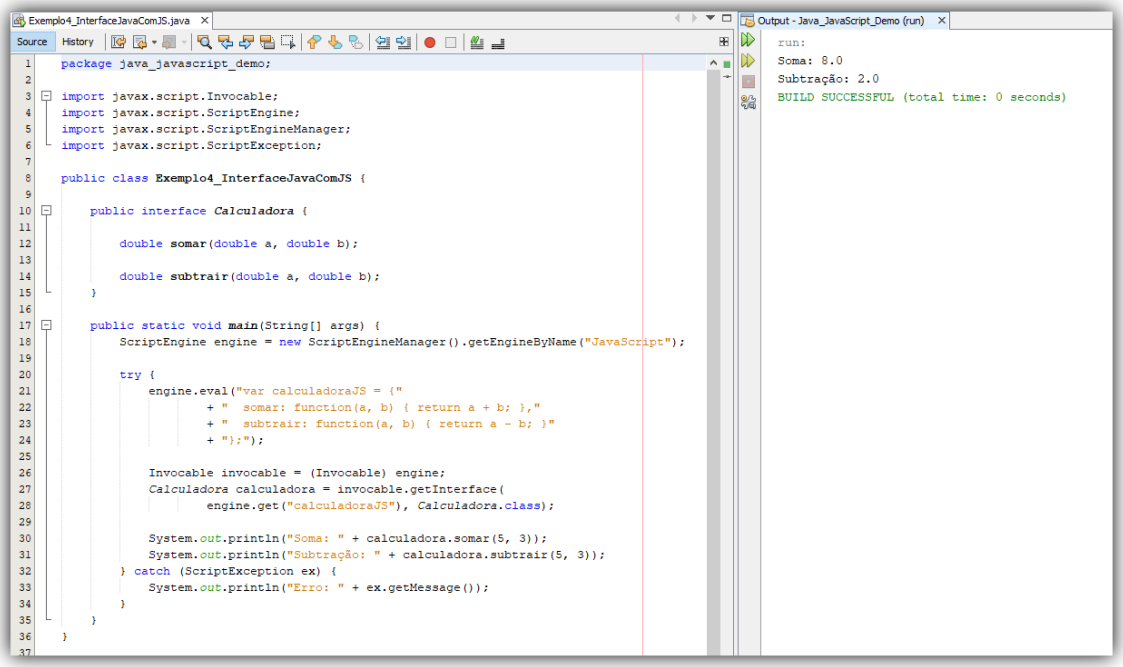
Exemplo 3: Passando Objetos Java para JavaScript

```
import javax.script.ScriptEngine;
import javax.script.ScriptEngineManager;
import javax.script.ScriptException;

public class Exemplo3_ObjetosJavaParaJS {

    public static void main(String[] args) {
        ScriptEngine engine = new ScriptEngineManager().getEngineByName("JavaScript");

        class Pessoa {

            String nome ;
            int idade;

            Pessoa(String nome, int idade) {
                this.nome = nome;
                this.idade = idade;
            }

            @Override
            public String toString() {
                return nome + " (" + idade + " anos)";
            }
        }

        Pessoa pessoa = new Pessoa("EducaCiencia FastCode", 45);

        try {
            engine.put("pessoa", pessoa);
            engine.eval("print('Nome: ' + pessoa.nome);");
            engine.eval("print('Idade: ' + pessoa.idade);");
            engine.eval("print('ToString: ' + pessoa.toString());");
            engine.eval("pessoa.idade = 31;");
            System.out.println("Nova idade: " + pessoa.idade);
        } catch (ScriptException ex) {
            System.out.println("Erro: " + ex.getMessage());
        }
    }
}
```



Exemplo 4: Implementando uma Interface Java com JavaScript

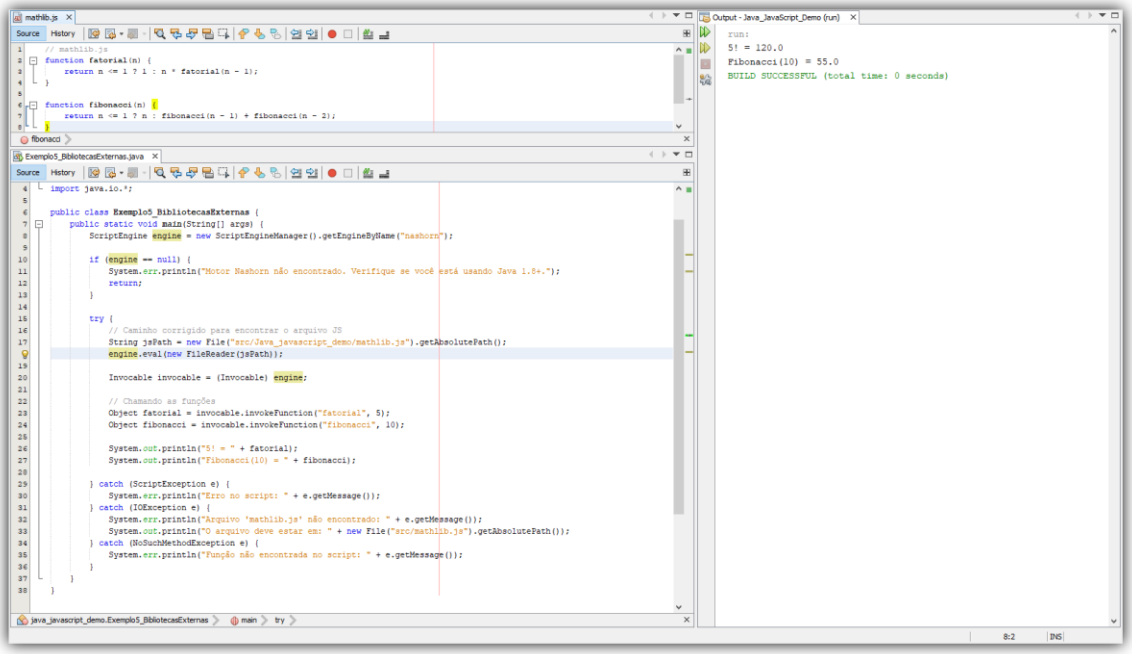
```
import javax.script.Invocable;
import javax.script.ScriptEngine;
import javax.script.ScriptEngineManager;
import javax.script.ScriptException;

public class Exemplo4_InterfaceJavaComJS {

    public interface Calculadora {
        double somar(double a, double b);
        double subtrair(double a, double b);
    }

    public static void main(String[] args) {
        ScriptEngine engine = new ScriptEngineManager().getEngineByName("JavaScript");

        try {
            engine.eval("var calculadoraJS = {"
                    + "  somar: function(a, b) { return a + b; },"
                    + "  subtrair: function(a, b) { return a - b; }"
                    + "};");

            Invocable invocable = (Invocable) engine;
            Calculadora calculadora = invocable.getInterface(
                engine.get("calculadoraJS"), Calculadora.class);

            System.out.println("Soma: " + calculadora.somar(5, 3));
            System.out.println("Subtração: " + calculadora.subtrair(5, 3));
        } catch (ScriptException ex) {
            System.out.println("Erro: " + ex.getMessage());
        }
    }
}
```



Exemplo 5: Usando Bibliotecas JavaScript Externas

```
// mathlib.js
function fatorial(n) {
    return n <= 1 ? 1 : n * fatorial(n - 1);
}

function fibonacci(n) {
    return n <= 1 ? n : fibonacci(n - 1) + fibonacci(n - 2);
}
```

```
package java_javascript_demo;

import javax.script.*;
import java.io.*;

public class Exemplo5_BibliotecasExternas {
    public static void main(String[] args) {
        ScriptEngine engine = new ScriptEngineManager().getEngineByName("nashorn");

        if (engine == null) {
            System.err.println("Motor Nashorn não encontrado. Verifique se você está usando Java 1.8+.");
            return;
        }

        try {
            // Caminho corrigido para encontrar o arquivo JS
            String jsPath = new File("src/\ava_javascript_demo/mathlib.js").getAbsolutePath();
            engine.eval(new FileReader(jsPath));

            Invocable invocable = (Invocable) engine;

            // Chamando as funções
            Object fatorial = invocable.invokeFunction("fatorial", 5);
            Object fibonacci = invocable.invokeFunction("fibonacci", 10);

            System.out.println("5! = " + fatorial);
            System.out.println("Fibonacci(10) = " + fibonacci);

        } catch (ScriptException e) {
            System.err.println("Erro no script: " + e.getMessage());
        } catch (IOException e) {
            System.err.println("Arquivo 'mathlib.js' não encontrado: " + e.getMessage());
            System.out.println("O arquivo deve estar em: " + new File("src/mathlib.js").getAbsolutePath());
        } catch (NoSuchMethodException e) {
            System.err.println("Função não encontrada no script: " + e.getMessage());
        }
    }
}
```



Considerações - EducaCiência FastCode

A integração entre Java e JavaScript via ScriptEngine oferece diversas vantagens:

- **Extensibilidade:** Permite scripts externos dinâmicos sem recompilar o código Java.
- **Prototipagem rápida:** Ideal para testar trechos lógicos em JavaScript.
- **Reaproveitamento:** Usa bibliotecas JS existentes dentro do ambiente Java.

Limitações

- **Desempenho:** Scripts são interpretados, podendo ser mais lentos.
- **Segurança:** Evite executar scripts externos não confiáveis.
- **Compatibilidade:** Recursos modernos do ECMAScript podem não estar disponíveis.

Esses cinco exemplos práticos mostram como integrar funcionalidades JavaScript diretamente em projetos Java no NetBeans.

Sinta-se à vontade para adaptá-los e expandi-los para projetos mais complexos!

EducaCiência FastCode para a comunidade