09

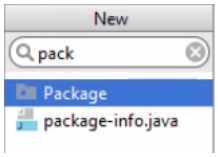
Criando código com o generate

Transcrição

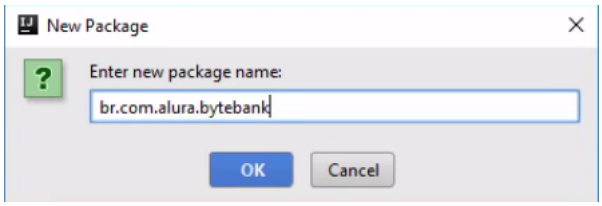
Tivemos o nosso primeiro contato com o *IntelliJ*, e aprendemos alguns atalhos, e como criar e executar uma classe, vamos dar continuidade com o nosso projeto.

Nosso passo agora é colocar uma classe para representar um funcionário no projeto. Mas antes de criar a classe precisamos organizar as nossas classes nas convenções estabelecidas pelo Java, que é por meio de pacotes.

Para criar um novo pacote, usamos "Alt + 1" para acessar a *project*, agora navegamos até a pasta `src` e usamos o atalho "Alt + Insert". No menu de opções, podemos escolher ***package***, ou utilizar o recurso de filtros do *IntelliJ*, que nada mais é do que digitar "package" e a própria IDE já ativa a filtragem.

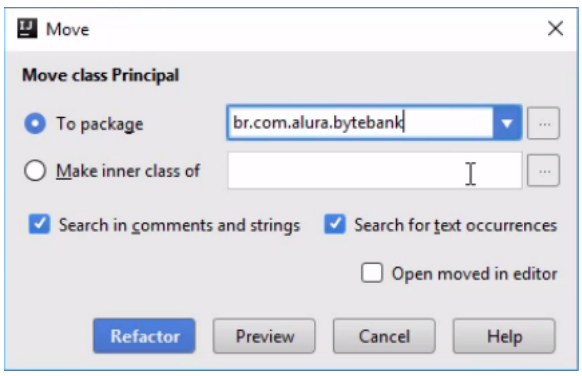


Com a opção *package* selecionada, clicamos em "Enter" para confirmar a opção. Na nova janela aberta colocamos o nome do pacote, por exemplo, `br.com.alura.bytebank` .



Criamos apenas o pacote, e não mandamos classe `Principal` para o novo pacote. Poderíamos clicar e arrastar a classe com o mouse, mas o interessante é usar o atalho para isso.

O atalho para mover a classe é o "F6" conhecido como ***Move***, que faz a mesma função que arrastar a classe. No campo ***To Package*** colocamos o pacote que queremos enviar, que no caso é `br.com.alura.bytebank` . Agora é só clicar em "*Refactor*".



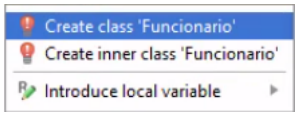
Simples, o *IntelliJ* moveu a nossa classe para o pacote. Inclusive ele já colocou na classe o pacote que ela pertence.

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

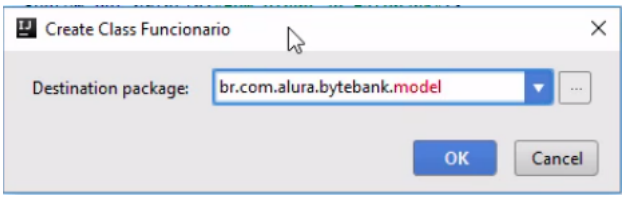
```
public class Principal {

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

Agora de fato vamos criar uma classe `Funcionario`. Poderíamos acessar a *view* do *Project* e usar os atalhos, mas existe uma outra forma interessante de fazer esse processo.

Vamos imaginar que durante o desenvolvimento tivemos a intenção de instanciar a classe com `new Funcionario()`. No momento ela não existe, e a IDE sinaliza dizendo não sabe trabalhar com esse simbolo. Nessas situações podemos usar as sugestões do *IntelliJ* com o atalho "Alt + Enter". Baseando no contexto do que estamos tentando fazer, o que foi sugerido é a criação da classe `Funcionario`.



Como queremos criar a classe, vamos selecionar a opção ***Create class 'Funcionario'*** e confirmar com a tecla "Enter". Uma janela surgirá solicitando o pacote que a nova classe vai ficar, como `Funcionario` é um model, vamos colocá-lo no pacote `br.com.alura.bytebank.model`.



Confirmando o destino do pacote, a classe `Funcionario` será criada dentro do pacote que escolhemos no passo anterior. Além disso, na classe `Principal`, já foi feito automaticamente o `import br.com.alura.bytebank.model.Funcionario`.

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

import br.com.alura.bytebank.model.Funcionario

public class Principal {

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

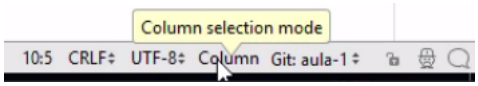
Vamos então trabalhar em nossa classe `Funcionario`. Um funcionário terá um nome `String nome`, uma matricula `int matricula`. Também terá um data de nascimento, como estamos configurados com **Java 8**, vamos usar a classe `LocalDate dataNascimento`, e repare que o podemos usar o ***autocomplete*** pressionando "Ctrl + Espaço", assim ao selecionar a classe `LocalDate` a IDE já faz o `import` automaticamente.

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

```
import java.time.LocalDate;

public class Funcionario {

    String nome;
    int matricula;
    LocalDate dataNascimento;

}
```

Como sabemos, uma boa prática é manter os atributos da classe encapsulados, por isso precisamos colocar os atributos como `private` . Mas imagine que temos muitos atributos nessa classe, daria muito trabalho colocar repetidas vezes o modificador de acesso `private` , por isso usaremos os atalhos.

Usaremos o atalho "Alt + Shift + insert". Esse atalho converte a maneira de seleção de linha para **coluna**. Podemos ver o modo em que estamos trabalhando na barra inferior do *IntelliJ*.



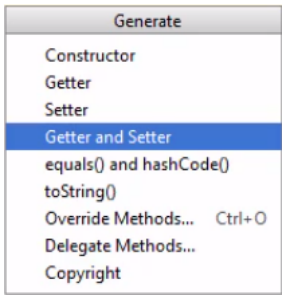
Com o modo de seleção para colunas, seguramos a tecla "Shift" e selecionamos todos os atributos que queremos colocar o modificador de acesso, agora é só escrever `private` que será repetido para todos os atributos.

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

import java.time.LocalDate;

public class Funcionario {

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

}
```

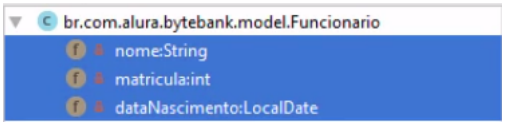
Para voltar ao modo de seleção de linha, basta repetir o atalho "Alt + Shift + insert".

Em **Java**, utilizamos dos métodos ***Getter*** e ***Setter***. Para evitar trabalho, podemos escrever `setNome` e usar o *autocomplete* para que ele já monte o método *Setter* inteiro para você. A mesma coisa pode ser feita para *Getter*.

É bem interessante o recurso do *autocomplete*, mas para classes com muitos atributos ainda ficaria trabalhoso. Então, podemos usar o recurso do atalho "Alt + insert". Dentro de uma classe, o atalho tem a função de ***Generate***, e no caso, vamos selecionar a opção ***Getter and Setter***.



Selecionando a opção, uma janela será aberta para que possamos selecionar os atributos que gostaríamos de ter o ***Getter and Setter***. Selecionamos todos e clicamos em "OK".

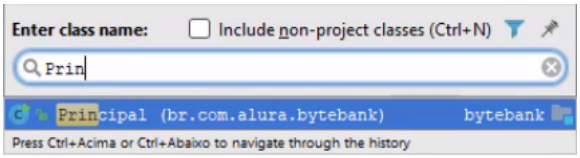


Pronto, temos todos os *Getter* e *Setter* criados, isso sem nenhum esforço. Para aumentar o espaço do editor, podemos usar o "Ctrl + Shift + F12", e para voltar a visualização original basta usar o mesmo atalho.

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

import java.time.LocalDate;

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 LocalDate getDataNascimento() {
        return dataNascimento;
    }

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

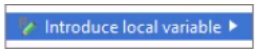
Vamos agora criar um objeto da classe `Funcionario` e executar o nosso programa. Mas, ainda estamos usando o mouse para trocar de classe, seria algo complicado se tivéssemos dezenas de classes.

Por conta disso podemos usar o atalho "Ctrl + n". Esse atalho é o que busca as classe no projeto, então basta digitarmos `Principal`, ou as iniciais para que ele retorne a classe.



Essa é a maneira rápida de buscar uma classe no projeto. Na classe `Principal`, agora que temos uma instância de `Funcionario`, precisamos guardá-lo em uma variável do tipo `Funcionario`.

Poderíamos escrever manualmente `Funcionario funcionario = new Funcionario();`, porém a IDE também nos ajuda nessa questão, basta colocarmos o cursos do editor em cima de `new Funcionario()` e usarmos o atalho "Alt + Enter", selecionando a opção "***Introduce a local variable***".



Dessa forma, o *IntelliJ* cria para gente uma variável que assina a instância que criamos. A variável já vem como nome padrão da classe, mas podemos alterar para o que desejarmos, por exemplo, `Funcionario jose = new Funcionario()`.

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

import br.com.alura.bytebank.model.Funcionario;

public class Principal {

    public static void main(String[] args) {

        System.out.println("Bem-vindo ao Bytebank");
        Funcionario jose = new Funcionario();
    }
}
```

Vamos agora colocar as informações do objeto usando os método `jose.setNome()`, `jose.setMatricula()` e `jose.setDataNascimento()`.

Mas o que passaremos para o `jose.setDataNascimento()`? Como o método espera um `LocalDate` podemos passar `LocalDate.of()` com os argumento de valor do ano, mês e dia.

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

import br.com.alura.bytebank.model.Funcionario;

public class Principal {

    public static void main(String[] args) {

        System.out.println("Bem-vindo ao Bytebank");
        Funcionario jose = new Funcionario();

        jose.setNome("José");
        jose.setMatricula(1);
        jose.setDataNascimento(LocalDate.of(1990, 2, 10));
    }
}
```

Vale notar que no `LocalDate.of()` o *IntelliJ* colocou ***labels*** (*rótulos*) para o que cada argumento passado representa. Esses *labels* são visíveis apenas no *IntelliJ* e serve apenas com o intuito de auxiliar o desenvolvedor a passar os valores corretos.

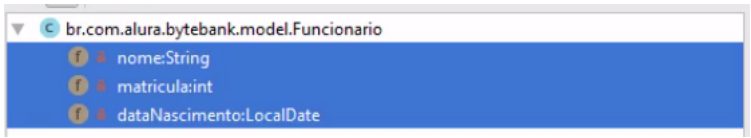
Podemos agora colocar o uma saída para que possamos visualizar o funcionário.

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

import br.com.alura.bytebank.model.Funcionario;

public class Principal {

    public static void main(String[] args) {

        System.out.println("Bem-vindo ao Bytebank");
        Funcionario jose = new Funcionario();

        jose.setNome("José");
        jose.setMatricula(1);
        jose.setDataNascimento(LocalDate.of(1990, 2, 10));

        System.out.println(jose);
    }
}
```

Executando com "Shift + F10" podemos ver que recebemos a mensagem de bem-vindo e o endereço de memória do objeto `jose` . Isso aconteceu por que foi chamado o método `toString()` da classe `Object` .

Vamos acessar a classe `Funcionario` . Dentro da classe podemos usar o atalho de *Generate* "Alt + insert" e selecionar a opção `toString()` ou digitar para que o *IntelliJ* faça a busca, assim como fizemos na criação de pacotes.

Agora a janela ***Generate toString()*** será aberta, podemos selecionar os atributos que queremos que seja usado no `toString()` , na caso, selecionaremos todos.



O `toString()` será sobrescrito e já vira com os atributos que selecionamos.

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

import java.time.LocalDate;
```

```
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 LocalDate getDataNascimento() {
        return dataNascimento;
    }

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

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

Podemos voltar a classe `Principal` e executar novamente a classe. Recebemos a mensagem de boas-vindas e a descrição do objeto `jose` .