INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

AULA:

Herança e Polimorfismo

Programação Orientada a Objetos

Alba Lopes, Profa.

http://docentes.ifrn.edu.br/albalopes
alba.lopes@ifrn.edu.br

Repetição de Código

- Tomemos como exemplo a classe Funcionario, que representa o funcionário de um banco:

```
public class Funcionario {
    private String nome;
    private String cpf;
    private double salario;

    public String getNome() {
        return nome;
    }

    public void setNome(String nomeFuncionario) {
        nome = nomeFuncionario;
    }

    public String getCpf() {
        return cpf;
    }
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Repetição de Código

- Além de um funcionário comum, há também outros cargos, como os gerentes.
  - Os gerentes guardam a mesma informação que um funcionário comum, mas possuem outras informações, além de ter funcionalidades um pouco diferentes
  - Vamos supor que, nesse banco, o gerente possui também uma senha numérica que permite o acesso ao sistema interno do banco

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Repetição de Código

- Classe Gerente:

```
public class Gerente {
    private String nome;
    private String cpf;
    private double salario;

    private int senha;

    public boolean autentica(int testarSenha) {
        if (testarSenha == senha) {
            System.out.println("Acesso Permitido!");
            return true;
        } else {
            System.out.println("Acesso Negado!");
            return false;
        }
    }

     public String getNome() {
        return nome;
    }
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Repetição de Código

- Ao invés de criar duas classes diferentes, uma para Funcionario e outra para gerente, poderíamos ter deixado a classe Funcionario mais genérica, mantendo nela senha de acesso.
  - Caso o funcionário não fosse um gerente, deixaríamos este atributo vazio (não atribuiríamos valor a ele).
- Mas e em relação aos métodos?
  - A classe Gerente tem o método autentica, que não faz sentido ser acionado em um funcionário que não é gerente.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Repetição de Código

- Se tivéssemos um outro tipo de funcionário, que tem características diferentes do funcionário comum, precisaríamos criar uma outra classe, e copiar o código novamente!
- Ou ainda, se um precisássemos adicionar uma nova informação (ex: data de nascimento) para todos os funcionários?
  - Todas as classes teriam que ser alteradas para receber essa informação
- SOLUÇÃO: Centralizar as informações principais do funcionário em um único lugar!

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Herança

- Existe uma maneira, em Java, de relacionarmos uma classe de tal maneira que uma delas herda tudo que a outra tem.
- Isto é uma relação de classe mãe e classe filha.
- No nosso caso, gostaríamos de fazer com que o Gerente tivesse tudo que um Funcionario tem:
  - Gostaríamos que ela fosse uma extensão de Funcionario
- Fazemos isto através da palavra chave extends

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Herança

```
public class Gerente extends Funcionario{

    private int senha;

    public boolean autentica(int testarSenha) {
        if (testarSenha == senha) {
            System.out.println("Acesso Permitido!");
            return true;
        } else {
            System.out.println("Acesso Negado!");
            return false;
        }
    }


}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Herança

- Todo momento que criarmos um objeto do tipo Gerente, este objeto possuirá também os atributos definidos na classe Funcionario, pois agora um Gerente é um Funcionario!

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Herança

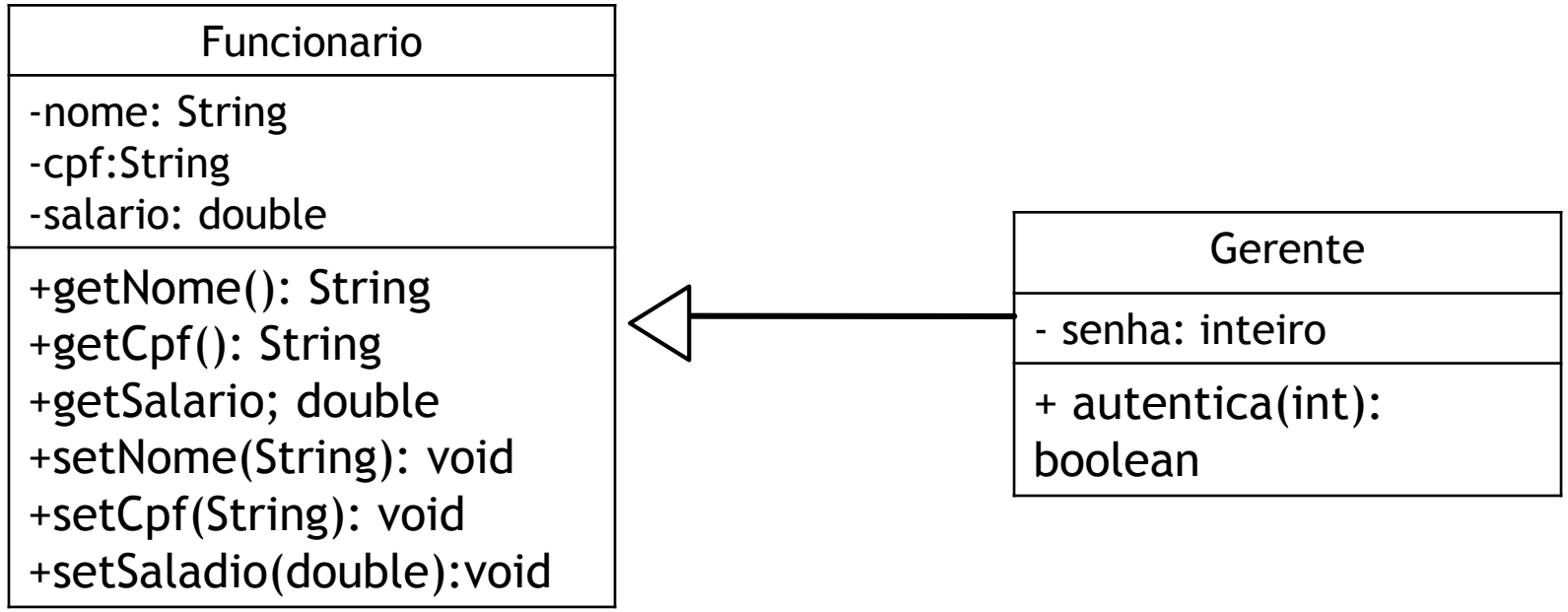
- Termos utilizados:

Classes que fornecem Herança	Classes que herdam de outras
Superclasse	Subclasse
Pai	Filha
Tipo	Subtipo

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Herança

- Exemplo de notação UML



Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Herança

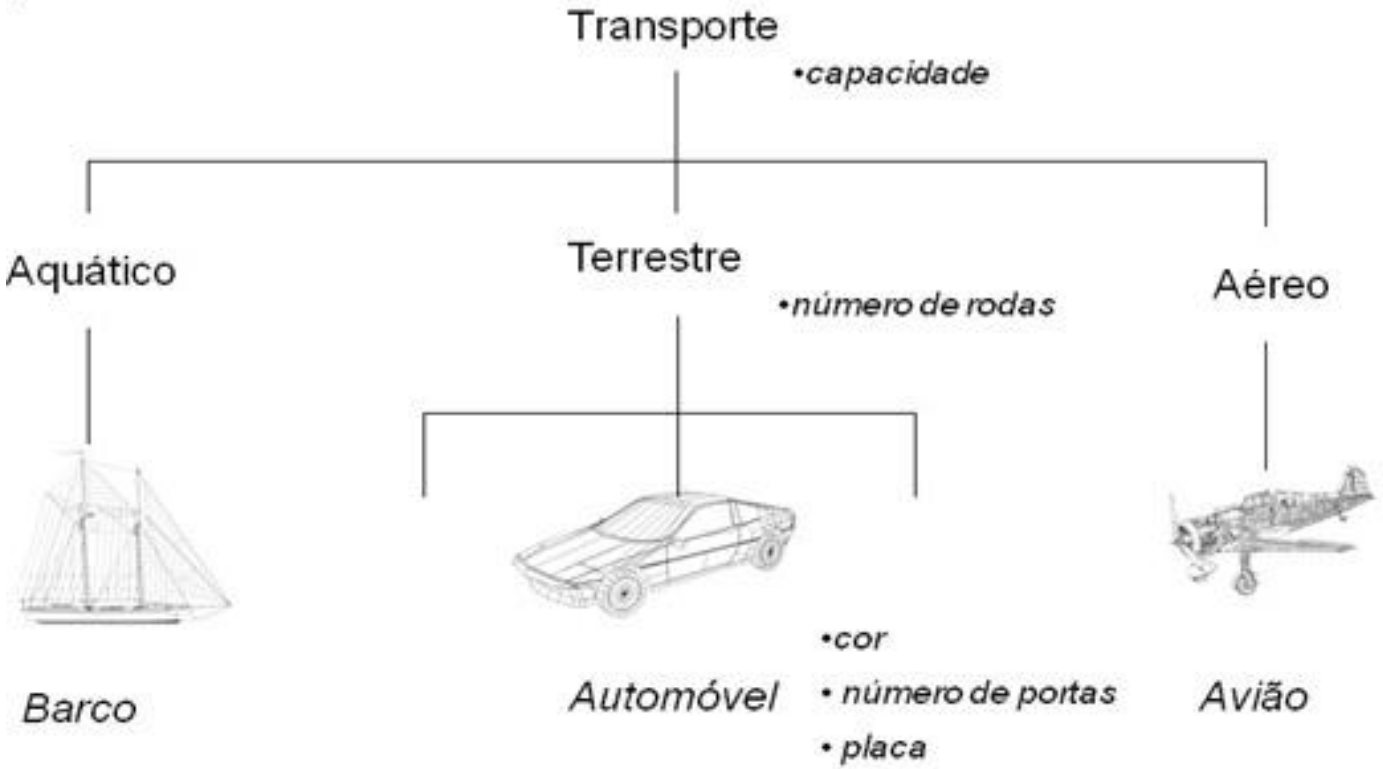
- Exemplo de Teste da classe:

```
public class TestarHeranca {
    public static void main(String [] args){
        Gerente gerenteBanco = new Gerente();
        gerenteBanco.setNome("João da Silva");
        gerenteBanco.setCpf("123456789101");
        gerenteBanco.setSenha(123456);
    }
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Exemplo 1

o



to

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Exemplo 1

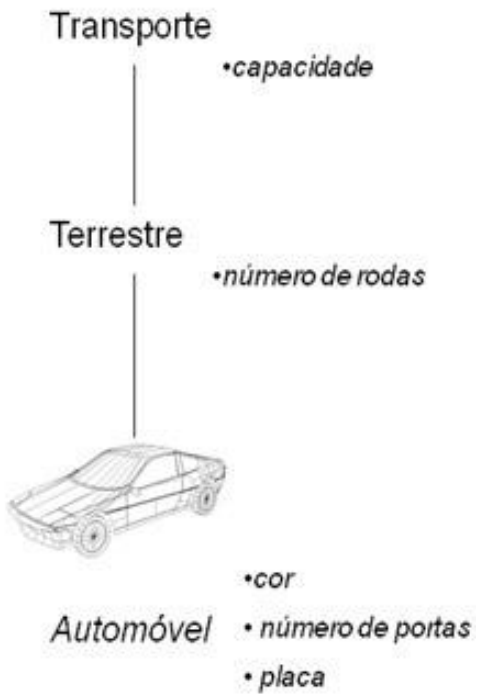
- Usando a lógica de herança responda:
  - Quantos e quais são os atributos da classe Terrestre?
  - E da classe Automóvel?

- Respondendo:
  - a classe Terrestre possui dois atributos:
    - capacidade (que é herdado de Transporte) e número de rodas.
  - Já a classe Automóvel possui cinco atributos:
    - capacidade (herdado de Transporte), número de rodas (herdado de Terrestre), cor, número de portas e placa.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Exemplo 1

- Como seria o código?




```
public class Transporte{
        private int capacidade;
}
```

```
public class Terrestre extends Transporte{
        private int numRodas;
}
```

```
public class Automovel extends Terrestre{
        private String cor;
        private int numPortas;
        private String placa;
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Exemplo 1

```
public class TestarTransporte {

    public static void main(String [] args){

        Transporte t1 = new Transporte();
        t1.setCapacidade(300);
        System.out.println("A capacidade de T1 é: " + t1.getCapacidade());

        Terrestre t2 = new Terrestre();
        t2.setCapacidade(30);
        System.out.println("A capacidade de T2 é: " + t2.getCapacidade());


        Automovel t3 = new Automovel();
        t3.setCapacidade(5);
        System.out.println("A capacidade de T3 é: " + t3.getCapacidade());

    }

}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Herança e Construtores

- Vimos em aulas passadas que podemos criar construtores que são chamados assim que o nosso objeto é criado.

```
public class Funcionario{
	private String nome;
	private String cpf;
	private double salario;
	...
	public Funcionario(String nome, String cpf, double salario){
		this.nome = nome;
		this.cpf = cpf;
		this.salario = salario;
	}
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Herança e Construtores

- Vimos em aulas passadas que podemos criar construtores que são chamados assim que o nosso objeto é criado.

```
public class TestarHeranca{
        public static void main(String [] args){
           Funcionario f = new Funcionario("Jose","123456789",2000);
           System.out.println(" Nome: " + f.getNome());
        }
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Herança e Construtores

- Quando trabalhamos com herança e um construtor foi definido para a superclasse, devemos, obrigatoriamente criar um construtor na subclasse que chame construtor da superclasse.
- Caso isso não seja feito, o código apresentará um erro.
- A chamada do método construtor da superclasse é feita através da palavra super e seguida dos argumentos específicos.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Herança e Construtores

- Sendo assim:

```
public class Gerente{
        private int senha;
        ...
        public Gerente(String nome, String cpf, double salario){
                super(nome, cpf, salario);
        }
}
```

- A chamada do método construtor da superclasse deve ser sempre o primeiro comando a aparecer no construtor da subclasse.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Herança e Construtores

- Mas o construtor da subclasse não precisar ter necessariamente os mesmos parâmetros do construtor da superclasse. Ex:

```
public class Gerente{
        private int senha;
        ...
        public Gerente(String nome, String cpf, double salario, int senha){
                super(nome, cpf, salario);
                this.senha = senha;
        }
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Herança e Construtores

```
public class TestarHeranca{

        public static void main(String [] args){
           Funcionario f = new Funcionario(“Jose”,“123456789”,2000);
           System.out.println(“ Nome Funcionario: ” + f.getNome());
           Gerente g = new Gerente(“Marcia”,“987654321”,3000, 12345);
           System.out.println(“ Nome Gerente: ” + g.getNome());
        }
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Exercício

Animal
-nome: String -comprimento:double -numDePatas: int - cor: String - ambiente : String - velocidadeMedia: double
+ toString(): String + Animal(String, double, int, String, String, double) gets e sets para todos os atributos

1. Crie uma classe Animal que obedeça à seguinte descrição:
   - possua os atributos nome (String), comprimento (double), número de patas (int), cor (String), ambiente (String) e velocidade média (double)
   - Crie um método construtor que receba por parâmetro os valores iniciais de cada um dos atributos e atribua-os aos seus respectivos atributos.
   - Crie os métodos get e set para cada um dos atributos.
   - Crie um método toString, que retorno os dados do animal

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Exercício

2. Crie uma classe **Peixe** que herde da classe Animal e obedeça à seguinte descrição:
   - possua um atributo **caracteristica**(String)
   - Crie um método construtor que receba por parâmetro os valores iniciais de cada um dos atributos (incluindo os atributos da classe **Animal**) e atribua-os aos seus respectivos atributos.
   - Crie ainda os métodos **get** e **set** para o atributo **caracteristica**.
   - Crie um método **toString** que retorno os dados do Peixe

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Exercício

3. Crie uma classe **Mamifero** que herde da classe Animal e obedeça à seguinte descrição:

- possua um atributo **alimento** do tipo String
- Crie um método construtor que receba por parâmetro os valores iniciais de cada um dos atributos (incluindo os atributos da classe **Animal**) e atribua-os aos seus respectivos atributos.
- Crie ainda os métodos **get** e **set** para o atributo **alimento**.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Exercício

4. Crie uma classe **TestarAnimais** possua um método main para testar as classes criadas.
   a) Crie um objeto **camelo** do tipo Mamífero e atribua os seguintes valores para seus atributos (utilize o construtor da classe para inicializar os atributos):
      - Nome: Camelo / Comprimento: 150 / Patas: 4
      - Cor: Amarelo / Ambiente: Terra / Velocidade: 2.0 / Alimento: leite
   b) Crie um objeto **tubarao** do tipo Peixe e atribua os seguintes valores para seus atributos
      - Nome: Tilapia / Comprimento: 20 / Patas: 0
      - Cor: Cinzento / Ambiente: Mar / Velocidade: 1.5
      - Caracteristica: cauda
   c) Crie um objeto **ursocanada** do tipo Mamifero e atribua os seguintes valores para seus atributos:

      Nome: Urso-do-canadá
      - Comprimento: 180 / Patas: 4 / Cor: Vermelho
      - Ambiente: Terra / Velocidade: 0.5 / Alimento: Mel

- Chame os método para toString de cada um dos objetos criados.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Sobrescrita de Método

- É a capacidade de redefinir um método com mesma assinatura em sua subclasse
- Tomemos como exemplo a classe Funcionário e incluir um novo método bonificacao. Esse método representa uma bonificação que todos os funcionários recebem no fim do ano e é referente a 10% do valor do salário. Porém, o gerente recebe uma bonificação de 15%.
- Como ficaria o código da nossa classe Funcionario?

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Sobrescrita de Método

```
public class Funcionario{
	private String nome;
	private String cpf;
	private double salario;

	...

	public double bonificacao(){
		double b = getSalario() * 0.10;
		return b;
	}

}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Sobrescrita de Método

- Se deixarmos a classe Gerente como está, ela vai herdar o método bonificacao da classe Funcionario:

```
public class TestarHeranca{
        public static void main(String [] args){
           Gerente g = new Gerente();
           g.setSalario(3000);
           System.out.println(“ A bonificacao é: ” + g.bonificacao);
        }
}
```

- Nesse caso, a bonificação do Gerente será 300 ao invés de 450, o que está errrado.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Sobrescrita de Método

- O método bonificacao deve ser reescrito na classe Gerente de modo que ele forneça o valor correto:

```
public class Gerente{
        private int senha;
        ...
        public double bonificacao(){
                double b = getSalario() * 0.15;
                return b;
        }
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Sobrescrita de Método

- Agora, se executarmos o mesmo código anterior, ele dará o valor correto!

```
public class TestarHeranca{
        public static void main(String [] args){
            Gerente g = new Gerente();
            g.setSalario(3000);
            System.out.println(“ A bonificacao é: ” + g.bonificacao);
        }
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Sobrescrita de Método

- Método toString
  - O método toString é definido na classe Object e pode ser sobrescrito para apresentar as informações de um determinado objeto
  - Ele tem a seguinte assinatura: public String toString()

```
public class Funcionario{
        private String nome;
        private String cpf;
        private double salario;
        ...
        public String toString(){
                String dados = "Nome: "+this.nome+"\n";
                dados += "Cpf: "+this.cpf+"\n";
                dados += "Salario: "+this.salario+"\n";
                return dados;
        }
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Polimorfismo

- Na herança, vimos que Gerente é um Funcionario, pois é uma extensão deste. Podemos referenciar a um Gerente como sendo um Funcionario.
- Se alguém precisa falar com um Funcionario do banco, pode falar com um Gerente!
- Por que? Pois Gerente é um Funcionario. Essa é a idéia da herança

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Polimorfismo

```
public class TestarHeranca2{

        public static void main(String [] args){
           Funcionario f = new Gerente(“Marcia”,“987654321”,3000, 12345);
                   System.out.println(“Nome Gerente: ” + f.getNome());
        }
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Polimorfismo

- Polimorfismo é a capacidade de um objeto poder ser referenciado de várias formas
  - CUIDADO: polimorfismo não quer dizer que o objeto fica se transformando.
  - Muito pelo contrario, um objeto nasce de um tipo e morre daquele tipo, o que muda é a maneira como nos referenciamos a ele.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Polimorfismo

- Mas e se tentarmos fazer:

```
public class TestarHeranca2{

        public static void main(String [] args){
            Funcionario f = new Gerente("Marcia","987654321",3000, 12345);
                    System.out.println("Nome Gerente: " + f.getNome());
            System.out.println("Bonificacao: " + f.bonificacao() );
        }
}
```

- Qual é o retorno do método bonificacao? 300 ou 450?

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Polimorfismo

- Nesse caso, a resposta seria 450, pois o tipo de dados que criamos ao fazer

```
Funcionario f = new Gerente(“Marcia”,“987654321”,3000, 12345);
```

foi Gerente. E o método bonificacao da classe Gerente faz o cálculo de 15%!

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Polimorfismo

- Mas em que situação isso seria útil?
  - Por exemplo, no caso de precisarmos passar um Funcionario como parâmetro de um método:

```
public class ControleDeGastos{
        private double gastosComBonificacao = 0;

        public void bonificacaoPorFuncionario(Funcionario f){
                gastosComBonificacao += f.bonificacao();
        }

        public void getGastosComBonificacao(){
                return gastosComBonificacao;
        }
}
```

  - Ou criar um array que armazene objetos do tipo Funcionario (ele poderá guardar tanto funcionarios como gerentes

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Polimorfismo

- Testando a classe:

```
public class TestarControleDeGastos {

    public static void main(String [] args ){
        ControleDeGastos cg = new ControleDeGastos();
        Funcionario f = new Funcionario("Jose","123456789",2000);
        Gerente g = new Gerente("Marcia", "987654321", 3000, 12345);
        cg.bonificacaoPorFuncionario(f);
        System.out.println("Parcial de Gastos 1: " + cg.getGastosComBonificacao());
        cg.bonificacaoPorFuncionario(g);
        System.out.println("Parcial de Gastos 2: " + cg.getGastosComBonificacao());


    }

}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Referências

- Caelum - Java e Orientação a Objetos
- Métropole Digital – Programação Orientada a Objetos: http://www.metropoledigital.ufrn.br/aulas/disciplinas/poo/aula_10.html

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br