IntelliJ IDEA: dicas e truques para usar no dia a dia

Akemi Alice
09/12/2022

Introdução

O IntelliJ é um ambiente de desenvolvimento integrado e com reconhecimento de contexto para trabalhar com **Java** e outras linguagens que rodam na **JVM**, como **Kotlin, Scala e Groovy**.

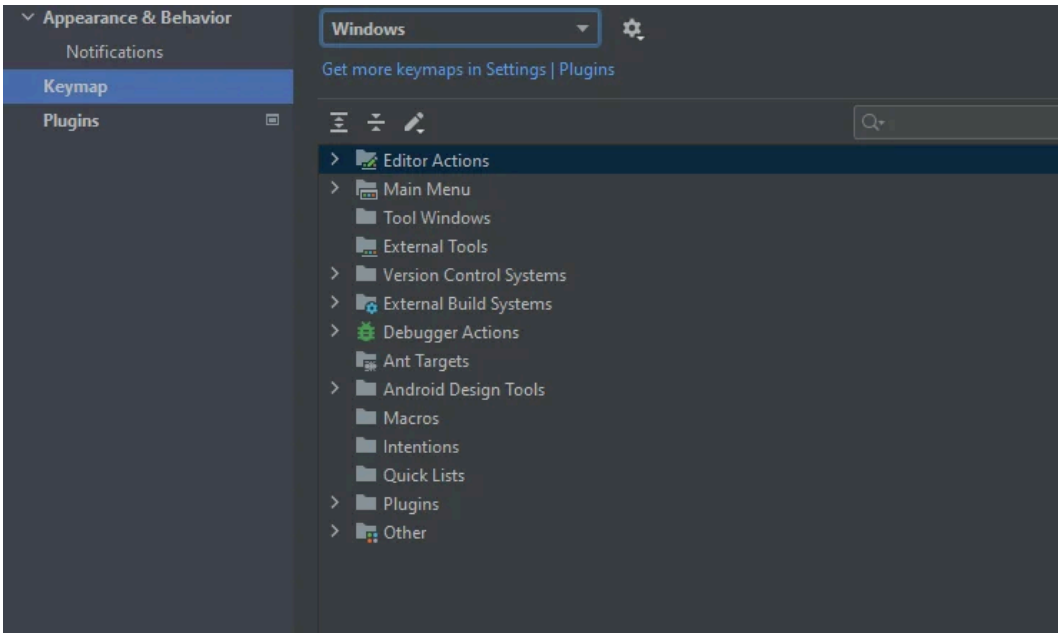
Um ponto importante a se pensar no momento da criação do código é a escolha da IDE. Por isso este artigo trará um **conjunto completo de informações sobre o Intellij IDEA**, vamos criar um novo projeto, navegar por ele, utilizar técnicas de refatoração e modo debug para analisar o código.

Aprenderemos também algumas dicas e truques que podemos usar no IntelliJ IDEA no cotidiano como pessoas desenvolvedoras, aumentando a produtividade no trabalho em projetos Java.

Configuração de atalhos: keymap

Um dos primeiros passos durante o uso do IntelliJ é a configuração do keymap, o mapa de atalhos que podemos utilizar durante o desenvolvimento com a IDE. O principal motivo para isso é conhecermos esses atalhos disponíveis, já que eles podem nos poupar tempo de trabalho.

A maneira tradicional de acessar o keymap é por meio do menu **File > Settings...**, então, filtramos por "keymap":



Keymap References

Além de verificar os atalhos por meio da opção "Settings", também podemos acessar o PDF que contém todos os atalhos de todos os sistemas operacionais selecionando "Help > Keymap Reference". Fique à vontade para consultar os atalhos da maneira que achar mais confortável.

Keymaps padrão entre os sistemas operacionais

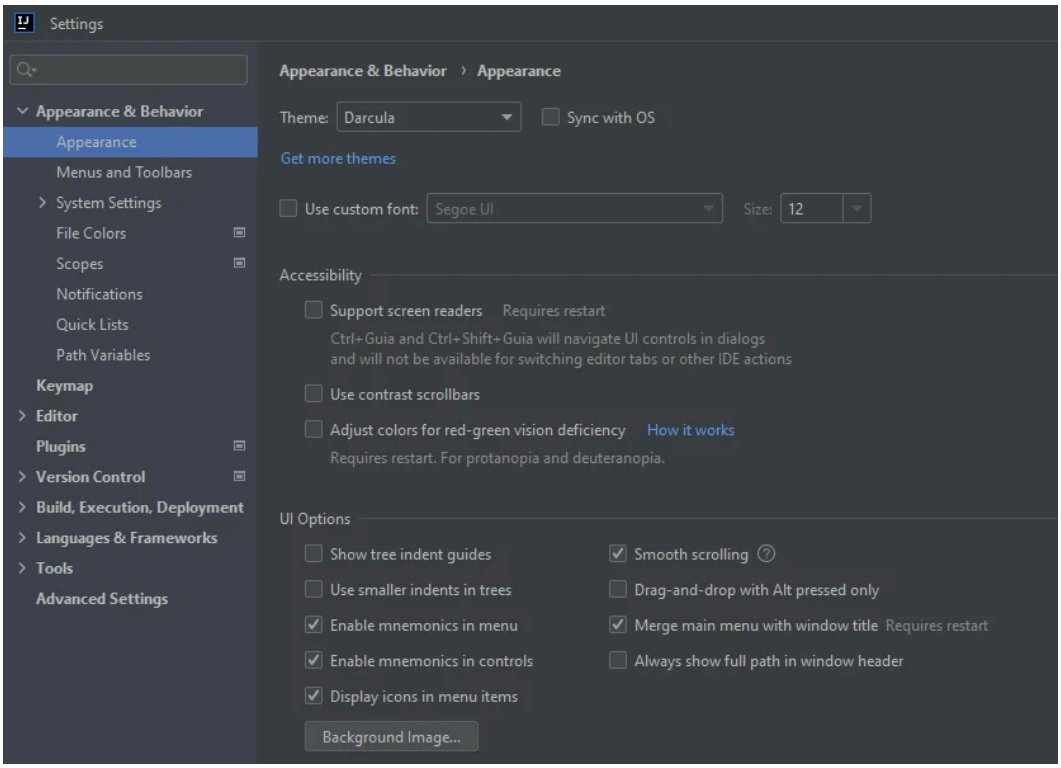
No geral temos os seguintes valores padrão, considerando os diferentes sistemas operacionais:

- Windows: Default
- Linux: Default for XWin
- Mac: Mac OS X 10.5+

Por padrão, quando apresentarmos os atalhos, vamos escrever primeiro como eles poderiam ser usados no Windows e depois, com uma barra, como podem ser usados no Linux ou Mac.

Modificando o tema

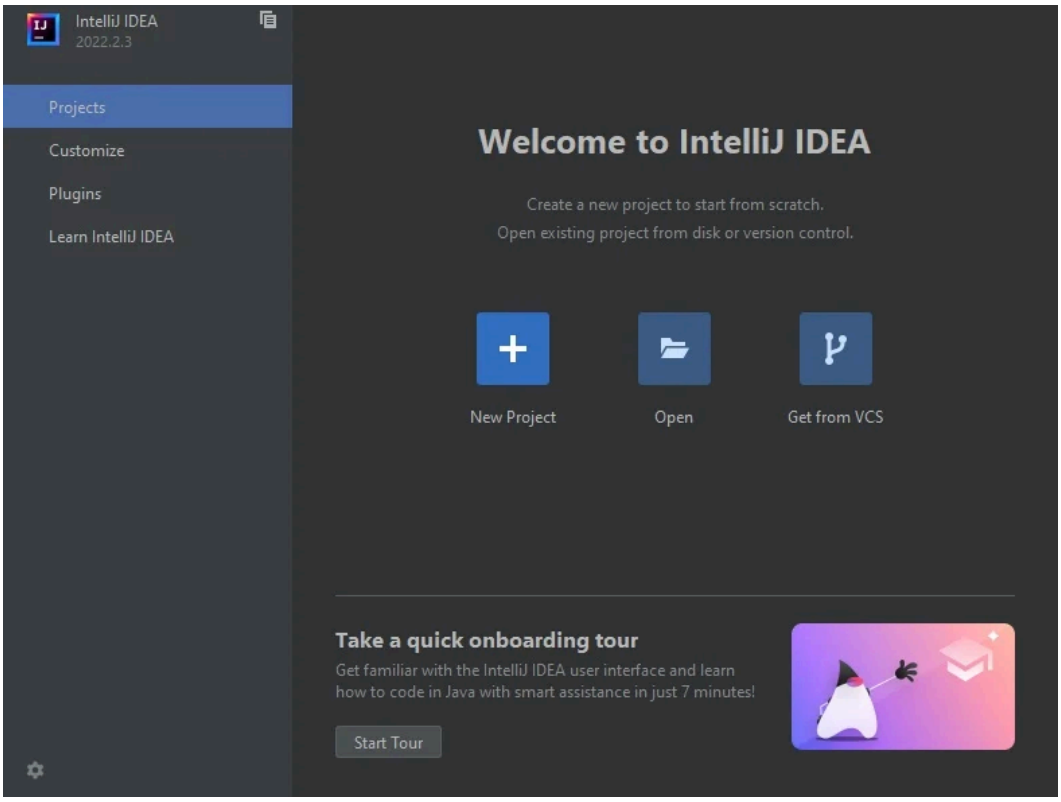
Uma configuração realizada por muitas pessoas antes de começar a escrever códigos é a personalização da aparência do IntelliJ. Para isso, selecione "**File > Settings**" e filtre por "**Appearance**":



Nessa janela, você pode fazer as alterações que achar mais adequadas para seu uso. No meu dia a dia, costumo manter o valor do "Theme" como **Darcula**.

Primeiros passos com a IDE: conhecendo os atalhos

Como estamos iniciando nosso desenvolvimento com a ferramenta, vamos selecionar a opção “New Project”:

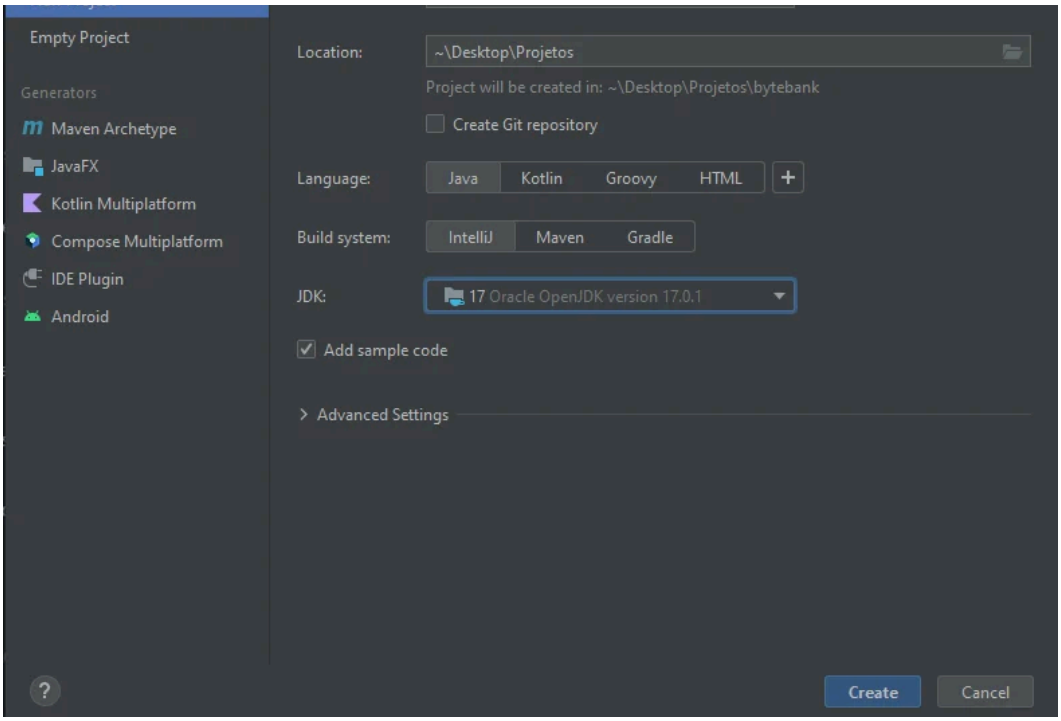


Assim uma nova janela será aberta, na qual podemos configurar o projeto. Vamos selecionar a opção "Java" nas opções de projeto.

Durante o artigo trabalharemos em um projeto de um banco digital, então vamos nomeá-lo no campo "Project name" como "**bytebank**". Em “Project location” fica o caminho onde se encontra nosso projeto.

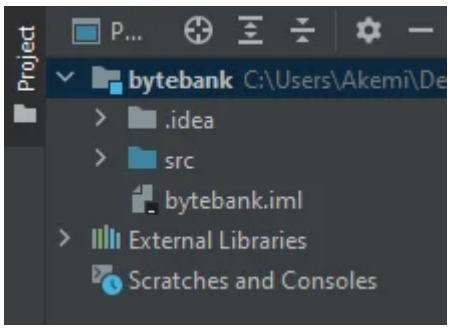
Em “Project SDK” deixamos a versão que está. Caso esta opção não apareça em seu projeto, basta clicar em "New" e selecionar o diretório de instalação do seu JDK.

Após essa configuração, basta clicar em "Create".



Conforme tudo carregar, teremos uma view chamada "Project" e poderemos visualizar toda a estrutura das pastas do projeto.

O primeiro diretório é o nosso, nomeado `bytebank`. Dentro dele há outro, o diretório `.idea` onde ficam todas as configurações do IntelliJ. Também temos a pasta `src` (source) onde colocaremos todas as nossas classes Java.

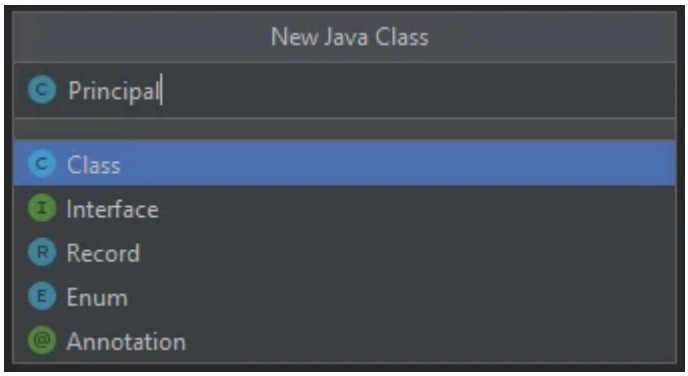


Para criar uma classe, bastaria clicarmos com o botão direito do mouse na pasta src e selecionar a opção "New" e em seguida "Java Class". Porém a grande vantagem da IDE é poder fazer os mesmos passos com atalhos. Vamos começar?

forma é possível ocultar ou mostrar a view. Após a visualização, uma boa opção é navegar pelos diretórios com as setas do teclado.

Selecionando a pasta `src` com os atalhos, podemos clicar **Alt + insert** para criar algum arquivo.Nesta pasta, o IntelliJ vai mostrar as opções de arquivos que faz sentido estarem nesse diretório.

Selecionando a opção "Java Class", colocaremos o nome da classe de "Principal".



Mas como faremos para executar a classe? Sabemos que para executar, precisamos de um método `main()`. Poderíamos escrever toda a assinatura do método, porém a IDE também consegue nos auxiliar, bastando escrever diretamente dentro de um arquivo. No caso, dentro da classe `Principal` escreva **"psvm"** e aperte a tecla **Tab** para que a assinatura seja escrita automaticamente:

```
public class Principal {

    public static void main(String[] args) {

    }
}
```

a mensagem "Bem vindo ao Bytebank".

```
public class Principal {

    public static void main(String[] args) {
        System.out.println("Bem vindo ao bytebank");
    }
}
```

Agora que temos a classe com o método `main()` pronta, vamos executá-la. Dentro da classe `Principal`, utilize o atalho **Alt + Shift + F10 / Ctrl + Shift + F10**Dessa forma, o IntelliJ vai interpretar que queremos executar a nossa classe atual.

Note que a princípio ele vai listar algumas opções de execução, portanto, escolha a classe `Principal`. O intelliJ executará e vai imprimir: "Bem vindo ao bytebank". Você já deu seus primeiros passos com o intelliJ!

Criando código com o `generate`

Inicialmente criamos a nossa classe `Principal` dentro do diretório raiz src. Porém, quando estamos desenvolvendo programas em Java, fazemos uso da convenção por pacotes que geralmente definem o domínio da nossa aplicação.

Sendo assim, vamos criar o pacote no nosso projeto para que a nossa classe Principal como também outras classes que iremos criar estejam organizadas.

Criando pacotes

Para criar os pacotes, acesse a view project **Alt + 1 / CMD + 1** e navegue até o diretório src, em seguida, utilize o generate **Alt + Insert / CMD + N**. Com a opção de New visível, digite p... (ou package) e observe que aparecerá a

Movendo a classes dentro do projeto

Agora que temos o pacote precisamos apenas mover a nossa classe `Principal` para dentro dele. Para isso, vá até a `view project` e navegue até a classe. Então utilize a feature “Move” do IntelliJ com o atalho *”F6”*.

Neste instante aparece a janela com o título “Move”, opção para mover arquivos dentro do projeto. Nesse caso, ele indica que estamos movendo a classe `Principal` . Defina o local de destino por meio do campo “to package” e coloque nele o valor do pacote que criamos, `br.com.alura.bytebank` . Em seguida, tecle “Enter”.

Observe que a classe foi movida para o pacote `br.com.alura.bytebank , o que é fácil de ser notado pela instrução *package* contida dentro do arquivo:

```
package br.com.alura.bytebank;

public class Principal {
  // código implementado
}
```

Agora, quando criarmos novas classes faremos uso desse pacote como raiz.

Interações com o IntelliJ dentro do código

Já organizamos nosso projeto, portanto, faremos com que o programa do Bytebank seja capaz de armazenar funcionários. Ou seja, vamos criar uma classe que represente um funcionário na vida real.

Abaixo do código que apresenta a mensagem de boas-vindas, digite "new Funcionario();". O IntelliJ vai reclamar, indicando que não conhece o símbolo

esperto o suficiente para entender que existe uma possibilidade de querermos criar uma classe.

Considerando a existência dessa e outras possibilidades, a IDE nos apresenta intenções de ação por meio da feature "***Show Intention Action***", ou melhor, por meio de suas sugestões.

Utilizando sugestões do IntelliJ

Para acessá-la, coloque o cursor em cima do símbolo que o IntelliJ não conhece, `Funcionario`, e utilize o atalho **Alt + Enter**.

Veja que aparece a opção `Create class 'Funcionario'`, tecle "Enter" e logo em seguida surgirá a possibilidade de definir o pacote no qual queremos que a classe seja criada. Isso acontece porque o IntelliJ não tem certeza absoluta que queremos criar uma classe no mesmo pacote.

Sendo assim, defina o pacote como `br.com.alura.bytebank.model`, pois o funcionário em si faz parte de um modelo do nosso projeto. Por fim, tecle "Enter".

O resultado após criar a classe `Funcionario` é o mesmo de quando criamos a classe `Principal`:

```
package br.com.alura.bytebank.model;

public class Funcionario {

}
```

A diferença é que desta vez temos podemos realizar menos passos para atingir essa solução.

Encapsulando atributos da classe

Para essa classe vamos considerar os atributos de nome, matrícula e data de nascimento. A princípio, `Funcionario` vai ficar assim:

```
public class Funcionario {

    String nome;
    int matricula;
    LocalDate dataNascimento;
}
```

De acordo com o princípio da orientação a objetos devemos de fato fazer com que os atributos das classes sejam encapsulados, evitando o acesso direto. Dessa forma evitamos comportamentos indesejados.

Assim sendo, precisamos deixar o modificador de acesso como `private`, ou seja, privado para cada um dos atributos em primeiro lugar. Porém, ao escrever o trecho `private` para cada um deles, percebemos que essa não é uma tarefa proveitosa, pois temos apenas que repetir o procedimento.

Alternando o modo de seleção para coluna

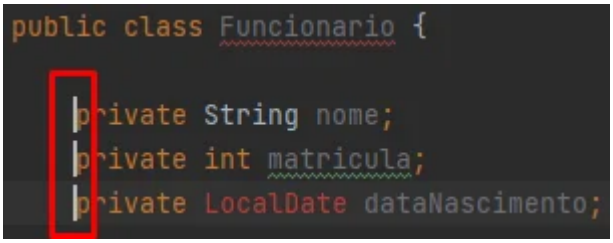
Sendo assim, vamos modificar o modo de seleção para **coluna**, tornando possível selecionar uma coluna inteira e modificar todos os pontos dela de uma vez. Para utilizar esse modo de seleção clique no atalho **Alt + Shift + Insert / CMD + Shift + 8**.

Perceba que na parte inferior direita da tela, o IntelliJ apresentará a mensagem “Column”.

Antes:

Depois:

Isso indica que o modo de seleção por coluna foi ativado. Agora selecione com o mouse ou com as setas + Shift a coluna que deixa o cursor na frente de todos os atributos, algo próximo do exemplo abaixo:



Agora quando escrever, note que o "private" será escrito para todos. Após realizar essa edição utilize o mesmo atalho para alternar o modo de seleção por linha que é o mais comum durante a edição de código de modo geral:

```
public class Funcionario {

    private String nome;
    private int matricula;
    private LocalDate dataNascimento;
}
```

métodos de acesso, isto é, tanto os getters como também os setters.

Generate

Considerando esse contexto, temos a possibilidade de agilizar o processo com o `generate`, por meio do atalho **(Alt + Insert / CMD + N)**. Dentro da classe `Funcionario` use o `generate` e escolha a opção "Getter and Setter", então, selecione todos os atributos e tecle "Enter".

Após realizar todos os passos, teremos uma classe `Funcionario` da seguinte maneira:

```
public class Funcionario {

    private String nome;
    private int matricula;
    private LocalDate dataNascimento;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public int getMatricula() {
        return matricula;
    }

    public void setMatricula(int matricula) {
        this.matricula = matricula;
    }
```

```
public void setDataNascimento(LocalDate dataNascimento) {
    this.dataNascimento = dataNascimento;
}
```

Conhecendo o find action

Agora que temos o nosso funcionário pronto, vamos utilizá-lo, voltando para a nossa classe `Principal`. Porém, em vez de usar a aba “Project”, vamos acessá-la diretamente.

Já que faremos um acesso direto à classe, não faz sentido mantermos a janela da “view project”, portanto, podemos escondê-la.

Escondendo todas as views

Novamente, poderíamos usar o mesmo atalho que acessa a view (**Alt + 1 / CMD + 1**), pois faz parte de um atalho alternativo (abre ou fecha). Entretanto, também podemos esconder todas as telas de uma view com o atalho **Ctrl + Shift + F12 / CMD + Shift + F12**.

Buscando classes

Agora, para acessar a classe `Principal` vamos utilizar o recurso de busca de classe do IntelliJ. Usaremos o atalho **Ctrl + N / CMD + O**, então comece a digitar "Pri" e note como a ferramenta já vai filtrar a classe `Principal`. Tecle Enter e observe que o IntelliJ já vai direto para a classe `Principal`.

Dentro do método `main()` adicione o funcionário “João” com as seguintes informações:

- Nome: João
- Matrícula: 1
- Data de nascimento: 12/02/1990

Em outras palavras, para criar e adicionar as informações de um funcionário simultaneamente, usaremos construtores personalizados que recebam todos os parâmetros necessários.

Sugestões do IntelliJ: Adicionando construtores personalizados

Podemos, por exemplo, adicionar os parâmetros no momento da instância:

```
Funcionario jose = new Funcionario("José", 1, LocalDate.of(1990, 2
```

Então, usamos as sugestões do IntelliJ (*"Alt + Enter"*) e escolhemos a opção "Create constructor".

Embora funcione, perceba que o construtor criado não faz um processo de *bind* automaticamente, isto é, vincular os valores recebidos pelo construtor com os atributos da classe.

Implementações personalizáveis com o generate

Pensando justamente em facilitar esse processo, podemos também usar recursos que já vimos, como o próprio `generate` (**Alt + Insert / CMD + N**).

Para isso, acesse novamente a classe `Funcionario` e dentro dela use o `generate`.

Criando construtor

Em seguida, escolha a opção "Constructor" selecione todos os campos pressionando o "Shift + setas" ou o clique do mouse e tecle "Enter".

classe `Funcionario`.

Por fim, faça a impressão do funcionário criado na classe `Principal` e veja se tudo está funcionando.

Após a implementação de todos os passos, teremos o seguinte resultado:

```
public class Principal {
    public static void main(String[] args) {
        Funcionario jose = new Funcionario("José", 1, LocalDate.of
        System.out.println(jose);
    }
}
```

Classe Funcionario:

```
public class Funcionario {

    private String nome;
    private int matricula;
    private LocalDate dataNascimento;

    public Funcionario(String nome, int matricula, LocalDate dataN
        this.nome = nome;
        this.matricula = matricula;
        this.dataNascimento = dataNascimento;
    }

    public String getNome() {
        return nome;
    }
```

```
    public int getMatricula() {
        return matricula;
    }

    public void setMatricula(int matricula) {
        this.matricula = matricula;
    }

    public LocalDate getDataNascimento() {
        return dataNascimento;
    }

    public void setDataNascimento(LocalDate dataNascimento) {
        this.dataNascimento = dataNascimento;
    }

    @Override
    public String toString() {
        return "Funcionario{" +
                "nome='" + getNome() + '\'' +
                ", matricula=" + getMatricula() +
                ", dataNascimento=" + getDataNascimento() +
                '}';
    }
}
```

Resultado após a execução da classe `Principal` :

```
Funcionario{nome='José', matricula=1, dataNascimento=1990-02-12}
```

código

Para praticarmos, vamos manipular o seguinte código:

```
import java.util.Scanner;
public class Saldo{ public static void main(String[] args) {Scanne
            System.out.println("Digite o saldo:");
    var saldo = leitura.nextDouble();double p = saldo * (10.0 / 10
  var valor = saldo + p;System.out.println("O saldo com o reajuste
```

Observe que inicialmente é bem difícil entender o que essa classe faz. Precisamos realizar alguns processos de refatoração para que a leitura do código seja mais compreensível para quem desenvolve.

Copiando uma classe

Antes de começarmos com as técnicas de refatoração, faremos uma cópia da `Saldo` para comparar com o resultado final nos exercícios a seguir. Coloque o cursor do mouse sobre o nome da classe ou a selecione na view project e use o atalho **"F5"**.

Em seguida, nomeie a cópia como **SaldoCopia** e deixe-a no mesmo pacote. Por fim, tecle "Enter" ou clique em **OK**. Pronto! Conseguimos copiar. Agora podemos começar com o nosso processo de refatoração na classe `Saldo`

Formatando o código automaticamente

A primeira das técnicas que podemos aplicar nesse instante é a famosa formatação automática do código (o que no IntelliJ é conhecido como **Reformat Code**). Com ela nós ajustamos a indentação. Para formatar o código basta usar o atalho **Ctrl + Alt + L / Alt + CMD + L**.

ficar legíveis para qualquer pessoa que programe. Em outras palavras, realize a seguinte mudança:

- `valor` para `valorFinal`
- `p` para `percentual`

Para realizar esse ajuste visando o processo de refatoração, podemos utilizar o **Rename** do IntelliJ com o atalho **Shift + F6**.

Com as alterações realizadas, a classe `Saldo` terá o seguinte código:

```
import java.util.Scanner;

public class Saldo {
    public static void main(String[] args) {
        Scanner leitura = new Scanner(System.in);
        System.out.println("Digite o saldo:");
        var saldo = leitura.nextDouble();
        double percentual = saldo * (10.0 / 100);
        var valorFinal = saldo + percentual;
        System.out.println("O saldo com o reajuste de 10% é: " + v
    }
}
```

Desse modo fica mais fácil compreender que estamos lidando diretamente com pagamentos, pois não temos mais aquelas variáveis que não deixam claro para que servem.

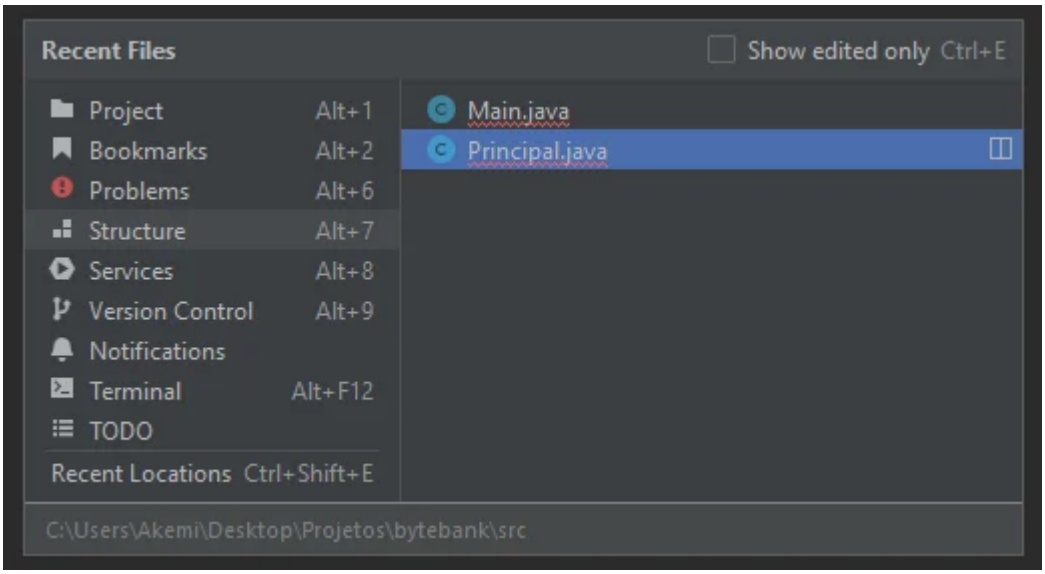
Search Everywhere

O Search Everywhere permite buscarmos qualquer coisa dentro do projeto, seja, classes, arquivos, símbolos e configurações.

Visualizar arquivos recentes

Isto pode ser útil em cenários que passamos por diversos arquivos e temos muitas abas, podendo visualizar aqueles últimos que foram abertos.

Ctrl + E / CMD + E



Busca de símbolos

Além da busca de qualquer item (“search everywhere”) e busca de arquivos, também podemos fazer uso da busca de símbolos. De forma geral, fazemos uso desse tipo de busca quando queremos encontrar classes, interfaces ou membros de ambas, isto é, atributos ou métodos.

Ctrl + Shift + Alt + N / CMD + Alt + O

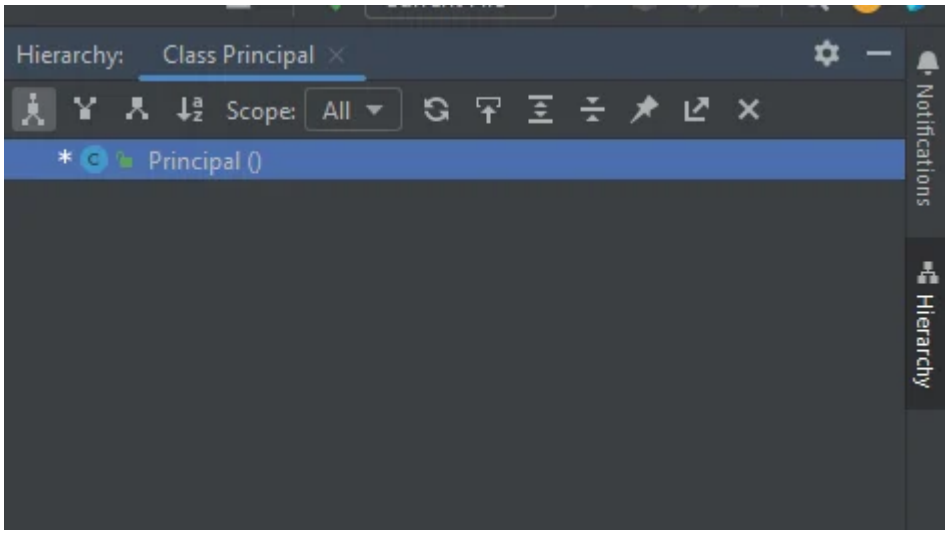
atalho:

Ctrl + Shift + F / CMD + Shift + F

Visualização de hierarquia

Entre dentro de uma classe qualquer e utilize o atalho “Ctrl + H”. Observe que será aberta a view “Hierarchy”, que representa a hierarquia das nossas classes.

Ctrl + H



Find Usages

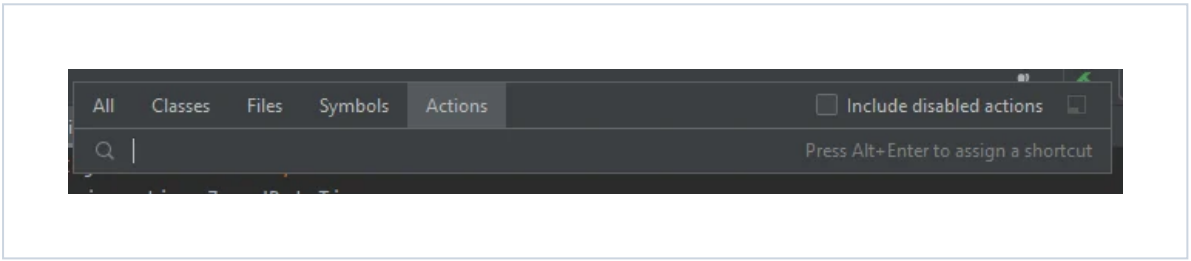
Por meio do “Find Usages” temos a capacidade de saber onde o nosso código está sendo utilizado.

Alt + F7

Find Action

Este é um recurso que permite buscar qualquer funcionalidade do IntelliJ. O atalho para ele é:

Ctrl + Shift + A / CMD + Shift + A.



Copiando linhas

Conseguimos copiar linhas teclando o seguinte atalho:

Ctrl + D / CMD + D

Navegação por declaração

O recurso “declaration” verifica a implementação do código por meio do atalho. Com ele navegamos diretamente até o trecho em que o código foi implementado.

Ctrl + B / CMD + B

Introduce to Local Variable

Como vimos, podemos tomar ações conforme as sugestões da IDE, outra função interessante é a Introduce to Local Variable, ela sugere a introdução da variável e depois é só nomeá-la.

Testando e debugando código

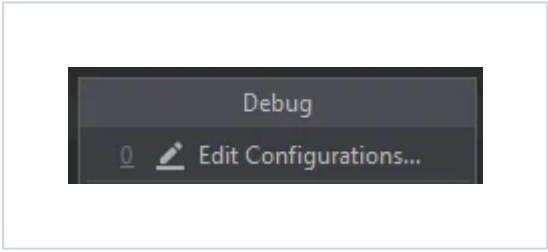
Uma das técnicas comuns para verificar o comportamento do nosso código é a execução em modo **debug**. Com ela temos a capacidade de inspecionar o nosso código a ponto de verificar o resultado de execução linha a linha.

Quando queremos executar nesse modo, precisamos indicar em que momento do nosso código queremos realizar a inspeção. Para isso, fazemos uso de **breakpoints**. Para marcar o breakpoint, clique no lado direito do número da linha do código que deseja inspecionar com o botão esquerdo do mouse.

Então, a partir do momento que executar em modo debug, observe que o IntelliJ apresenta uma view de debug.

Modo debug

Para executar o modo debug, fazemos o mesmo procedimento usado para as execuções, o "Run...,". Porém, mudamos o atalho final, portanto, use o seguinte atalho **Alt + Shift + F9 / Ctrl + Shift + D**.



O IntelliJ vai parar exatamente no ponto do breakpoint e é possível utilizar as funcionalidades disponíveis para este modo.

Step into

A primeira delas é justamente o "Step Into", que executa cada trecho do código. Fazemos isso usando o atalho **F7**.

Esse modo é muito eficaz no ponto do código que queremos entender detalhe por detalhe do que está acontecendo.

"Step Into", em alguns momentos ele pode não ser tão desejado, pois durante a execução do modo debug, pode não ser de nosso interesse entender aquele código tão detalhadamente.

Sendo assim, também temos a capacidade de executar a linha inteira do código com o "Step Over" por meio do atalho **F8**. Essa funcionalidade é útil justamente nos trechos que queremos apenas avançar sem nos aprofundar mais nas informações internas.

Step Out

Um outro cenário comum é quando entramos, sem querer, dentro de um método que não queremos inspecionar. Para casos como esses, podemos usar o "Step Out" para sair do método com o atalho **Shift + F8**.

Variáveis

Para criar uma variável por atalhos, temos as seguintes possibilidades:

- Variável local: **Ctrl + Alt + V / CMD + Alt + V**
- Atributo: **Ctrl + Alt + F / CMD + Alt + F**
- Constante: **Ctrl + Alt + C / CMD + Alt + C**

Note que os atalhos são bem similares, a única diferença é que muda a letra no final.

> *Dica: Para ficar fácil de lembrar o atalho lembre-se que C vem de constant (constante), F de field (campo ou atributo) e V de variable (variável, no caso a variável local)*

Tabela

Para auxiliar, confira essa tabela que mostra os principais atalhos que vimos durante os estudos de IntelliJ:

Alt + 1	Ocultar e mostrar a view
Alt + insert	Permite criar algum arquivo
“psvm” + Tab	Escreve o método main automaticamente
Alt + Shift + F10 / Ctrl + Shift + F10	Executar a classe atual
Alt + Enter	Possibilidades de ação por meio da feature Show Intention Action - por meio de suas sugestões.
Alt + Shift + Insert / CMD + Shift + 8	Modificar o modo de seleção para coluna
Ctrl + N / CMD + O	Busca de classe
Alt + Shift + setas para cima ou baixo	Mover linhas de código
Ctrl + Y / CMD + Delete/Backspace	Apagar linhas de código
Ctrl + Shift + A / CMD + Shift + A	Find action: permite buscar qualquer funcionalidade do IntelliJ
Ctrl + B / CMD + B	Navegação por declaração
Alt + setas (direita ou esquerda) / CMD + Shift + { ou }	Navegação por tabs
Ctrl + F4 / CMD + W	Fecha a tab atual
Alt + Shift + X	Fecha todas tabs
F5	Copia uma classe

Ctrl + Alt + L / Alt + CMD + L	ajustando a indentação
Shift + F6	Renomeando variáveis Rename
Ctrl + Alt + M / CMD + Alt + M	Extração de método
Shift + Shift.	Search everywhere: permite buscarmos qualquer coisa dentro do projeto
Ctrl + E / CMD + E	Visualizar todos os arquivos recentes
Ctrl + Shift + Alt + N / CMD + Alt + O	Busca de símbolos
Ctrl + Shift + F / CMD + Shift + F	Buscando trechos
Ctrl + H	Visualização de hierarquia
Alt + F7	Find Usages
Alt + Home / CMD + seta pra cima	Navegação por barra
Ctrl + D / CMD + D	Copiando linhas
Ctrl + Shift + A / CMD + Shift + A	Find Action
F7	Step Into
F8	Step Over
Shift + F8	Step Out
Ctrl + barra(/) / CMD + barra(/)	Comentários

V	Variável Local
Ctrl + Alt + F / CMD + Alt + F	Atributo
Ctrl + Alt + C / CMD + Alt + C	Constante

Conclusão

Nesse artigo, vimos que podemos fazer as ações que utilizamos frequentemente de maneira mais objetiva por meio de atalhos. Aprendemos a criar um projeto, navegar por ele, utilizar técnicas de refatoração e o modo debug para analisar mais precisamente o código.

Conhecer a IDE que você usa em seus projetos é fundamental na programação, já que o uso dela impacta diretamente na **produtividade** e **otimização**. Ao utilizá-la da melhor forma, o tempo de trabalho pode ser menor porque o processo se torna mais ágil.

E você, tem algum atalho que mais utiliza? Bons estudos e até mais!

Se esse conteúdo te interessou, você pode acessar os links abaixo para potencializar sua aprendizagem:

- Saiba tudo sobre o IDE - Integrated Development Environment
- Formação Java e Orientação à Objetos
- Formação Kotlin

Esse artigo é baseado em um conteúdo desenvolvido pelo Alex Felipe, em 2017.

Akemi Alice

Akemi faz parte do Scuba Team na Escola de Programação & DevOps da Alura, com foco em Java, e é formada no curso técnico em Informática pelo Instituto Federal de São Paulo (IFSP).

Artigo Anterior Próximo Artigo

Veja outros artigos sobre
Programação

Quer mergulhar em tecnologia e aprendizagem?

Receba a newsletter que o nosso CEO escreve pessoalmente, com insights do mercado de trabalho, ciência e desenvolvimento de software

Escreva seu email

ME INSCREVA

Nossas redes e apps