Programadores sabem que mudar o contexto entre linguagens de programação é algo difícil. Se você pretende programar em Java e JavaScript criando micro serviços, extraindo o melhor de cada linguagem, e deseja chavear o contexto rapidamente, será interessante conhecer o que as linguagens têm em comum e suas particularidades. Vamos estudar neste capítulo a sintaxe das linguagens, e comparar suas estruturas básicas.

Bem, comentamos anteriormente que ambas linguagens derivam de C/C++. Então, a sintaxe básica das linguagens é a mesma, excetuando-se particularidades relacionadas a orientação para objetos.

A declaração de variáveis, por exemplo, tem uma regra bem simples e comum. Em ambas linguagens uma variável deve se iniciar com uma letra, underline (_) ou cifrão ($). Da segunda letra em diante, pode ser qualquer letra, número, underline ou cifrão. Ou seja, uma variável não deve se iniciar com um número, caso contrário as duas linguagens irão tentar interpretar como um número e não como variável.

Seria prudente que de alguma forma você armazenasse as palavras reservadas abaixo, porque embora JavaScript tenha menos restrições, seria melhor guardar uma regra geral para formação de variáveis nas duas linguagens. Em resumo, não use os nomes abaixo como variáveis em nenhuma das duas linguagens.

abstract	**continue**	**for**	**new**	**switch**
assert	**default**	**goto**	**package**	**synchronized**
boolean	**do**	**if**	**private**	**this**
break	**double**	**implements**	**protected**	**throw**

byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while
function(JS)	var (JS)			

Em Java, não existe o modificador var. Por outro lado, em JavaScript ele indica que a variável está sendo declarada naquele momento e naquele contexto. Por exemplo:

```
// javascript
function processa() {
    var numeroPrimo=7;
}
```

Neste caso, a variável **numeroPrimo** está declarada dentro do método. Se não colocamos var, ela irá tentar encontrar em algum outro local, e caso não encontre acabará ficando como global, o que já vimos que é ruim.

Uma boa prática é sempre declarar as variáveis como var dentro dos módulos Node.JS

Por outro lado, em Java não existe var, já que a linguagem é fortemente tipada. Você deve declarar uma variável com um tipo, que pode ser byte, **short**, **int**, **long**, **float**, **double**, **boolean**, **char**, sendo que:

byte,short,int,long – tipos inteiros
float,double – números em ponto flutuante

boolean – true ou false (estes também são constantes)
char – um caractere (é tratado como um número)

O tipo **void**, ao contrário do que se poderia pensar, não gera uma variável sem tipo em Java. Ele indica que um método não terá retorno (ou melhor, todo método retorna algo em Java, e caso o retorno seja "nada" será do tipo **void**).
Em Java, todas as variáveis são declaradas dentro de uma classe, e podem ser **public**, **private** ou **protected**. Public indica que a variável pode ser acessada de qualquer lugar (é como todas variáveis JavaScript são). Private indica que somente de dentro da classe ela pode ser acessada, e protected indica que a variável pode ser acessada de dentro de uma hierarquia ou herança de classes.
Em JavaScript podemos criar variáveis globais, embora seja desejável encapsular tudo em módulos utilizando closures. Em Java não há esta opção, tudo são classes e objetos. Além disto, existem alguns modificadores especiais em Java: **volatile**, **final**, **static**, **transient** e **const**, com as seguintes finalidades:

volatile – Esta variável pode ser acessada por mais de uma thread (esqueça, raramente utilizado)
final – Não pode ser modificada uma vez atribuída (uma constante)
static – Não pertence a classe, pode ser acessada mesmo sem uma instância disponível (lembre-se do método main)
transient - Os mecanismos de persistência e serialização do Java irão ignorar esta variável
const – é reservado em Java, mas não utilizado

Dá para perceber que Java tem muito mais modificadores do que JavaScript, afinal é uma linguagem fortemente tipada!

Uma variável em Java pode ser declarada como

```
public static final int valor = 0;
```

ela é visível por outras classes (**public**) mesmo sem uma instância da classe (**static**) e seu valor não pode ser alterado após atribuído (**final**), além de ser um inteiro de 32bits com valor igual a zero.
Neste ponto, você pode estar imaginando que trabalhar com as duas linguagens será um inferno, mas na verdade daqui em diante quase tudo é muito parecido.
Seria uma boa estratégia padronizar os nomes de variáveis e funções nas duas linguagens. O mais aceito e comum nas duas linguagens é o **CamelCase**. Cada uma das palavras de uma variável terá sua primeira letra em maiúscula, excetuando-se a primeira palavra. Por exemplo:

```
umaCamaDeCasal
seuImpostoDeRenda
```

Estruturas de controle permitem iterar ou tomar decisões sobre um conjunto de elementos e são fundamentais. O mais comum é a estrutura do tipo **if/else**. Ela tem a sintaxe

```
// comentário em Java e JavaScript (uma linha apenas)
if ((diaMes == 10) || (diaSemana == 0))
{
  return "é dia 10 ou é domingo !";
}
else
{
  return "Não é dia 10 nem domingo";
```

```
}
```

A abertura e fechamento dos parênteses definem o início e final da condição. Dentro dele é necessária alguma comparação booleana, ou uma expressão que gere **true** ou **false**. A abertura e fechamento de chaves define o que deve ser executado nesta condição, e caso seja apenas uma linha, não são necessários (uma boa prática é sempre colocar as chaves).
O **else** indica a condição a ser executada caso o **if** não seja **true**, e pode ser omitido sem problema.
Uma variante desta estrutura é o **if/else if/else**

```
/* Comentário em JavaScript e Java
    com mais de uma linha */
if ((diaMes == 10) || (diaSemana == 0)) {
  return "é dia 10 ou é domingo !";
}
else if (diaMes == 10)
{
  return "é dia 10";
}
else if (diaMes == 11)
{
  return "é dia 11";
}
else
{
  return "nao é dia 10 nem 11 nem dia 10 e domingo";
}
```

Os operadores que podem ser utilizados nas duas linguagens são:

==	Igual comparação	&	AND binário
===	Estritamente igual (JAVASCRIPT)	|	OR binário
!=	Diferente comparação	^	XOR binário
!==	Estritamente diferente comparação (JAVASCRIPT)	~	NOT binário
<, <=, >, >=	Menor, menor ou igual, maior, maior ou igual	<<	Shift left binário
&&	AND comparação	>>	shift right binário
||	OR comparação	>>>	shift right binário sem carry
!	Not comparação		

O único operador existente no JS e não no Java é o comparador estrito. Quando você compara dois valores, como

```
true == 'true'
```

o JS tenta converter os valores para um mesmo tipo, e o resultado é **true**, embora neste exemplo anterior estejamos comparando um booleano com uma string. Para forçar uma comparação de mesmo valor e mesmo tipo, utilizamos os comparadores estritos

```
true === 'true'
```

Será **false**, porque embora os dois sejam **true**, um valor é booleano e outro é **string**.
Em Java não existe este operador porque a comparação entre tipos diferentes não é permitida implicitamente.
A comparação de **strings** em Java tem uma particularidade.

```
"umapalavra" == "umapalavra"
```

é totalmente válido em JS. Em Java, **strings** são objetos (existe uma classe **java.lang.String** !). Quando fazemos a comparação acima, estamos na verdade comparando a referência (o ponteiro de memória) que não necessariamente precisam ser iguais. A forma correta de comparar conteúdo em Java é através do método **equals**, disponível na classe String.

```
"umapalavra".equals("umapalavra")
```

Strings em Java sempre ficam entre aspas duplas, mas em JS são aceitos tanto aspas simples quanto aspas duplas. Se colocamos um caractere dentro de aspas simples, no Java significam um único caractere (char). Um char é diferente de uma String em Java. Um char é um tipo básico (não objeto) enquanto que uma String é um objeto. Além disto, o char em Java é tratado como número, e pode ser somado ou subtraído de outros valores. Uma boa estratégia é tentar usar strings em JavaScript sempre entre aspas duplas.

Ambas linguagens utilizam ponto e virgula como finalizador de linha. Embora existam situações onde não é

necessário, o ideal é sempre finalizar linhas com ;

A estrutura de controle **while** é igual nas duas linguagens:

```
while (x < 10) {
  // o que fazer com o valor de x
  x++;
}
```

O for, utilizado quando se sabe a condição inicial e condição de saída é idêntico em ambas linguagens.

```
for (declaração_inicial; expressão_lógica; salto ou incremento) {
    // aqui vão as instruções
}
```

Esta estrutura é bem flexível, sendo que todos os elementos dentro dos parênteses são opcionais.

```
for (; ; ) {
    // loop sem condição de saída nem condição inicial
    // loop infinito até que encontre algum break.
}
for (; ; x++) {
    // o incremento é em x (x deve existir anteriormente)
}
for (int x=0;x<10 ;x++ ) {
    // O mais comumente utilizado
    // A variável I é declarada e existe apenas dentro do for
}
```

A estrutura **do/while** também é comum:

```
do {
```

```
    // o que executar
    x++;
  } while (x < 10);
```

Para efeitos práticos, a única diferença desta estrutura para o **while** é que neste modelo ao menos uma vez o loop será executado. No **while** ele pode sair antes mesmo de executar uma vez (o teste acontece antes de entrar no loop).

Existe uma estrutura **for-in** para navegar em coleções de objetos:

```
// JavaScript
for (elemento in colecao)
{
  // loop
}
```

```
// Java
for (String item : someList) {
  // loop
}
```

A estrutura do tipo switch tem a seguinte aparência:

```
switch (month) {
    case 1:  monthString = "January";
             break;
    case 2:  monthString = "February";
             break;
    default: monthString = "Invalid month";
             break;
}
```

No caso do Java, existe uma restrição para os tipos que podem ser utilizados em switch. Eles são: o que puder ser convertido para inteiros (**byte**, **short**, **char**, **int**), **Strings** (a partir do Java 7) e **enums**. Esta restrição não se aplica ao JavaScript.

Os operadores aritméticos são aqueles comuns da linguagem C: + para soma, - para subtração, * para multiplicação, / para divisão, % para módulo (resto da divisão), ++ incremento e -- para decremento.

Além disto, as duas linguagens suportam os conjugados, que são as combinações de operadores com atribuição. Por exemplo:

```
valor += 1;
// o mesmo que
valor = valor + 1;
// isto vale para os outros operadores também
```

Ainda as duas linguagens suportam uma variação confusa do if, que não é recomendado, mas é necessário explicar:

```
mensagem = (numero % 2 == 0) ? "número par" : "número impar"

// equivalente a
if (numero % 2 == 0)
    mensagem = "número par";
else
    mensagem = "número impar";
```

No que se refere a declaração de métodos, a comparação é

muito similar à declaração de variáveis descrita anteriormente.
Em JavaScript uma função pode ser declarada como

```
function soma( numero1, numero2)
```

Já em Java, como é uma linguagem tipada, o método seria:

```
<modificadores> <retorno> soma(<tipo> numero1, <tipo>numero2)
// por exemplo
public static int soma (int numero1, int numero2)
```

O **function** não é necessário e não existe em Java, e os tipos e modificadores para o retorno e parâmetros são obrigatórios.

Utilizamos o termo função quando não está atrelado a uma classe (JavaScript) e método quando está dentro de uma classe (Java), mas esta é apenas uma convenção adotada há muito tempo. Funcionalmente os dois blocos são similares.

Ainda as duas linguagens utilizam uma sintaxe "mais moderna" e menos prolixa que permitem economizar algum espaço ao digitar. É bom conhece-la, mesmo que não for adota-la, pois você encontrará códigos na internet com esta notação. Em JavaScript recebem o nome de "arrow functions".

Como é muito comum criarmos métodos em JavaScript e atribuirmos a métodos, faz sentido economizarmos digitação. Por exemplo, ao invés de algo como:

```
var temp = function soma(a,b) { return a + b; }
```

pode-se adotar a sintaxe

```
var temp = (a,b) => a+b;
```

Ela economiza em digitação, e dizem alguns programadores que é menos prolixa.
Em Java, temos uma estrutura similar que é chamada de lamba. É muito comum passar uma instância de classe para um processamento como:

```
btn.setOnAction(new EventHandler<ActionEvent>() {
  @Override
  public void handle(ActionEvent event) {
      System.out.println("Hello World!");
  }
});
```

Isto pode ser reduzido com lambda a:

```
btn.setOnAction(
    event -> System.out.println("Hello World!")
);
```

Ao invés de criarmos uma inner class com o método EventHandler, apenas passamos o valor para ser processado, e a seta elimina a necessidade de declarar a classe e o método. Neste caso funciona perfeitamente porque EventHandler tem apenas um método.

Ambas linguagens também possuem uma estrutura do tipo for

que permite navegar por uma coleção de tamanho delimitado. Em JavaScript podemos navegar com:

```
var valores = ["valor1", "valor2", "valor3"];
for (umvalor in valores) {
  // processa um valor
}
```

E Java permite estrutura similar com:

```
String[] valores = new String[] { "valor1", "valor2", "valor3"};
for (String umvalor : valores) {
   // processa um valor
}
```

A escolha destas duas linguagens é sábia porque além de muito próximas, acaba sendo fácil chavear entre elas sem muito impacto. Como guardam o mesmo ancestral comum que é C++, acabaram adotando as mesmas estruturas de construção ao longo de sua evolução.