Unidade Acadêmica: Faculdade de Computação – FACOM
Disciplina: Programação Orientada a Objetos I
Professor: Fabiano Azevedo Dorça

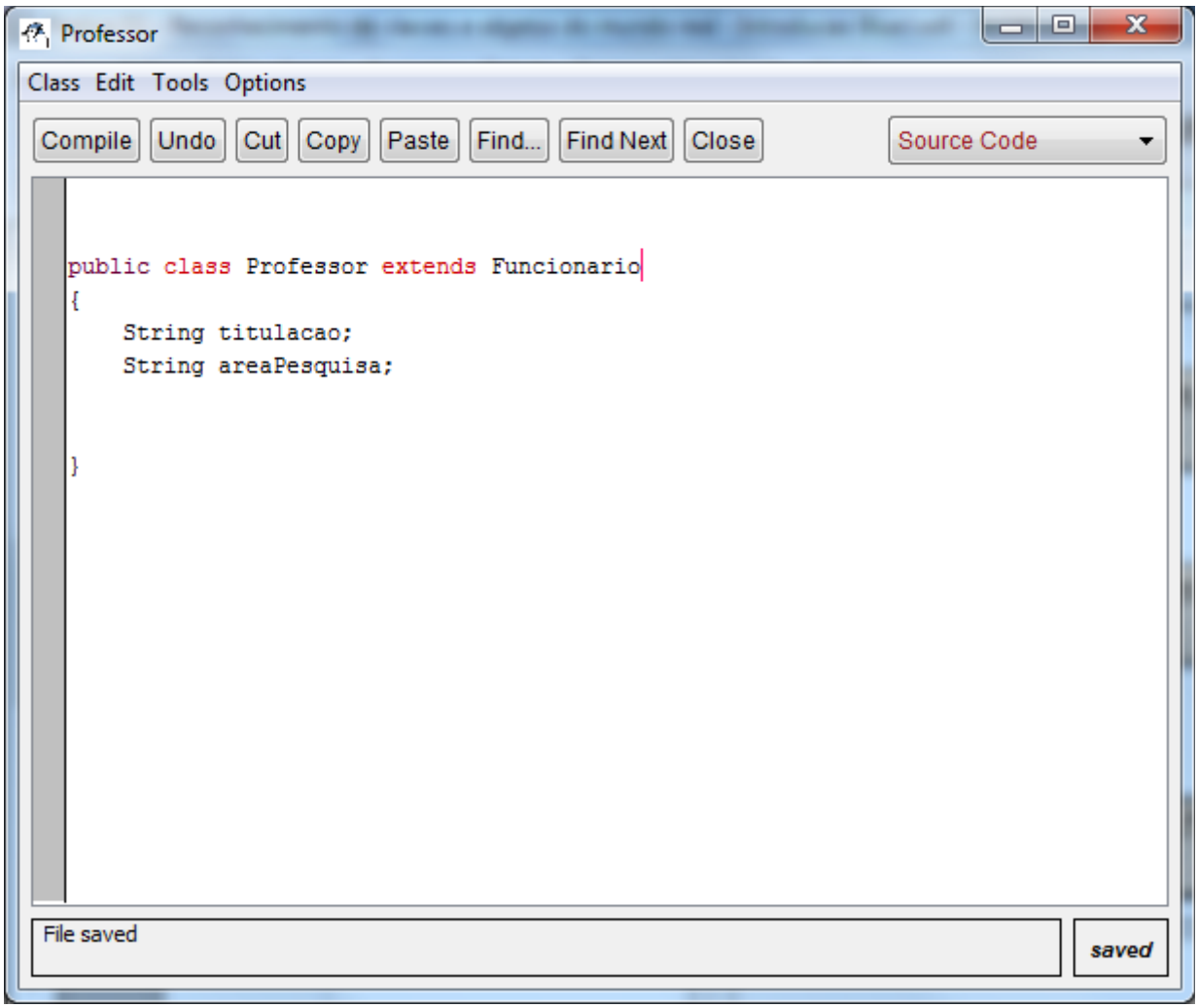
Prática 02 – Herança

Agora, vamos utilizar herança para evitar replicação de código, criando as abstrações (classes mais gerais), e a partir delas criar as classes mais específicas.

Uma possibilidade interessante é criar um relacionamento de herança entre as classes “Professor” e “Funcionario”. No caso, a descrição do problema deixa claro que um professor é um funcionário, ou seja, deve possuir todos os atributos de um funcionário, além de seus próprios atributos.

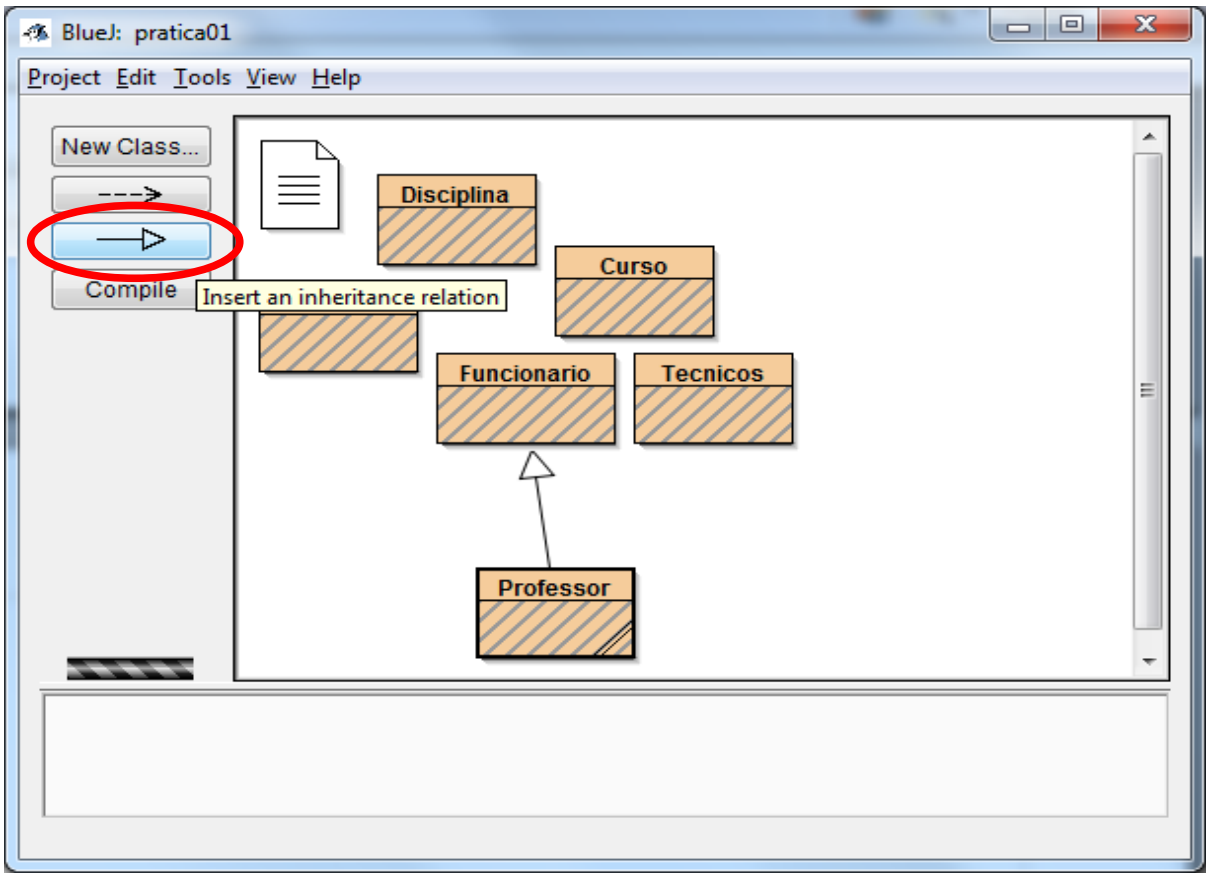
Na programação orientada a objetos, este é um relacionamento de HERANÇA, que possibilita que uma classe herde toda a implementação de uma outra classe. Em nosso caso, a classe Professor deve herdar todo o código de funcionário.

Para implementar este relacionamento, basta modificar a declaração da classe professor para:



A palavra "**extends**" identifica que determinada classe A estende a implementação de uma classe B, ou seja, quando especificamos que Professor **extends** Funcionario, estamos especificando que a classe "Professor" herda todas as especificações da classe "Funcionario", estendendo-a com suas próprias especificações, criando uma sub-classe da classe "Funcionario", mais específica. Neste caso, dizemos que a classe "Professor" é uma sub-classe da classe "Funcionario".

Ao proceder esta modificação, o diagrama de classes é alterado, apresentando graficamente a relação de herança entre Professor e Funcionário:



Uma outra forma de implementar a herança é usar a ferramenta de inserção de herança visual do BlueJ (destacada em vermelho na figura anterior). Para usar esta ferramenta, selecione-a e em seguida clique na classe “Professor” e arraste a seta até a classe “Funcionario”. O código fonte é automaticamente atualizado.

Agora, crie a relação de herança entre “Tecnicos” e “Funcionario”.

Agora vamos criar o método lerDados() para a classe Professor, reutilizando o código já pronto na classe “Funcionário”. Para isto, implemente o seguinte método **dentro da classe “Professor”:**

```
public void lerDados() {
      super.lerDados();
```

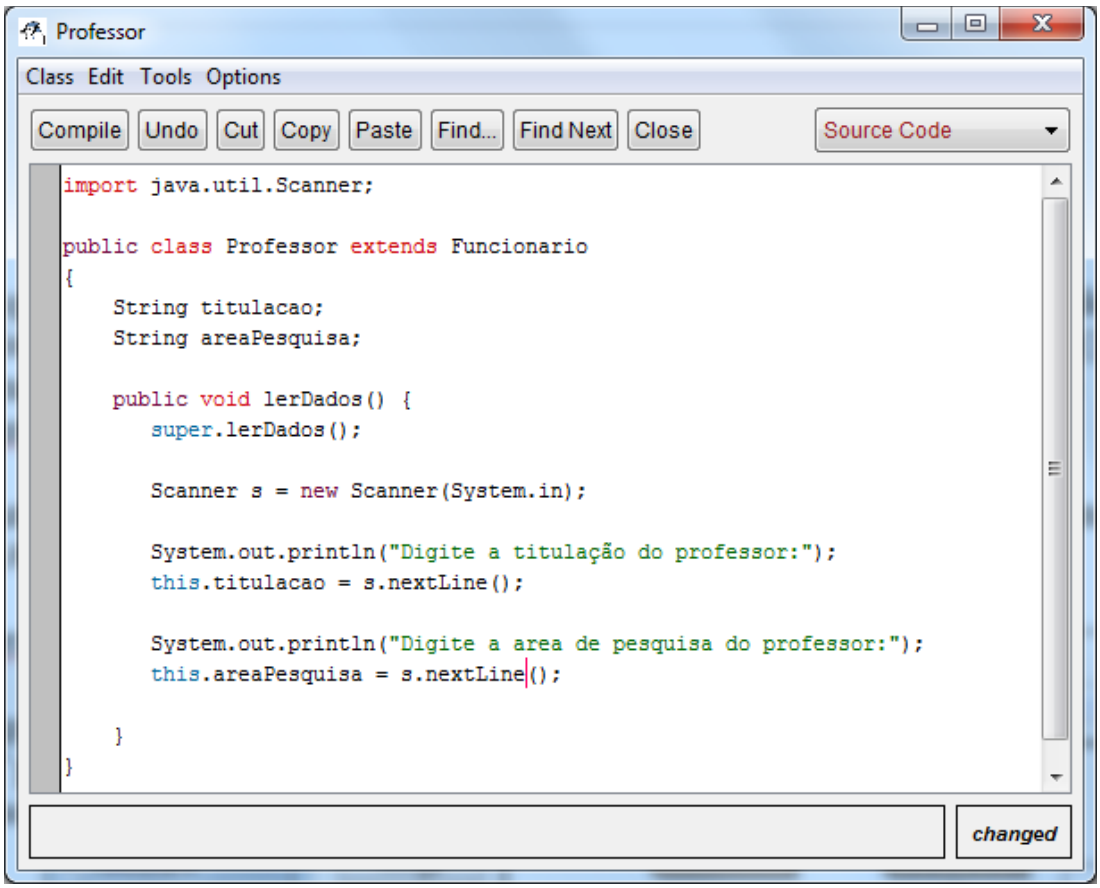
```
        Scanner s = new Scanner(System.in);

        System.out.println("Digite a titulação do professor:");
        this.titulacao = s.nextLine();

        System.out.println("Digite a area de pesquisa do professor:");
        this.areaPesquisa = s.nextLine();
}
```

A novidade nesta implementação é que foi utilizado o modificador de acesso “super” para invocar o método lerDados() da classe “Funcionário”.

Com este modificador de acesso podemos acessar o método “lerDados” da superclasse.

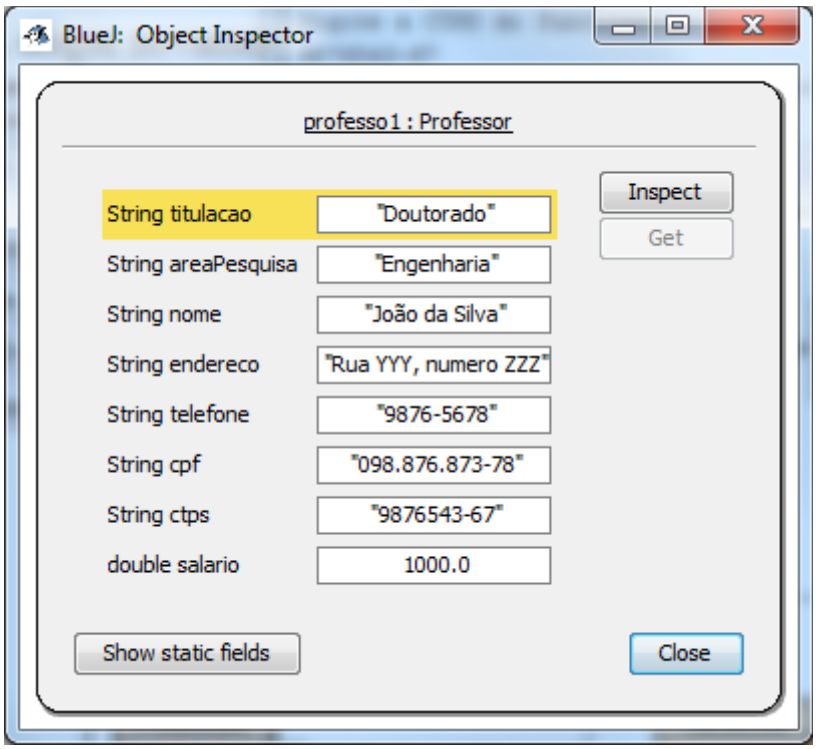
É importante ressaltar que neste caso o uso do super é obrigatório. Caso não fosse utilizado, seria realizada uma chamada recursiva ao método “lerDados” da própria classe Professor.

O resultado é o seguinte:



Vamos agora testar este método. Para isto, compile a aplicação, e clique com o botão direito na classe “Professor” e selecione a opção “new Professor()” para instanciar um objeto desta classe. Agora clique com o botão direito no objeto criado (em vermelho) e clique em “void lerDados()”. O método será executado em uma console, possibilitando que os dados sejam inseridos.

Após digitar todos os dados, clique novamente com o botão direito no objeto criado e acesse a opção *Inspect*. Lá é possível observar o valor de todos os atributos do objeto em determinado momento, conforme ilustra a figura a seguir:



Note que um objeto do tipo “Professor” possui todos os atributos de sua superclasse (“Funcionario”) e mais os atributos próprios de “Professor”.

Uma outra classe que gera gerando redundância de implementação e replicação de código é a classe Aluno. Esta classe tem alguns atributos semelhantes à classe Funcionário.

Porém, não podemos criar uma relação de herança entre as classes Aluno e Funcionário, já que **um aluno não é um funcionário**. Desta forma, a classe Aluno estaria herdando diversas informações desnecessárias. Isto prejudica o nível de coesão da classe.

Desta forma, uma saída é criar uma nova superclasse, por exemplo, Pessoa, e implementar nesta classe tudo o que é comum às classes Funcionário e Aluno. Desta forma, as classes Funcionário e Aluno poderiam herdar diretamente desta classe. Implemente isto no código.

Agora, faça os métodos lerDados() e mostrarDados() da classe Aluno.

Note que além dos atributos do aluno, é preciso ler os dados do curso. Para isto, é necessário que a classe curso também implemente os métodos lerDados() e mostrarDados():

```
import java.util.*;
public class Aluno extends Pessoa{
    String matricula;
    Curso curso;

    public void lerDados() {
        super.lerDados();
        Scanner s = new Scanner(System.in);
        System.out.println("Digite a matricula do aluno:");
        this.matricula = s.nextLine();

        System.out.println("Digite os dados do curso do aluno:");
        this.curso = new Curso();
        this.curso.lerDados();
    }
}
```

Claro que não é prático ler os dados do curso toda vez que se cadastra um aluno. Em breve faremos uma busca em uma base de cursos pré-cadastrados ao invés de ler seus dados repetidamente.

Agora, crie uma nova classe “AlunoPosGraduacao”. Esta classe deve possuir todos os atributos da classe Aluno, e mais o atributo “graduacao”, para armazenar qual é a graduação do aluno (engenharia, administração, etc...) e “instituicao”, para armazenar em qual instituição o aluno se graduou.
Além disto, são necessários atributos para armazenar o ano de início e ano de conclusão da graduação.

Ainda, a classe “AlunoPosGraduacao” deve ter uma associação com a classe “Curso”, para que possamos saber qual curso de pós-graduação este aluno esta cursando em nossa universidade (já está sendo herdado da classe “Aluno”). Implemente corretamente o lerDados() e mostrarDados().
