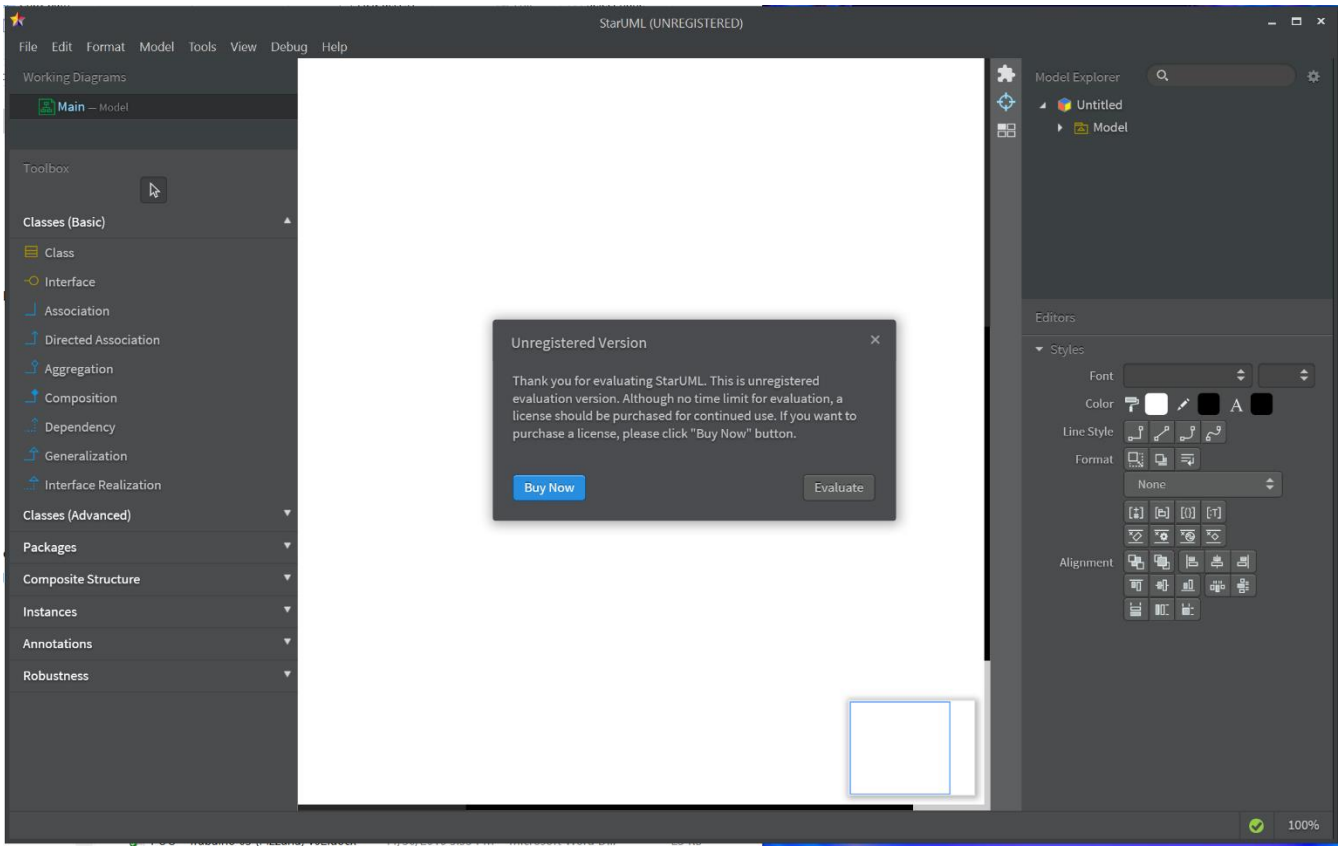
Roteiro 01 – Atividade 01/04

1. Procure pela ferramenta StarUML (), a qual deve estar instalada nos equipamentos do laboratório. Esta é uma ferramenta de modelagem UML (*Unified Modeling Language*). Para quem deseja instalá-la em seu computador pessoal, o link é http://staruml.io/. Existem versões para diferentes sistemas operacionais (Windows, Linux, MacOS). Ao executá-la, selecione **Evaluate**. Como nosso foco é somente educacional, não iremos comprá-la (não há limite de tempo para avaliação). Entretanto, caso você venha a utilizá-la comercialmente, por favor, verifique as orientações no site.



StarUML é considerada uma **ferramenta CASE (*Computer Aided Software Engineering* – Engenharia de Software Assistida por Computador)**. Qualquer ferramenta que apoie algum aspecto do ciclo de desenvolvimento de software pode ser considerada uma ferramenta CASE. Além de diagramas UML, a ferramenta ainda suporta modelos entidade-relacionamento (para bancos de dados relacionais). Entretanto, nesta disciplina, utilizaremos somente dois diagramas UML:

- **Diagrama de classe** (visão estrutural, estática) – permite modelar visualmente classes, seus membros (atributos e operações) e seus relacionamentos (dependência, generalização/especialização, associação, agregação, composição).
- **Diagrama de sequência** (visão comportamental) – permite modelar visualmente um ou mais métodos. Na UML, um método é a implementação de uma operação. O método especifica o algoritmo ou procedimento associado com uma operação.

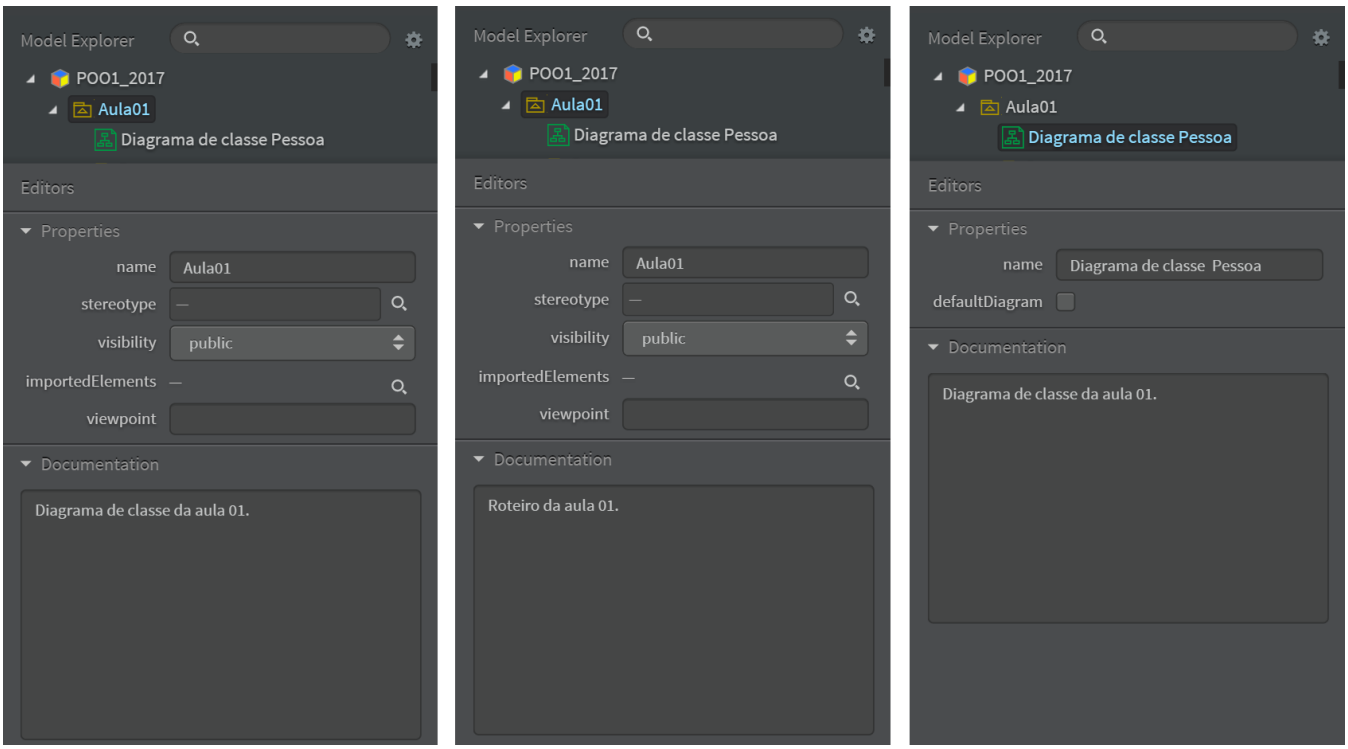
O objetivo deste roteiro é apresentar conceitos básicos para utilização da ferramenta, os quais permitirão a execução das atividades planejadas. Entretanto, recomenda-se que você explore a ferramenta (extraclasse) e fique familiarizado com ela. Você pode começar pelo link http://staruml.io/support.

> **ATENÇÃO**
>
> Todos os diagramas solicitados nesta disciplina **devem estar obrigatoriamente** no formato da ferramenta StarUML 2.8.0 ou superior. **Não serão considerados** arquivos em **outros formatos**, incluindo imagens. **Trabalhos idênticos** ou que apresentarem **inconsistências de autoria** também **não serão considerados**.

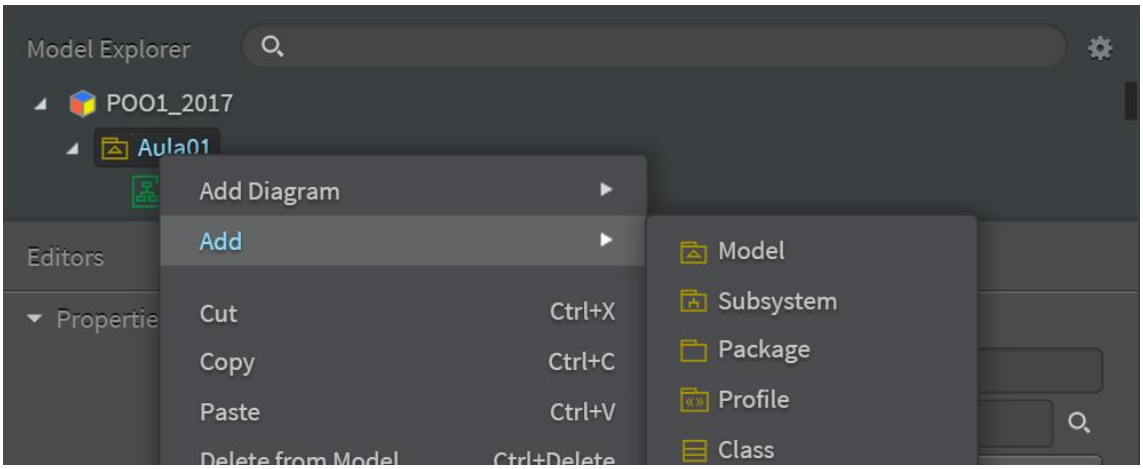
Roteiro 01 – Atividade 01/04

2. Note que um projeto default é aberto sem nome (*Untitled*). A área **Model Explorer** (lado direito) é utilizada para a organização da documentação do seu projeto. Cada projeto é armazenado em um arquivo único com extensão "**.mdj**". Um projeto representa o nível mais alto na hierarquia de documentação. Um projeto pode ser descrito em um ou mais modelos (**Model**). Modelos, por sua vez, podem conter diferentes elementos (ex: diagramas, pacotes, classes, etc.).

   Para utilizar a ferramenta, você precisa conhecer a diferença entre um modelo e um diagrama. Um modelo é a descrição de qualquer aspecto do sistema (estrutura, comportamento, requisitos, etc.) e pode ser representado de várias formas (textualmente, matematicamente ou visualmente). Um diagrama é uma representação visual de um modelo. Desta forma, um modelo pode ser representado por um ou mais diagramas que exploram diferentes aspectos (ex: diagrama de classe e um diagrama de sequência). Além do projeto, a ferramenta cria, automaticamente, um modelo "*Model*" e um diagrama "*Main*". Na figura abaixo, as propriedades destes elementos foram alteradas. Experimente fazer o mesmo.

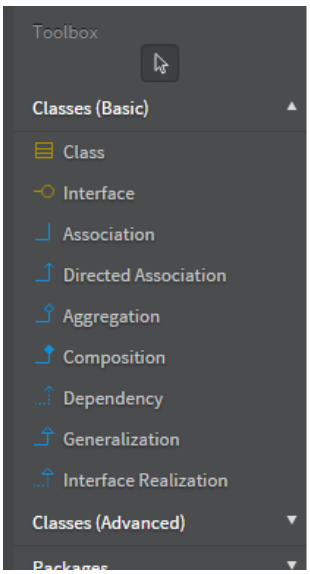


Ao clicar com o botão direito sobre um destes elementos, é apresentado um menu de contexto com várias opções, inclusive a inclusão de outros diagramas.



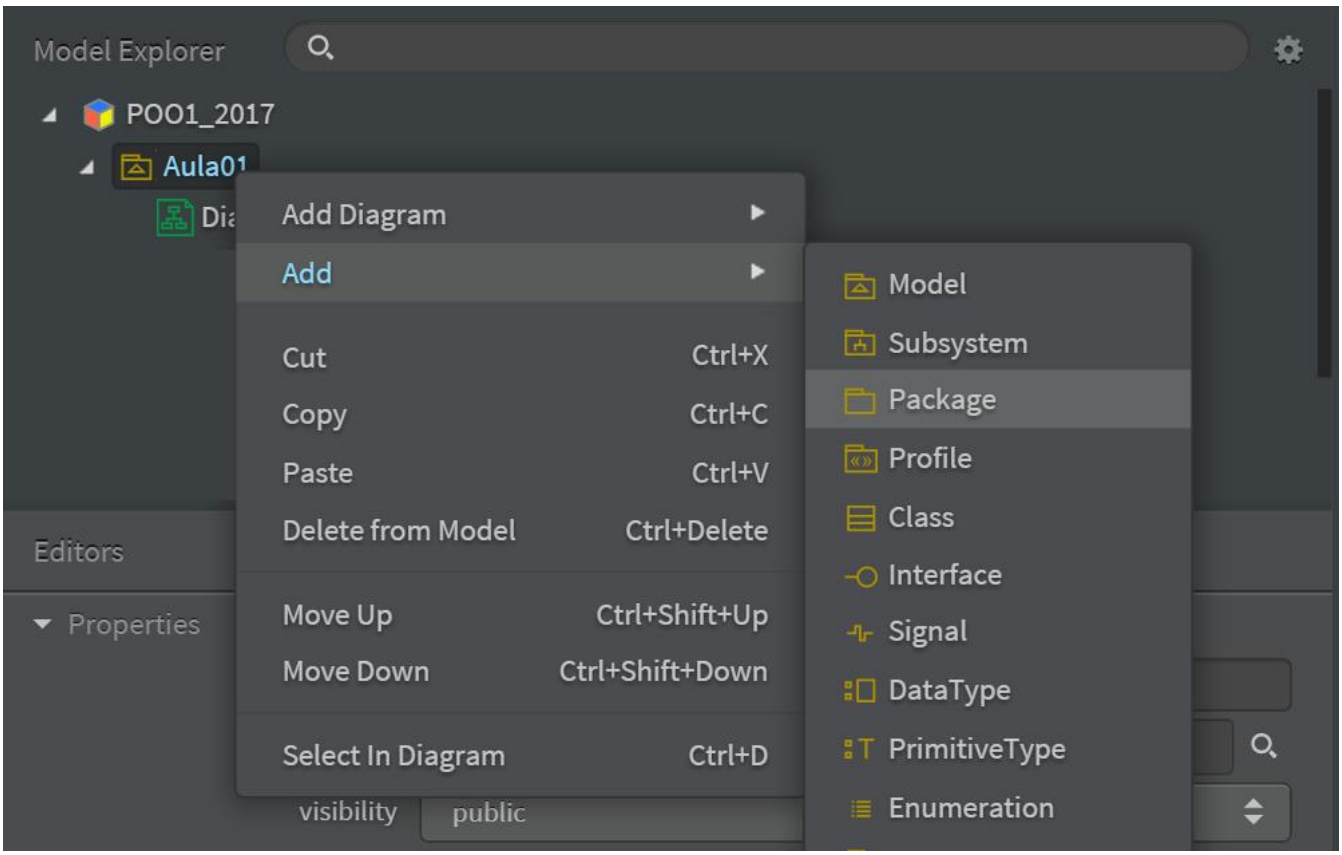
Roteiro 01 – Atividade 01/04

3. Repare também que além da área central, onde os diagramas são editados, há uma outra área chamada **Toolbox** (caixa de ferramentas) que fica do lado esquerdo. Esta área apresenta uma paleta de elementos de notação conforme o tipo de diagrama selecionado. Como o diagrama default é um diagrama de classe, os elementos disponibilizados são aqueles compatíveis com este tipo de diagrama.



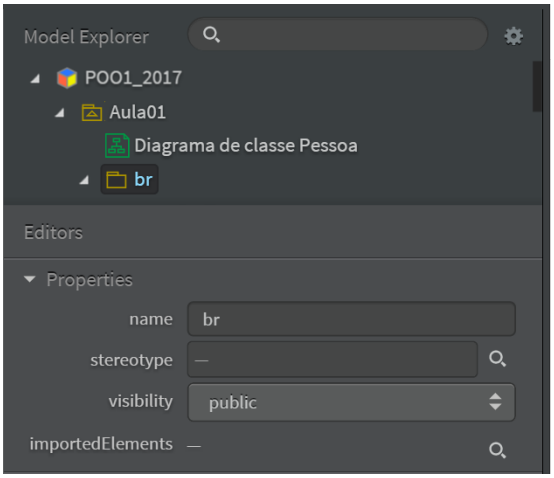
Clique no elemento de notação desejado e depois clique no diagrama. O elemento será criado na posição indicada. No caso de relacionamentos, é necessário que você tenha, pelo menos, dois elementos. Para fazer a ligação, clique no elemento de origem e arraste o relacionamento até o elemento destino.

4. Para modelarmos a nossa primeira classe, iremos inicialmente criar uma estrutura de pacotes. Um pacote (*package*) é um mecanismo para organizar diagramas, classes, pacotes (subpacotes) e outros elementos dentro de espaços de nome (*namespaces*). Um espaço de nome é uma forma de garantir, por exemplo, que uma classe tenha um nome único. Pense em um pacote como sendo algo similar a um diretório contendo classes e outros arquivos relacionados (o espaço de nome seria o nome do arquivo com seu caminho completo). Da mesma forma que um diretório, se você excluir um pacote do modelo, tudo que estiver dentro daquele pacote também será excluído. Para criar um pacote, com o botão direito sobre o modelo, selecione [Add | Package].



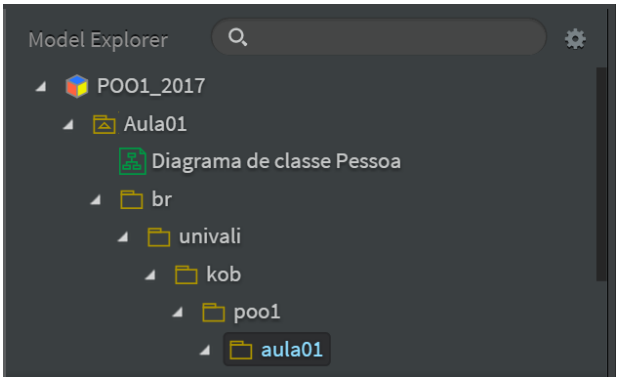
Roteiro 01 – Atividade 01/04

5. Edite o nome do pacote para "**br**". Este pacote raiz será utilizado para conter todas as nossas classes.

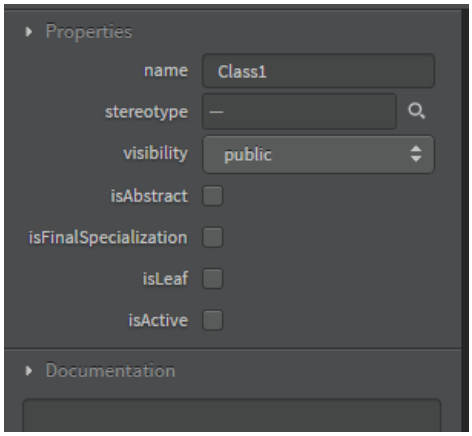


> Durante a disciplina, adotaremos a boa prática de nomear nossa estrutura de pacotes a partir da URL invertida da organização (ex: br.univali.kob.poo1...). Como uma URL é única em todo o mundo, podemos garantir que nossas classes terão nomes únicos. Isso permite a coexistência de duas classes com o mesmo nome, desde que em pacotes diferentes (os *namespaces* serão diferentes). Isso é particularmente útil quando você utiliza bibliotecas de terceiros. Estas bibliotecas podem ter utilizado um mesmo nome que você havia adotado. Entretanto, o espaço de nome (*namespace*) gerado pelo pacote garante a individualidade.

6. Repita o processo anterior, criando a estrutura de pacotes **br.univali.kob.poo1.aula01**.



7. Agora, podemos modelar nossa primeira classe: **Person**. Durante a disciplina, a maior parte dos exemplos de modelos e códigos estarão em Inglês (comentários e orientações estarão em Português para facilitar o entendimento). Escrever o código em Inglês é uma boa prática para permitir colaboração internacional e trocas de ideias em fóruns técnicos. O ideal seria que os comentários também fossem em Inglês, mas isso ficará para mais tarde. Aproveite para melhorar seu Inglês, pois esta competência é fundamental para a nossa área. Entretanto, se você ainda quiser escrever seu código em Português, fique à vontade. Para criar uma classe, clique no elemento **Class** na área **Toolbox** e clique no diagrama. Note que, ao fazer isso, o elemento é inserido no diagrama em modo de edição. Passe o mouse sobre as várias opções (não discutiremos todas agora) de atalho. Este recurso aumenta a produtividade. Note também que as propriedades (lado inferior direito) foram ajustadas para uma classe.

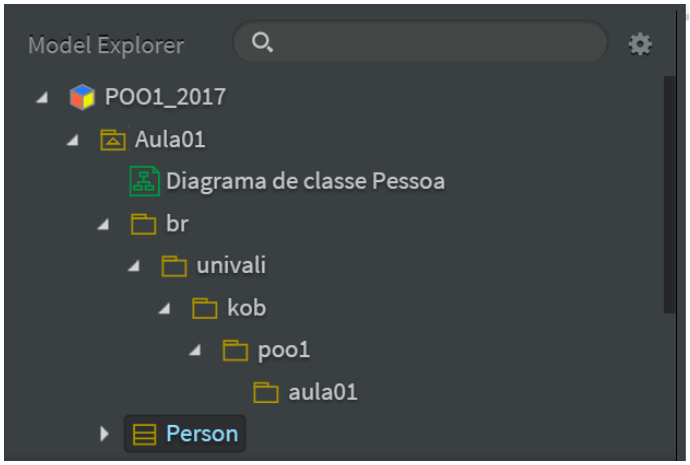


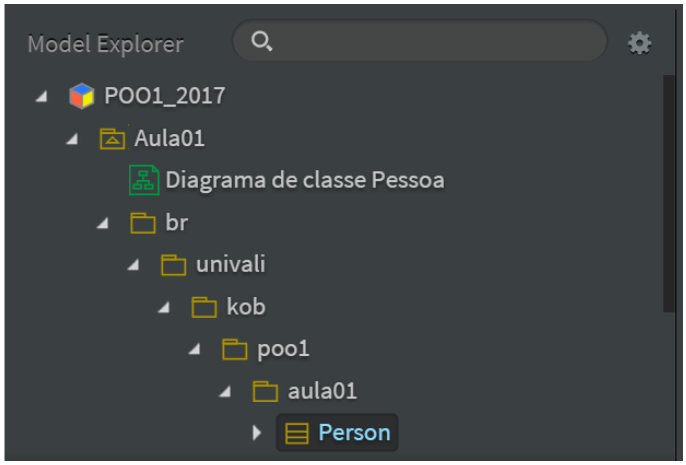
Roteiro 01 – Atividade 01/04

As **propriedades** disponíveis para uma **classe** são definidas pela **linguagem de modelagem UML**:

- **Name** – Nome da classe.
- **Stereotype** (estereótipo) – Recurso que permite estender a UML (não será utilizado agora).
- **Visibility** – Indica a visibilidade da classe (pública, protegida, privada ou *package*). Por enquanto, trabalharemos apenas com classes públicas.
- **isAbstract** – Indica se a classe é abstrata (não permite que objetos sejam instanciados a partir dela). Objetos só podem ser criados a partir de suas especializações (subclasses).
- **isFinalSpecialization** – Indica que a classe não pode ser especializada (não pode ter subclasses). Você percebeu que não faz sentido marcar esta opção se a classe for abstrata?
- **isLeaf** – Mesmo que **isFinalSpecialization** (a especificação UML não deixa claro o motivo para duas propriedades com o mesmo objetivo). Quando uma estiver marcada, não esqueça de marcar a outra também (manter consistência).
- **isActive** – Uma classe pode ser ativa ou passiva. Uma classe é ativa quando cada uma de suas instâncias são ativas, ou seja, seus objetos têm controle total sobre sua execução. Uma classe é passiva quando seus objetos são executados no contexto de algum outro objeto. Não utilizaremos classes ativas por enquanto. Logo, deixe sempre esta propriedade desmarcada.

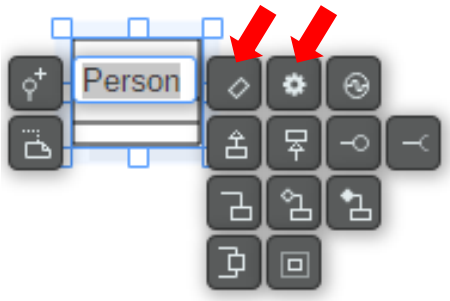
8. Altere o nome da classe para **Person** e pressione a tecla **ENTER** ou clique em alguma outra área do diagrama. Veja no **Model Explorer** que a classe criada está fora do pacote **br.univali.kob.poo1.aula01**. Para resolver este problema, basta arrastar a classe (no **Model Explorer**) para dentro do pacote desejado. Podemos fazer isso com qualquer elemento, facilitando a reorganização da documentação quando necessário.





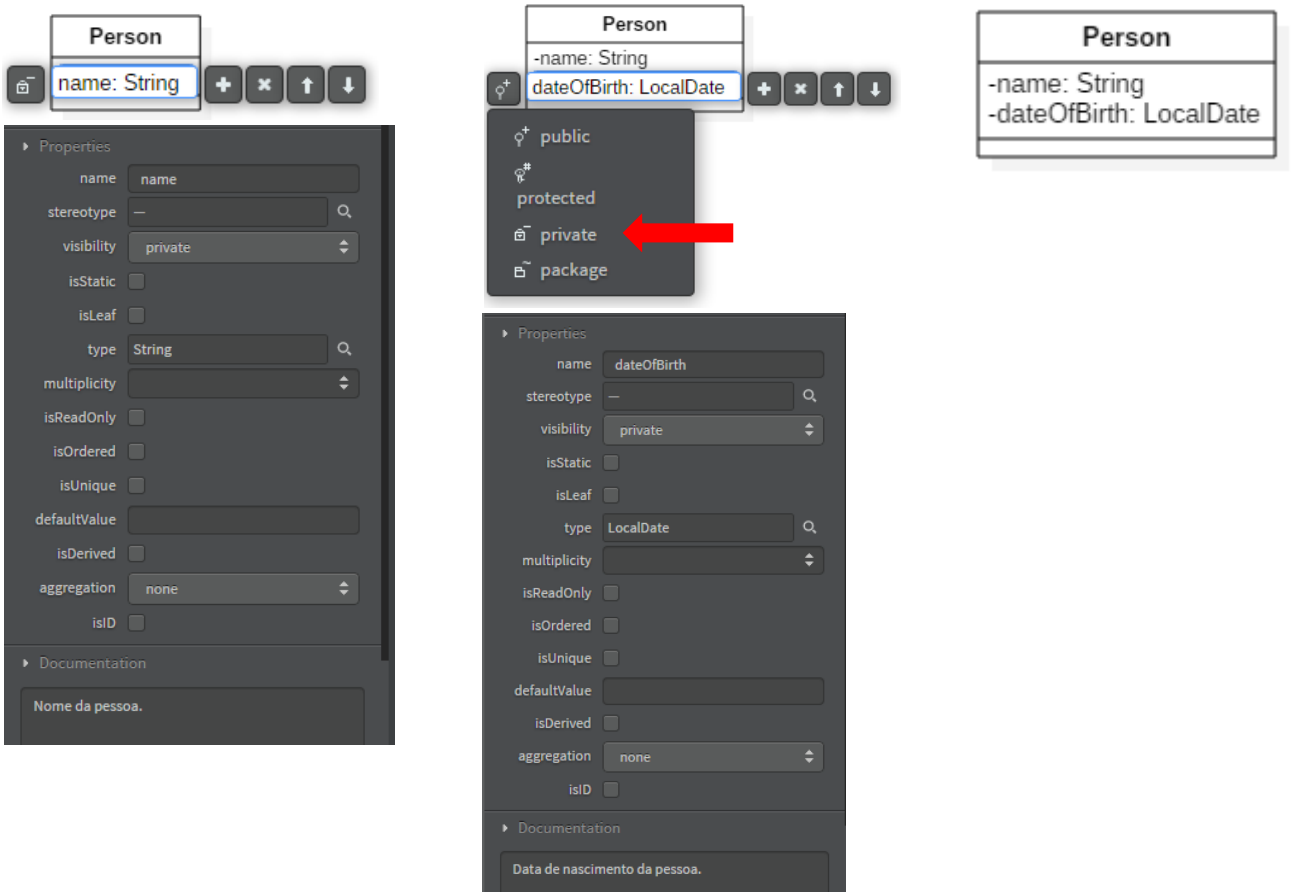
9. Se você quiser alterar as configurações de visualização (por exemplo, desativar o grid), selecione a opção [File | Preferences | General].

10. Se você fizer um duplo clique sobre a classe, ela voltará a entrar em modo de edição com vários atalhos disponibilizados. O primeiro atalho do lado direito permite adicionar um atributo e o segundo, uma operação.



Roteiro 01 – Atividade 01/04

11. Adicione dois atributos PRIVADOS (atributos são sempre **PRIVADOS**).



As **propriedades** disponíveis para um **atributo** são definidas pela **linguagem de modelagem UML**:

Name – Nome do atributo.

Visibility – Atributos são sempre PRIVADOS.

i**sStatic** – Atributo de classe. O valor de um atributo de classe (estático) é compartilhado entre todos os objetos.

isLeaf – Não pode mais ser redefinido em subclasses (equivale ao modificador final em Java).

type – Tipo conforme a linguagem adotada (no nosso caso, utilizaremos tipos válidos em Java ou nossas classes).

multiplicity – 1 (um único valor do tipo especificado).
0..1 (pode ser null).
* (uma coleção de valores do tipo especificado).
1..* (uma coleção que contém pelo menos um valor).
n .. m (uma coleção que contém entre n e m valores).

isReadOnly – O atributo não pode ter seu valor modificado (equivale ao modificador final em Java).

isOrdered – Os valores da coleção estão ordenados (multiplicidade precisa ser maior do que 1).

isUnique – Não existem valores duplicados na coleção (multiplicidade precisa ser maior do que 1).

defaultValue – Valor inicial (garantir que o atributo tem este valor logo após a criação do objeto).

isDerived – O valor do atributo é calculado a partir dos valores de outros atributos. Não deveria permitir alteração direta.

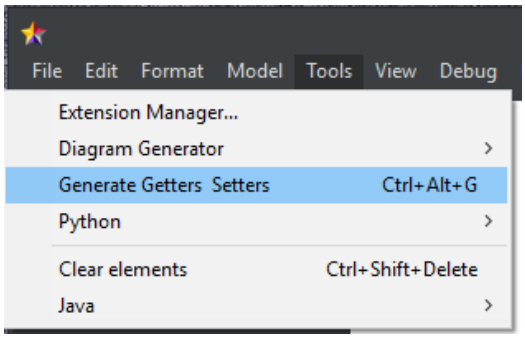
aggregation – Usada para indicar o tipo de agregação que se aplica ao atributo (veremos isso mais tarde).

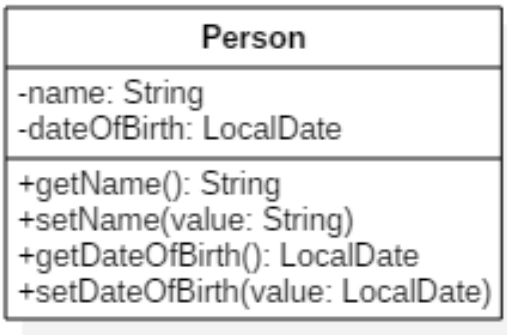
isID – Indica que o atributo pode ser usado como um identificador único para os objetos da classe (ou parte deste identificador/similar a uma chave primária em banco de dados).

Roteiro 01 – Atividade 01/04

12. Além das propriedades, você deve ter notado que o campo **Documentation** foi preenchido. Não deixe de fazer o mesmo, pois este campo será utilizado como comentário na geração do código a partir da ferramenta.

13. Podemos agora modelar as operações. Como, neste momento, queremos apenas **getters** (operações de acesso) e **setters** (operações modificadoras), podemos utilizar um recurso da ferramenta. Selecione a classe e depois selecione a opção [Tools | Generate Getters Setters Ctrl+Alt+G]. A ferramenta cria automaticamente as operações, utilizando sempre o nome **value** no parâmetro dos **setters**.





Esta extensão, assim como a extensão para geração de código em Java, não está habilitada inicialmente (mas o pessoal do laboratório já deve ter habilitado). Para habilitá-las, você deve selecionar antes a opção [Tools | Extension Manager] e instalar as extensões desejadas (Java, Generate Getters and Setters).

14. Se você clicar em uma operação criada ou precisar incluir uma operação específica, você também precisará trabalhar as propriedades.

As **propriedades** disponíveis para uma **operação** são definidas pela **linguagem de modelagem UML**:

Name – Nome da operação.

Visibility – Visibilidade da operação (pública, protegida, privada ou *package*).

i**sStatic** – Operação de classe. Operações especiais que podem ser invocadas a partir da classe, sem a necessidade de instanciar um objeto. Podem manipular somente atributos de classe (estáticos).

isLeaf – Não pode mais ser redefinida em subclasses (equivale ao modificador final em Java).

raisedExceptions – Indica quais as exceções que podem ser disparadas a partir da operação (similar à cláusula *throws* do Java).

isQuery – Sua execução não altera o estado do objeto (operação somente de acesso).

isAbstract – Indica se a operação é abstrata, ou seja, não é implementada nesta classe (não tem método). A implementação (método) é delegada para as subclasses.

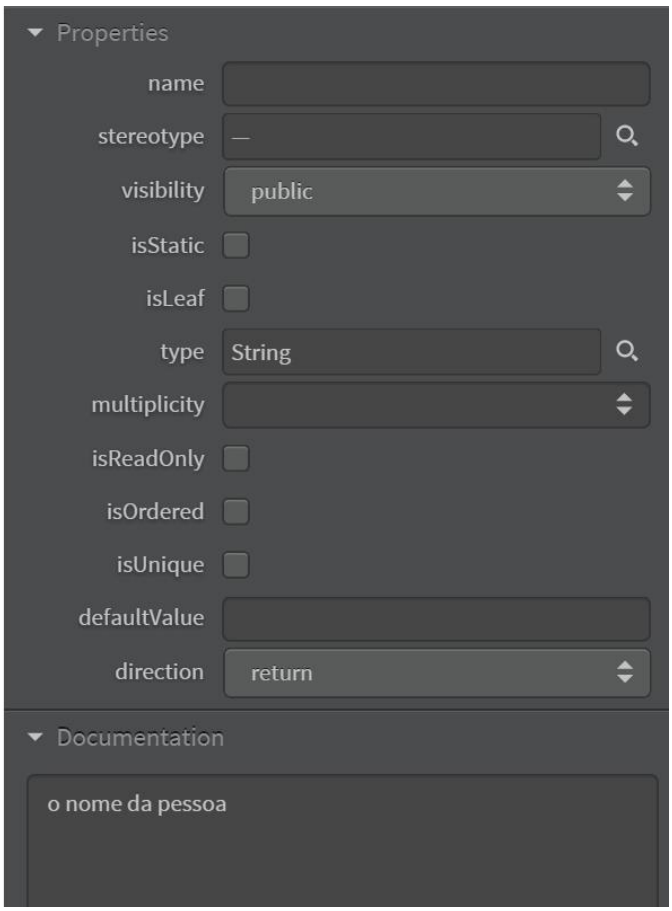
As propriedades dos **parâmetros** (inclusive o de retorno) são basicamente as mesmas de um atributo. A diferença é que a propriedade **isID** não está presente (ela não faz sentido em parâmetros) e que há uma nova propriedade:

direction – indica o tipo de passagem de parâmetro (valor ou referência).

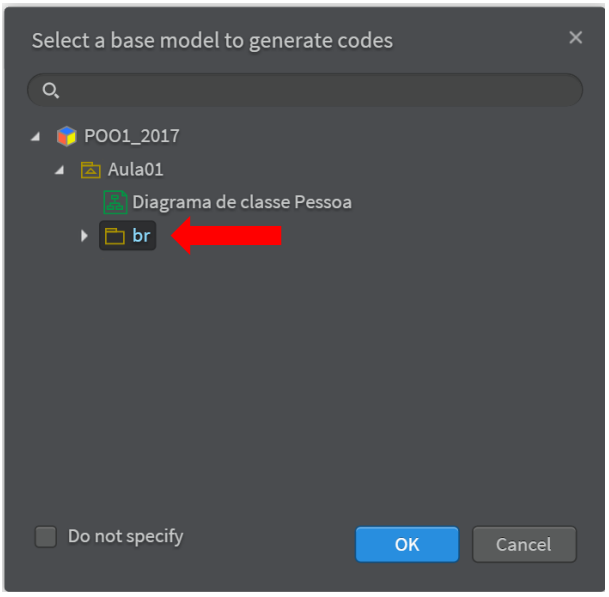
- **in** - Parâmetro de entrada que não pode ser modificado.
- **out** - Parâmetro de saída que pode ser modificado e novo valor pode ser acessado pelo chamador.
- **inout** - Parâmetro de entrada que também pode ser modificado e acessado pelo chamador.
- **return** - representa o valor de retorno de uma chamada.

Roteiro 01 – Atividade 01/04

15. Para ter acesso aos parâmetros, você pode fazer um duplo clique sobre a operação no diagrama ou navegar pelo **Model Explorer**. Novamente, perceba que a descrição do parâmetro foi preenchida no campo **Documentation**. Quando o código for gerado, este campo também será utilizado para documentar o código.



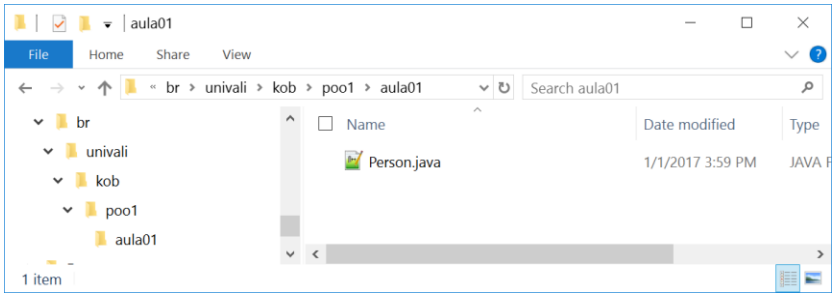
16. Antes de utilizarmos a extensão para geração de código, você deve garantir que os nomes dos pacotes não possuem acentos nem espaços (use "_" quando for conveniente). Confirmado que tudo está ok, selecione a opção [Tools | Java | Generate Code...]. Na janela de diálogo, selecione o pacote desejado (todas as classes a partir deste pacote serão geradas). No nosso caso, selecione o primeiro pacote **br** (veja a imagem abaixo) e pressione Ok. Escolha o diretório onde o código será gerado e confirme.



Uma limitação da ferramenta é que o diretório selecionado não pode ter um subdiretório com o mesmo nome do pacote selecionado. Por exemplo, se o diretório que você escolheu já tiver um subdiretório chamado **br**, o gerador de código não gerará as classes. Outro problema de qualidade desta extensão é que o gerador não dá feedback (sucesso ou falha).

Roteiro 01 – Atividade 01/04

17. Abra o diretório que você selecionou no passo anterior e note que o arquivo **Person.java** foi criado, seguindo a mesma estrutura de pacotes que você utilizou na ferramenta.



18. Embora a formatação da classe não seja exatamente a que utilizaremos (adotaremos o padrão do NetBeans), você está vendo a implementação da classe. Em ferramentas CASE mais completas, é possível configurar várias características da geração de código. Entretanto, conseguiremos importar esta classe para o NetBeans.

```
package br.univali.kob.poo1.aula01;

import java.util.*;

/**
 * @author Marcello Thiry
 */
public class Person {

    /**
     * Default constructor
     */
    public Person() {
    }

    /**
     * Nome da pessoa.
     */
    private String name;

    /**
     * Data de nascimento da pessoa.
     */
    private LocalDate dateOfBirth;

    /**
     * @return o nome da pessoa
     */
    public String getName() {
        // TODO implement here
        return "";
    }

    /**
     * @param value o nome da pessoa
     */
    public void setName(String value) {
        // TODO implement here
    }

    /**
     * @return a data de nascimento da pessoa
     */
    public LocalDate getDateOfBirth() {
        // TODO implement here
        return null;
```

Nome do pacote já está correto.

Sempre fixo: precisa alterar conforme classes utilizadas (ex: java.time.LocalDate).

A ferramenta sempre gera um **construtor default**. Por enquanto, não precisamos modifica-lo.

Comentários **javadoc** gerados a partir do campo **Documentation**.

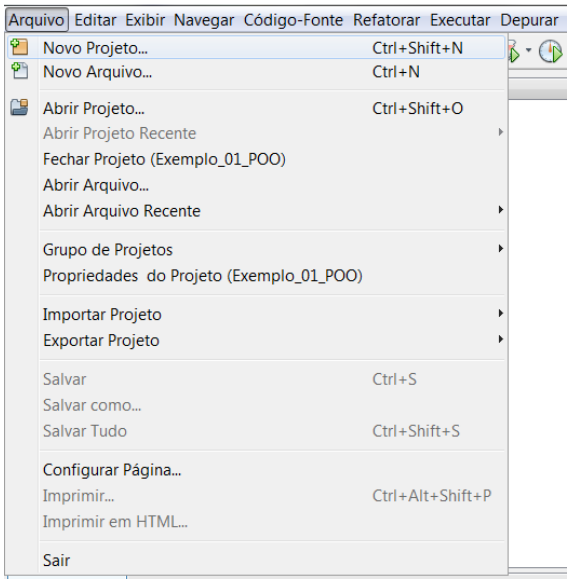
Revisar os métodos. Lembre-se que um método é a implementação de uma operação.

Roteiro 01 – Atividade 04/04

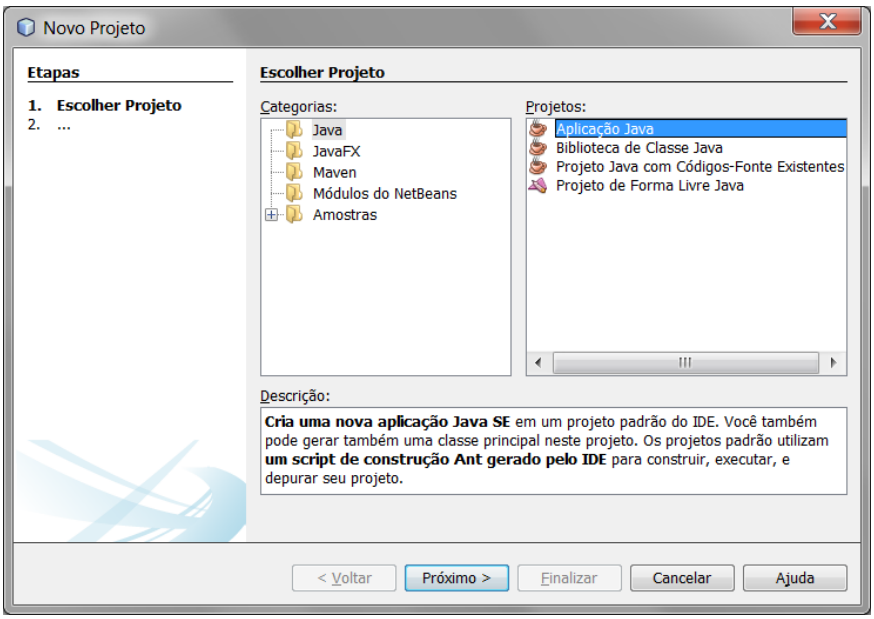
1. Inicie o NetBeans IDE (*Integrated Development Environment* – Ambiente Integrado de Desenvolvimento). Para mais informações sobre o NetBeans, consulte https://NetBeans.org/kb/index.html

> **ATENÇÃO**
>
> Todos os programas solicitados nesta disciplina **devem estar obrigatoriamente** no formato da ferramenta NetBeans 8.1 ou superior, considerando JDK 8 ou superior. **Não serão considerados** programas em **outros formatos**. Também **não devem ser utilizadas bibliotecas e/ou frameworks de terceiros**, a não ser que tenha sido dada orientação explícita do seu professor. Antes de submeter sua atividade, tenha certeza que o programa abrirá e executará corretamente neste IDE. **Trabalhos idênticos** ou que apresentarem **inconsistências de autoria** também **não serão considerados**.

2. Escolha [Arquivo | Novo Projeto (Ctrl-Shift-N)], como mostrado na figura abaixo.



3. No assistente "Novo Projeto", expanda a categoria "Java" e selecione "Aplicação Java", como mostrado na figura abaixo. Em seguida, clique em [Próximo >].



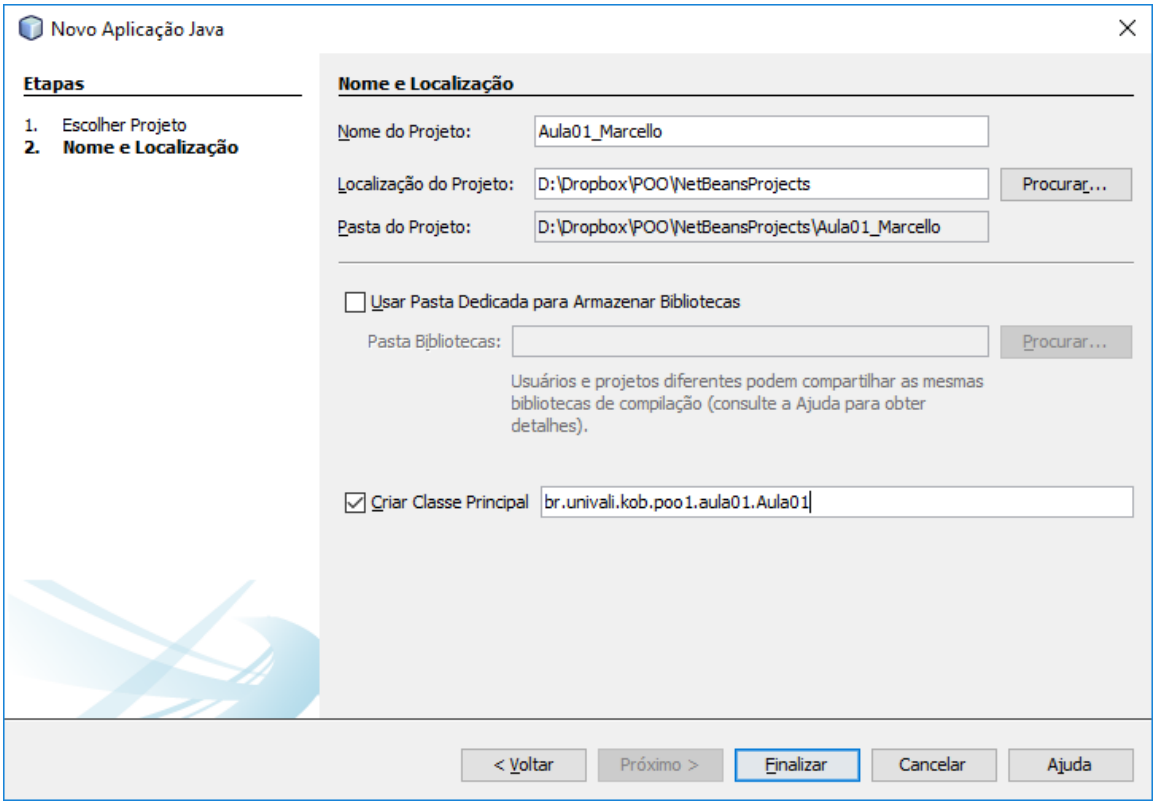
Roteiro 01 – Atividade 04/04

4. Na página “Nome e Localização” do assistente, adote o seguinte procedimento (veja figura abaixo):

   a) No campo “Nome do Projeto”, digite “Aula01_NomeAluno”, onde <NomeAluno> deve ser substituído pelo seu nome. Se houver mais de um aluno fazendo o trabalho com você, coloque o nome de cada um, separando pelo caractere “_” (*underscore*).

   b) Deixe desmarcada a caixa de seleção “Utilizar Pasta Dedicada para Armazenar Bibliotecas”.

   c) Marque a opção “Criar Classe Principal”, digite ao lado `br.univali.kob.poo1.aula01.Aula01`. Este nome corresponde ao nome do pacote onde a classe principal Java “Aula01” será criada.



5. Clique em Finalizar.

   O projeto deve ter sido criado e aberto no IDE. Agora você deve visualizar os seguintes componentes:

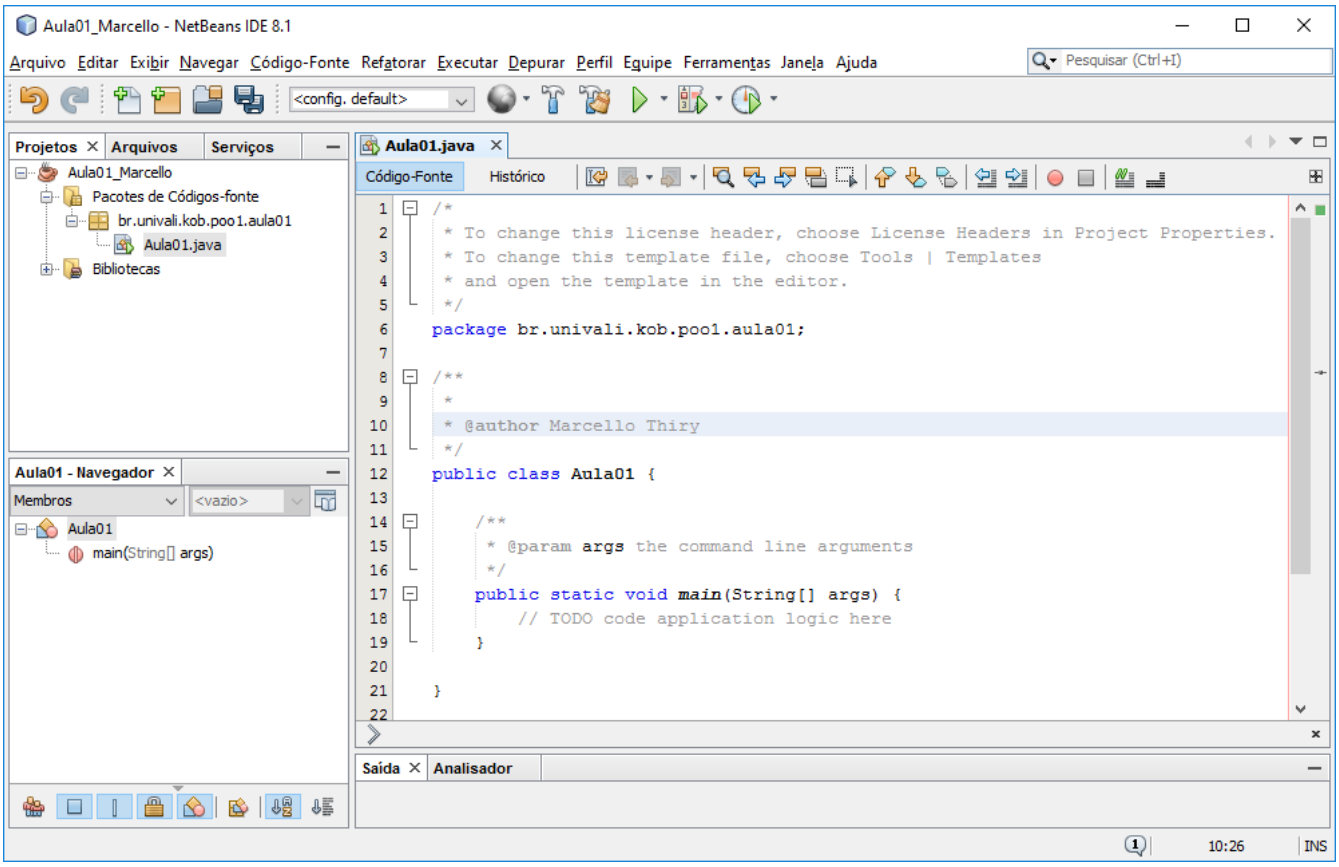
   - A janela “Projetos”, contendo uma estrutura em árvore dos componentes do projeto, incluindo arquivos de código-fonte, bibliotecas das quais seu código depende, e assim por diante.

   - A janela “Editor de Código-fonte” com um arquivo chamado **Aula01.java** é aberta. Cada arquivo é aberto numa aba específica.

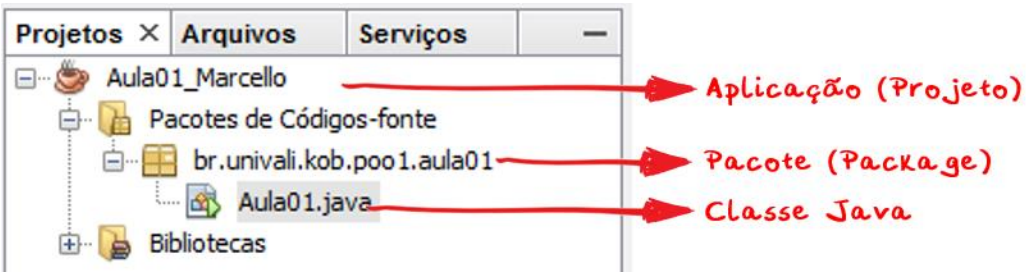
   - A janela “Navegador” que pode ser utilizada para navegar rapidamente entre elementos dentro da classe selecionada.

6. Abra o arquivo **POO1 - Slides de referência Java (Marcello Thiry).pdf**. Este documento apresenta uma série de informações sobre a sintaxe da linguagem Java, bem como dicas e orientações. Ele não tem a pretensão de substituir a documentação da Oracle, mas oferece um guia rápido com exemplos. Mantenha este arquivo sempre aberto quando estiver trabalhando nas atividades da disciplina.

Roteiro 01 – Atividade 04/04



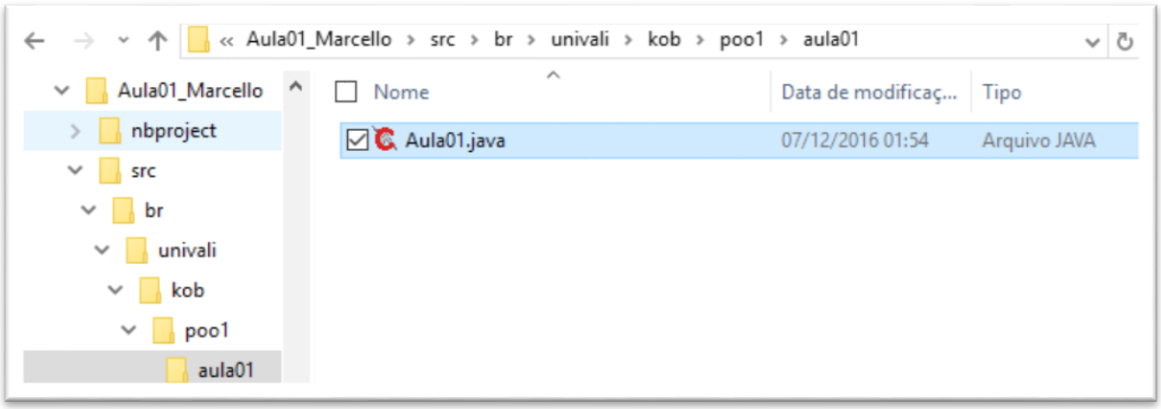
7. Você notou que o NetBeans gerou a linha "`package br.univali.kob.poo1.aula01;`". Isso foi resultado da ação que você executou em (4c). Esta linha indica que a classe **Aula01** está inserida no pacote **br.univali.kob.poo1.aula01**. Assim como visto na UML, um pacote (*package*) Java é um mecanismo para organizar classes Java (e outros elementos) dentro de espaços de nome (*namespaces*). Pense no pacote como sendo algo similar a um diretório contendo classes e outros arquivos relacionados.



Fisicamente, o NetBeans utiliza uma estrutura de diretórios para mapear os pacotes de código-fonte. Por default (pode ser alterado na instalação), o NetBeans cria um diretório chamado **NetBeansProjects** dentro do diretório padrão **Documentos** (Windows). Ao criar um novo projeto, o NetBeans aponta, por default, para este diretório (novamente, é possível modificar a localização do projeto – ver passo 4).

Na figura abaixo, você pode visualizar a estrutura de diretórios criada para o projeto criado **(...\src\br\univali\kob\poo1\aula01\Aula01.java**). Note que o NetBeans mantém toda a estrutura de pacotes de código-fonte a partir de um diretório chamado **src** (do Inglês *source*, fonte). Esta estrutura é a mesma utilizada na primeira atividade deste roteiro, na documentação UML.

Roteiro 01 – Atividade 04/04



8. Verifique onde o NetBeans criou o projeto que você acabou de implementar e analise a estrutura de diretórios criada. Veja se você realmente entendeu o que feito até agora.

9. Agora, observe o código gerado para o arquivo **Aula01.java**. Um arquivo "**.java**" deve ter o mesmo nome da classe pública que ele contém. Lembre-se que ao gerarmos o código na ferramenta CASE para a classe **Person**, foi criado o arquivo **br\univali\kob\poo1\aula01\Person.java**. As primeiras 5 linhas do arquivo são comentários sobre como utilizar modelos de código e podem ser excluídas. (é recomendável que você se aprofunde no uso da ferramenta depois)

```
package br.univali.kob.poo1.aula01;

/**
 *
 * @author Marcello Thiry
 */
public class Aula01 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    }

}
```

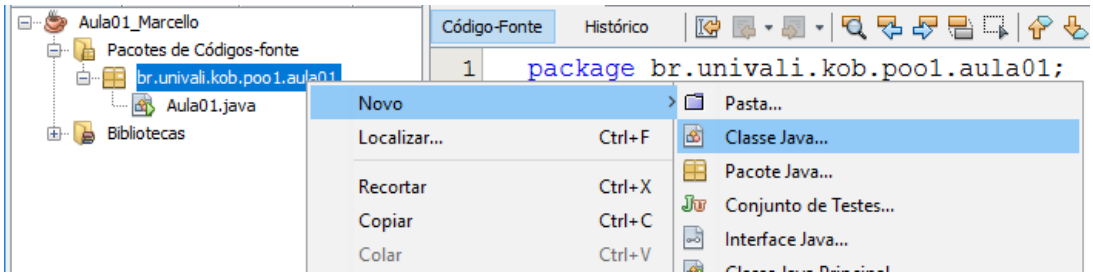
Temos a definição do pacote na primeira linha, conforme discutido anteriormente. Depois, o trecho comentado está no formato **javadoc** (para detalhes sobre este padrão de documentação leia os slides de referência da linguagem Java). Aí temos a classe **Aula01**. Veja que ela possui um método estático chamado **main** (lembre-se que um método estático não precisa de uma instância para ser invocado). **Sempre que uma aplicação Java for executada, ela iniciará na primeira linha deste método**. É comum chamarmos a classe que contém este método especial de classe principal do programa. Entretanto, como boa prática, esta classe sempre terá muito pouco código. Apenas o suficiente para iniciar o programa.

10. O método **main** ainda define um parâmetro chamado **args** que é um vetor de **String**. Este vetor é preenchido automaticamente com os argumentos passados para o programa quando ele é executado via console. Não utilizaremos este recurso durante a disciplina.

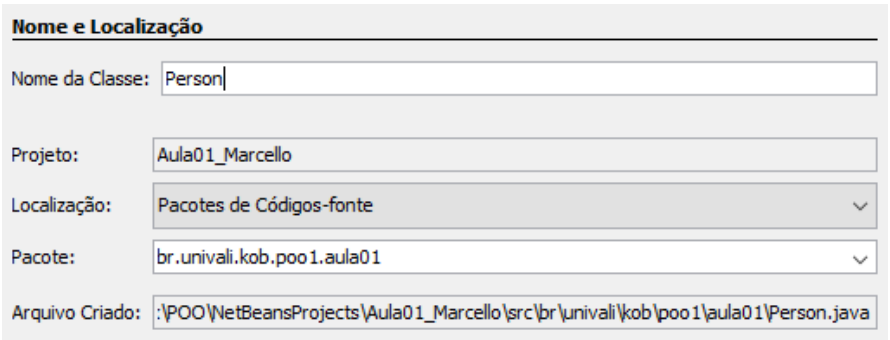
Roteiro 01 – Atividade 04/04

11. Agora leia o arquivo **br.univali.kob.poo1.aula01.Person.pdf** (disponibilizado com este roteiro) com bastante atenção, considerando os comentários e observando o código implementado. Compare com o código gerado pela ferramenta CASE. O nome do pacote no arquivo PDF tem uma diferença no último subpacote, mas se você seguiu as orientações até aqui, o pacote que você utilizará é **br.univali.kob.poo1.aula01**. Outra diferença está na cláusula **import**. A ferramenta não reconheceu **LocalDate** como uma classe existente no Java e ainda gera uma linha **import** fixa que deve ser sempre ajustada. Note também que faltam os métodos (implementação das operações), pois a ferramenta CASE gera somente a estrutura das classes e não o comportamento esperado.

12. Implemente a classe Person conforme o código apresentado (você pode manter apenas o **javadoc** relevante para a classe). O NetBeans possui várias formas para criar uma classe ou importar código fonte. É recomendável que você explore seus recursos durante a aula e também fora dela. Aprenda atalhos, dicas, etc. Leia tutoriais sobre a ferramenta. Isso tornará seu trabalho mais produtivo. Agora, iremos utilizar o botão direito sobre o pacote "br.univali.kob.poo1.aula01" e selecionar [Novo | Classe Java...].

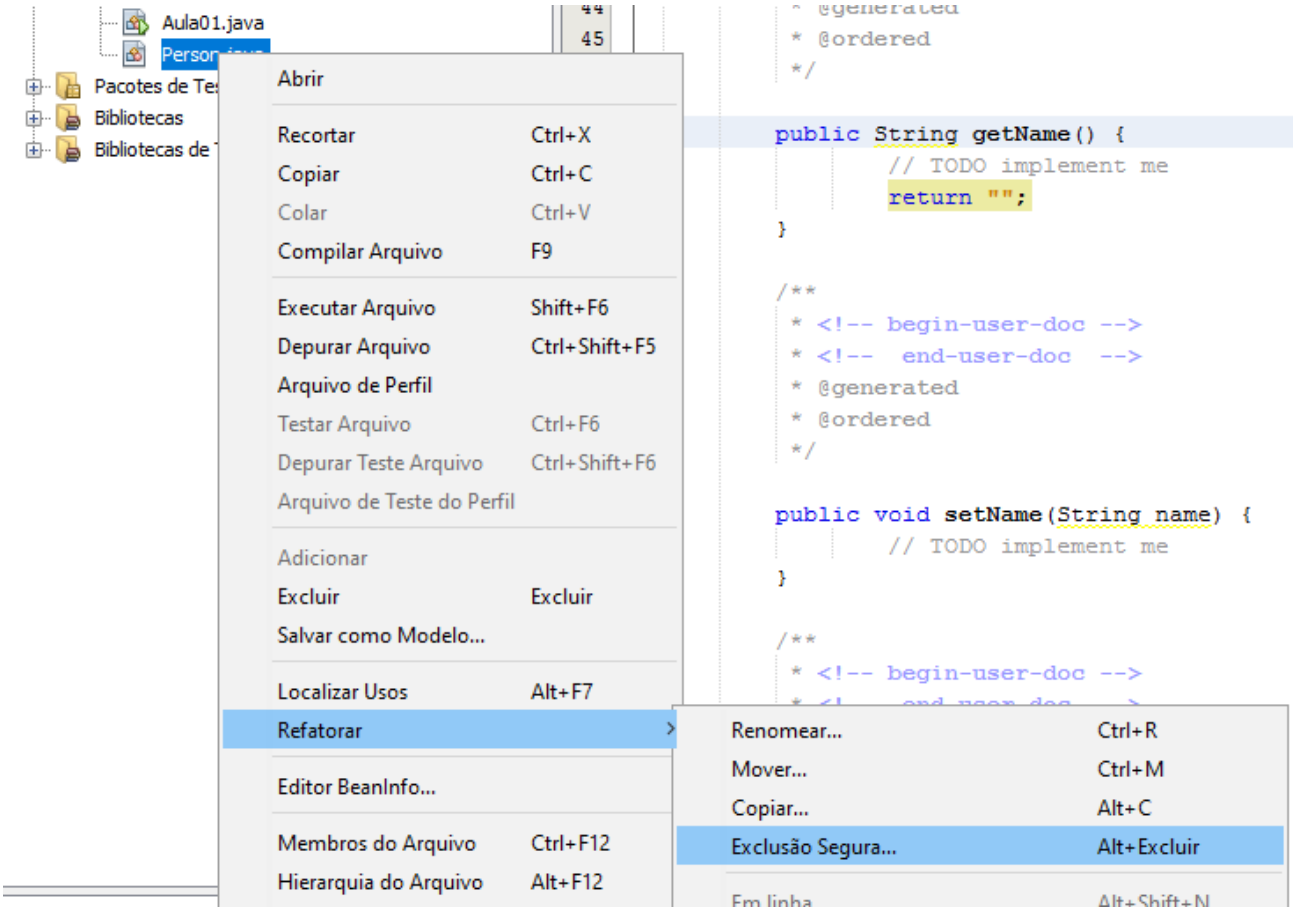


13. Como comentado anteriormente, a maior parte dos exemplos de códigos apresentados na disciplina estarão em Inglês (comentários estarão em Português para facilitar o entendimento). No diálogo apresentado pela ferramenta, você pode alterar também o pacote onde a classe será implementada. Para efeitos práticos agora, implementaremos a classe no mesmo pacote. Trocar o local (pacote) onde uma classe está é muito simples e podemos fazer isso depois.

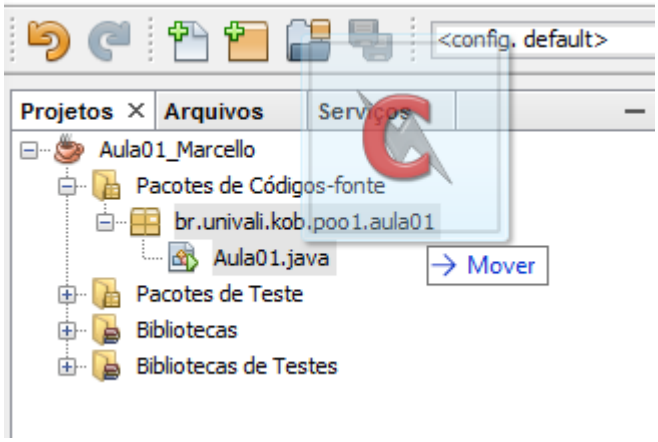


14. Pronto. Agora basta digitar seu código. Mas espere um pouco! E aquele arquivo que geramos a partir da ferramenta CASE? Será que não podemos utilizá-lo? Claro que podemos. Porém, antes de mais nada, teremos que excluir a classe que acabamos de criar (não fique impaciente, pois a ideia deste roteiro é fazer com que você conheça um pouco mais o ambiente que você usará durante todo o semestre). Para apagar uma classe do seu projeto, você pode clicar com o botão direito sobre a classe e selecionar a opção [Excluir]. Entretanto, esta opção pode não ser segura, uma vez que podem existir várias dependências ou comentários relacionados. Sempre que você quiser renomear, mover, copiar ou excluir um elemento, recomenda-se que seja utilizada a opção [Refatorar], também acessível pelo botão direito. Veja na imagem abaixo como selecionar a opção para exclusão segura. Quando o diálogo aparecer, confirme clicando no botão [Refatorar].

Roteiro 01 – Atividade 04/04

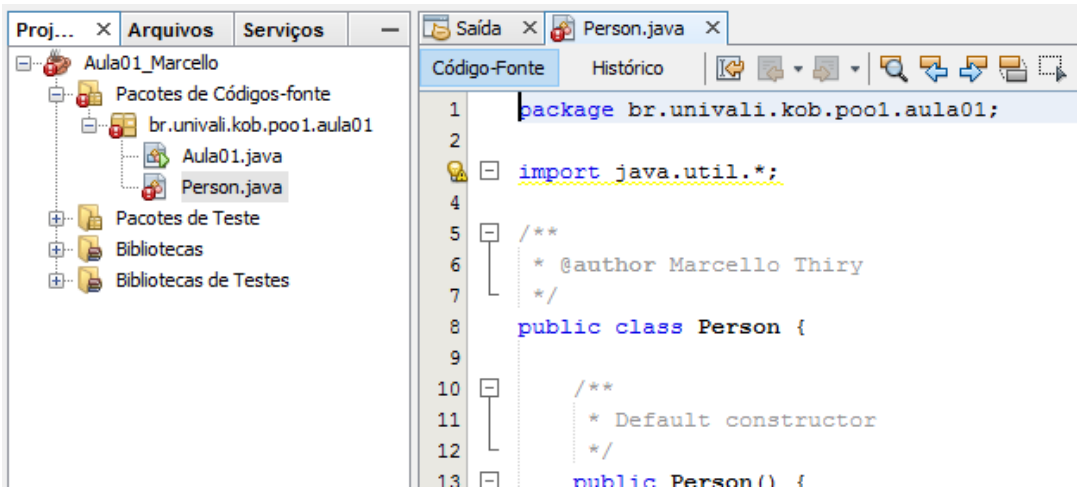


15. Você pode arrastar o arquivo **Person.java** para o pacote **br.univali.kob.poo1.aula01** no NetBeans (veja a figura abaixo). Se você copiar o arquivo (ou o diretório completo "**br...**" para o diretório **src**) fisicamente para o diretório correspondente do seu projeto, o efeito será o mesmo.

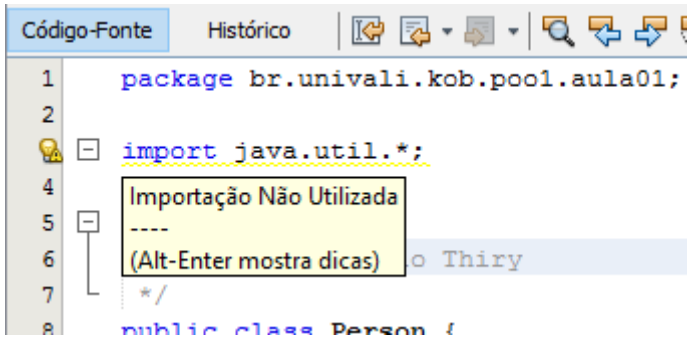


Roteiro 01 – Atividade 04/04

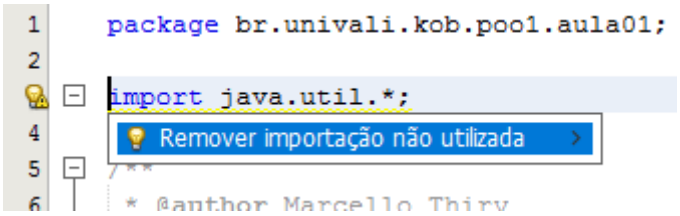
16. O arquivo deve fazer parte do seu projeto agora. Porém, há uma indicação de erro (exclamação dentro de um hexágono vermelho). Abra o arquivo **Person.java** no editor para investigarmos o que está acontecendo.



17. Antes de verificar o erro, note que há um alerta (*warning*) na linha 3 (`import java.util.*`). O alerta é indicado pelo marcador (lâmpada + símbolo de atenção). Além disso, a linha também está sublinhada (amarelo). Alertas são mensagens que avisam sobre alguma situação que pode levar a um problema futuro, mas que não impedem a compilação e a execução do programa. Como boa prática, devemos sempre analisar as mensagens de alerta. Quando a lâmpada aparecer sem o símbolo de atenção, o marcador é apenas uma dica/sugestão (você precisa avaliar caso a caso, pois nem sempre a sugestão é adequada). Passe o mouse sobre o marcador de alerta e veja que a mensagem está nos avisando que a importação não está sendo utilizada (não utilizamos nenhuma definição que está no pacote `java.util`). Já sabíamos disso, pois esta linha é sempre gerada pela ferramenta StarUML.

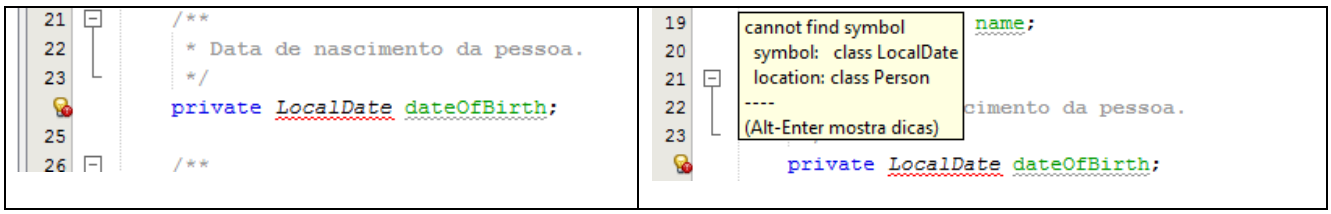


18. Agora, clique no marcador de alerta e observe que o IDE oferece uma sugestão de ação. Neste caso, iremos aceitar a sugestão e remover esta linha. Sempre observe se a sugestão é adequada para a situação. Não confie cegamente na ferramenta. Você é que deve saber o que deve ser feito.

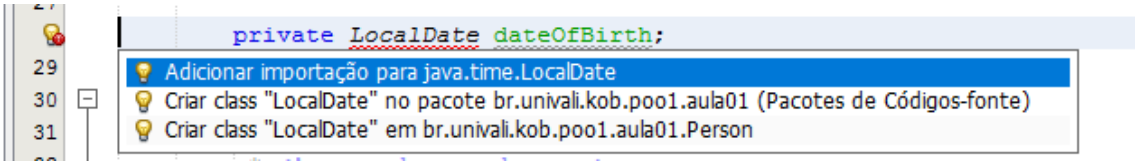


19. Agora sim, nós podemos procurar pelo erro. Veja que erro está na linha 24 (na declaração do tipo LocalDate). Também já havíamos falado sobre isso. O erro acontece porque esta classe é disponibilizada pelo Java no pacote "java.time". Podemos incluir a linha "`import java.time.LocalDate;`" manualmente no início do arquivo ou utilizar um recurso do NetBeans que fará isso automaticamente.

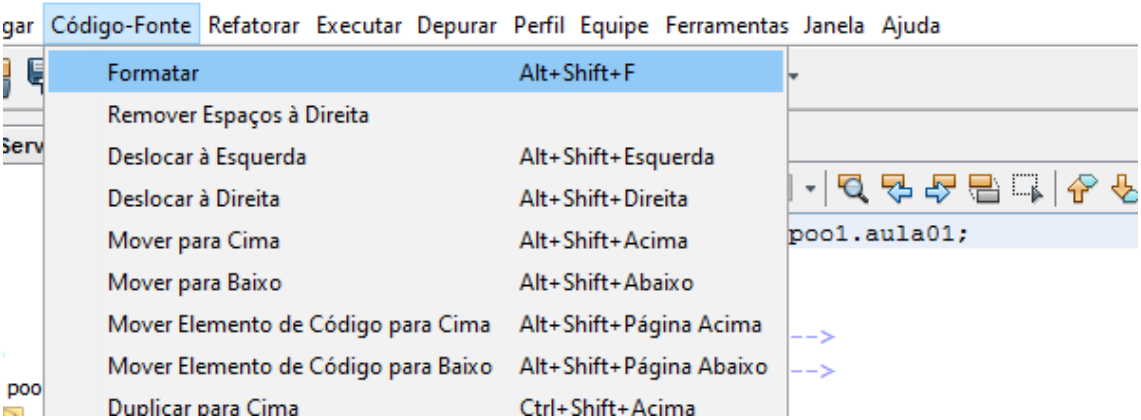
Roteiro 01 – Atividade 04/04



20. Ao clicarmos no marcador (lâmpada + símbolo de erro), o NetBeans mostra possíveis soluções para o problema. No nosso caso, devemos optar pela primeira (Adicionar importação para ...). Feito isso, não teremos mais erros. Entretanto, falta completar o código. Lembre-se que a ferramenta CASE gera apenas o esqueleto da classe. Precisamos escrever os métodos e ajustar quando a geração do código não for exatamente o que queremos.



21. Podemos também já deixar o código com a formatação do NetBeans. Para isso, vá até o menu principal e selecione a opção [Código-Fonte | Formatar]. Faça isso periodicamente quando estiver digitando o código para garantir que o estilo de programação está padronizado. Lembre-se que, eventualmente, você trabalhará com outros colegas e um estilo de código comum facilita a comunicação.

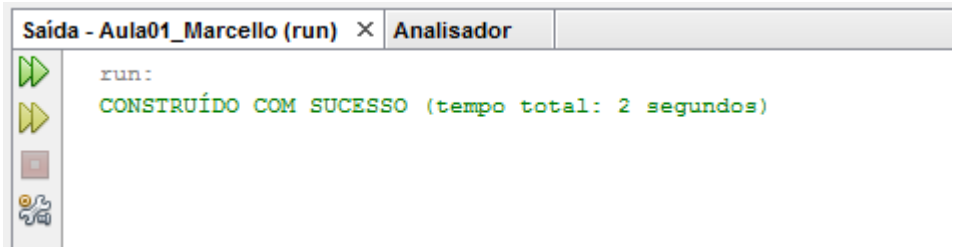


22. Volte a ler o arquivo **br.univali.kob.poo1.aula01.Person.pdf** e faça os ajustes necessários no seu código. Assim que você tiver concluído as alterações, podemos compilar e executar o programa. Devido à funcionalidade "Compilar ao Salvar" do IDE, não é necessário compilar manualmente o projeto para que ele seja executado no IDE. Quando um arquivo de código-fonte Java é salvo, ele é compilado automaticamente pelo IDE. A funcionalidade "Compilar ao Salvar" pode ser desativada na janela Propriedades do Projeto. Clique com o botão direito do mouse no projeto (Janela de Projetos) e selecione "Propriedades". Na janela "Propriedades", escolha a guia "Compilação". A caixa de seleção "Compilar ao Salvar" está na parte superior. Observe que, na janela Propriedades do Projeto, é possível configurar várias definições para o projeto: bibliotecas do projeto, encapsulamento, construção, execução, etc. Se você for até a opção "Cabeçalho da Licença" e selecionar no combo "Usar licença global" o valor "NetBeans CDDL/GPL", a ferramenta não gerará mais aquelas linhas iniciais a cada arquivo gerado. Para executar o programa:

    - Escolha Executar > Executar Projeto (F6). Você pode usar também os ícones. Passe em cada um e veja o hint que a ferramenta apresenta. Isso irá ajudá-lo a conhecer melhor a ferramenta e ser mais produtivo.

Roteiro 01 – Atividade 04/04

A figura abaixo mostra o que você deve ver agora:



Você deve ter percebido que seu programa não fez nada. Isso, porque nós realmente não implementamos o programa que utiliza a classe desenvolvida. Para resolver o problema, veja o arquivo **PersonTest.java** disponibilizado com o material deste roteiro. Este arquivo contém uma classe que irá testar a nossa classe Person. Iremos importar este arquivo agora para o nosso projeto. Copie (ou arraste como explicado anteriormente) o arquivo para o diretório **...\NetBeansProjects\Aula01_Marcello\src\br\univali\kob\poo1\aula01**, considerando que este é o pacote desejado para a classe.

23. Retorne para a IDE e perceba que a classe **PersonTest** está no pacote. Porém, poderá aparecer um marcador de erro, caso o nome do pacote na implementação da classe **PersonTest** esteja diferente do nome do pacote para o qual você copiou o arquivo. Abra a classe **PersonTest** e faça a alteração necessária. Pronto, agora podemos executar novamente o programa? Ainda não. Veja que o método **main** da nossa classe principal **Aula01** continua vazio. Logo, o programa não tem nada para fazer. Vamos resolver este problema. Veja a imagem abaixo:

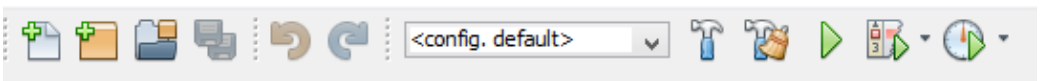
```
package br.univali.kob.poo1.aula01;

/**
 *
 * @author Marcello Thiry
 */
public class Aula01 {

    public static void main(String[] args) {
        PersonTest test = new PersonTest();
        test.run();
    }

}
```

Agora sim, você pode executar o projeto. Se houver erros de compilação, eles serão marcados na margem do editor. É possível passar o mouse sobre o marcador para visualizar sua descrição. O IDE oferece ainda uma barra de atalhos com os comandos mais utilizados (veja a figura abaixo). Passe o mouse em cada ícone e preste atenção na dica (*hint*) apresentada. Você encontrou o ícone para executar o projeto?



24. Estude o código da classe **PersonTest** e os resultados na caixa "Saída". Você deve entender o que está acontecendo.

25. Depois de escrever e executar seu primeiro programa em Java, você pode utilizar o comando "Limpar e Construir" para construir a aplicação para implantação. Quando o comando "Limpar e Construir" é utilizado, o IDE executa um script de construção que realiza as seguintes tarefas:
    a) Apaga todos os arquivos compilados anteriormente e outras saídas de construção.

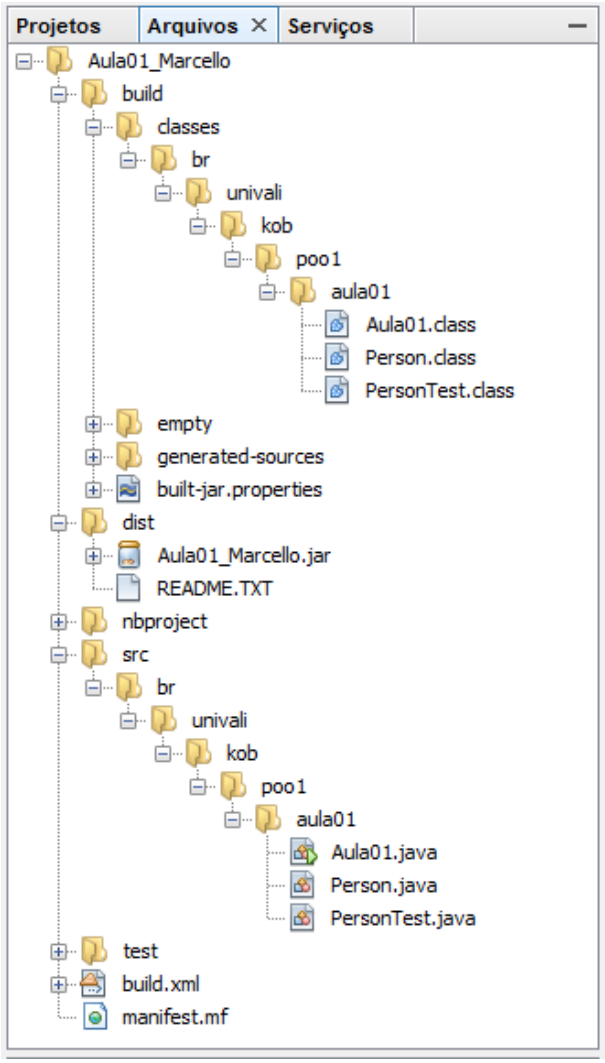
Roteiro 01 – Atividade 04/04

b) Recompila a aplicação e constrói um arquivo JAR que contém arquivos compilados. Um arquivo JAR é um arquivo compactado e tem a mesma estrutura de um arquivo ZIP. O objetivo é colocar vários arquivos compilados num mesmo arquivo para facilitar a distribuição e também para aumentar a eficiência na carga do programa.

Para construir sua aplicação:

- Escolha Executar > Limpar e Construir Projeto Principal (Shift-F11)

É possível exibir as saídas de construção abrindo a aba "Arquivos" (margem esquerda da IDE) e expandindo a pasta "Aula01_Marcello". O arquivos bytecode (código de máquina para a JVM – máquina virtual do Java) "*.class" estão na subpasta "build\classes\br\univali\kob\poo1\aula01". O arquivo JAR que contém o projeto "Aula01_Marcello" está na pasta "dist" (distribuição): "Aula01_Marcello.jar".



26. Abra o console do Windows (linha de comando) e posicione-se na subpasta "dist" (ver acima). Quando você estiver no mesmo diretório do arquivo JAR criado, digite o seguinte comando:

- java -jar Aula01_Marcello.jar

O programa deve mostrar no console a mesma saída visualizada anteriormente na IDE. Se você utilizou acentos, antes de executar o comando anterior, execute o comando chcp 1252 no console. Ela muda a página de código (conjunto codificado de caracteres) para o alfabeto latino.