



Domain Driven Design - Java

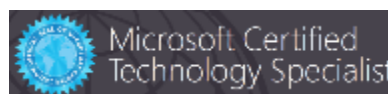
Prof. Gilberto Alexandre das Neves
profgilberto.neves@fiap.com.br

Apresentação



Formação:

- Técnico em Eletrônica
- Graduado em Ciências da Computação
- Licenciatura em Computação
- Pós Graduado em Produção e Programação em Games
- Mestre em Tecnologias Emergentes em Educação



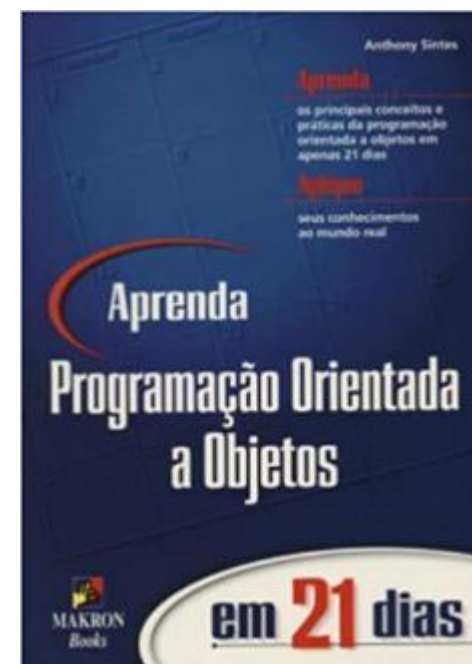
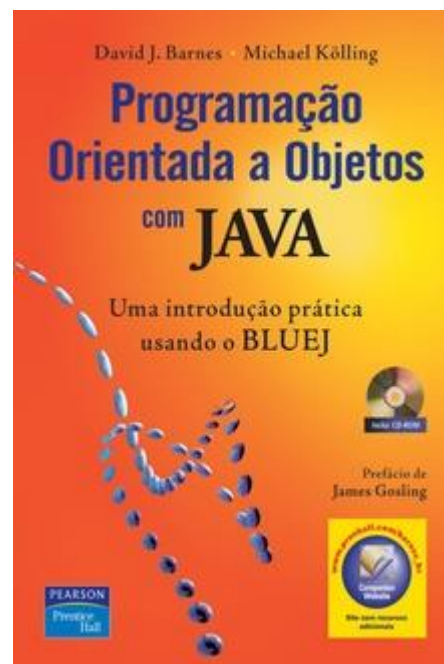
Experiência Acadêmica:



Competências:

- Aplicação do conceito de orientação a objetos em Java que favoreça a reutilização de código e o mínimo de acoplamento através dos *Patterns*. Comunicação com banco de dados utilizando JDBC.

Conteúdo	
1º semestre	2º semestre
Conceito de POO	Collections
Classes, atributos e métodos	Exceptions
Manipulação de objetos	Manipulação de arquivos
Entrada e saída	Abstração
Encapsulamento	JDBC
Herança	MVC e DAO



As notas semestrais da Fiap são compostas:

- 3x Checkpoints (duas maiores notas)
- 2x Sprints (Challenge)
- 1x Global Solution

MS = 40% (CP+CHG) + 60% (GS)

Média anual da Fiap é composta de:

MA = 40% MS1 + 60% MS2

CRITÉRIOS DE APROVAÇÃO

Média Anual	Situação
0 a 3.9	Reprovado
4.0 a 5.9	Exame
6.0 a 10	Aprovado

02

FEVEREIRO

- 10 Início das Aulas (veteranos)
- 24 Aula Inaugural (calouros)
- 24 A 28 FIAP First Week (calouros)

03

MARÇO

- 3 E 4 Carnaval (aulas suspensas)
- 5 Quarta-feira de Cinzas (aulas suspensas)
- 6 Início das Aulas (calouros)
- 10 A 14 Período para solicitação de mudança de turma e curso
- 10 A 14 Período para solicitação de dispensa de disciplina
- 17 A 21 Divulgação dos pedidos de mudança de turma e curso
- 17 A 21 Período para regulamentação das disciplinas em regime de dependência
- 31 Divulgação das dispensas das disciplinas

04

ABRIL

- 17 Quinta-feira Santa (aulas suspensas)
- 18 Paixão de Cristo (aulas suspensas)
- 20 Páscoa
- 21 Tiradentes

05

MAIO

- 01 Dia Mundial do Trabalho
- 02 Aulas suspensas
- 26 Início do período de avaliação semestral (Global Solutions)

06

JUNHO

- 03 A 13 Período de solicitação de avaliações substitutivas regulares e de dependência (cursos presenciais)
- 06 Fim do período de avaliação semestral (Global Solutions)
- 09 A 13 Período de avaliação semestral de disciplinas de dependência
- 16 A 18 Período de avaliações substitutivas regulares e de dependência
- 19 Corpus Christi (aulas suspensas)
- 20 Aulas suspensas
- 23 A 27 Vistas de provas
- 30 Final das Fases/divulgação dos resultados das avaliações semestrais

07

JULHO

- 01 A 31 Período de férias

Instalando o Java

Existem diversas maneiras de preparar o ambiente para o desenvolvimento de aplicações em Java. Consideraremos inicialmente apenas o kit de ferramentas da Oracle: o **JDK**.

Recomendado sempre buscar versões estáveis mais recentes do Java (LTS).

Dica: uma boa versão JDK é da empresa Oracle (basta buscar no google: JDK LTS oracle). No resultado da pesquisa você deve encontrar esse link:

<https://www.oracle.com/br/java/technologies/downloads/archive/>

Busque pela versão mais recente para seu sistema operacional e fique atento a arquitetura de sua máquina (32 ou 64 bits).

Após realizar o download basta executar a instalação ou descompactar o conteúdo, como por exemplo, em: **C:\Program Files\Java** (para sistemas operacional Windows)

Instalando o IntelliJ IDEA Community

IntelliJ IDEA é uma IDE desenvolvida pela empresa JetBrains. Lançada pela primeira vez em janeiro de 2001, a versão **Community** é **gratuita** e de código aberto, oferecendo uma ampla gama de funcionalidades para desenvolvedores Java.

Vantagens:

- **Refatoração Inteligente:** Suporta refatorações avançadas e sugere melhorias no código.
- **Suporte a Várias Linguagens:** Além de Java, suporta outras linguagens como Kotlin, Groovy, Scala e mais.
- **Integração com VCS:** Suporta ferramentas de controle de versão como Git, SVN e Mercurial.
- **Ferramentas de Desenvolvimento Web:** Suporte para HTML, CSS, JavaScript, entre outras.
- **Plug-ins e Extensões:** Possui uma vasta coleção de plug-ins para adicionar funcionalidades extras.

- Para instalar acesse o site oficial da JetBrains:
<https://www.jetbrains.com/idea/download/>
- Selecione a versão **Community** e clique em "Download".

We're committed to giving back to our wonderful community, which is why IntelliJ IDEA Community Edition is completely free to use



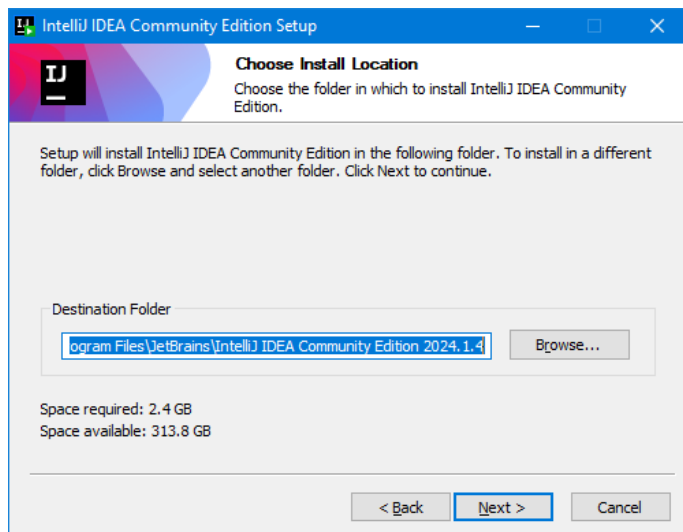
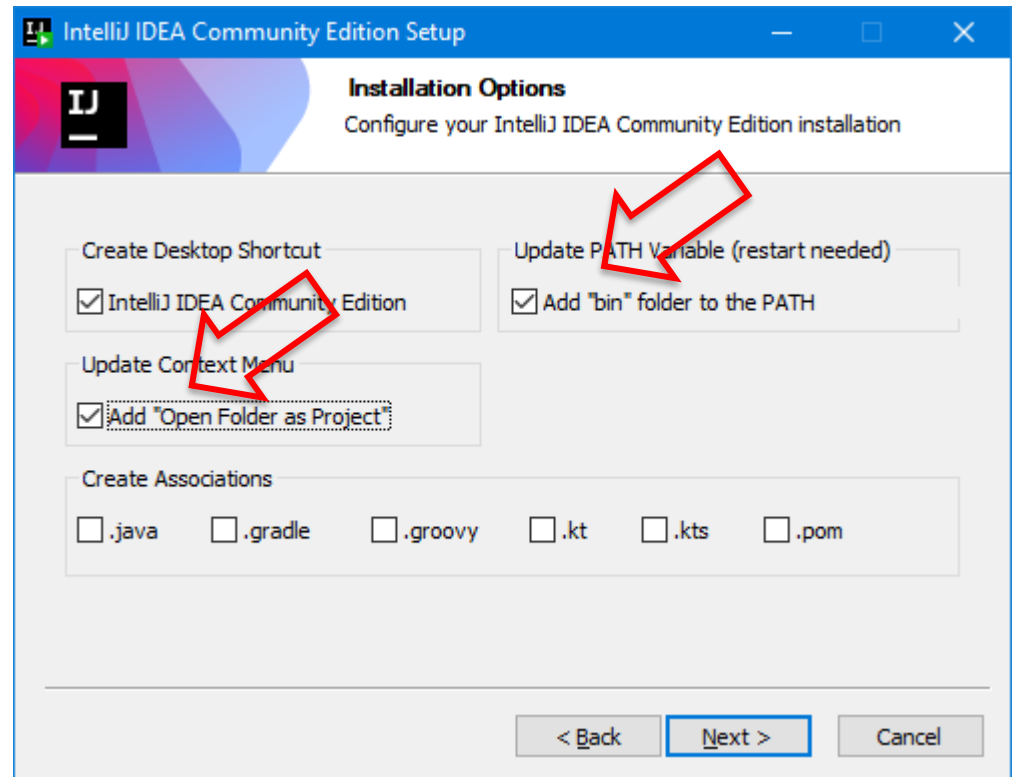
IntelliJ IDEA Community Edition

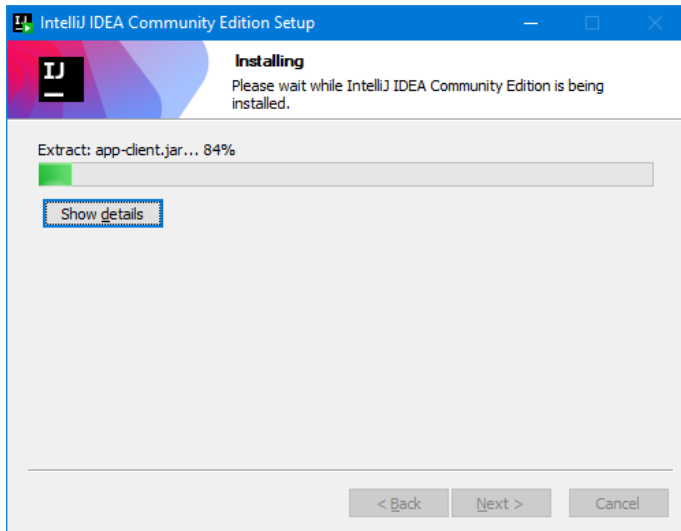
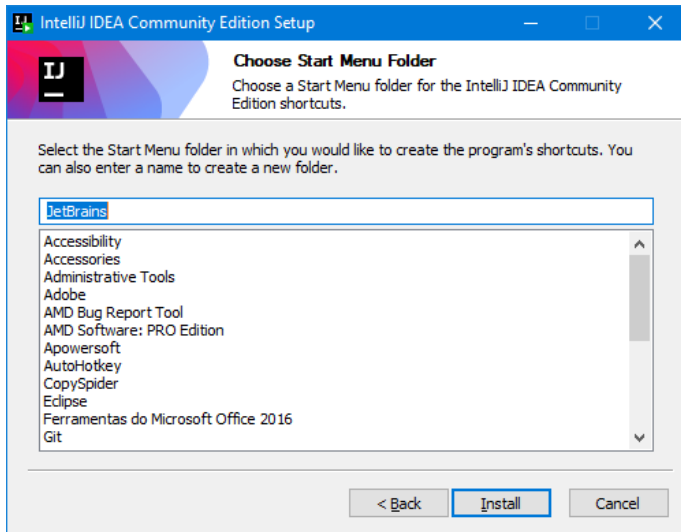
The IDE for Java and Kotlin enthusiasts

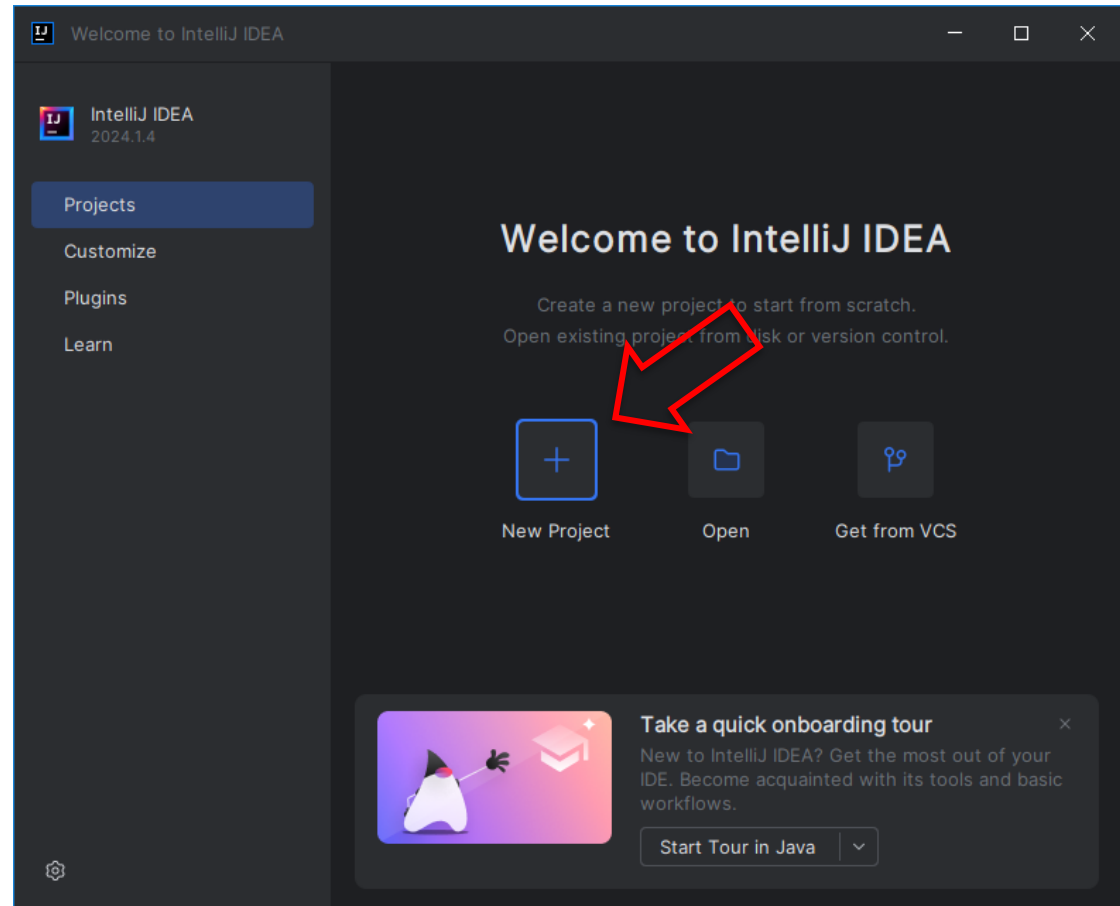
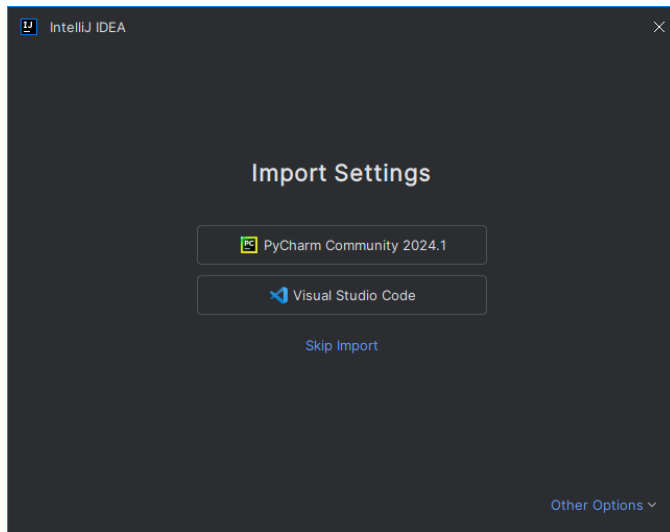
Download

.exe (Windows) ▼

Free, built on open source



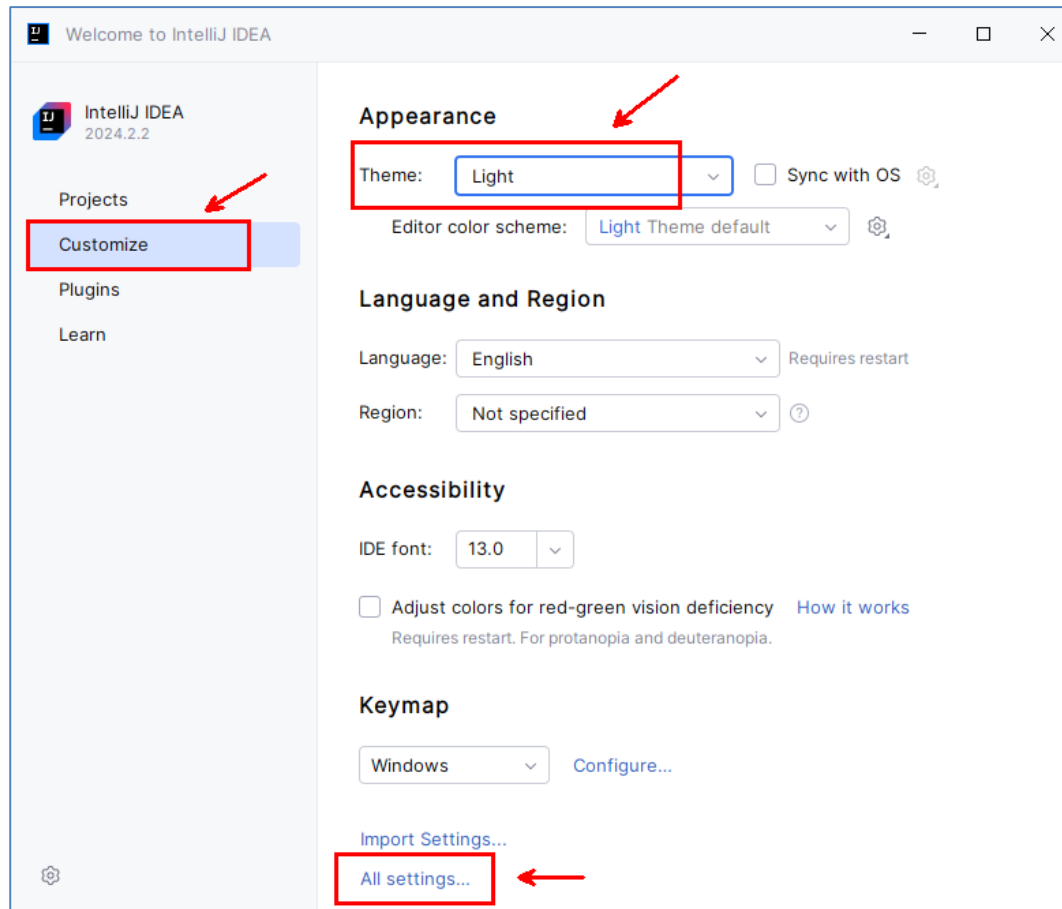




Configurando a IDE

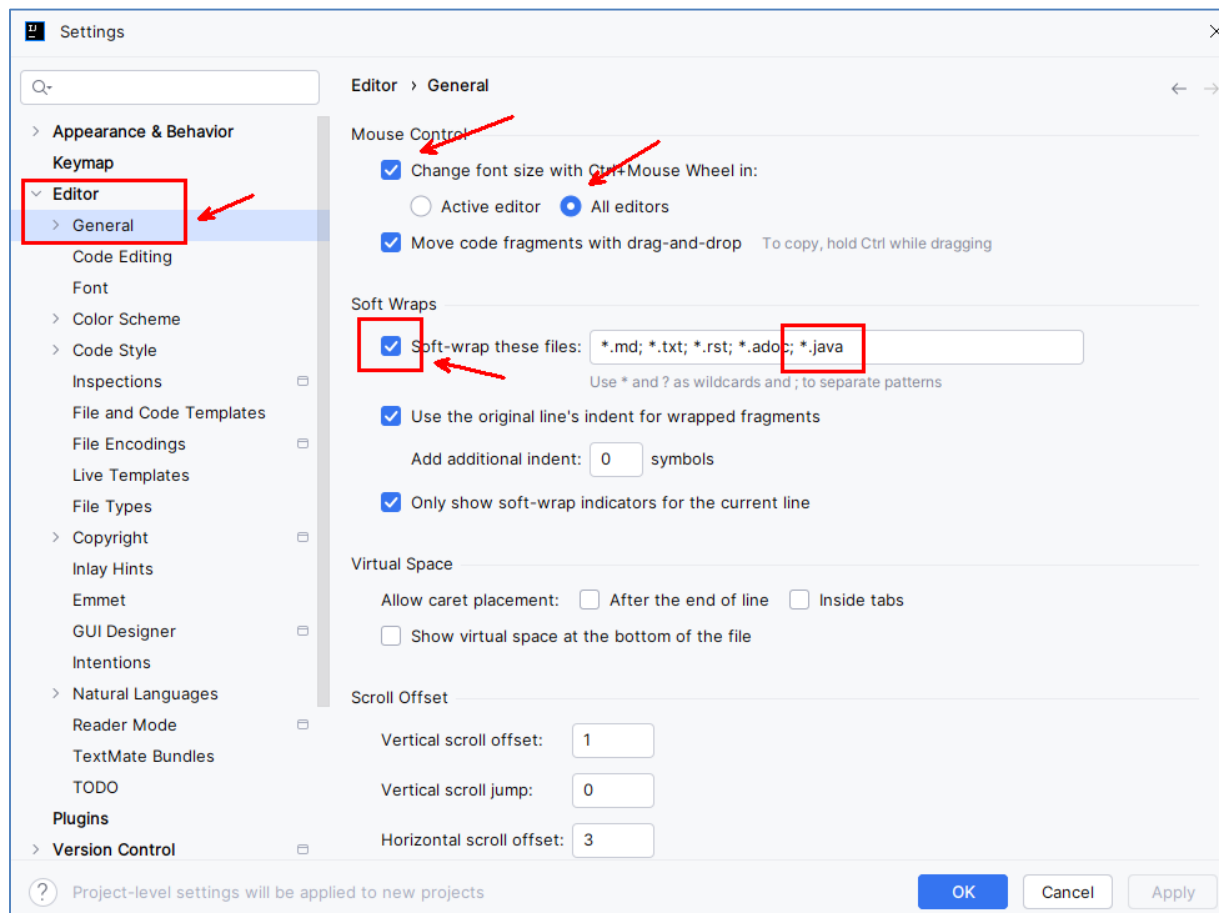
Vamos configurar a **IDE** para melhor aproveitamento, na tela inicial vá na categoria **Customize** (aqui é possível alterar o tema da IDE).

Para mais opções clique no link **All settings...**



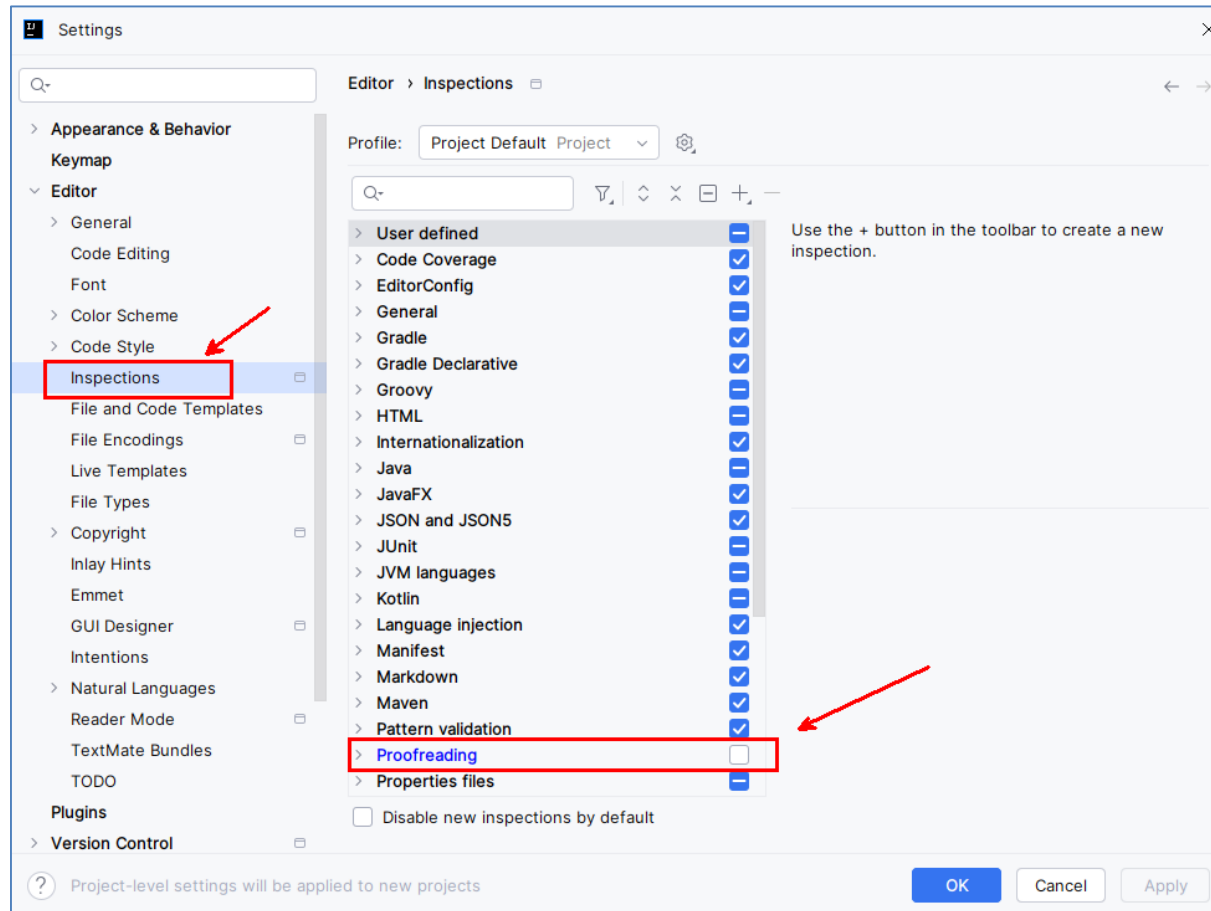
Configurando a IDE

Na categoria **Editor** selecione **General**. Ative as opções mostradas abaixo para permitir o **zoom** com o **Control + Scroll do Mouse** (rodinha do mouse) e a quebra automática de linha para arquivos do Java (necessário digitar **;.java** na caixa de texto).



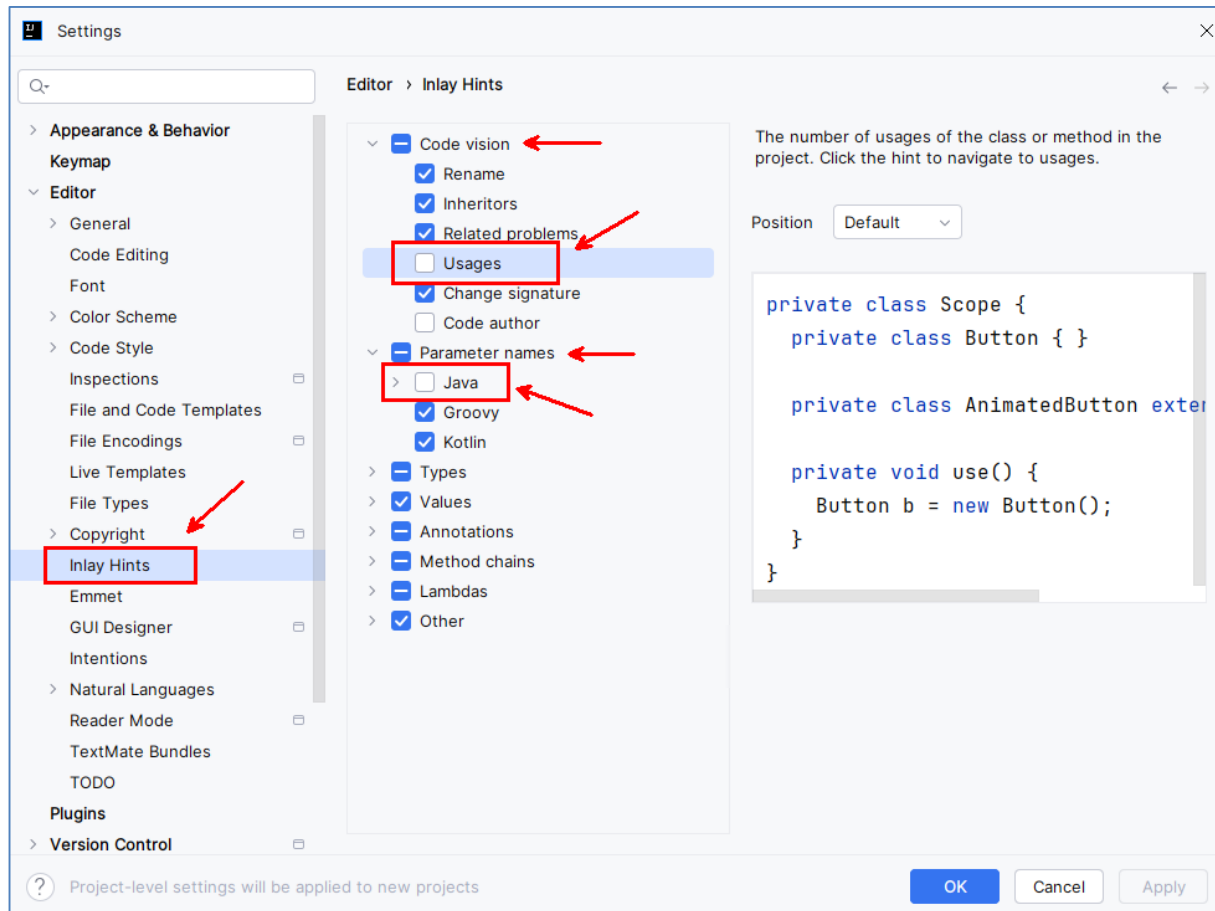
Configurando a IDE

Agora selecione **Inspections**. Desative a opção **Proofreading** para desativa a sugestão de correção ortográfica.



Configurando a IDE

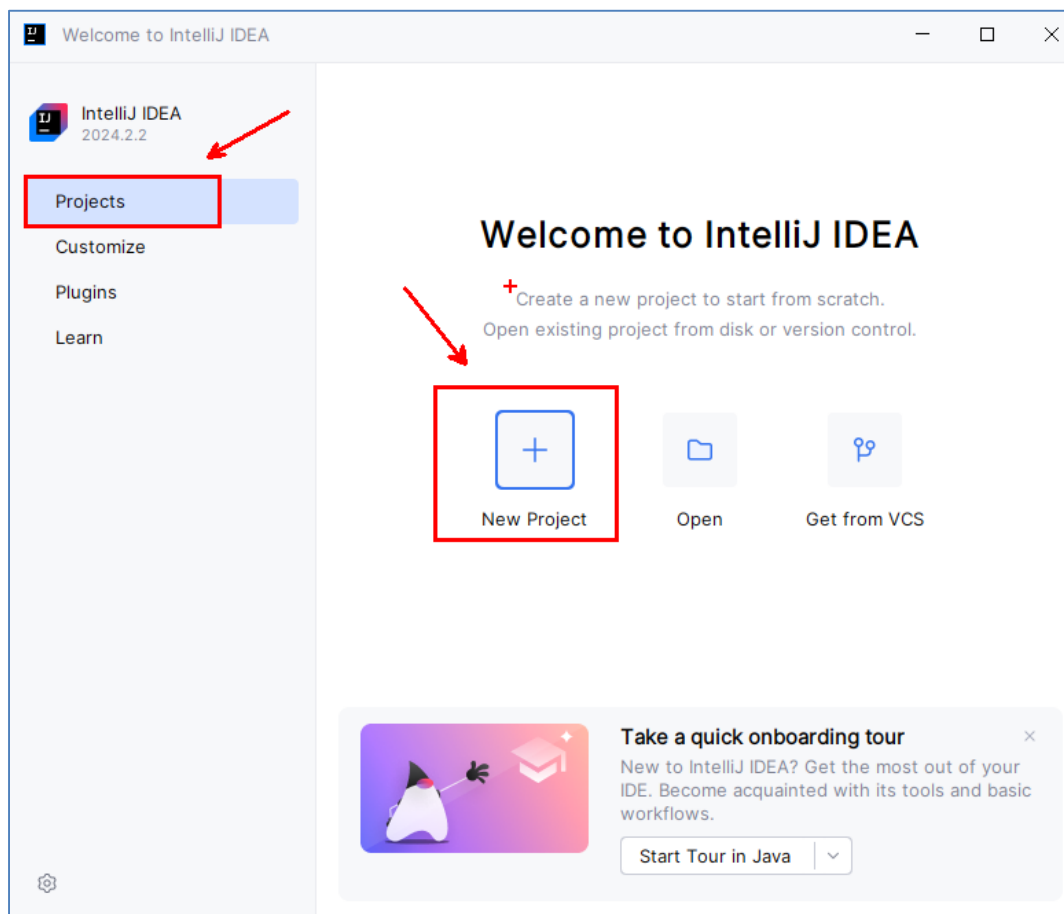
Agora selecione **Inlay Hints**. Desative a opção **Usages** (em **Code vision**) e desative a opção **Java** (em **Parameter names**). Vamos desativar essas opções na não nos confundir quando estivermos codificando.



Programando em Java

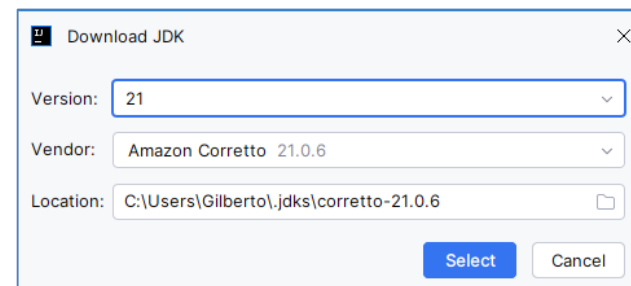
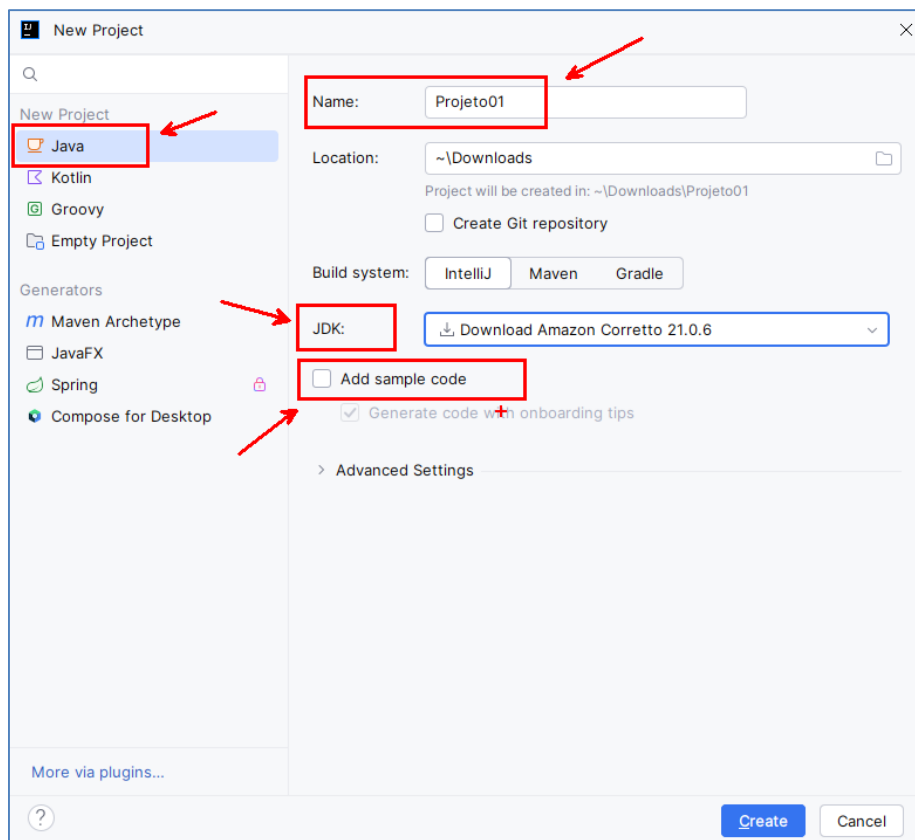
Criando um novo projeto

Vamos iniciar um novo projeto, volte para a categoria **Projects** e clique em **New Project**.



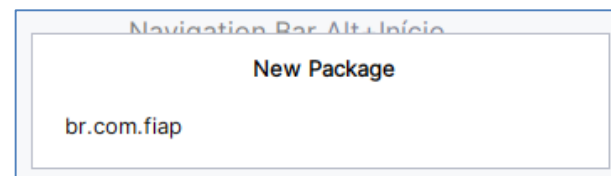
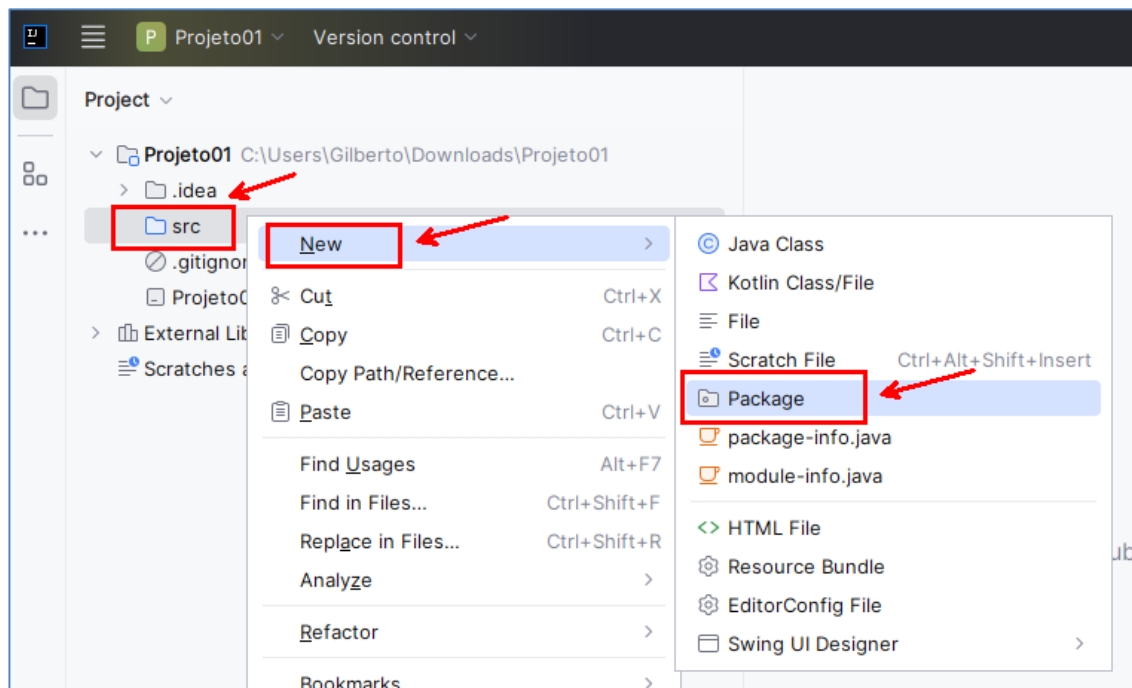
Criando um novo projeto

Vamos criar um projeto **Java**. Dê um **nome ao seu projeto** e selecione o **JDK** adequado (na primeira vez será necessário realizar o download clicando em **Download JDK...**). Em nossa aulas vamos utilizar a **versão 21 da Amazon (Amazon Corretto)**. Desative a opção **Add sample code**. Então clique em **Create**.



Criando um novo projeto

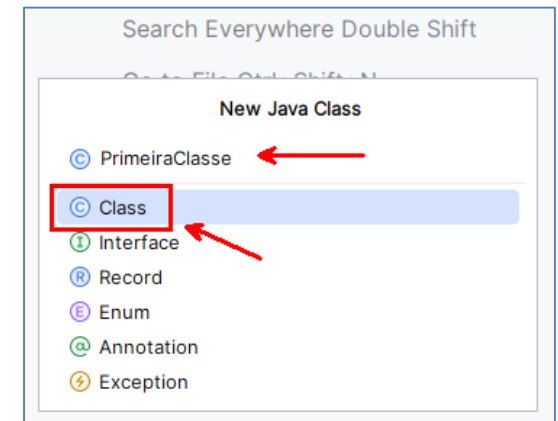
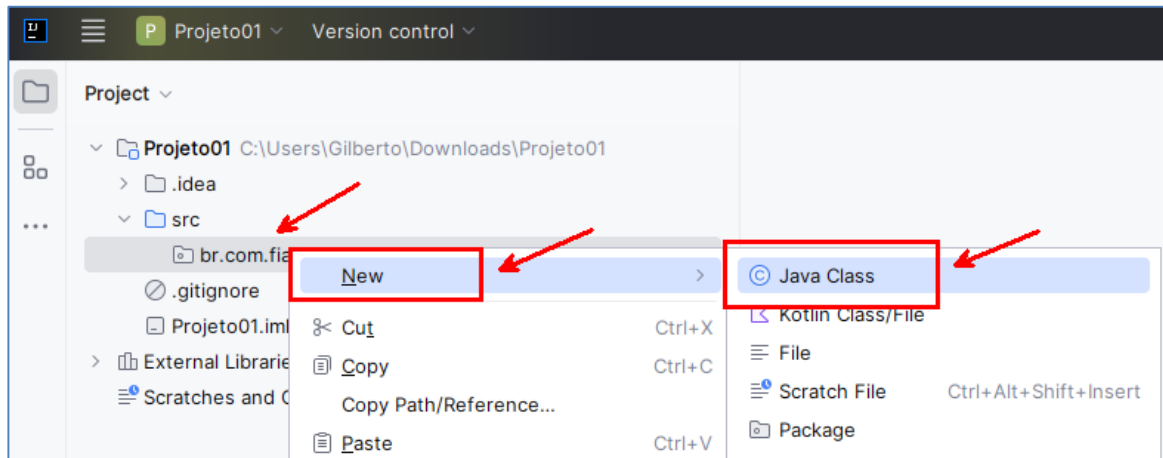
A **pasta de nosso projeto**, neste exemplo, é **Projeto01** (fica no topo em negrito). Já o código-fonte deve ficar dentro da pasta **src**. Porém por questões de organização, colocamos os arquivos de código-fonte dentro de **pacotes**. Vamos então primeiro criar um pacote chamado **br.com.fiap** dentro da pasta **src** (clcando com o botão direito do mouse).



Criando uma Classe

Como o **Java** é **orientado à objetos**, nosso código-fonte fica em uma **Classe**. Nesta aula vamos apenas criar classe dentro do pacote criado anteriormente com o botão direito do mouse (em aula futuras vamos ver mais a fundo este e outros conceitos da orientação à objetos).

Único detalhe aqui, já vamos começar a seguir a **convenção de nomenclatura** para as classes (boas práticas de programação). **TODA** classe deve ter a **primeira letra** de cada palavra em **maiúsculo** (e não deve ter espaços entre as palavras).



O método main

Em uma **Classe** onde precisamos **testar** (executar) seu código para ver seu funcionamento, precisamos criar o método **main** nesta classe.

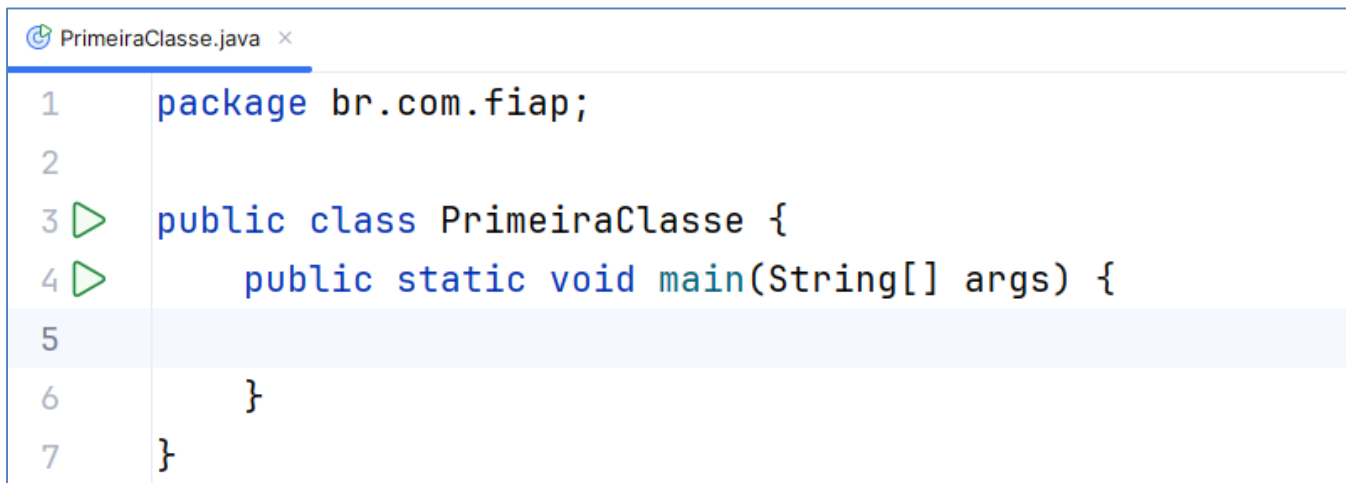
Basta digitar a palavra **main** dentro da Classe (dentro das **chaves** que indicam início e fim do código da classe) e pressionar a tecla **Tab** ou **Enter**.



```
1 package br.com.fiap;
2
3 public class PrimeiraClasse {
4     main
5 }
6
```

main() method declaration

Press Enter to insert, Guia to replace Next Tip



```
1 package br.com.fiap;
2
3 public class PrimeiraClasse {
4     public static void main(String[] args) {
5
6     }
7 }
```

public

É um qualificador usado em diversos identificadores em Java (classes, atributos, métodos), significa que esse método será visível a outras classes

static

Trata-se de um qualificador que indica que o método pertence à classe (ele é estático à classe que o definiu).

void

É o valor de retorno do método. Quando não há nenhum valor a ser retornado por quem chamou o método, ele retorna *void*, uma espécie de valor vazio.

main

Esse é o nome do método que indica o ponto inicial da execução da classe. Por convenção, uma classe que contém o método *main* é considerada uma aplicação, um programa que pode ser executado. Em um sistema real existem muitas classes, no entanto, apenas uma classe normalmente possui o método *main*.

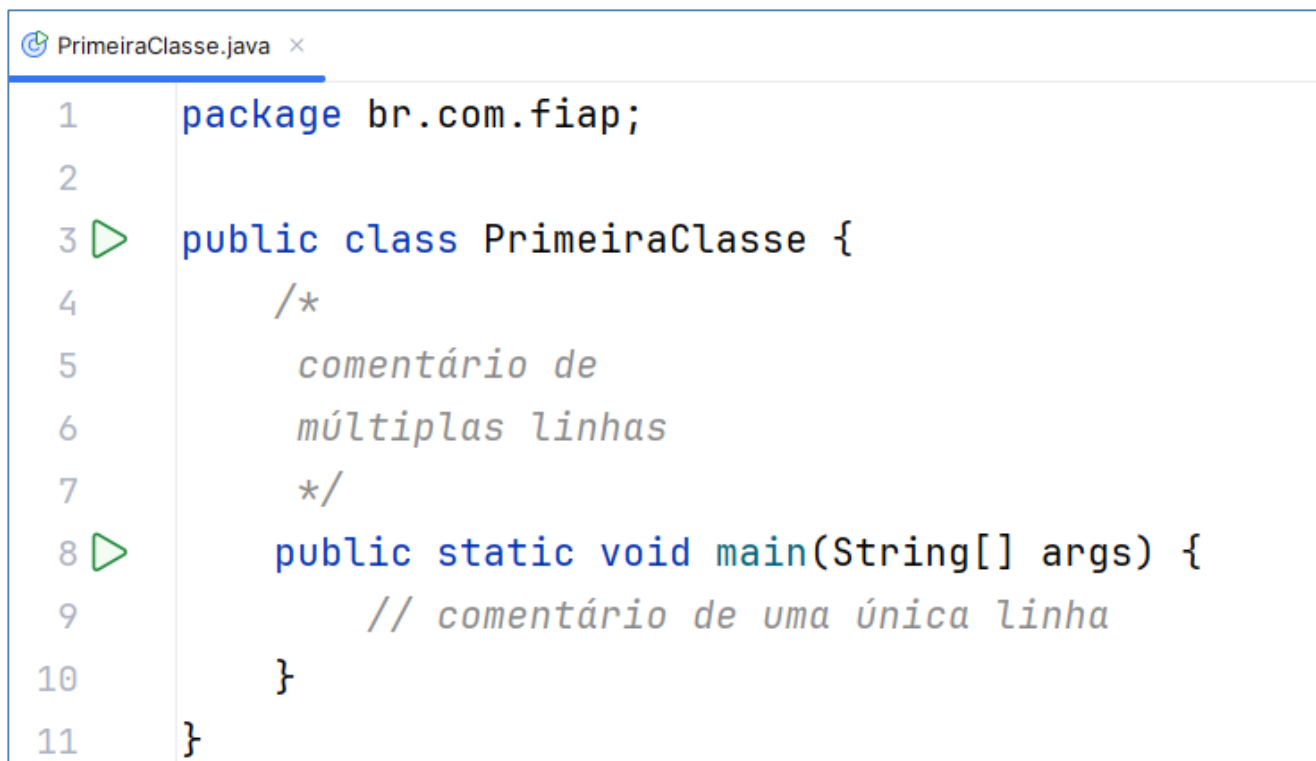
(String[] args)

É o argumento do método principal (*main*); trata-se de um vetor de strings responsável por receber valores que podem ser processados internamente à classe. Outra variação existente nessa sintaxe é inverter a posição dos colchetes ([]) que aparecem do lado direito da palavra String (***String args[]***).

{ ... }

“Abre-chaves” e “fecha-chaves” delimitam um bloco de código.

Os comentários são linhas adicionadas ao programa que servem para facilitar seu entendimento por parte do programador, ou ainda por uma outra pessoa que o consulte. Essas linhas não afetam o programa em si, pois não são consideradas parte do código. O Java aceita dois tipos de comentário: de **uma única linha** e de **múltiplas linhas**.

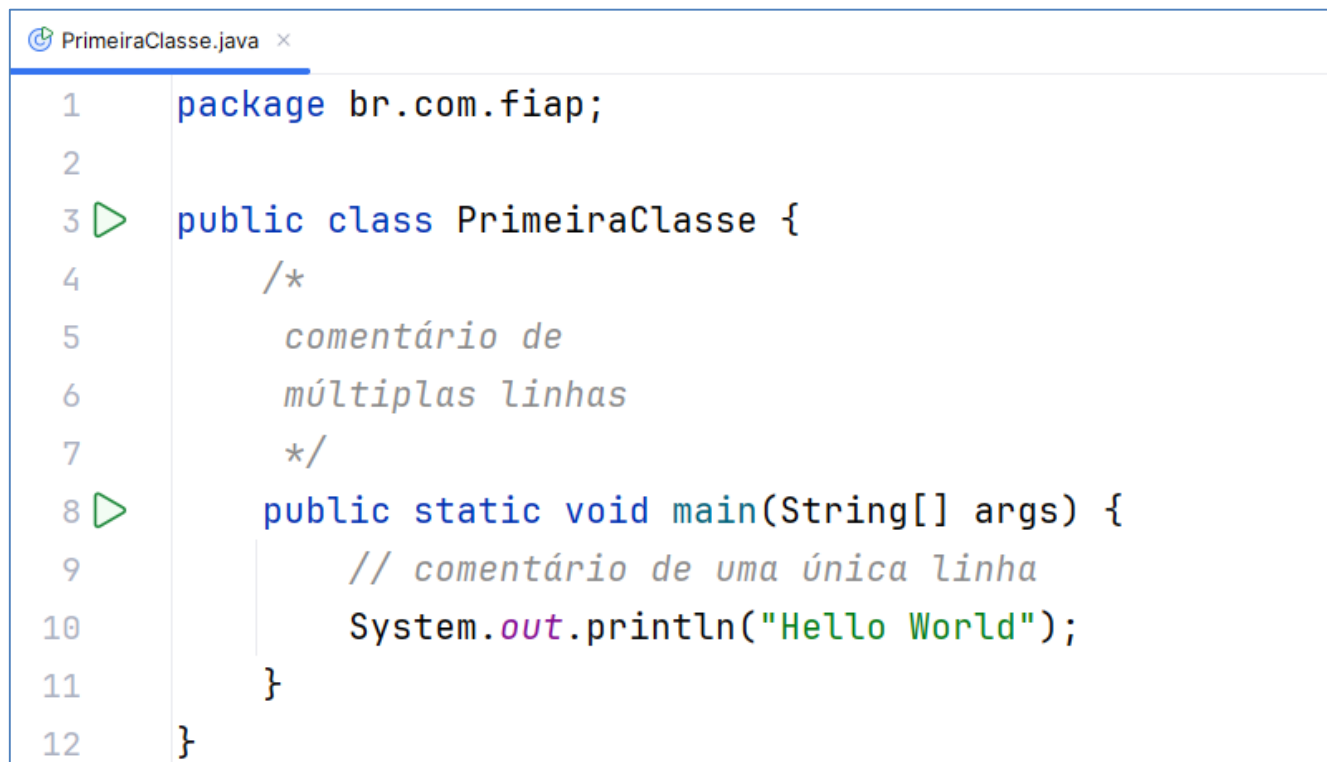


```
1 package br.com.fiap;
2
3 public class PrimeiraClasse {
4     /*
5      comentário de
6      múltiplas linhas
7     */
8     public static void main(String[] args) {
9         // comentário de uma única linha
10    }
11 }
```

Exibindo mensagens no vídeo

Podemos utilizar o método **println()** (da classe **System.out**) para exibir mensagens no vídeo (console do computador). Para isso, dentro do método **main**, digite **sout** e pressione **Tab** ou **Enter** que a IDE completa a linha de comando para você.

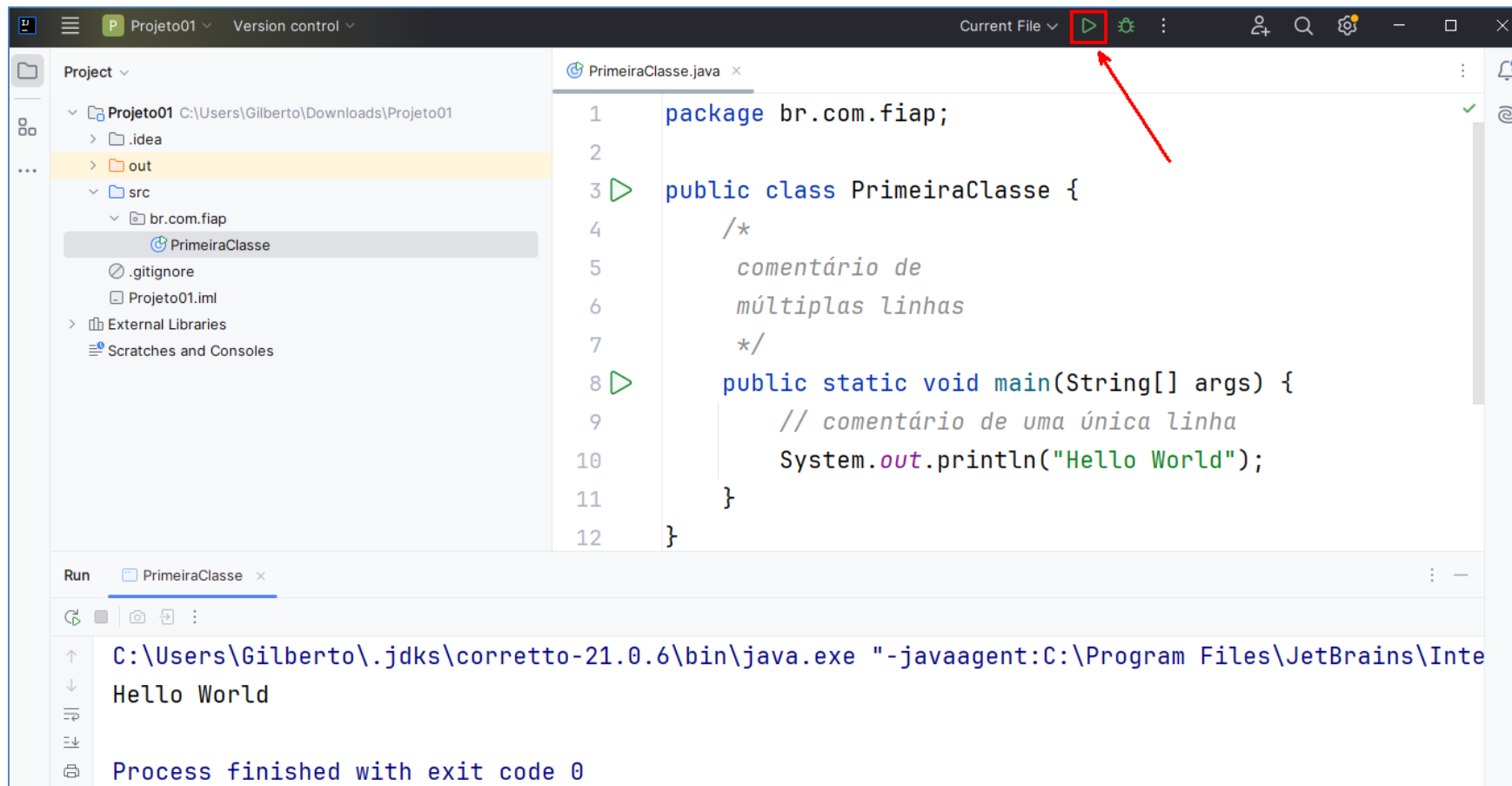
Então, basta digitar a mensagem desejada dentro do parêntese **entre aspas** como, por exemplo: **“Hello World!”**.

A screenshot of a Java IDE window titled 'PrimeiraClasse.java'. The code is as follows:

```
1 package br.com.fiap;
2
3 public class PrimeiraClasse {
4     /*
5      * comentário de
6      * múltiplas linhas
7      */
8     public static void main(String[] args) {
9         // comentário de uma única linha
10        System.out.println("Hello World");
11    }
12 }
```

Executando um programa

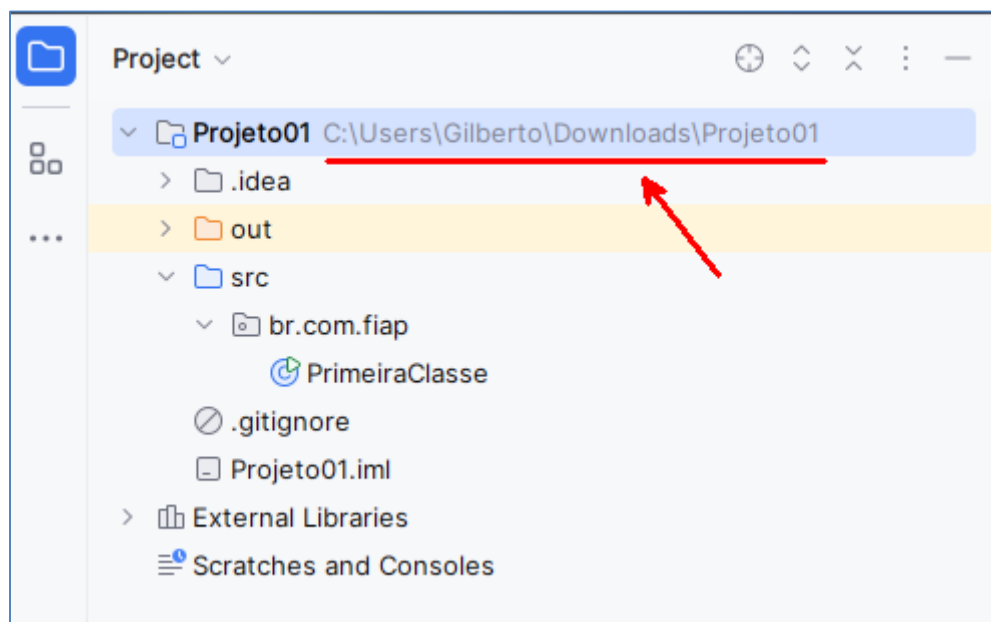
Para executar seu programa, basta clicar no botão **Run** da barra de título e você verá o resultado exibido na **janela do console** na parte inferior da **IDE**.



Salvando o programa

Seu projeto **já esta salvo** (você escolheu o local no momento da criação do projeto) e a IDE possui o recurso de salvamento automático.

Caso necessite copiar o projeto para levar para outro computador por exemplo, basta copiar a **PASTA DO PROJETO**. Isso pode ser feito diretamente na IDE com botão direito **Copy** (**Control** + **C**) ou ir até onde o projeto está salvo (você pode verificar isso pela própria IDE que exibe o caminho onde está o projeto).



Orientação à Objetos

Nos anos 60 nasceu a programação estruturada. Esse é o método estimulado por linguagens como C e Pascal.

Usando-se linguagens estruturadas, foi possível, pela primeira vez, escrever programas moderadamente complexos de maneira razoavelmente fácil.

Entretanto, com programação estruturada, quando um projeto atinge um certo tamanho, torna-se extremamente difícil e muito custoso efetuar sua manutenção e fazer qualquer modificação.

A primeira linguagem a incorporar facilidades para definir classes de objetos genéricos na forma de uma hierarquia de classes e subclasses foi a linguagem Simula, que foi idealizada em 1966, na Noruega.

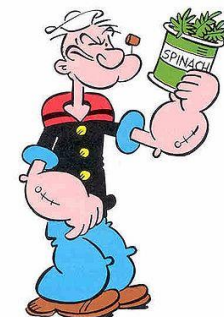


A Programação Orientada à Objetos aproveitou as melhores ideias da programação estruturada e combinou-as com novos conceitos, permitindo que um problema seja mais facilmente decomposto em subgrupos relacionados. Então, usando-se a linguagem, pode-se traduzir esses subgrupos em objetos.

A Programação Orientada à Objetos utiliza os conceitos que aprendemos no jardim de infância: objetos e atributos, todos e partes, classes e membros.

No entanto, cabe ressaltar que o conceito de Orientação à Objetos depende mais da mentalidade do programador do que da linguagem de programação que está sendo utilizada. Tomemos como exemplo a frase:

“O navio atraca no porto e descarrega sua carga.”



“O navio atracado no porto e descarrega sua carga.”

Se analisássemos esta frase estruturadamente, pensaríamos logo em como o navio atracado no porto e como ele faz para descarregar sua carga, ou seja, pensaríamos na ação que está sendo feita (que na frase é representada pelos verbos) para transformá-la em procedimento.

Em orientação objeto, o enfoque com que se encara a frase é diferente: primeiro pensaríamos no objeto navio, no objeto porto e no objeto carga, pensando como eles seriam e procurando definir seu comportamento. Após isto é que pensaremos em como o navio se relaciona com o porto e com a carga, e como o porto se relaciona com carga.



O mundo real é algo extremamente complexo. Quanto mais de perto o observamos, mais claramente percebemos sua complexidade.

A orientação a objetos tenta gerenciar a complexidade inerente dos problemas do mundo real, ***abstraindo*** conhecimento relevante e ***encapsulando-o*** dentro de objetos.



Abstração

Uma das principais formas do ser humano lidar com a complexidade é através do uso de abstrações.

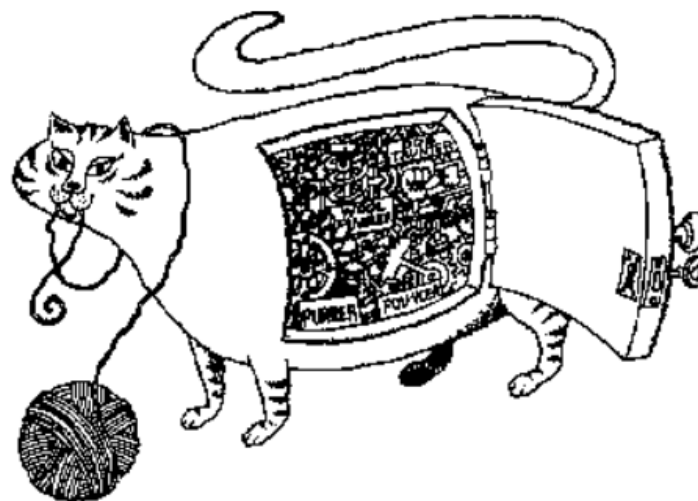
As pessoas tipicamente tentam compreender o mundo, construindo modelos mentais de partes dele. Tais modelos são uma visão simplificada de algo, onde apenas elementos relevantes são considerados.

Exemplo: Mapa de um território.



No mundo real, um objeto pode interagir com outro sem conhecer seu funcionamento interno. Uma pessoa, por exemplo, geralmente utiliza uma televisão sem saber efetivamente qual a sua estrutura interna ou como seus mecanismos internos são ativados. Para utilizá-la, basta saber realizar algumas operações básicas, tais como ligar/desligar a TV, mudar de um canal para outro, regular volume, cor, etc.

O encapsulamento consiste na separação dos aspectos externos de um objeto, acessíveis por outros objetos, de seus detalhes internos de implementação, que ficam ocultos dos demais objetos.



Muitos métodos de construção de software buscam obter sistemas modulares, isto é, construídos a partir de elementos que sejam autônomos, conectados por uma estrutura simples e coerente.

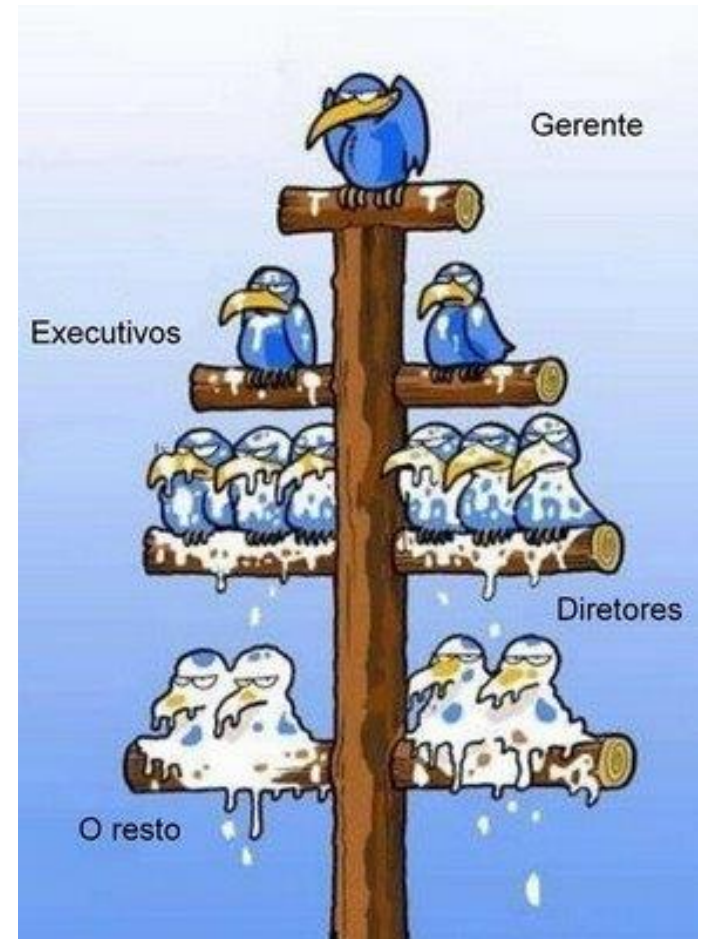
Modularidade é crucial para se obter re-usabilidade e extensibilidade.



Hierarquia

Um conjunto de abstrações frequentemente forma uma hierarquia e, pela identificação dessas hierarquias, é possível simplificar significativamente o entendimento sobre um problema.

Em suma, hierarquia é uma forma de arrumar as abstrações.



Uma classe é um **molde** ou **modelo** que define as características (**atributos**) e os comportamentos (**métodos**) de um objeto.

Para entender melhor o conceito de classe, vamos analisar suas **instâncias**, conhecidas como **objeto**. Um objeto é um termo que usamos para representar uma entidade do mundo real (fazemos isso através do exercício da abstração).

Vamos usar como exemplo o cachorro *Muttley*.



Podemos representá-lo em termos de **atributos**:

- Seu ***tamanho*** é pequeno
- Sua ***cor*** predominante é castanha

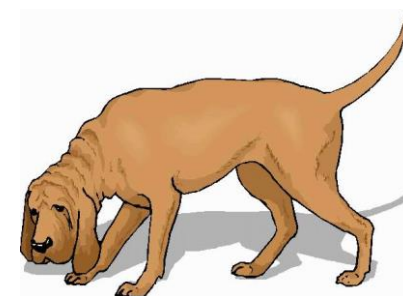
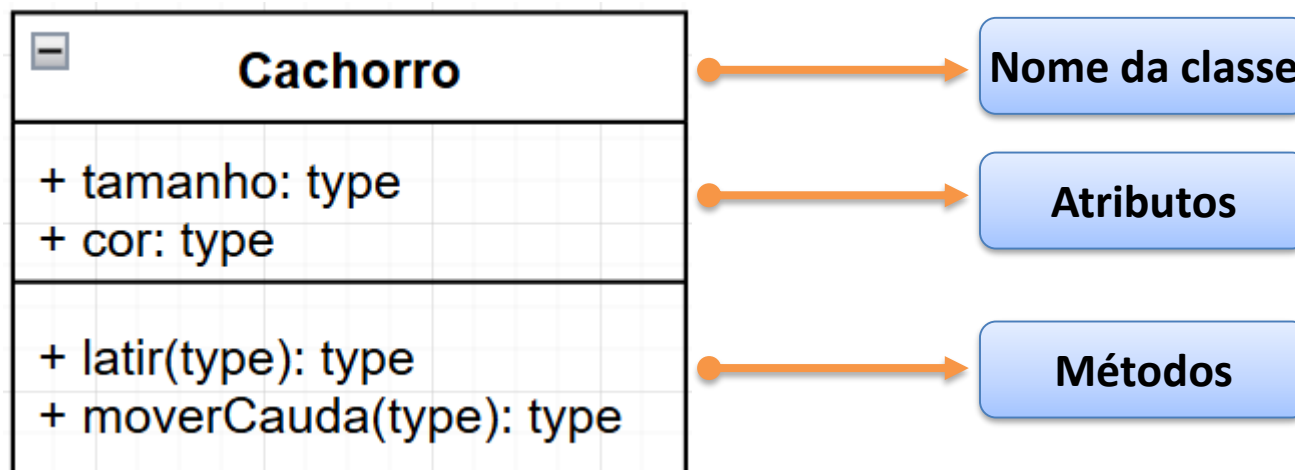
Podemos representá-lo também através de seu “comportamento” (**métodos**):

- Ele é capaz de ***latir***
- Ele é capaz de ***balançar a cauda***



Identificação de Classes

Logo, *Muttley* é um objeto da classe **Cachorro** da qual ele faz parte.



Atenção para a **convenção de nomenclatura** para os **atributos** e **métodos**, devem começar com letra minúscula e a primeira letra de cada nova palavra maiúscula (não deve ter espaços).

A **UML** (Unified Modeling Language) é uma linguagem visual padronizada para modelar sistemas de software. Ela ajuda a representar a estrutura e o comportamento de um sistema de forma clara e compreensível.

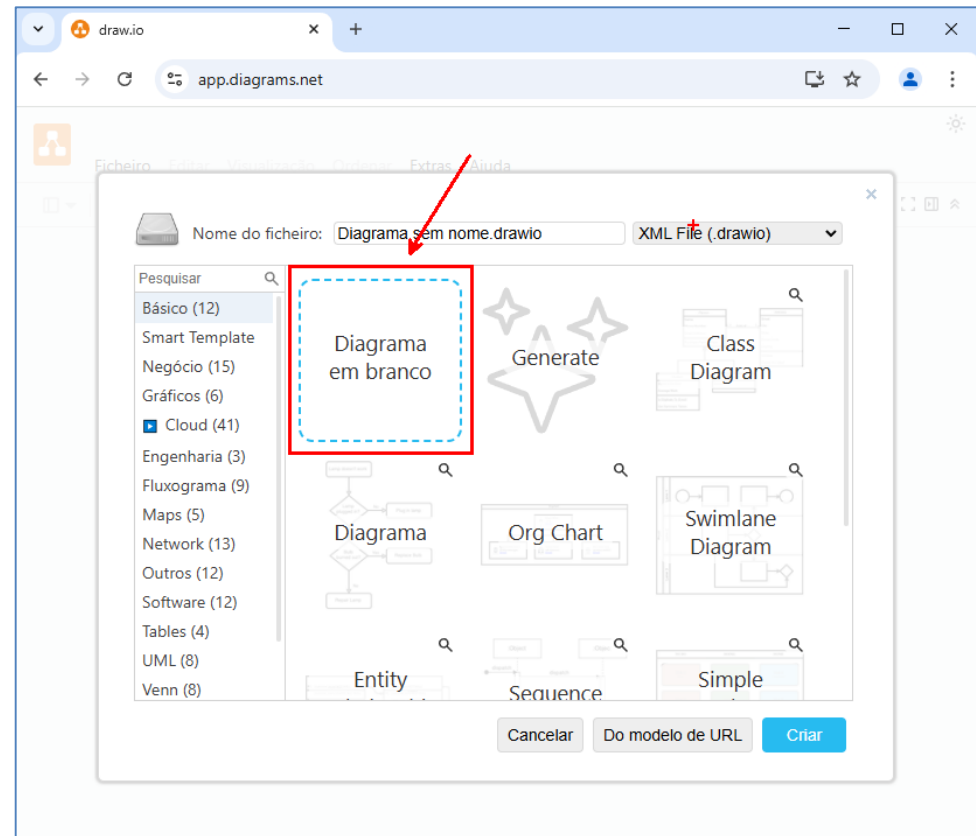
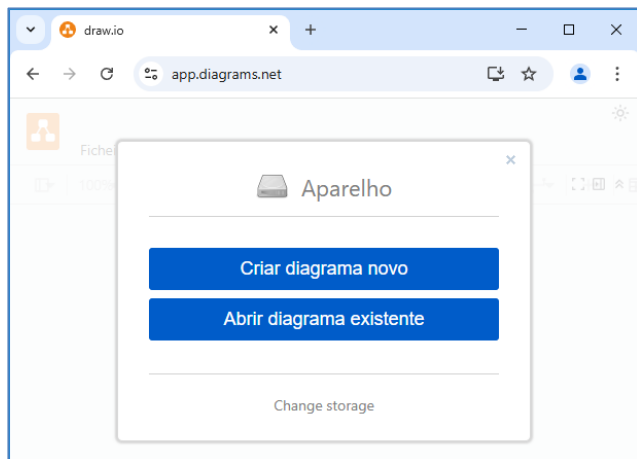
◆ **Objetivo:** Facilitar a comunicação entre desenvolvedores, arquitetos de software e *stakeholders*.

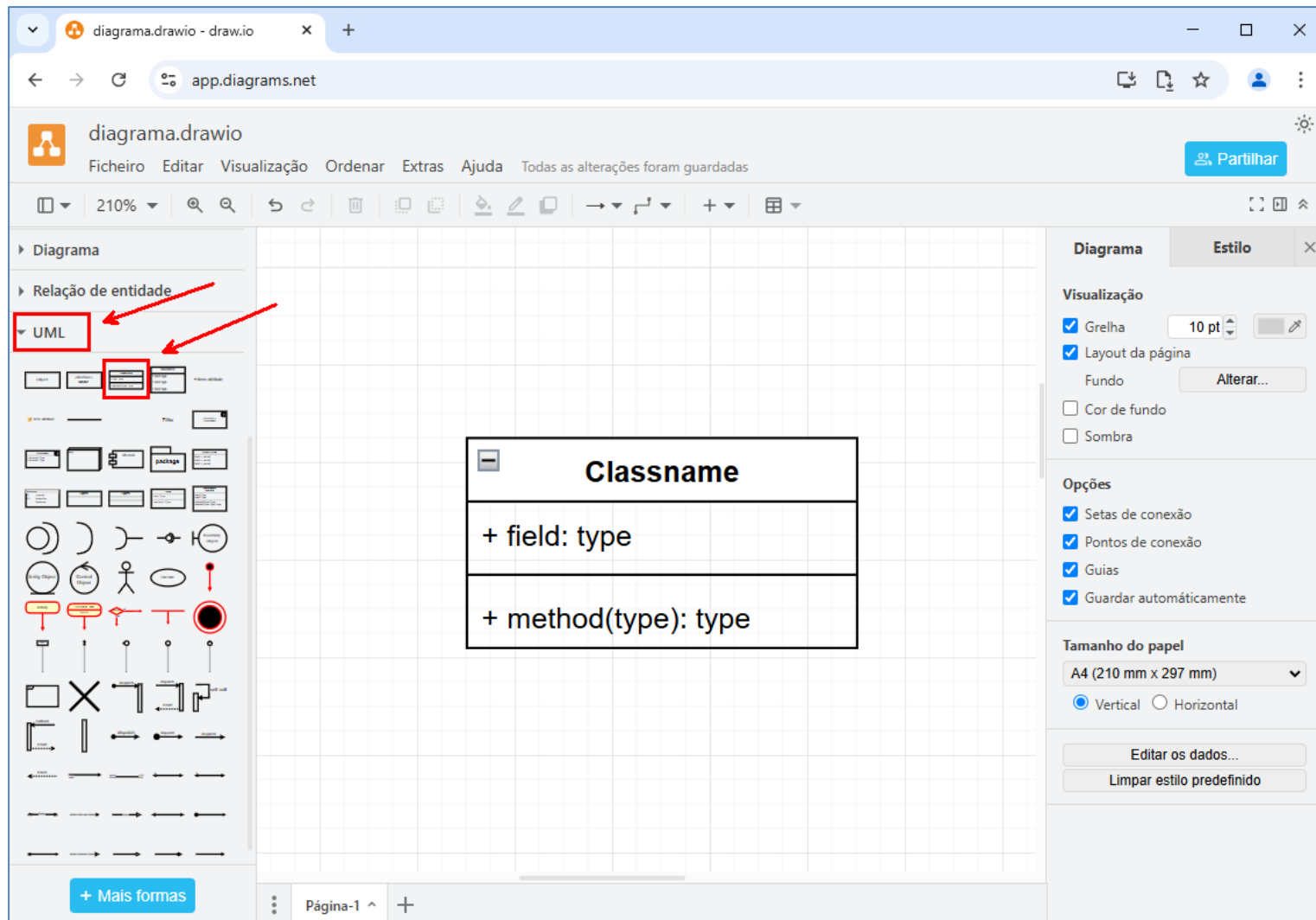
◆ **Principais Diagramas:** UML possui vários diagramas, como diagrama de classes, casos de uso, sequência, atividades, entre outros.

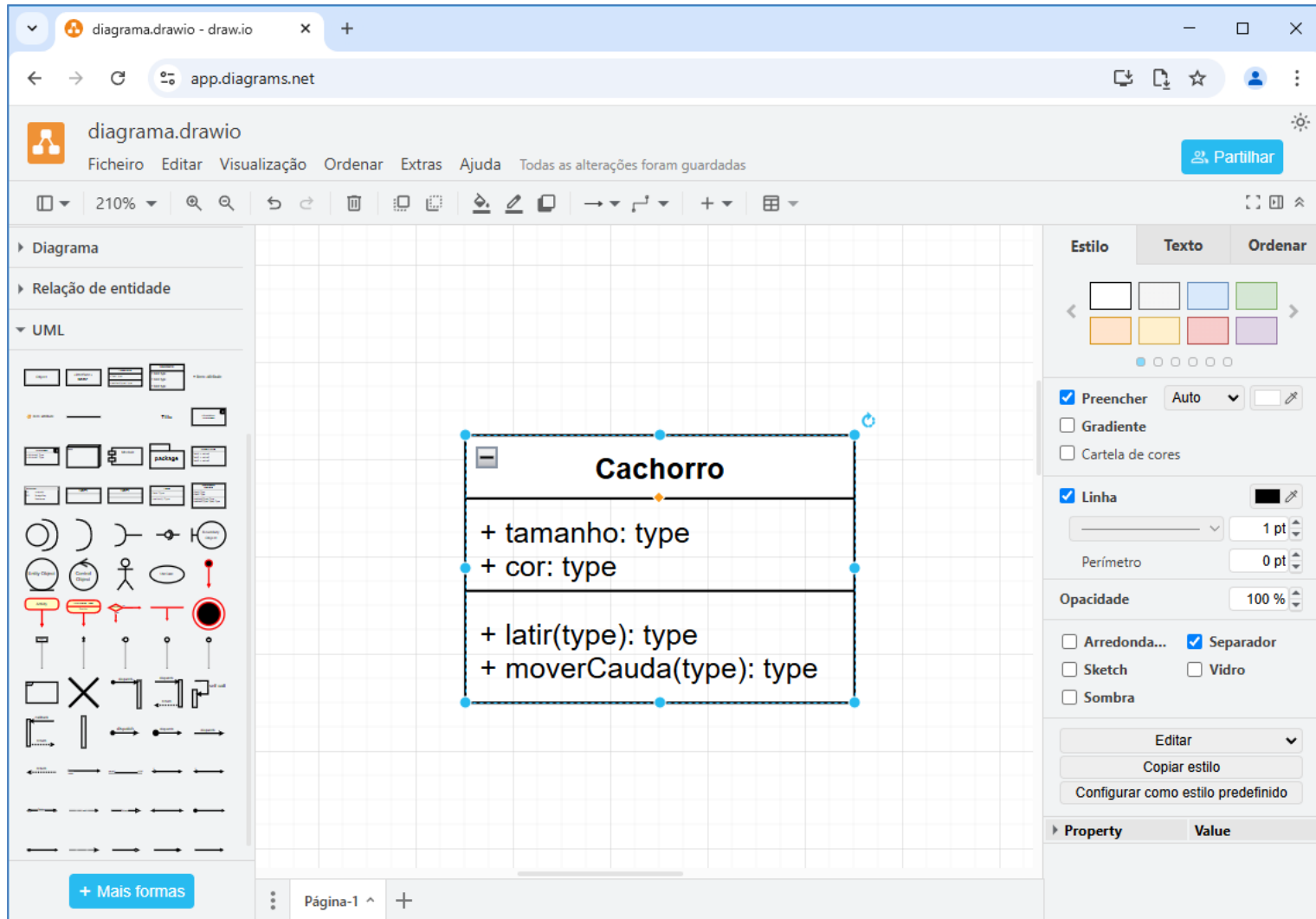
O **diagrama de classes** é um dos mais importantes da UML. Ele representa a estrutura estática do sistema, mostrando:

- ✓ Classes e seus atributos
- ✓ Métodos (comportamentos das classes)
- ✓ Relacionamentos entre as classes (associação, herança, agregação, composição)

Para desenvolver **diagrama de classes** com a linguagem **UML** recomendo o uso da ferramenta **app.diagrams.net** (antigo **draw.io**). Ferramenta de desenho de diagramas **online** que não necessita instalação.

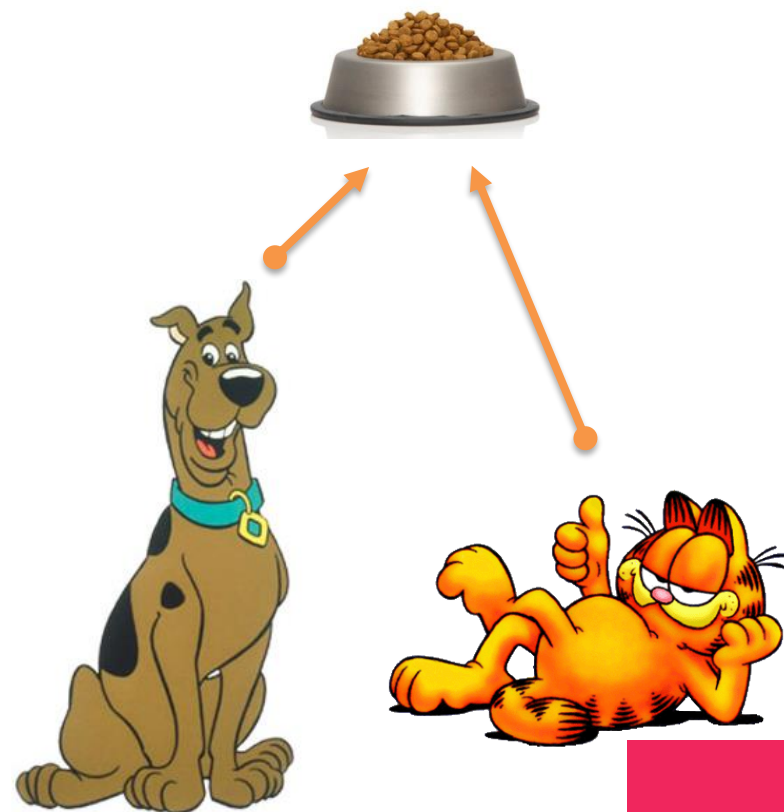
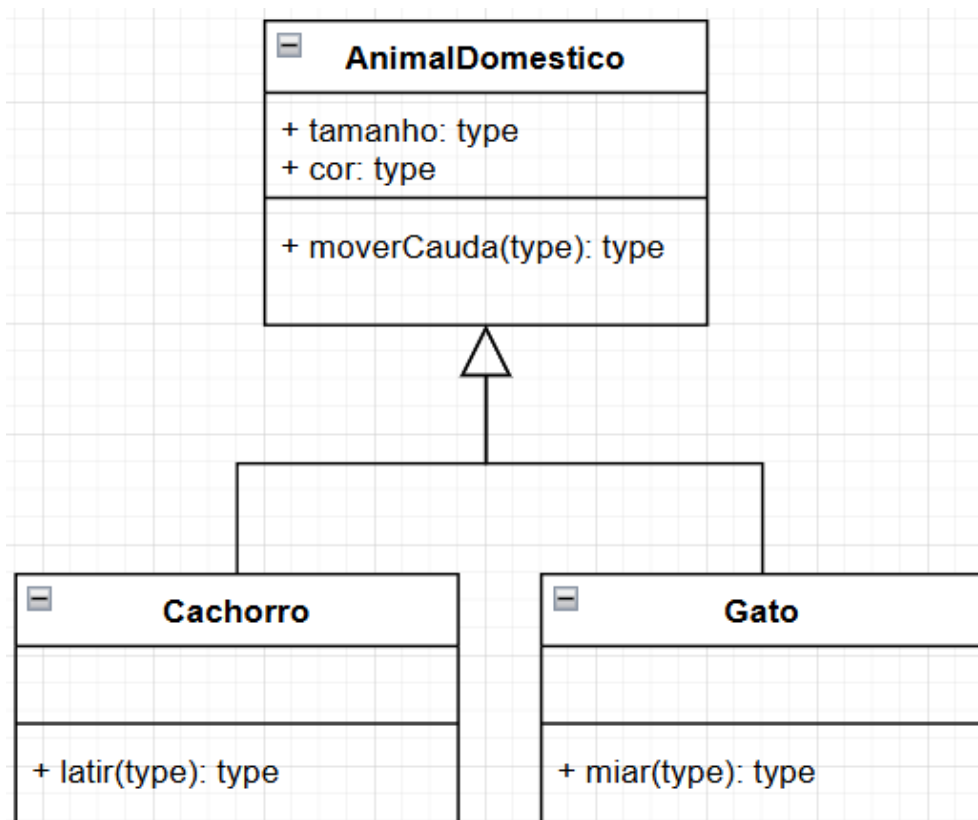


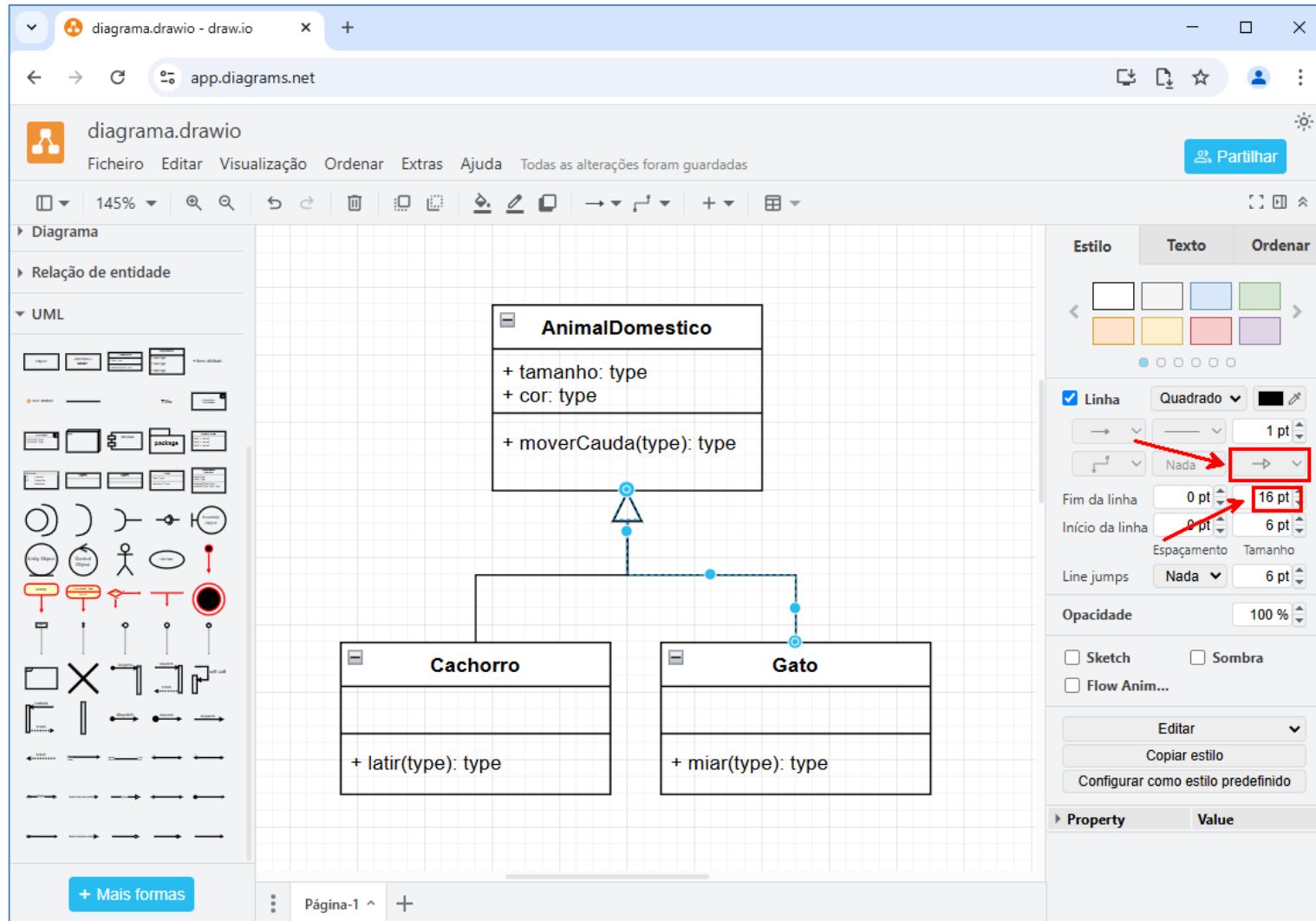




Em orientação a objeto podemos criar uma classe base e, a partir desta classe, criar subclasses relacionadas. As subclasses **herdarão** todos os atributos e métodos da classe base e poderão ter seus próprios atributos e métodos.

Exemplo:







Java como programar. Paul Deitel e Harvey Deitel. Pearson, 2011.

Java 8 – Ensino Didático : Desenvolvimento e Implementação de Aplicações. Sérgio Furgeri. Editora Érica, 2015.

Até breve!