Chain of Responsibility

padrão de projeto

Carla gabrIella albuquerque

Francielly Tineli

idarlete furtado de miranda

Colatina

2016

**SUMÁRIO**

[1. INTRODUÇÃO 3](#_Toc451360188)

[2. Descrição do padrão factory method 4](#_Toc451360189)

[3. diagrama de classes original do padrão. 5](#_Toc451360190)

[4. dIagrama de CLASSES DO EXEMPLO APRESENTADO 6](#_Toc451360191)

[5. DescrIção em que o padrão auxilia no desenvolvimento do projeto e em que auxiliou no exemplo apresentado 7](#_Toc451360192)

[6. Código do aplicativo comentado. Explicado em DETALHES 8](#_Toc451360193)

[7. Caminho para vídeo no YOUTUBE explicando o padrão e o exemplo desenvolvido com o padrão de projeto. 9](#_Toc451360194)

1. **INTRODUÇÃO**

"Em engenharia de software, padrões de projeto de comportamento são padrões de design que identificam padrões comuns de comunicação entre os objetos e percebe esses padrões. Ao fazer isso, esses padrões aumentam a flexibilidade na execução.

Os padrões comportamentais são os padrões que incidem sobre as interações entre objetos. As interações entre os objetos de cooperação devem ser de tal forma que eles estejam se comunicando, mantendo o acoplamento fraco, que é a chave para arquiteturas de n-tier.

1. **Descrição do padrão Chain of Responsibility**

O Chain of Responsibility permite que mais de um objeto lide com um pedido, sem o conhecimento mútuo. Evitamos o acoplamento entre o remetente do pedido e os receptores possíveis, ao dar a mais de um objeto a oportunidade de tratar a solicitação.

Assim, é possível variar praticamente todos os componentes sem grandes danos ao projeto. Cada elemento implementa sua própria maneira de responder a requisição, e estas podem ser alteradas facilmente.

É preciso tomar cuidado para garantir que as chamadas sejam realmente respondidas. Para isso é necessário fazer uma investigação, a partir da mensagem enviada para o objeto, e assim determinar se esta mensagem pertence ao protocolo da classe verificada ou a alguma das superclasses da hierarquia.

Para que se possa otimizar e reduzir o espaço de procura pelo padrão, é importante definir algumas regras:

• limitar à procura a implementações de métodos com o mesmo nome em todos os casos. Chain of Responsibility não requer que o mesmo nome de método seja invocado em todos os elementos da cadeia. Com esta restrição, o problema remove dependências semânticas;

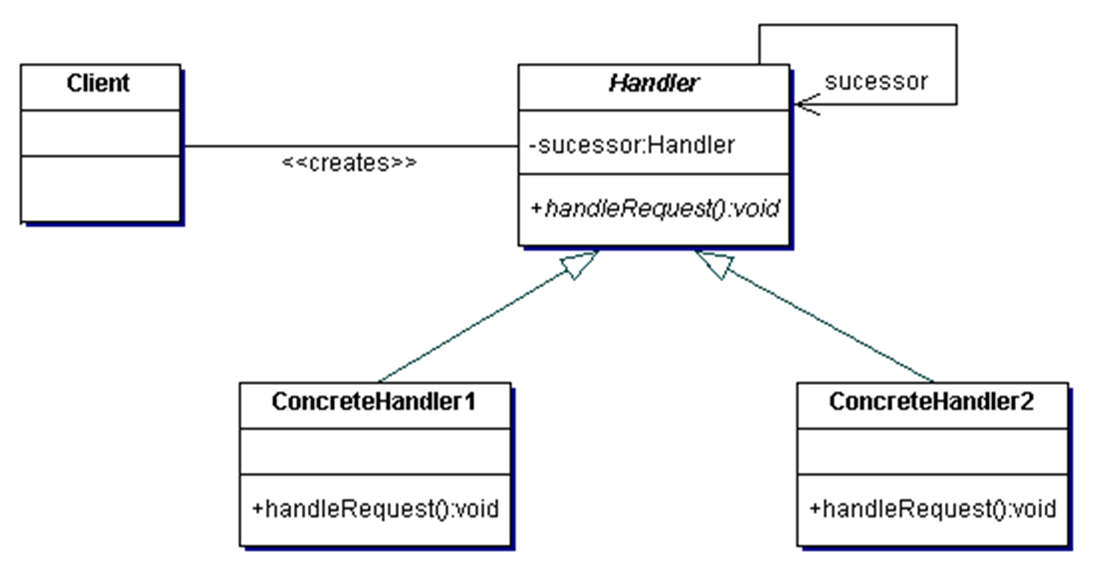
• restringir a procura a mensagens enviadas a receptores conhecidos. Uma vez que foram identificados e conhecidos atributos para armazenar valores de um conjunto particular de classes, é importante concentrar esforço nestas variáveis que podem participar de uma cadeia;

• ignorar mensagens enviadas quando os objetos são passados como argumentos ou como variáveis temporárias.

A formalização para a identificação do padrão Chain of Responsibility é elaborada a partir do objeto concreto receptor da mensagem de execução. A mensagem é recebida e, caso, o objeto não a reenvie, está será executada. Caso o objeto faça o reenvio da mensagem, está será recebida por outro objeto da cadeia que poderá propor a sua execução ou o seu reenvio a outro.

1. **diagrama de classes original do padrão.**

O padrão Chain of Responsibility, da forma como foi descrito no livro Design Patterns: Elements of Reusable Object-Oriented Software, contém os seguintes elementos:

****

**Handler:** Define uma interface para tratar as solicitações

**ConcreteHandler:** Trata a solicitação pela qual é responsável

- Repassa a solicitação se não couber a ele o tratamento

**Client:** Inicia a solicitação para um objeto ConcreteHandler da cadeia

Cada objeto no encadeamento (ConcreteHandlerA, ConcreteHandlerB) age como um tratador de pedidos e conhece seu sucessor.

Se um dado objeto é capaz de tratar o pedido ele o faz, caso contrário o pedido é passado ao próximo tratador no encadeamento de objetos.

1. **dIagrama de CLASSES DO EXEMPLO APRESENTADO**

## C:\Users\Bruxo\Documents\5=Periodo\Projeto de Software= IGOR\Trabalho II - Padroes de Projeto\Composite\composIte.png

Figura 1: Diagrama de Classe \_ Acadêmico

1. **DescrIção em que o padrão auxilia no desenvolvimento do projeto e em que auxiliou no exemplo apresentado**

O Chain of Responsibility é muito utilizando quando mais de um objeto é capaz de tratar um pedido, e este objeto não é conhecido antecipadamente.

O objeto tratador do pedido deve ser reconhecido automaticamente.

Um pedido deve ser tratado por um entre vários objetos sem que o objeto tratador seja especificado explicitamente.

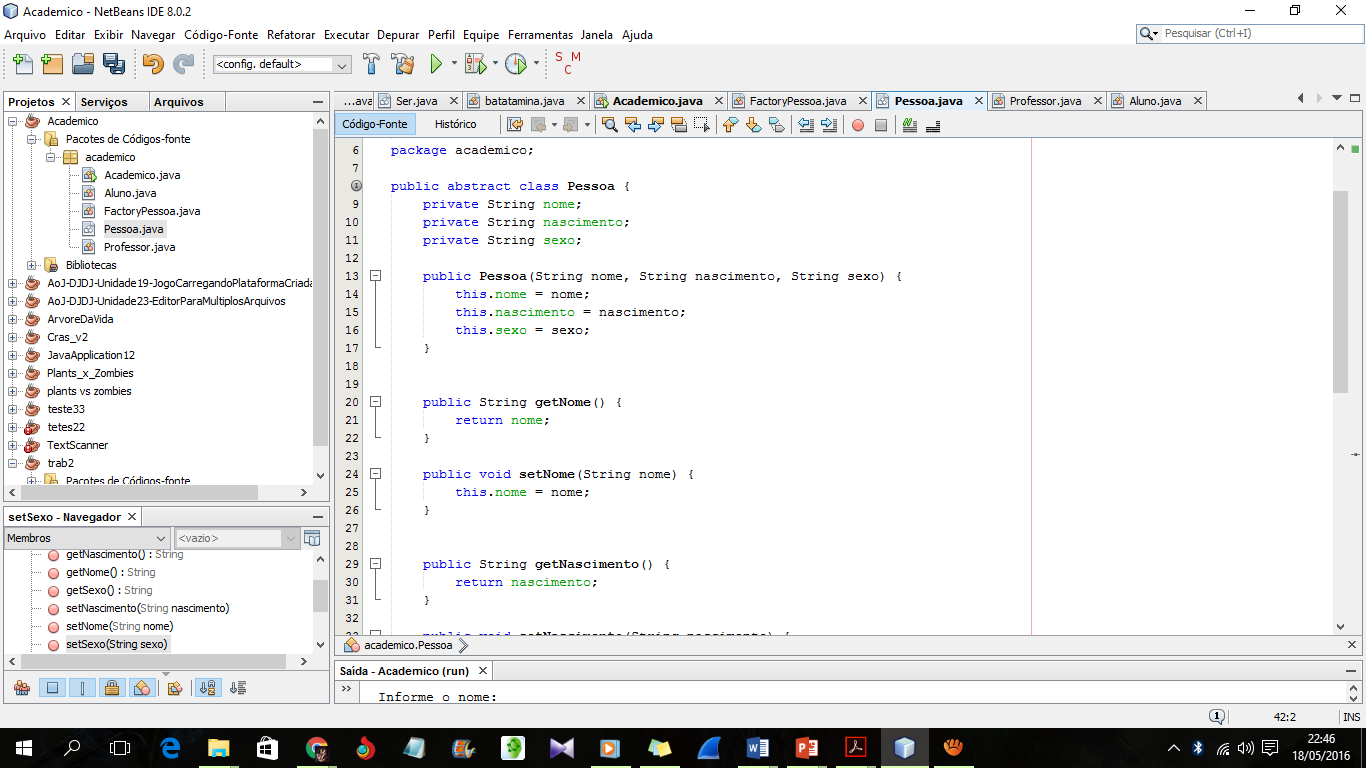
**Benefícios**

* Aumenta a flexibilidade de lidar com um pedido.
* Reduz o acoplamento = Evita acoplamento do transmissor de uma requisição com seus receptores, fazendo com que mais de um projeto tenha a chance de manipular a requisição.
* Encadeia os objetos receptores e passa a requisição ao longo dessa cadeia até que um objeto possa manipulá-lo.
* Reduz a vinculação.
* Permite que um conjunto de classes atue como uma classe; eventos produzidos em uma classe podem ser enviados para outras classes dentro da composição.

**Desvantagens / consequências**

A recepção não é garantida pois o pedido não tem receptor explícito, e não há garantia de que vai ser tratado (o pedido pode sair pelo final da cadeia sem ter sido tratada) a menos que a cadeia esteja configurada corretamente.

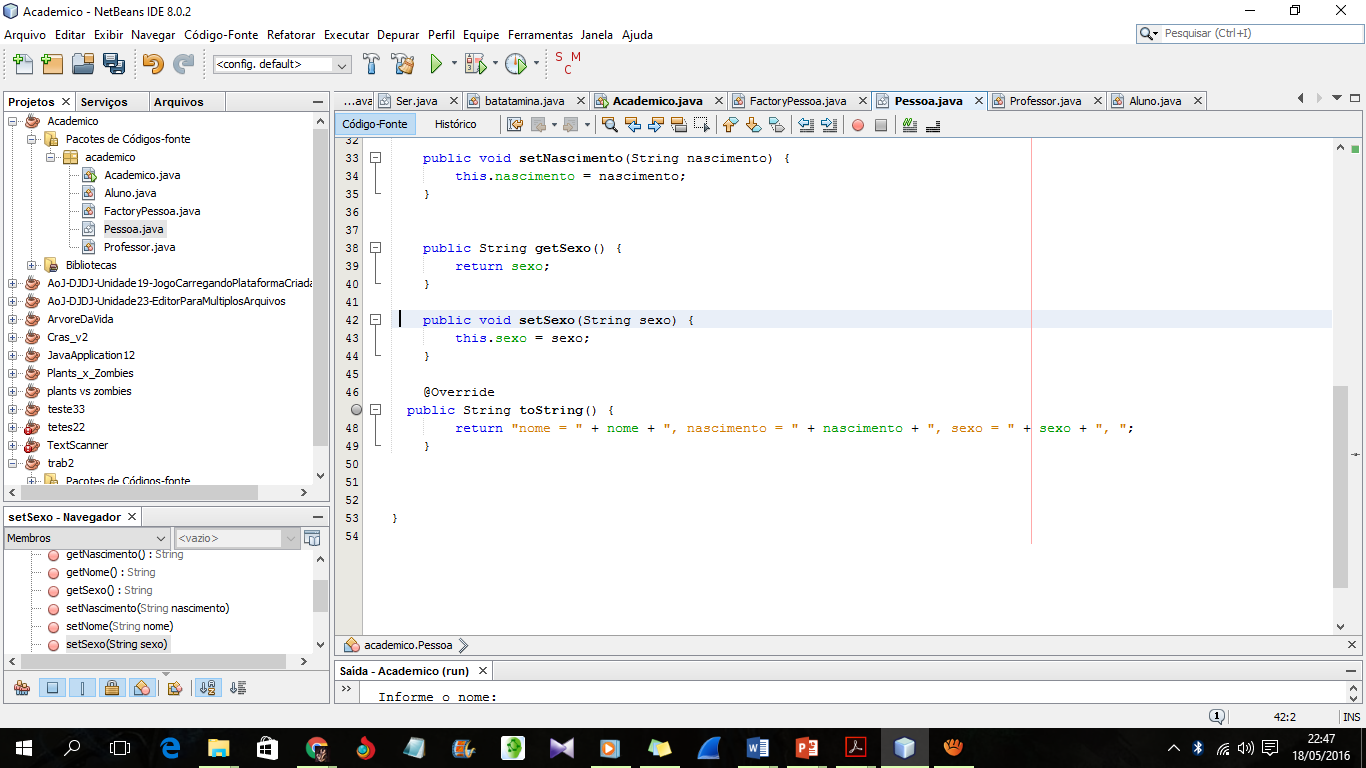
1. **Código do aplicativo comentado. Explicado em DETALHES**



Construtor da superclasse pessoa

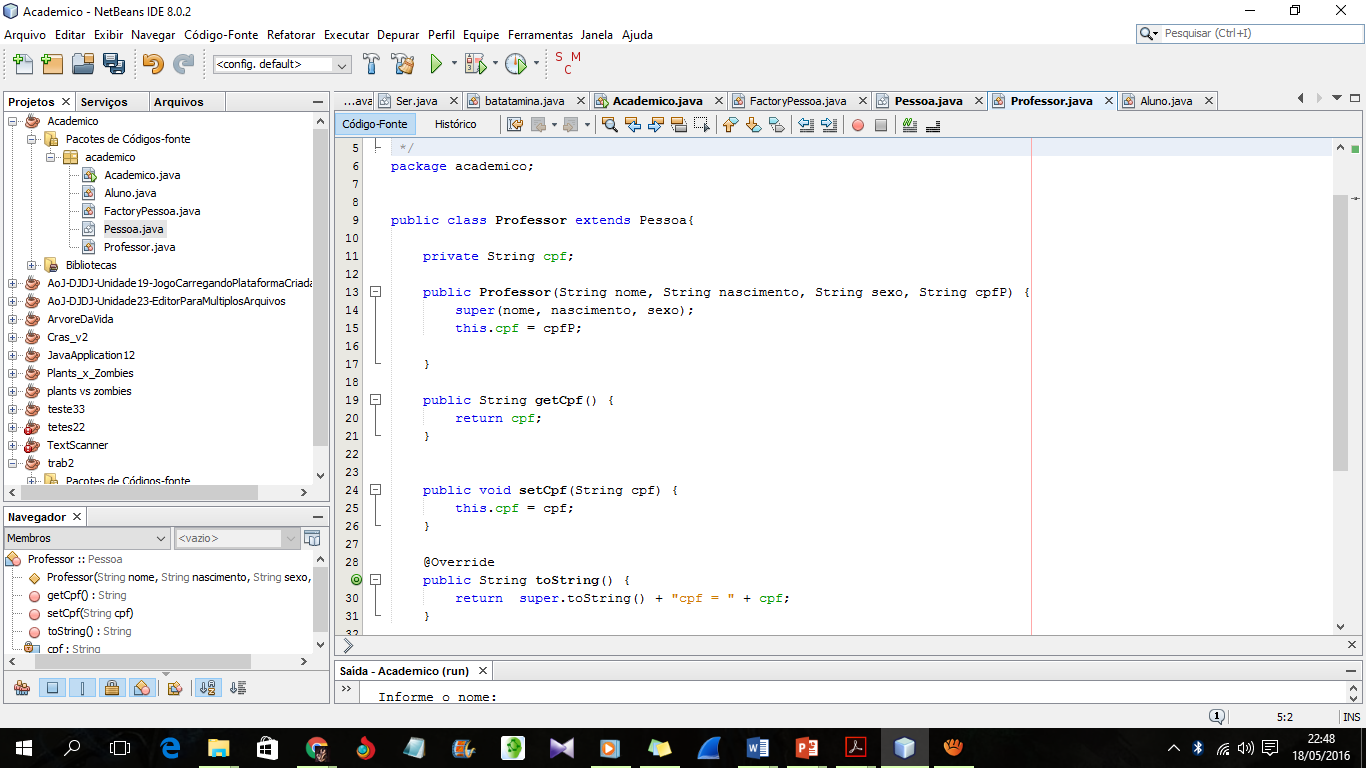
Classe Abstrata Pessoa,

Atributos, nome, sexo, d-n



Sobrescreve o construtor para receber as variáveis passadas na classe academinco

Figura 2: classe Abstrata (pessoa) com métodos

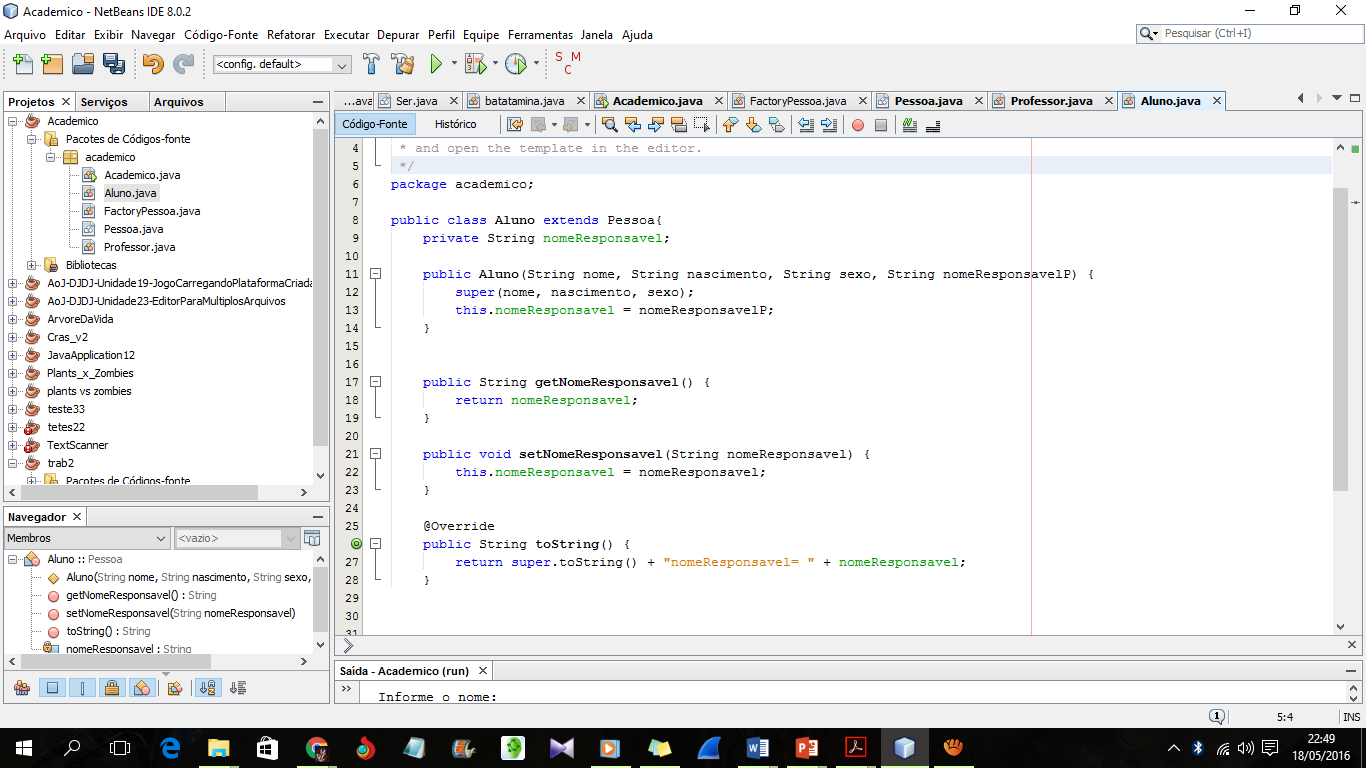


Sobrescreve o construtor para receber as variáveis passadas na classe acadêmico

Atributo da Classe Professor

Herda Atributos da Classe ancestral pessoa

Figura 3: classe Concreta (professor) herda da superclasse pessoa

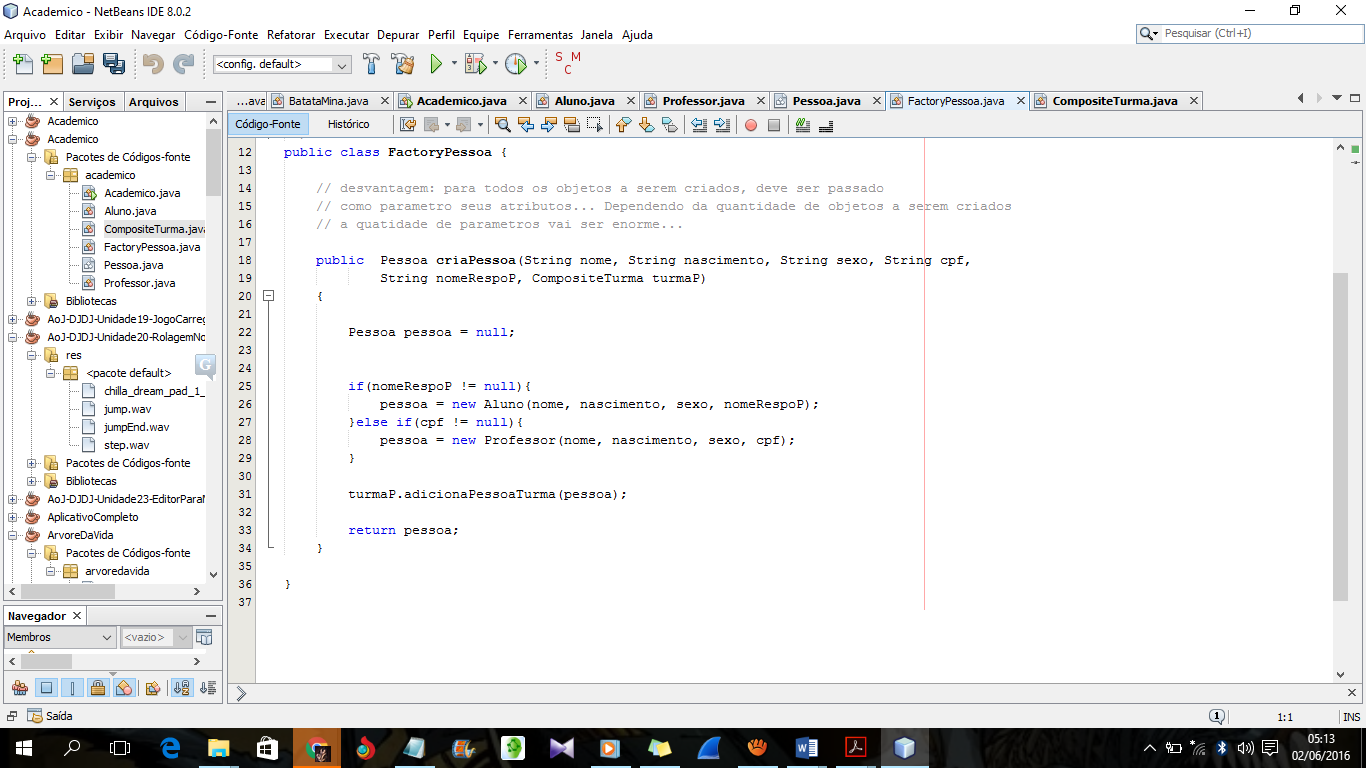


Atributos da Classe Aluno

Herda Atributos da Classe ancestral pessoa

Herda Atributos da Classe ancestral pessoa

Figura 4: classe Concreta (aluno) herda da superclasse pessoa

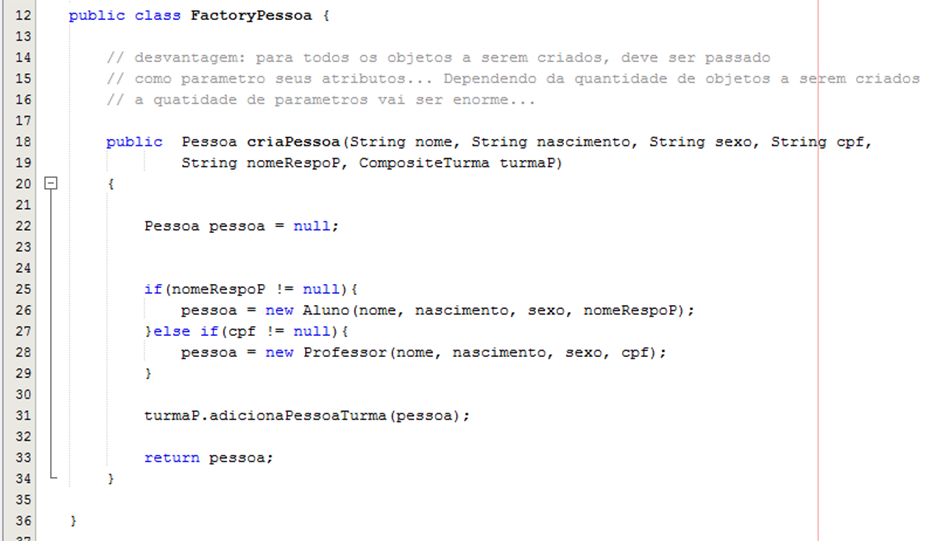


Atributo da classe Aluno

Método criaPessoa recebe todos os atributos das classes professor e aluno

Atributo da classe professor

Figura 5: FactoryPessoa (Fábrica de métodos)

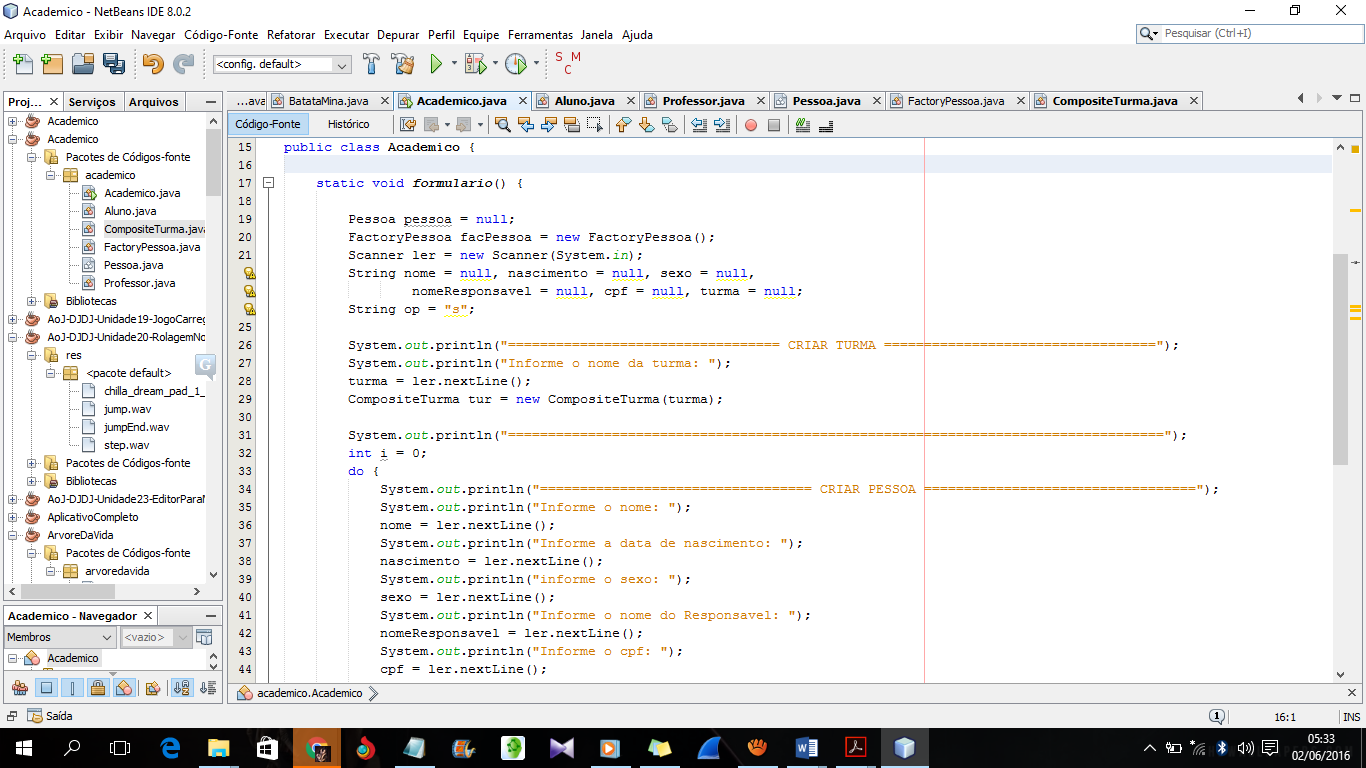


Compara os atributos passados como parâmetros, para saber qual está vazio, e assim criar apenas a pessoa que tiver todos os atributos preenchidos

Retorna o objeto pessoa, cujo os atributos foram passados. Se for passado os atributos de professor vai retornar uma pessoa do tipo professor, se os atributos forem de aluno vai retornar uma pessoa do tipo aluno

Figura 6: FactoryPessoa (Fábrica de métodos)

obs.: não serão passados todos os atributos para a criação de uma nova pessoa, quando quisermos criar um objeto do tipo professor, o atributo **nomeRespoP** que pertença apenas ao objeto aluno ficará inativo, (não será passado)



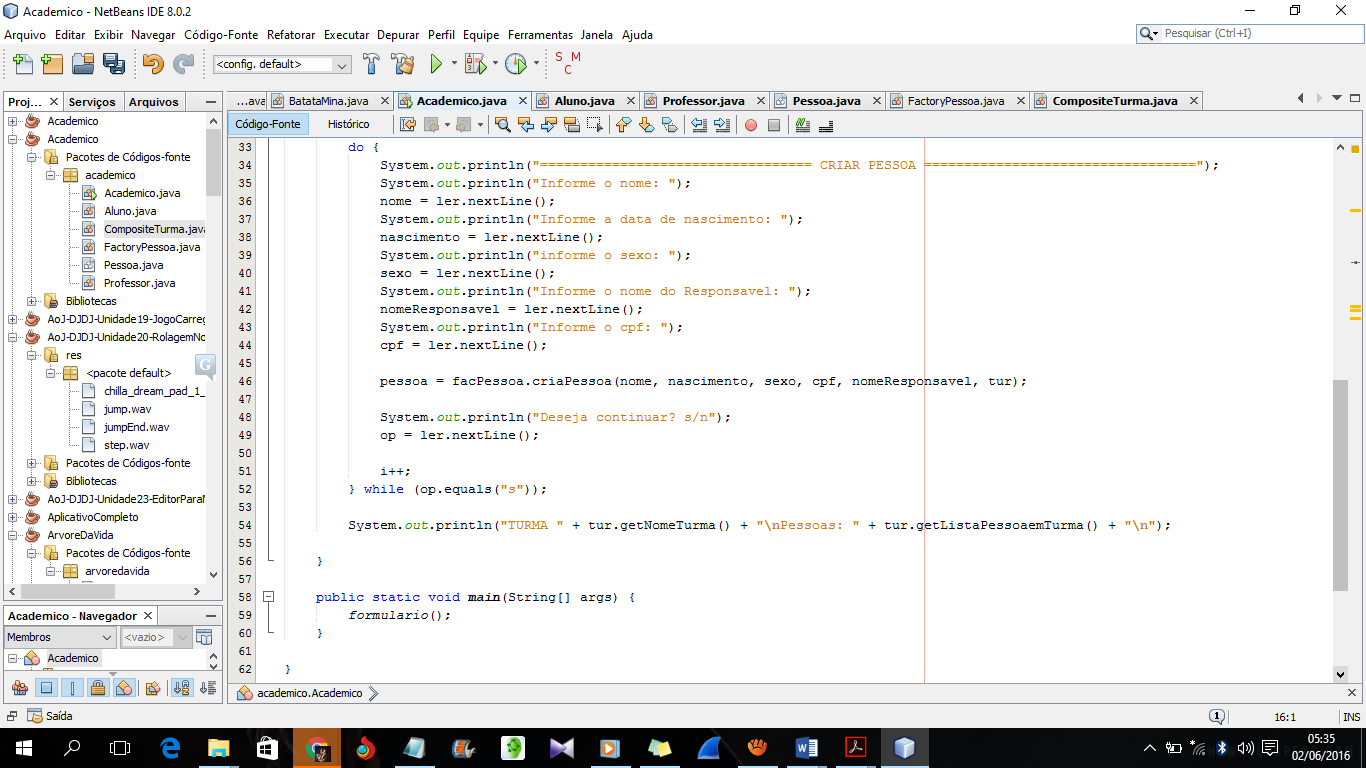
Informações passadas para as variáveis

Variáveis iniciadas como nulas

Cria um objeto factoryPessoa do tipo factoryPessoa

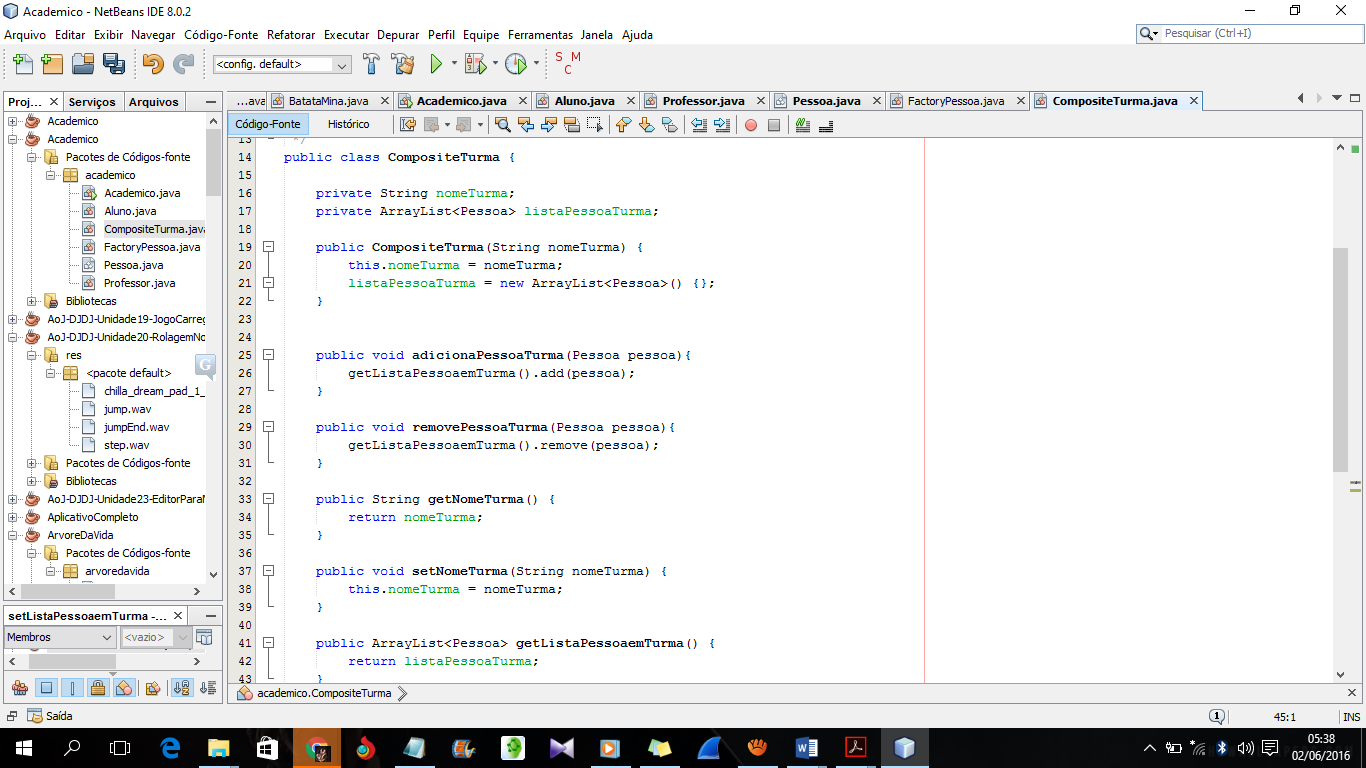
Variável do tipo pessoa iniciada como nula

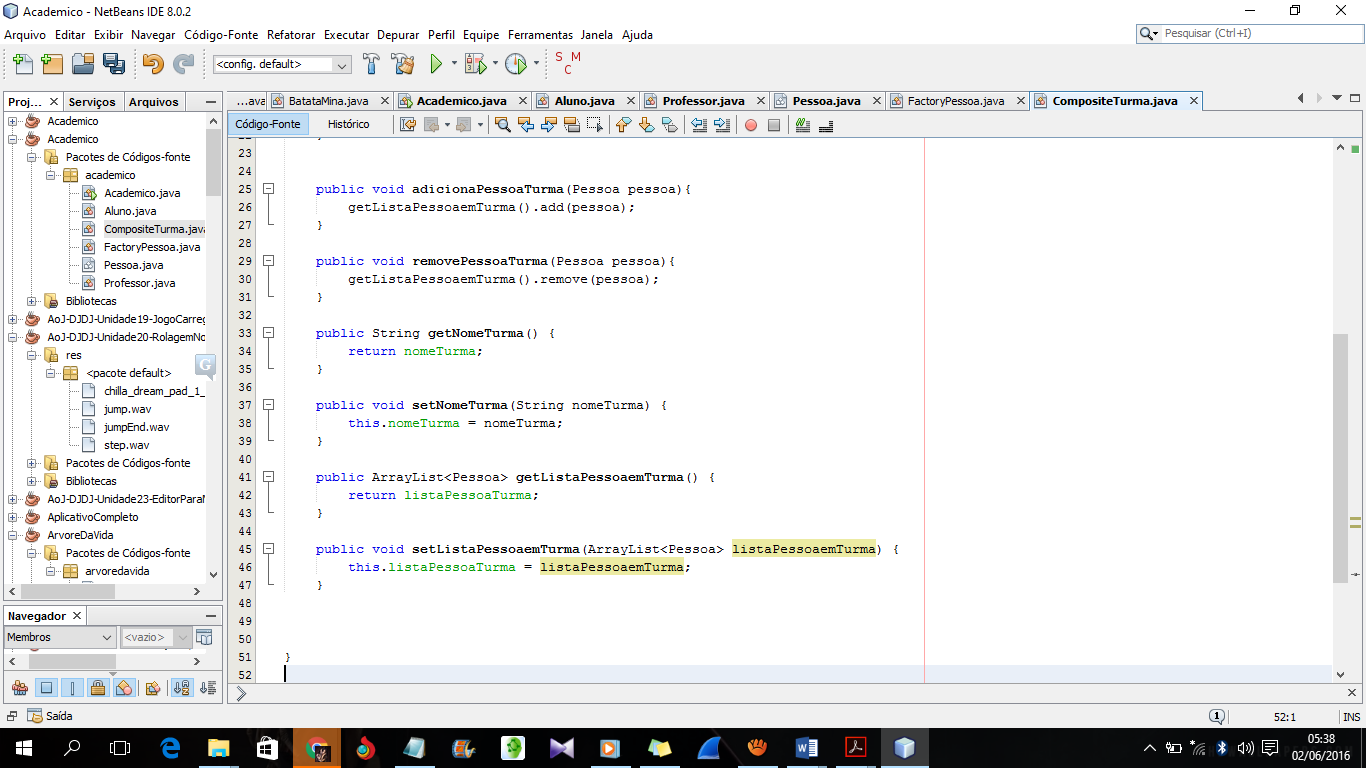
Classe principal onde está o main

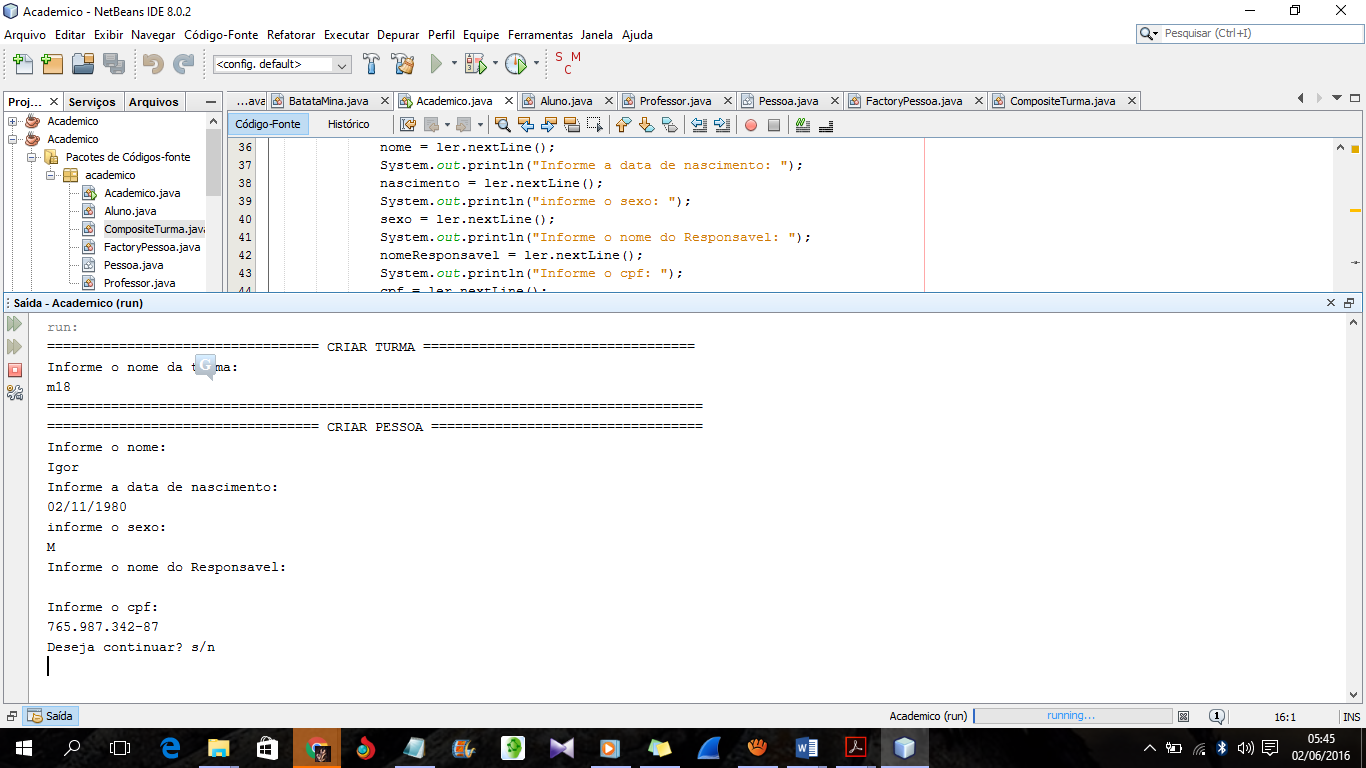


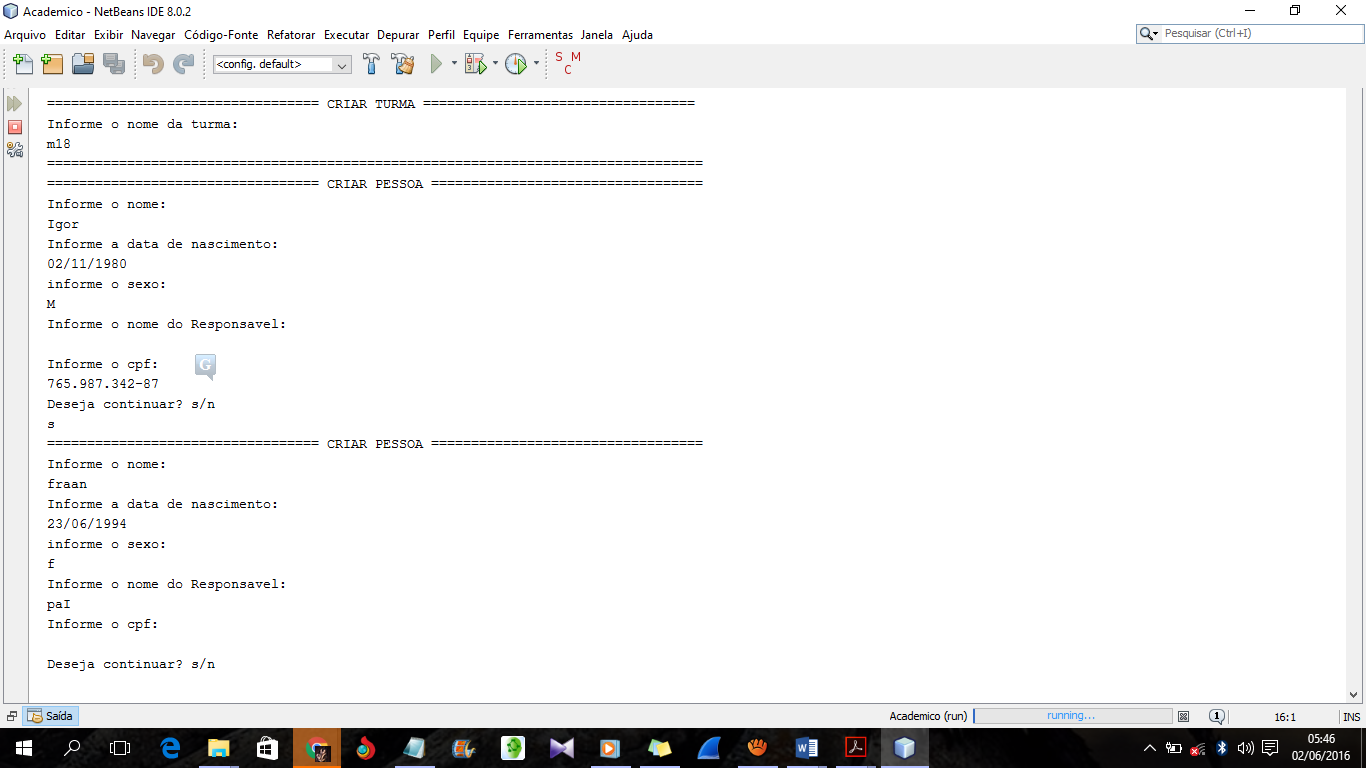
Chamando a função menu, que através de outras funções Ira criar, uma nova pessoa

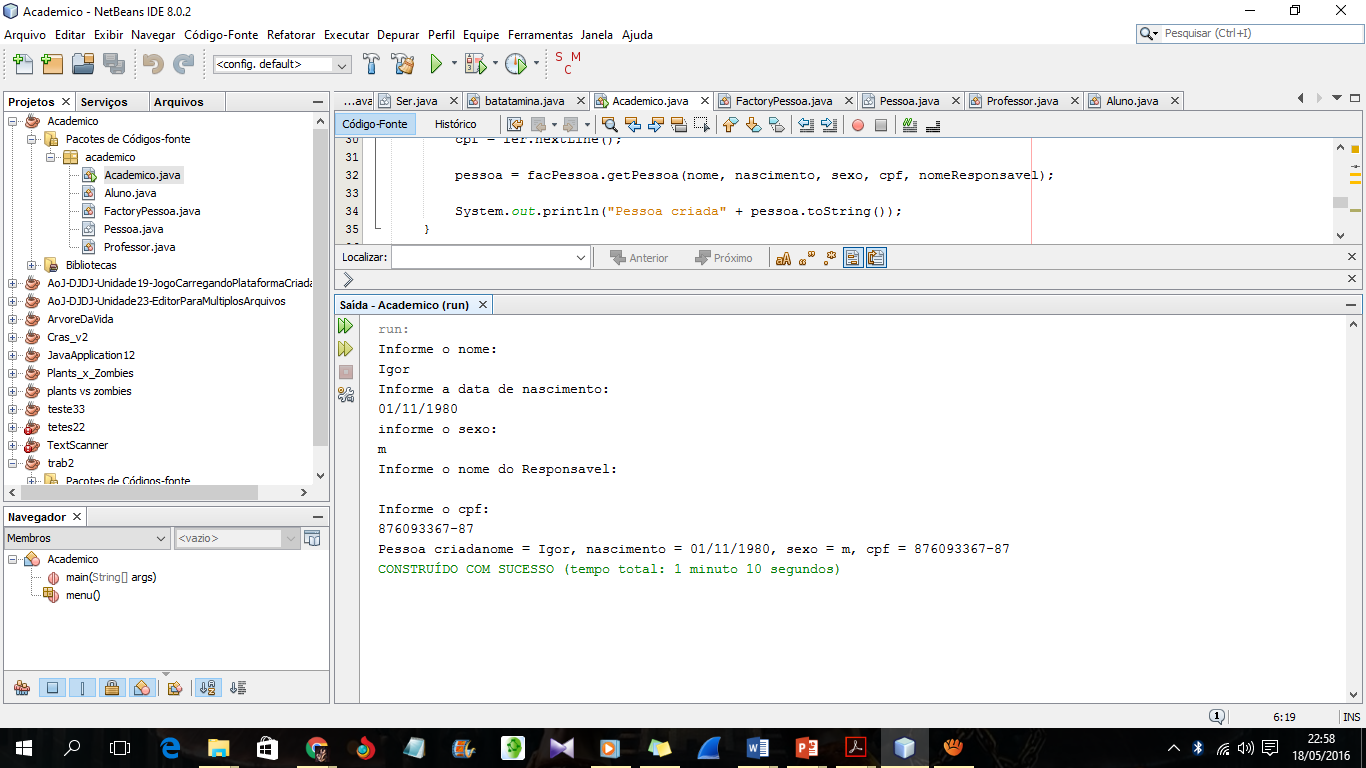
Figura 7: Acadêmico







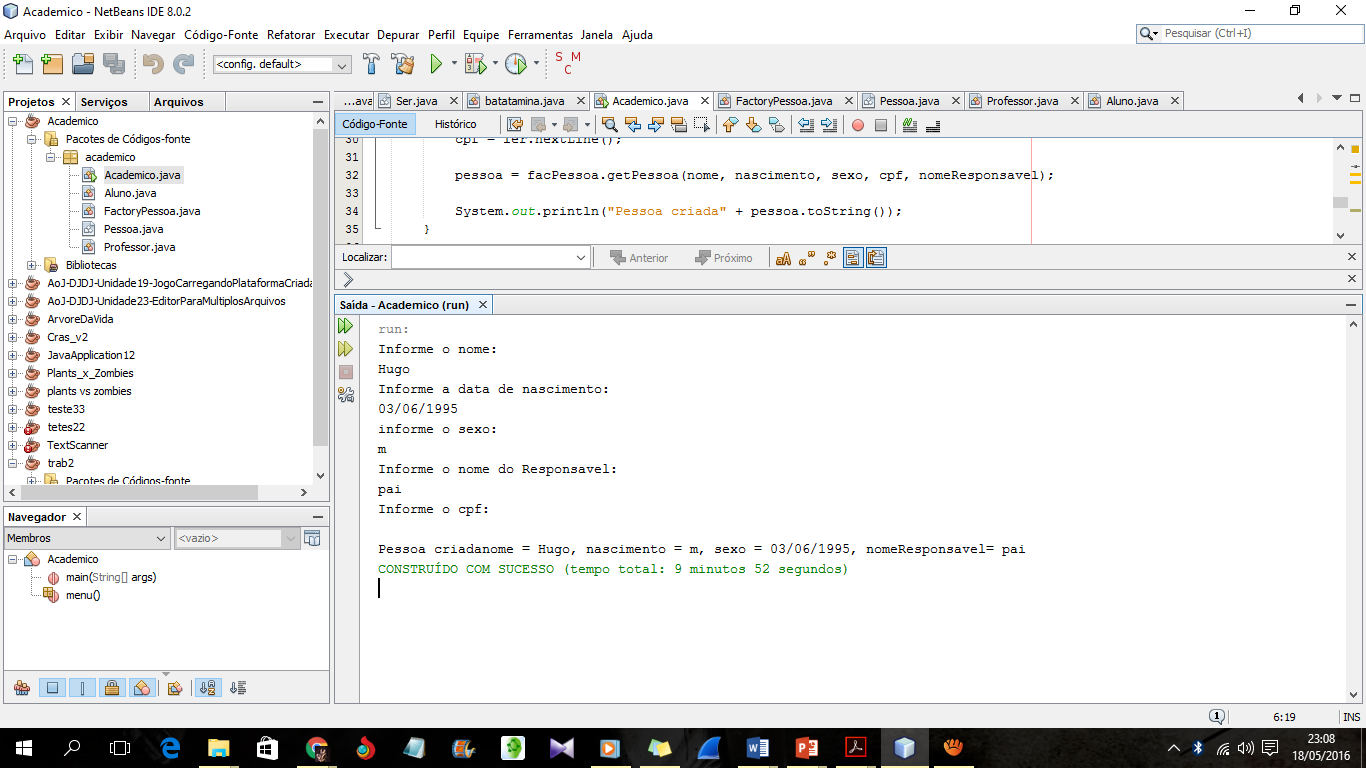




Pessoa do tipo professor criada

O campo nome do responsável não é passado, pois não existe

Figura 7: Saída do código tipo professor



Pessoa do tipo Aluno criada

O campo CPF não é passado, pois não existe

Figura 8: Saída do código tipo Aluno

1. **Caminho para vídeo no YOUTUBE explicando o padrão e o exemplo desenvolvido com o padrão de projeto.**

Vídeo passado diretamente para o professor.