DOS BLOCOS AO TEXTO:

DICAS PARA PROGRAMAR COM BLUEJ

RELATÓRIO TÉCNICO

Alison de Quadros Carnetti
João Benno Weber - Lucia Giraffa

CNPq
Conselho Nacional de Desenvolvimento Científico e Tecnológico

EDITORA & LIVRARIA
VECHER EDUCOM

Alison de Quadros Carnetti
João Benno Weber - Lucia Giraffa

DOS BLOCOS AO TEXTO:

Dicas para programar com BlueJ

RELATÓRIO TÉCNICO

EDITORA & LIVRARIA
VECHER EDUCOM

Apoio do Conselho Nacional de Desenvolvimento Científico
e Tecnológico (CNPq), através da Bolsa de Produtividade
em Pesquisa - PQ - Processo: 312864/2020-5.

ISBN: 978-65-993242-9-1

DOI: 10.47585/dosblocosaotexto

Editoração: Marcelo Rodríguez

Vecher
Avenida Paulista, 171, 4º andar
CEP 01.311-904
São Paulo, SP
www.editora.vecher.com.br

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISBD

C289d Carnetti, Alison de Quadros

Dos blocos ao texto [recurso eletrônico]: Dicas para programar com BlueJ / Alison de Quadros Carnetti, João Benno Weber, Lucia Giraffa. – São Paulo : Vecher, 2021.
54 p. : PDF ; 4,6 MB.

Inclui bibliografia, índice e apêndice.
ISBN: 978-65-993242-9-1 (Ebook)

1. Linguagem de programação. 2. Ensino – Programação. I. Weber, João Benno. II. Giraffa, Lucia. III. Título.

CDD 005.133
2021-3679 CDU 004.43

Elaborado por Odilio Hilario Moreira Junior - CRB-8/9949

Índice para catálogo sistemático:
1. Ciência da Computação: linguagem de programação 005.133
2. Ciência da Computação: linguagem de programação 004.43

SUMÁRIO INTERATIVO

INTRODUÇÃO

Na hora de ensinar ou aprender programação, existe uma etapa importante na qual deve ser escolhido o ambiente para iniciar o desenvolvimento dos programas. Desse modo, além das diversas possibilidades associadas aos diferentes paradigmas de programação, é disponibilizada para cada um deles uma série de opções em que são integrados recursos que facilitam a tarefa de organizar e testar seu código.

Assim sendo, esse manual foi idealizado a partir de pesquisas provenientes das Bolsas de Iniciação Científica (**CNPq** e **PUCRS-BPA**), pelos autores João Benno e Alison Carnetti, respectivamente.

Neste guia, são apresentados alguns ambientes para se ensinar programação, desde o nível mais iniciante (para aqueles que nunca programaram) até níveis mais avançados, como o ensino considerando o paradigma Programação Orientada a Objetos (POO).

Nessa perspectiva, seu objetivo é auxiliar na escolha de ambientes para o ensino de programação, denominados de IDE (*Integrated Development Environment*), o quais se caracterizam por combinar recursos para criação de código, em determinada linguagem em uma única interface gráfica (GUI), buscando, com isso, facilitar o desenvolvimento da programação.

Ademais, esse guia, resultado de uma atividade de pesquisa envolvendo ensino de programação orientada a iniciantes no paradigma de OO, foca na linguagem Java, uma vez que é a adotada nos cursos de Computação da PUCRS, nos primeiros dois semestres.

A abordagem metodológica visa apoiar o ensino de programação OO para iniciantes e se baseia no desenvolvimento do Pensamento Computacional (PC). Nesse sentido, ao conceituar o PC, Brackmann (2017) o define como "uma distinta capacidade criativa, crítica e estratégica humana de saber utilizar os fundamentos da computação, nas mais diversas áreas do conhecimento, com a finalidade de identificar e resolver problemas, de maneira individual ou colaborativa, através de passos claros, de tal forma que uma pessoa ou uma máquina possam executá-los eficazmente."

De acordo com Brackmann, o PC deve ser entendido como uma alternativa para solução de problemas que utiliza o pensamento crítico unido aos conceitos da computação. Em outras palavras, é uma competência associada ao conjunto de capacidades humanas usadas na resolução de problemas. Baseia-se na decomposição de problemas, no reconhecimento de padrões, na abstração e nos conceitos e práticas relacionados à construção, ao uso e à avaliação de algoritmos. Isso posto, esses quatro pilares podem ser assim entendidos:

- **Decomposição:** o indivíduo consegue separar um problema em partes menores. Ou seja, é decomposto para facilitar o seu entendimento e poder organizar a sua resolução.

- **Reconhecimento de Padrões:** o indivíduo identifica similaridades e reconhece aquilo que

já conhece (ou precisa conhecer) para poder utilizar na construção da sua solução.

- **Abstração:** o indivíduo identifica e consegue separar aquilo que é importante para poder resolvê-lo. Assim, diferencia os processos mais importantes do problema ao invés de priorizar o detalhe, por exemplo.

- **Algoritmos:** o indivíduo cria uma sequência de passos feitos especialmente para solucionar um problema. Com isso, desenvolve instruções em ordem para a execução de uma tarefa.

Cabe aqui também ressaltar que, ao apoiar o ensino de programação na abordagem da construção do conjunto de habilidades e competências para "pensar computacionalmente", busca-se incentivar a criatividade dos alunos e não a condicioná-los ao ambiente do IDE. Ou seja, após a construção da sua solução é que ele/ela irá transpor o seu planejamento para código e, aí sim, buscar resolver questões de sintaxe, semântica e forma de organizar seu código em função das características da linguagem e, por conseguinte, do IDE escolhido.

Nessa lógica, para organização deste guia, foram pesquisados e analisados diversos IDEs. Como critério de escolha, foram selecionados aqueles cuja finalidade era de introduzir a programação de forma amistosa, indo do nível mais

básico (usando blocos) até o nível intermediário (usando texto), de modo que a transição de IDEs seja a mais suave possível. Para tanto, essa pesquisa utilizou como referência o trabalho de Michael Kölling (https://nms.kcl.ac.uk/michael.kolling/).

O QUE É IDE?

O termo IDE, do inglês *Integrated Development Environment* (Ambiente de Desenvolvimento Integrado), define qualquer programa de computador que integre ferramentas de desenvolvimento de programas em um único local, facilitando, assim, o desenvolvimento de aplicações. Um IDE consiste em:

- **Editor de código:** é a parte que permite escrever ou editar o código de um programa, colorindo partes da sintaxe (palavras e comandos da linguagem). Possui também preenchimento automático e indicação de erros.

- **Compilação automática:** Ao terminar de escrever um programa, é necessário compilar, isto é, juntar todas as suas partes em uma ordem para o processador executá-las. Assim, a compilação automática facilita a vida de iniciantes astronomicamente, já que o processo manual requer um conhecimento mais avançado da linguagem.

- **Debugger:** O termo *debugger* significa desinfetar ou remover '*bugs*', que na programação são defeitos no programa. Os *debuggers* existem nos IDEs para, antes ou durante a compilação, testar o programa e apontar onde estão localizados os defeitos do código.

Os IDEs servem também para ajudar os desenvolvedores a programar de forma mais rápida, uma vez que todos os utilitários listados anteriormente não precisam ser adicionados manualmente durante a configuração do programa, uma vez que já estão todos lá. Além disso, os desenvolvedores ganham muito tempo, já que não precisam ficar horas aprendendo a usar cada uma das ferramentas, na medida em que estão todas integradas no ambiente de programação e podem ser usadas de forma simples e intuitiva, muitas vezes com apenas um clique. Sem dúvida, isso é de extrema utilidade para quem está começando, pois facilita tanto a usabilidade que acaba se tornando indispensável na hora de ensinar programação.

Ademais, existem outras funcionalidades comuns nos IDEs para organizar e manter o fluxo de trabalho do programador. Um exemplo é a leitura do código em tempo real para localizar erros humanos e, dependendo do IDE, corrigi-los automaticamente, mantendo quem está escrevendo o código focado. A maioria dos IDEs também conta com o destaque da sintaxe do programa, quer dizer, pinta de cores diferentes as palavras reservadas da linguagem para facilitar a visualização. E ainda, como os ambientes de desenvolvimento são integrados, torna-se desnecessário usar outro programa para compilar ou executar o seu código, já que o próprio IDE possui essa funcionalidade integrada.

Apesar de exigir um conhecimento avançado em programação, existe a possibilidade de um desenvolvedor fazer seu próprio IDE juntando utilitários como compilador, editor

e *debugger*. A vantagem de fazer um IDE próprio é a grande possibilidade de personalização, permitindo que o desenvolvedor crie um IDE que atenda necessidades mais específicas.

Atualmente, existem IDEs específicos para atender a diversos tipos de uso como: técnicos, educacionais ou empresariais. As características desses IDEs se baseiam no nível de conhecimento do seu público-alvo. Assim, os principais tipos de IDEs são baseados em Blocos (*Block-Based*), outros em Quadros (*Frame-Based*) e alguns baseados em Texto (*Text-Based*).

Assim sendo, na sequência, os tipos de IDEs serão exemplificados, detalhados, e as vantagens e desvantagens de cada um serão apontadas.

BLOCK BASED

Uma das grandes dificuldades quando se começa a programar é o entendimento da sintaxe da linguagem utilizada. Nesse sentido, erros são comuns em razão da falta de intimidade com os comandos e notações. Por conta disso, a temática *Block-Based* simplifica a tarefa de programar apenas lógica, com comandos autoexplicativos organizados em blocos, além de ajudar com a familiaridade de diversos comandos, recorrentes durante o exercício da programação. Contudo, mesmo com essa simplificação de comandos, ainda existe o desafio lógico.

Os editores baseados em blocos têm foco totalmente na interação e visualização, como o *Scratch*, deixando de lado a parte da sintaxe da linguagem em si e dando mais atenção à ideia abstrata de programação e resolução de problemas. Desse modo, permite ao usuário gaste mais do seu tempo entendendo como a programação funciona ao invés de se preocupar com a sua sintaxe, dando menos chances aos erros de sintaxe, tais como parênteses e ";".

[Imagem 1] Exemplo de Scratch de um laço de repetição.

Com um pequeno exemplo em *Scratch* [Imagem 1], é possível mostrar as funcionalidades de um laço de repetição, além da definição de variáveis locais e da lógica proposicional, utilizando conectivos lógicos “Se” e “Então”.

Mesmo sendo o *Scratch* utilizado nos níveis introdutórios da programação, sua interface gráfica permite o desenvolvimento de códigos mais complexos e até mesmo jogos, como por exemplo o Pong.

Apesar de serem bastante intuitivos e possibilitarem o exercício do pensamento computacional para pessoas que nunca programaram, os IDEs baseados em blocos se mostram ineficazes ao longo do tempo, quando o objetivo é se tornar um programador. Quer dizer, esses IDEs, ao sacrificarem a maioria dos elementos de uma linguagem comum com o objetivo de simplificar a programação, acabam criando uma linguagem própria totalmente distinta, dificultando a transição para um IDE, como o *Text-Based*.

Em suma, uma linguagem *Block-Based* é adequada para a introdução aos conceitos básicos de lógica computacional, oferecendo uma dinâmica interativa ao código, mas que, para além disso, sempre será necessária a transição para uma metodologia mais avançada, utilizando IDEs baseados em texto.

A plataforma *Scratch* é totalmente gratuita, e, por não exigir nenhum conhecimento em programação, é ideal para iniciantes, podendo ser usada tanto com crianças quanto com adolescentes que querem começar no mundo da computação. Com essa ferramenta, as

possibilidades de exercícios e atividades básicas são quase ilimitadas como a criação de jogos, histórias animadas (o *Scratch* fornece funções que adicionam caixas de diálogos e personagens), labirintos a serem resolvidos com pensamento computacional e muito mais.

TEXT BASED

A metodologia *Text-Based* é a mais usada em ambientes profissionais de programação. É também a mais dinâmica e apropriada para toda e qualquer codificação. Por necessitar de conhecimento prévio da linguagem utilizada, cada uma com sua sintaxe e identidade própria, esse tipo de IDE é melhor aproveitado por seus usuários, após eles terem certo nível de experiência com programação.

Para a metodologia *Text-Based* não existe um IDE único ou algum que se sobressaia em comparação aos outros, é possível até mesmo programar em um simples arquivo de texto.

Em comparação com as outras metodologias apresentadas, a *Text-Based* não possui uma vantagem no ensino de programação. Mas, em compensação, as possibilidades que o completo uso desse tipo de IDE abre para o usuário são ilimitadas.

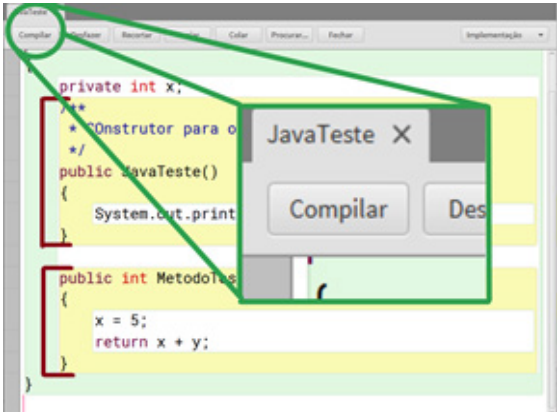
Depois que se familiarizar com os conceitos básicos comuns a todas as linguagens, não tendo como foco a memorização de comandos, mas sim o entendimento de suas funcionalidades, o usuário tem todas as ferramentas necessárias para o desenvolvimento de qualquer tipo de programa.

A ideia de se criar IDEs baseados em blocos ou *Framed* busca fornecer ao iniciante em programação um ambiente que facilite organizar e formatar código, agrupar diferentes comandos (que possuam a mesma finalidade) e ocultar os diferentes tipos de ações

associadas ao processo de compilação.

Dentre os editores baseados em texto pesquisados, destaca-se (para o aprendizado de Java) o IDE de programação *BlueJ*. Criado como um *software* para o auxílio no aprendizado de programação em Java, na universidade de *King's College London*, o *BlueJ* hoje é utilizado por universidades ao redor do mundo como uma ferramenta para assistir a visualização e abstração dos conceitos básicos de programação orientada a objetos (POO).

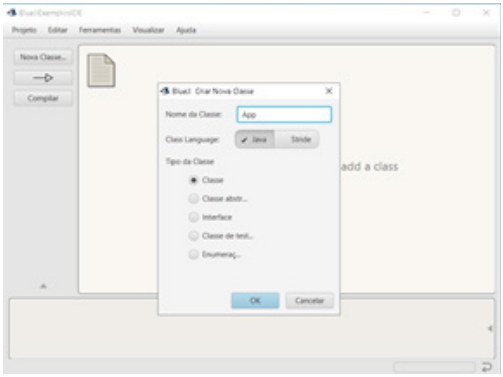
Aliás, o *BlueJ* pode ser utilizado como um IDE profissional de desenvolvimento de programas, por implementar todos os recursos que um IDE profissional possui, além das ferramentas de auxílio [Imagem 2] para iniciantes, como visualização de cada indentação (estrutura do código) com marcadores coloridos, melhor explicação nas mensagens de erro e visualização de diagramas de classe criados automaticamente.



[Imagem 2] Ilustra a forma com a qual o programa *BlueJ* torna a programação mais amigável para iniciantes.

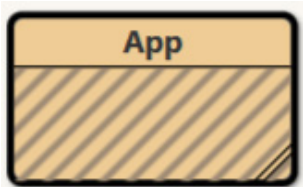
Dessa forma, são citadas abaixo algumas dessas funcionalidades:

- Ao invés de precisar definir por código a classe, o *BlueJ* oferece um menu com as opções mais comuns [Imagem 3], como por exemplo, Interfaces e Enumeradores. Além disso, o usuário pode escolher como deseja codificar: no modelo *Text-Based* do *BlueJ* básico ou no modelo *Frame-Based* do *Stride* (a ser analisado em seguida).

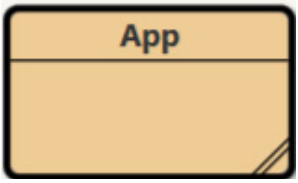


[Imagem 3] Criação de classes no *BlueJ*.

- O menu de relações de classes do *BlueJ* auxilia na visualização das classes integradas ao código, além de mostrar quais classes estão ou não compiladas e prontas para serem executadas [Imagem 4 & 5].



[Imagem 4] Uma classe não compilada.

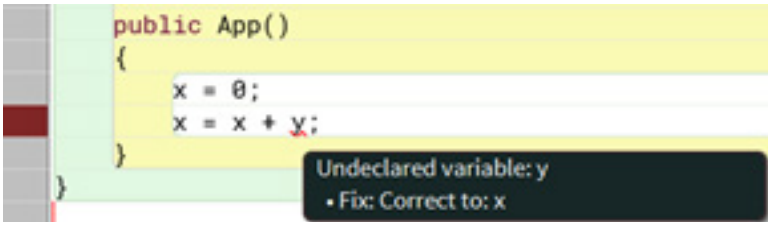


[Imagem 5] Uma classe compilada.

- A parte codificável da classe, no modelo de visualização do *BlueJ*, utiliza cores para definir a indentação de classes e métodos (em verde claro e amarelo, respectivamente [Imagem 6]).

[Imagem 6] Exemplo da indentação com cores do *BlueJ*.

- Quando ocorre um erro, o *BlueJ* marca em vermelho a linha, diz qual o problema e também aponta uma alternativa de solução. Contudo, como o *BlueJ* apenas sugere uma solução, é preciso estar atento: essa pode não ser a desejada. Por exemplo: na [Imagem 7], o usuário deseja realizar a soma de duas variáveis (**x** e **y**) e guardar o resultado em **x**. O *BlueJ* sugere que a variável **y** seja trocada por **x**, porém isso não resolveria o problema, visto que não se estaria fazendo a soma de duas variáveis distintas, e sim elevando uma delas ao quadrado. Nesse caso, para corrigir o erro, é preciso definir propriamente a variável **y** [Imagem 8].



[Imagem 7] Parte do código com erro.

```
public App()
{
    x = 0;
    int y;
    y = 1;
    x = x + y;
}
}
```

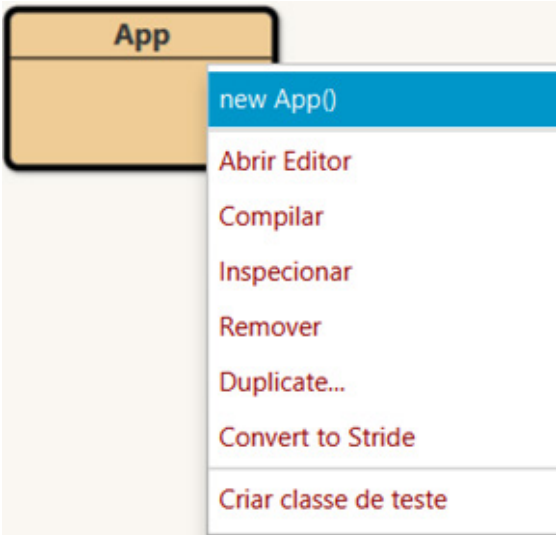
[Imagem 8] Variável devidamente definida.

- A linguagem utilizada pelo *BlueJ*, Java, permite exportar pacotes com mais funções para auxiliar no desenvolvimento do código. Ao invés de precisar elevar um número da forma básica, multiplicando-o por ele mesmo, é possível chamar métodos exportados pelo pacote *Math* para auxiliar, nesse caso, o método *Pow*, que recebe como argumento o número que desejamos elevar a certa potência e a potência desejada. Todo pacote possui um manual de instruções, mostrando como utilizar cada método, além de suas funcionalidades [Imagem 9].

```
import java.lang.Math;
public class App
{

    /**
     * COnstrutor para objetos da classe App
     */
    public App()
    {
        double x = Math.pow(4,2);
        double y = Math.pow(3,2);
        x = Math.sqrt(x+y);
    }
}
```

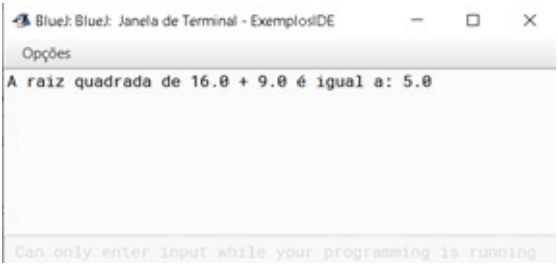
[Imagem 9] Uma função que utiliza os métodos do pacote *Math* para elevar ao quadrado e tirar a raiz quadrada da soma de duas variáveis.

- Após compilar o código, pode-se, então, rodar o programa. Para isso, no menu de classes, é necessário selecionar o método principal no qual foi desenvolvido o programa, como no exemplo a seguir [Imagem 10].



[Imagem 10] Classe compilada e pronta para ser executada ao selecionar o comando "*new App* ()".

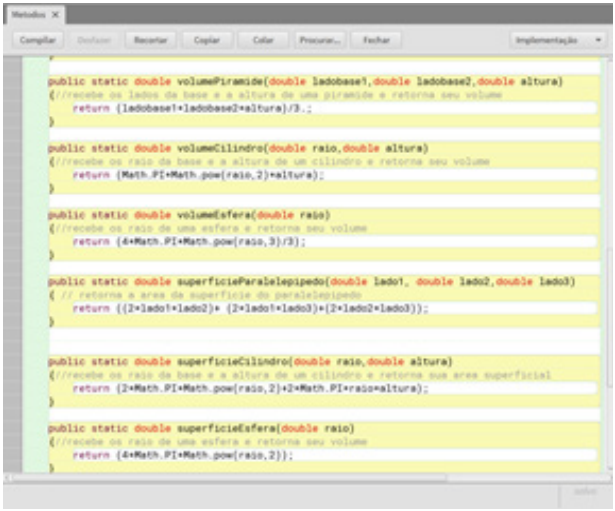
- Se não possuir mais erros, será aberta uma nova janela com o funcionamento do código [Imagem 11].



[Imagem 11] Janela de terminal criada pelo *BlueJ*.

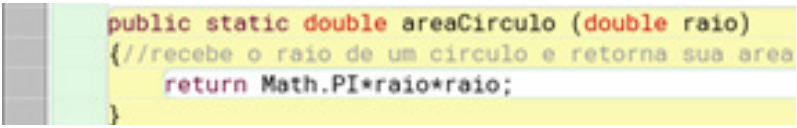
Além de tudo, pode-se utilizar a programação para auxiliar em problemas nas escolas, por exemplo:

- Uma biblioteca própria que auxilia nas equações de áreas de diversas formas geométricas [Imagem 12].

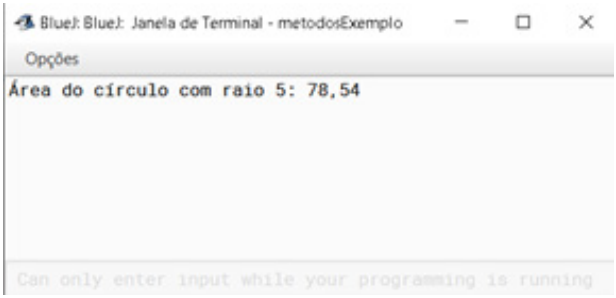


[Imagem 12] Biblioteca personalizada com equações geométricas, cedidas pela professora Lúcia Giraffa.

- Com essa biblioteca, além do exercício da programação, também podemos, por exemplo, calcular o valor da área de um círculo com raio igual a 4 [Imagem 13 e 14].



[Imagem 13] Equação da área de um círculo, utilizando o pacote *Java Math* para obtermos o valor de Pi através do método *Math.PI*.



[Imagem 14] Resultado da chamada do método área Círculo passando por parâmetro o valor 5.

Por fim, cabe ressaltar que, apesar de existirem ferramentas de ensino e serem ótimos para programação avançada, os IDEs *Text Based* não são recomendados para iniciantes na programação, por possuírem convenções complicadas e não serem intuitivos o suficiente para existir um desenvolvimento pleno do pensamento computacional. De mais a mais, a transição de IDEs baseados em blocos para os baseados em texto é extremamente complicada. Isso porque, ao passar de um IDE sem sintaxe nenhuma para linguagens cheias de convenções e detalhes, diminui o interesse de muitos alunos por parecer complexa demais, vindo de um IDE baseado em blocos. Tais dificuldades geram a necessidade de um tipo de IDE que acomode os alunos nessa transição. Isto é, algum IDE que se acomode entre *Block-Based* e *Text-Based*. Esses IDEs já existem e se chamam *Frame-Based*.

FRAME BASED

A metodologia *Frame-Based* tenta mesclar o melhor dos dois mundos, permitindo praticar a sintaxe básica do *Text-Based* com a praticidade da *Block-Based*. Isso, certamente, torna o aprendizado de programação mais leve e confortável e possibilita uma melhor transição entre IDEs básicos e avançados.

Nesse sentido, será usado nesta seção, como exemplo, o IDE *Stride*, que é integrado como uma modificação ao *BlueJ*, citado anteriormente.

Além disso, algumas dificuldades que estudantes tendem a enfrentar durante a transição de uma IDE *Block-Based* para uma *Text-Based* foram enumeradas por Michael Kölling, idealizador do *BlueJ*. De acordo com ele, tais dificuldades são:

1. **Legibilidade reduzida:** por conta da complexidade reduzida dos IDEs baseados em blocos, os baseados em texto tornam-se assustadores por serem muito mais complexos e totalmente diferentes.

2. **Paradigmas de programação diferentes:** para cada paradigma de programação existe uma organização do código que obedece a uma sintaxe e a um conjunto de regras, as quais, muitas vezes, são complexas para o iniciante em programação. Portanto, a ideia do sistema de representação *Frame Based* é auxiliar a "esconder" parte disso e fazer com que o programador se atende aos conceitos e não apenas à forma.

3. **Manusear layouts:** manusear um código, antes da transição, era muito mais visível e organizável. Com o novo formato podem ocorrer confusões recorrentes, por conta da maior dificuldade de organização do código.

4. **Mensagens de erros:** blocos mal encaixados ou *loops* infinitos tornaram-se uma dor de cabeça maior com a nova sintaxe, ocasionando erros muito mais comuns e difíceis de entender por serem mais complexos.

5. **Sistema de digitação:** ao trabalhar com blocos, a única parte em que existe digitação é no momento de definir o valor de variáveis ou caixas de diálogo. Já nas baseadas em texto, como o nome sugere, a dependência da digitação é total, visto que não existem blocos de encaixar.

6. **Entendimento de elementos:** outro problema que surge é na hora de entender como diversas partes dos IDEs baseados em texto se conectam para formar declarações compostas.

7. **Adequação:** também é necessária a adequação de uma chamada de método, definindo, assim, cada característica desse método para que possa ser chamado em qualquer parte do código. Algo que não foi trabalhado nos IDEs baseados em blocos.

8. **Maiores possibilidades:** comparado ao *Block-Based*, o número de comandos disponíveis pode abrir diversas portas ao programador, como também fazer surgir mais dúvidas.

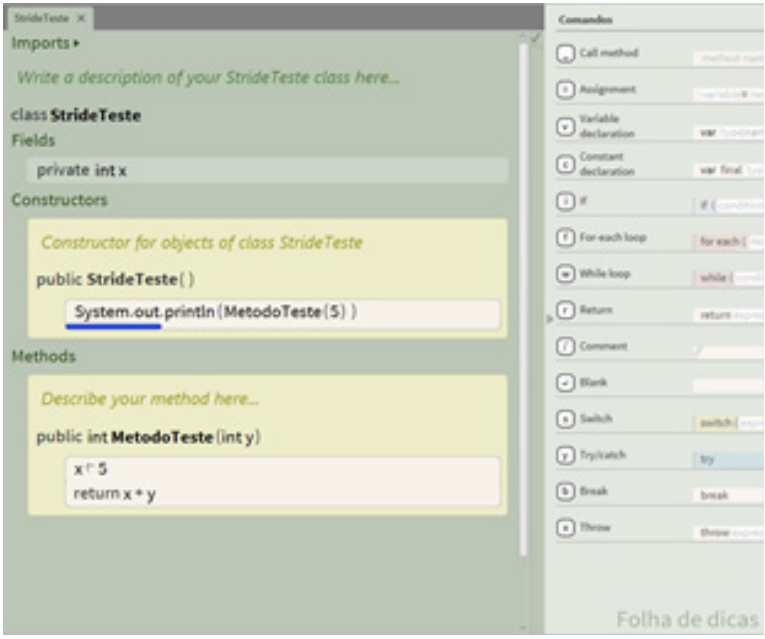
9. **Habilidade de escrita e digitação:** para uma melhor fluência na prática de programação, é necessária uma certa prática com escrita de código, e no idioma inglês, predominante nas linguagens de programação.

10. **Memorização:** ao invés da possibilidade de selecionar um comando através de uma tabela, é necessário memorizar tais comandos e sua sintaxe, além da forma de aplicação.

Em resumo, a partir de tais dificuldades emergiram duas questões: como auxiliar os estudantes nessa transição? Qual seria uma alternativa para a incorporação da temática *Block-Based* para um IDE puramente *Text-Based*?

Dessa maneira, no sentido de atender às necessidades demandadas por intermédio desses questionamentos, o *Stride* foi desenvolvido. Mas, afinal, o que é o *Stride*?

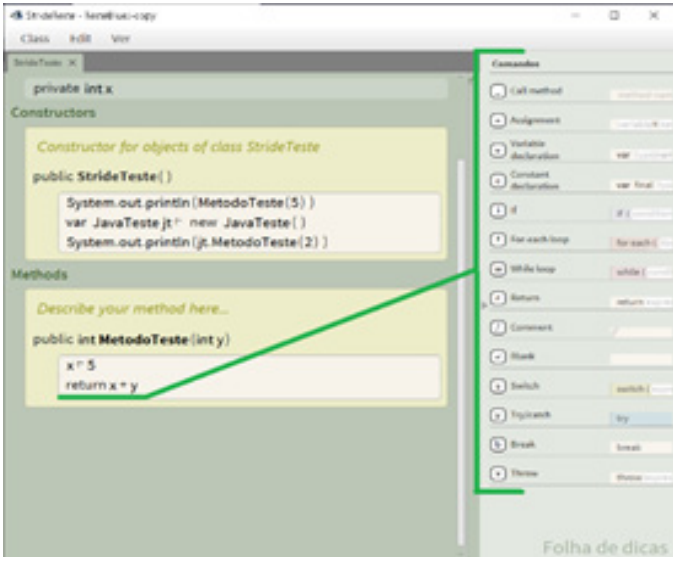
Stride é uma linguagem de programação baseada em Java, que utiliza um editor de código *Frame-Based*, permitindo uma melhor compreensão do código e da sua funcionalidade, deixando de lado pequenos erros de sintaxe (que normalmente ocorrem com aqueles que estão sendo introduzidos ao ensino de programação) e reduzindo, assim, a complexidade que envolve decorar pequenos detalhes da linguagem.



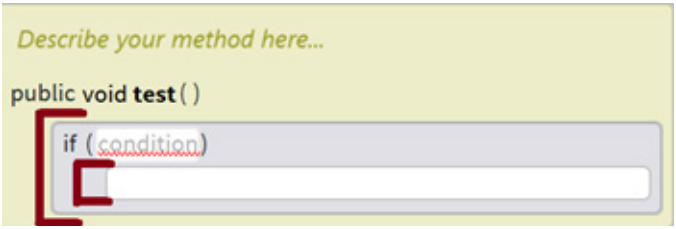
[Imagem 15] Visualização de uma classe em *Stride*.

Nesse caso, algumas funcionalidades únicas ao Stride requerem ser citadas:

- Indentação do código dividida por blocos coloridos;
- Uma “Folha e Dicas” que auxilia na escrita do código [Imagem16];
- Autocorretor de erros de sintaxe;
- Escopo dos métodos sempre visíveis para auxiliar na visualização das possibilidades.



[Imagem 16] Folha de dicas do *Stride*.



[Imagem 17] *Stride* torna o ambiente mais visual e fácil de organizar

Tais funcionalidades tornam o *Stride* ideal para a transição entre IDEs básicos (baseados em blocos) para mais complexos (baseados em texto), contornando muitas dificuldades encontradas e fazendo com que ocorra uma familiarização mais suave, potencializando, assim, a compreensão de linguagens mais complexas baseadas em texto, sem assustar os alunos

com barreiras tão altas para se escalar. Por exemplo:

[Imagem 18] Em Java, na IDE *BlueJ*, um código que define a equação da fórmula da área de um círculo, note a sintaxe necessária para dividir os comandos e assim eles serem aceitos para leitura.

O *Stride* auxilia na construção do código. Isto é, ao invés de precisar definir entre colchetes a indentação do código, são criados blocos para cada um dos campos fundamentais desse código, sendo o bloco "*Methods*" [Imagem 19] o principal deles, no qual é realizado a maior parte do código.

A seguir, está o comparativo do mesmo trecho de código em *BlueJ* e em *Stride*. No formato *Text-Based* Java do BlueJ é necessário definir a indentação do código à mão, algo muito mais suscetível a erros, além de precisar dos importes, caso sejam usados métodos de pacotes externos.

[Imagem 19] Em *Frame-Based*, no *Stride*, um código que define a equação da fórmula da área de um círculo, sem precisar definir os escopos de cada método, apenas o que será realizado em cada área.

CONCLUSÃO

Durante a pesquisa foram notadas diversas abordagens sobre como deve ser iniciado o ensino de programação para iniciantes, considerando o paradigma OO.

Nesse sentido, têm se a edição de programas baseados em texto, como *BlueJ* e *Greenfoot*, em que o foco é tornar o ambiente de programação mais visual e interativo, sem renunciar às convenções da linguagem. Já os editores baseados em blocos têm foco totalmente na interação e visualização, como o *Scratch*.

Com isso, observa-se o objetivo de cada IDE, indo de ensino introdutório (*Block-Based*) até o foco no desenvolvimento de programas mais avançados (*Text-Based*), com os IDEs de tipo *Frame-Based*, oferecendo um meio termo entre esses, para facilitar o aprendizado sem ter um pulo de complexidade muito grande.

Assim sendo, é recomendável utilizar um IDE de metodologia *Block-Based* para a introdução aos conceitos de programação, fazendo a transição para *Text-Based* aos poucos, podendo utilizar uma *Frame-Based* para auxiliar em tal transição.

Mesmo havendo melhores meios para o ensino dos termos iniciais, a introdução direta ao *Text-Based*, de preferência utilizando *BlueJ*, pode também ser uma alternativa, pois auxilia o aluno numa das partes mais complexas da transição, a memorização dos comandos e sintaxes, sem excluir as partes fundamentais, tais quais o funcionamento de um código, hierarquia de classes e a

orientação a objetos, já que é de prática comum a introdução à programação ser feita utilizando Java.

No caso de não existir conhecimento algum em programação, iniciar com IDEs *Text-Based* pode ter um efeito negativo por fazer parecer que programar é algo muito desafiador. Por isso, é recomendado o uso de ambientes baseados em blocos, tanto para crianças quanto para leigos sobre o assunto.

Desse modo, quando houver uma necessidade de mudar de IDE, para que não existam dificuldades nessa transição, o uso de ambientes *Frame-Based* é fundamental, já que inicia uma familiarização com ambientes baseados em texto de forma auxiliada e visual, derrubando barreiras no aprendizado de programação e suavizando a curva de aprendizagem.

BIBLIOGRAFIA

- ALTADMRI, Amjad; KÖLLING, Michael; BROWN, Neil C. C. "The Cost of Syntax and How to Avoid It: Text versus Frame-Based Editing," 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), Atlanta, GA, 2016, pp. 748-753. https://doi.org/10.1109/COMPSAC.2016.204.

- AURELIANO, Viviane Cristina Oliveira; TEDESCO, P. C. A. R. Avaliando o uso do Scratch como abordagem alternativa para o processo de ensino-aprendizagem de programação. In: XX Workshop sobre Educação em Computação. 2012. p. 10.

- KÖLLING, Michael. 2010. The Greenfoot Programming Environment. ACM Trans. Comput. Educ. 10, 4, Article 14 (November 2010), 21 pages. https://doi.org/10.1145/1868358.1868361.

- KÖLLING, Michael *et al*. (2019). Stride in BlueJ - Computing for All in an Educational IDE. 63-69. https://doi.org/10.1145/3287324.3287462.

- KÖLLING, Michael; BRONW, Neil C. C.; ALTADMRI, Amjad. 2015. Frame-BasedEditing: Easing the Transition from Blocks to Text-Based Programming. InPro-ceedings of the Workshop in Primary and Secondary Computing Education (WiPSCE'15). ACM, New York, NY, USA, 29–38. https://doi.org/10.1145/2818314.2818331.

- MALONEY, John *et al*. 2010. The Scratch Programming Language and Environment. ACM Trans. Comput. Educ. 10, 4, Article 16 (November 2010), 15 pages. https://doi.org/10.1145/1868358.1868363.

- UTTING, Stephen Cooper an *et al*. 2010. Alice, Greenfoot, and Scratch - A Discussion. ACM Trans. Comput. Educ. 10, 4, Article 17 (November 2010), 11 pages. https://doi.org/10.1145/1868358.1868364.

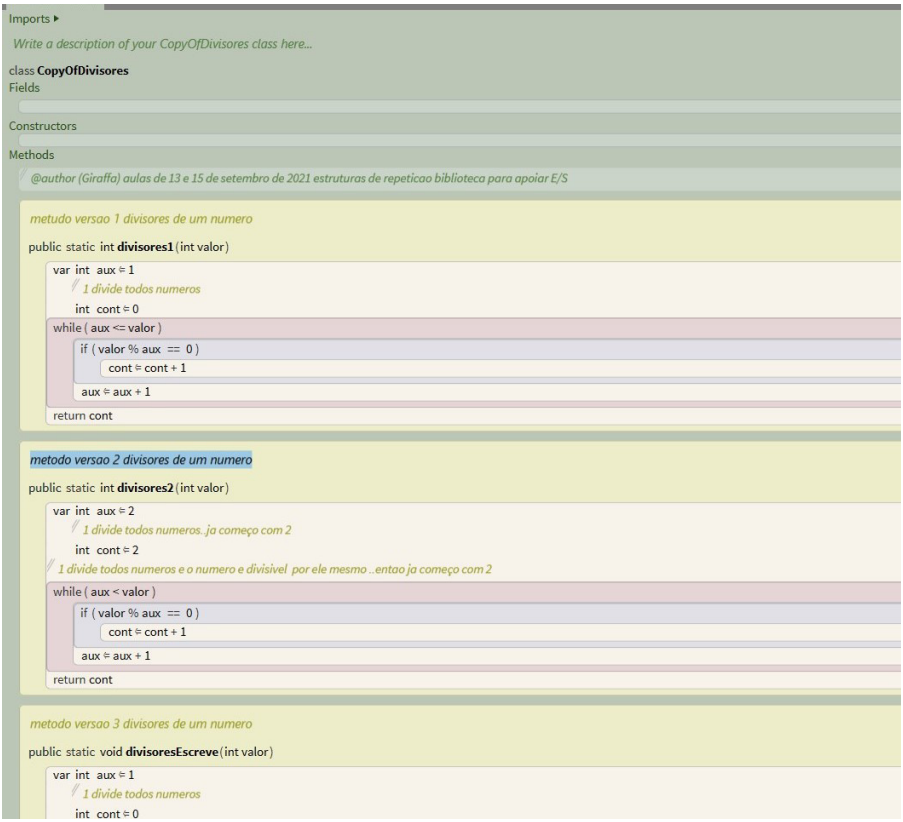
APÊNDICE

Código por extenso padrão texto:

```
import java.util.Scanner;//biblioteca para apoiar E/S
public class Divisores{

    public static int divisores1(int valor)
    {
    int aux=1;//1 divide todos numeros
    int cont=0;
        while (aux <=valor)
        {
         if(valor%aux==0) {cont++;}
         aux++;
        }
        return cont;
    }

    public static int divisores2(int valor)
    {
    int aux=2;//1 divide todos numeros..ja começo com 2
    int cont=2;//1 divide todos numeros e o numero e divisivel  por ele mesmo ..entao ja começo com 2
        while (aux <valor)
        {
         if(valor%aux==0) {cont++;}
         aux++;
        }
        return cont;
    }

    public static void divisoresEscreve(int valor)
    {
    int aux=1;//1 divide todos numeros
    int cont=0;
        while (aux <=valor)
        {
         if(valor%aux==0) {System.out.print( aux+ " ");}
         aux++;
        }
    }
```

```
        {
         if(valor%aux==0) {cont++;}
         aux++;
        }
        return cont;
    }

    public static void divisoresEscreve(int valor)
    {
    int aux=1;//1 divide todos numeros
    int cont=0;
        while (aux <=valor)
        {
         if(valor%aux==0) {System.out.print( aux+ " ");}
         aux++;
        }
    }


    public static void main(String args[])
    {
        Scanner teclado=new Scanner (System.in); // objeto teclado para poder acessar metodos da classe
        System.out.print("\f Iniciando a execucao do programa \n   ***************************\n");//"anulo" o efeito print usando \n
        int numero;
        //atencao para este trecho
        do{
        System.out.print("\n Digite um numero natural maior que zero: ");
        numero=teclado.nextInt();
        } while (numero<=0);

        System.out.println("\n A quantidade de divisores versao 1: \t"+ divisores1(numero));
        System.out.println("\n A quantidade de divisores versao 2: \t"+ divisores2(numero));
        System.out.println("\n Os divisores sao:");
        divisoresEscreve(numero);

    }
```

Versão do mesmo código no *Stride*:



```
public static int divisores2(int valor)
    var int aux = 2
        // 1 divide todos numeros..ja começo com 2
        int cont = 2
    // 1 divide todos numeros e o numero e divisivel por ele mesmo ..entao ja começo com 2
    while ( aux < valor )
        if ( valor % aux == 0 )
            cont = cont + 1
        aux = aux + 1
    return cont
```

```
metodo versao 3 divisores de um numero

public static void divisoresEscreve(int valor)
    var int aux = 1
        // 1 divide todos numeros
        int cont = 0
    while ( aux <= valor )
        if ( valor % aux == 0 )
            System.out.print(aux + " " )
        aux = aux + 1
```

```
chamando métodos

public static void main(String[ ] args)
    var Scanner teclado = new Scanner(System.in)
    // objeto teclado para poder acessar metodos da classe
    System.out.print( "\f Iniciando a execucao do programa \n ***************************\n" )
        // "anulo" o efeito print usando \n
    var int numero
    // atencao para este trecho
    System.out.print( "\n Digite um numero natural maior que zero: " )
    numero = teclado.nextInt( )
    System.out.println( "\n A quantidade de divisores versao 1: \t" + divisores1(numero) )
    System.out.println( "\n A quantidade de divisores versao 2: \t" + divisores2(numero) )
    System.out.println( "\n Os divisores sao:" )
    divisoresEscreve(numero)
```

Identificação do **BlueJ**, ambiente de programação utilizado para o trabalho apresentado nesta obra:

Disponível em: <https://bluej.org>

Obra produzida com apoio do
Conselho Nacional de Desenvolvimento
Científico e Tecnológico (CNPq),
através da Bolsa de Produtividade
em Pesquisa - PQ - Processo: 312864/2020-5.

CNPq
Conselho Nacional de Desenvolvimento
Científico e Tecnológico

EDITORA
& LIVRARIA
VECHER EDUCOM