**Linguagem de Programação: C#**

Departamento de Sistemas de Informação

**Universidade Federal de Sergipe (UFS) – Itabaiana, SE – Brasil**

Carlos Henrique Lima de Jesus

Charles Dayan da Conceição Costa

Jorge Matheus dos Santos

**1. Introdução**

O C# (pronuncia-se "C Sharp") é uma linguagem de programação moderna, orientada a objeto e fortemente tipada. O C# permite que os desenvolvedores criem muitos tipos de aplicativos seguros e robustos que são executados no .NET.

O C# é uma linguagem de programação orientada a objetos e orientada a componentes. C# fornece construções de linguagem para dar suporte diretamente a esses conceitos, tornando C# uma linguagem natural para criação e uso de componentes de software. Desde sua origem, o C# adicionou recursos para dar suporte a novas cargas de trabalho e práticas emergentes de design de software. Em sua essência, o C# é uma linguagem orientada a objetos. Você define os tipos e o comportamento deles.

Vários recursos do C# ajudam a criar aplicativos robustos e duráveis. A [***coleta de lixo***](https://learn.microsoft.com/pt-br/dotnet/standard/garbage-collection/) recupera automaticamente a memória ocupada por objetos não utilizados inacessíveis. [***Tipos anuláveis***](https://learn.microsoft.com/pt-br/dotnet/csharp/nullable-references) são protegidos contra variáveis que não se referem a objetos alocados. O [***tratamento de exceções***](https://learn.microsoft.com/pt-br/dotnet/csharp/fundamentals/exceptions/) fornece uma abordagem estruturada e extensível para detecção e recuperação de erros. [***As expressões Lambda dão suporte a técnicas***](https://learn.microsoft.com/pt-br/dotnet/csharp/language-reference/operators/lambda-expressions) de programação funcional. [***Consulta Integrada à Linguagem (LINQ)***](https://learn.microsoft.com/pt-br/dotnet/csharp/linq/) a sintaxe cria um padrão comum para trabalhar com dados de qualquer fonte. O suporte à linguagem para [***operações assíncronas fornece sintaxe***](https://learn.microsoft.com/pt-br/dotnet/csharp/programming-guide/concepts/async/) para a criação de sistemas distribuídos. C# tem um [***sistema de tipo unificado***](https://learn.microsoft.com/pt-br/dotnet/csharp/fundamentals/types/). Todos os tipos do C#, incluindo tipos primitivos, como int e double, herdam de um único tipo de object raiz. Todos os tipos compartilham um conjunto de operações comuns. Valores de qualquer tipo podem ser armazenados, transportados e operados de maneira consistente. Além disso, o C# dá suporte a [**tipos de referência**](https://learn.microsoft.com/pt-br/dotnet/csharp/language-reference/builtin-types/reference-types) e [**tipos de valor**](https://learn.microsoft.com/pt-br/dotnet/csharp/language-reference/builtin-types/value-types) definidos pelo usuário. O C# permite a alocação dinâmica de objetos e o armazenamento em linha de estruturas leves. O C# dá suporte a métodos e tipos genéricos, que fornecem maior segurança e desempenho do tipo. O C# fornece iteradores, que permitem que os implementadores de classes de coleção definem comportamentos personalizados para o código do cliente.

O C# enfatiza o controle de versão para garantir que programas e bibliotecas possam evoluir ao longo do tempo de maneira compatível. Aspectos do design do C# que foram diretamente influenciados pelas considerações de controle de versão incluem os modificadores separados virtual e override, as regras de resolução de sobrecarga de método e suporte para declarações explícitas de membro de interface.

**2. Lexemas do Java**

Utilizamos, em Java, as seguintes regras para criação do identificador:

1. não pode ser uma palavra-reservada (palavra-chave);

2. não pode ser true nem false - literais que representam os tipos lógicos (booleanos); 3. não pode ser null - literal que representa o tipo nulo;

4. não pode conter espaços em brancos ou outros caracteres de formatação; 5. deve ser a combinação de uma ou mais letras e dígitos UNICODE-16. Por exemplo, no alfabeto latino, teríamos:

• letras de *A* a *Z* (de *\u0041* a *\u005a*);

• letras de *a* a *z* (de *\u0061* a *\u007a*);

• sublinha *\_* (*\u005f*);

• cifrão *$* (*\u0024*);

• dígitos de *0* a *9* (de *\u0030* a *\u0039*).

Observação 01: caracteres compostos (acentuados) não são interpretados igualmente aos não compostos (não acentuados). Por exemplo, *História* e *Historia* não são o mesmo identificador. Observação 02: letras maiúsculas e minúsculas diferenciam os identificadores, ou seja, *a* é um identificador diferente de *A*, *História* é diferente de *história*, etc.

**2.1 Comentários**

Os comentários em Java são declarações que não são executadas pelo compilador ou interpretador. Os comentários podem ser usados para prover informações ou explanações sobre as variáveis, métodos, classes ou alguma declaração.

Existem três tipos de comentários em Java:

• Comentário de linha única: // Isso é uma única linha comentada

• Comentário de múltiplas linhas: /\* Isso é um comentário

de múltiplas linhas \*/

• Comentário de documentação: /\*\* Isto é uma documentação documentada \*/

**2.2 Palavras reservadas**

Em programação, *palavras-chave*, ou *palavras reservadas*, são as palavras que não podem ser usadas como identificadores. Em outras palavras, não podem ser usadas como nome de variáveis, nome de classes, etc. Estas palavras são assim definidas ou porque já têm uso na sintaxe da linguagem ou porque serão usadas em alguns momentos, seja para manter compatibilidade com versões anteriores ou mesmo com outras linguagens. No caso do Java temos as seguintes *palavras reservadas*:

• **abstract:** usada para declarar a classe abstrata. A classe abstrata pode fornecer a implementação da interface. Pode ter métodos abstratos e não abstratos.

• **assert:** descreve um predicado (uma declaração de verdadeiro ou falso) colocado em uma aplicação Java para indicar se o valor passado para o predicado é verdadeiro ou falso. • **boolean:** usada para declarar uma variável como um tipo boolean. Pode conter apenas valores True e False.

• **break:** utilizada para interromper o loop ou alternar a instrução. Ele interrompe o fluxo atual do programa na condição especificada.

• **byte:** a palavra-chave byte é usada para declarar uma variável que pode conter valores de dados de 8 bits.

• **case:** a palavra-chave case é usada com as instruções switch para marcar blocos de texto. • **catch:** a palavra-chave catch é usada para capturar as exceções geradas pelas instruções try. Ele deve ser usado somente após o bloco try.

• **char:** a palavra-chave char é usada para declarar uma variável que pode conter caracteres Unicode de 16 bits não assinados.

• **class:** utilizada para declarar uma classe.

• **const:** reservada, porém, sem função.

• **continue:** utilizada para continuar o loop. Ele continua o fluxo atual do programa e ignora o código restante na condição especificada.

• **default:** utilizada para especificar o bloco de código padrão em uma instrução switch. • **do:** a palavra-chave do é usada na instrução de controle para declarar um loop. Ela pode iterar uma parte do programa várias vezes.

• **double:** usada para declarar uma variável que pode conter números de ponto flutuante de 64 bits.

• **else:** utilizada para indicar as ramificações alternativas em uma instrução if. • **enum:** utilizada para definir um conjunto fixo de constantes. Os construtores Enum são sempre privados ou padrão.

• **extends:** usada para indicar que uma classe é derivada de outra classe. • **final:** utilizada para indicar que uma variável mantém um valor constante. É aplicado com uma variável. É usado para restringir o usuário.

• **finally:** indica um bloco de código em uma estrutura try-catch. Este bloco é sempre executado independentemente de a exceção ser tratada ou não.

• **float:** usada para declarar uma variável que pode conter um número de ponto flutuante de 32 bits.

• **for:** usada para iniciar um loop for. Tem o objetivo de executar um conjunto de instruções / funções repetidamente quando algumas condições se tornam verdadeiras. Se o número de iterações for fixo, é recomendável usar o loop.

• **goto:** tal palavra-chave não é utilizada, porém encontra-se reservada.

• **if:** testa a condição. Executa o bloco if se a condição for verdadeira.

• **implements:** usada para implementar uma interface.

• **import:** torna as classes e interfaces disponíveis e acessíveis ao código-fonte atual. • **instanceof:** utilizada para testar se o objeto é uma instância da classe especificada ou implementa uma interface.

• **int:** usada para declarar uma variável que pode conter um número inteiro assinado de 32 bits. • **interface:** a palavra-chave interface é usada para declarar uma interface. Pode ter apenas métodos abstratos.

• **long:** a palavra-chave long é usada para declarar uma variável que pode conter um número inteiro de 64 bits.

• **native:** a palavra-chave native é usada para especificar que um método é implementado no código nativo usando JNI (Java Native Interface).

• **new:** usado para criar uma instância de um objeto de classe ou matriz. • **package:** utilizada para declarar um pacote Java que inclui as classes. • **public:** modificador de acesso. É usado para indicar que um item está acessível em qualquer lugar. Possui o escopo mais amplo entre todos os outros modificadores.

• **private:** modificador de acesso. É usado para indicar que um método ou variável pode ser acessado apenas na classe em que é declarado.

• **protected:** modificador de acesso. Pode ser acessível dentro e fora do pacote, mas apenas por herança. Não pode ser aplicado na classe.

• **return:** usada para retornar um método quando sua execução estiver concluída. • **short:** a palavra-chave short é usada para declarar uma variável que pode conter um número inteiro de 16 bits.

• **static:** indica que uma variável ou método é um método de classe. A palavra-chave estática em Java é usada principalmente para gerenciamento de memória.

• **strictfp:** restringi os cálculos de ponto flutuante para garantir a portabilidade. • **super:** variável de referência usada para referenciar o objeto da classe pai. Pode ser usado para invocar o método imediato da classe pai.

• **switch:** a palavra-chave switch contém uma instrução switch que executa o código com base no valor do teste. A instrução switch testa a igualdade de uma variável em relação a vários valores.

• **synchonized:** especifica seções ou métodos críticos no código multithread. • **this:** referencia o objeto atual em um método ou construtor.

• **throw:** a palavra-chave throw é usada para lançar explicitamente uma exceção. A palavrachave throw é usada principalmente para lançar exceções personalizadas. É seguido por uma instância.

• **throws:** usada para declarar uma exceção. A exceção marcada pode ser propagada com lançamentos.

• **transient:** a palavra-chave transient é usada na serialização. Se você definir qualquer membro de dados como transient, ele não será serializado.

• **try:** utilizada para iniciar um bloco de código que será testado para exceções. O bloco try deve ser seguido pelo bloco catch ou finalmente.

• **void:** a palavra-chave void é usada para especificar que um método não possui um valor de retorno.

• **volatile:** indica que uma variável pode ser alterada de forma assíncrona.

• **while:** usada para iniciar um loop while. Este loop itera uma parte do programa várias vezes. Se o número de iterações não for fixo, é recomendável usar o loop while.

**2.3 Literais reservados**

• **null**: define que uma variável não está apontando para nada.

• **false**: valor lógico que indica falso.

• **true**: valor lógico que indica verdadeiro.

**2.4 Operadores e delimitadores**

**Operador** em Java é um símbolo que é usado para executar operações. Por exemplo: +, -, \*, / etc.

Existem muitos tipos de operadores em Java que são fornecidos abaixo:

• Operador Unário,

• Operador aritmético,

• Operador shift,

• Operador relacional,

• Operador bit a bit (bitwise),

• Operador lógico,

• Operador ternário e

• Operador de atribuição.

Precedência de operadores em Java

| Tipo de  Operador | Categoria | Precedência |
| --- | --- | --- |
| Unário | sufixos | *expr*++ *expr*-- |
| prefixo | ++*expr* --*expr* +*expr* -*expr* ~ ! |
| Aritmética | multiplicativo | \* / % |
| aditivo | + - |
| Mudança | mudança | << >> >>> |
| Relacional | comparação | < > <= >= instanceof |

|  | igualdade | == != |
| --- | --- | --- |
| Bit a bit | bit a bit AND | & |
| bit a bit OR  exclusivo | ^ |
| bit a bit  inclusive OU | | |
| Lógico | AND lógico | && |
| OR lógico | || |
| Ternário | ternário | ? : |
| Atribuição | atribuição | = += -= \*= /= %= &= ^= |=  <<= >>= >>>= |

**2.5 Literais String**

Literais strings são representações de código que representam um valor de um objeto String. Existem várias maneiras de se representar um objeto String, em Java.

**Ex.:** *String* nome = “Carlos Silva”; // Atribui “Carlos Silva” a variável nome *System.out.println*(“Carlos” + “Silva”); // Imprime Carlos Silva

**2.6 Literais Numéricos**

**2.6.1 Literais Integer**

É possível representar números do tipo integer de três formas em Java: decimal (base 10), octal (base 8) e hexadecimal (base 16).

**2.6.1.1 Literal Decimal**

Os decimais (base 10) em Java, são representados apenas com números, sem prefixo ou sufixo. Exemplo: *int* idade = 62; // 62 é uma literal decimal

**2.6.1.2 Literal Octal**

Octal, em Java, utiliza números entre 0 e 7. Para representar os números em octais, deve colocar um “0” na frente de qualquer inteiro. Ex.: *int* x = 06; // 06 é uma literal octal

**2.6.1.3 Literal Hexadecimal**

Hexadecimais, ou hex, são números construídos com 16 símbolos distintos**.** sendo que do 10 ao 15 são representados pelas letras do alfabeto: a, b, c, d, e, f. Respectivamente, ficando desta forma: 0 1 2 3 4 5 6 7 8 9 a b c d e f.

Um detalhe importante a ser acrescentado, é que java não é caseSensitive quanto ao reconhecimento das letras em hex. tanto faz você utilizar *123Cafe* quando *123CAFE*. Todos representarão 12312101514.

**2.6.4 Literais de ponto flutuante**

Literais de ponto flutuante são representadas por números, sendo eles decimais, e outros números representando a fração. As literais de ponto flutuante são, por padrão, do tipo primitivo double. Ex.: *float* f = 3.14f;

**2.6.5 Literais Booleanas**

Literais booleanas são representações de código para os valores booleanos. Sendo assim, seus valores só poderão ser true ou false.

**Ex.:**

boolean a = true; // Ok!

a = false; // Ok!

a = 0; // Erro de compilação

a = 1; // Erro de compilação

**2.6.6 Literais Char:**

São representadas por um único caractere. Por exemplo: *char* x = ‘s’. Perceba que não é utilizado duplas aspas neste caso. Mas sim, aspas simples.

**Referências:**

JAVA Tutorial. [*S. l.*], 2018. Disponível em: https://www.w3schools.com/java/java\_intro.asp.

MALAQUIAS, J.R. Construção de Compiladores. [*S. l.*: *s. n.*], 2011. Disponível em: http://www.decom.ufop.br/romildo/bcc328.2011-1/praticas/pratica.java.pdf.

OPERADORES em Java. [*S. l.*], 2017. Disponível em: https://www.javatpoint.com/pt/operadores-em java.

PALAVRAS reservadas do Java. [*S. l.*], 2014. Disponível em: http://excript.com/java/palavras reservadas-java.html.