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

AULA:

Classes Abstratas e Interface

Programação Orientada a Objetos

Alba Lopes, Profa.

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

Agenda

- Atributos e métodos estáticos
- Classe Object
- instanceof
- Classes Abstratas
- Atributos e métodos final
- Interfaces

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

Atributos e métodos de classe

- Vimos até agora que atributos e métodos pertencem aos objeto
  - Não se faz nada sem antes criar um objeto (new)!
- No entanto, há situações que você quer usá-los sem ter que criar objetos:
- Deseja-se um atributo associado a uma classe como um todo (todos os objetos compartilham a mesma variável, similar a uma “variável global”);
- Deseja-se chamar um método mesmo que não haja objetos daquela classe criados.

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

Atributos e métodos de classe

- Usando a palavra-chave static você define um atributo ou método de classe (“estático”):
  - Atributo/método pertence à classe como um todo;
  - Pode-se acessá-los mesmo sem ter criado um objeto;
  - Objetos podem acessá-los como se fosse um membro de objeto, só que compartilhado;
  - O contrário não é possível: métodos static não podem acessar atributos/métodos não-static diretamente (precisa criar um objeto).

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

Exemplos (Atributos *static*)

```
public class ClasseStatic {
      public static int i = 47;
      public int j = 26;
}
```

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

Exemplos (Atributos *static*)

```
public class TesteClasseStatic{
        public static void main(String[] args) {
                ClasseStatic ts1 = new ClasseStatic();
                ClasseStatic ts2 = new ClasseStatic();
                // 47 26
                System.out.println(ts1.i + " " + ts1.j);
                // 47 26
                System.out.println(ts2.i + " " + ts2.j);
                ts1.i++;
                ts1.j++;
                // 48 27
                System.out.println(ts1.i + " " + ts1.j);
                // 48 26
                System.out.println(ts2.i + " " + ts2.j);
                ClasseStatic.i = 100;
                System.out.println(ts1.i); // 100
                System.out.println(ts2.i); // 100
        }
}
```

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

Exemplos (Atributos *static*)

```
public class ClasseStatic{
        static int i = 47;
        int j = 26;

        static void imprime(String s) {
                System.out.println(s);
        }

        static void incrementaI() { i++; }

        void incrementaJ() { j++; }
}
```

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

Exemplos (Métodos static)

```
public class TesteClasseStatic{

        public static void main(String[] args) {
                ClasseStatic ts1 = new ClasseStatic();
                ClasseStatic.incrementaI(); // OK
                ts1.incrementaI(); // OK
                // TesteStatic.incrementaJ(); //ERRO!
                ts1.incrementaJ(); // OK
                // 50 27
                imprime(ts1.i + " " + ts1.j);
        }
}
```

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

A classe Object

- Em Java, todos os objetos participam de uma mesma hierarquia, com uma raiz única;
- Esta raiz é a classe java.lang.Object.

```
class Produto { }
```

```
class Produto extends Object { }
```

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

A classe Object

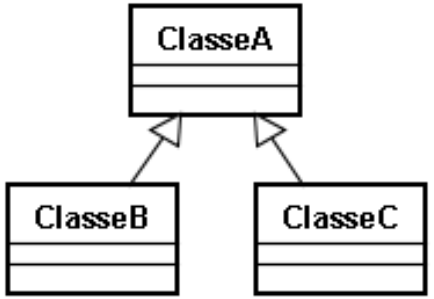
- Possui alguns métodos úteis:
  - clone(): cria uma cópia do objeto (uso avançado);
  - equals(Object o): verifica se objetos são iguais;
  - finalize(): chamado pelo GC (não é garantia);
  - getClass(): retorna a classe do objeto;
  - hashCode(): função hash;
  - notify(), notifyAll() e wait(): para uso com threads;
  - toString(): converte o objeto para uma representação como String.

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

instanceof

- Operador
  - Determina se um objeto é de uma determinada classe
    - retorna um valor lógico (booleano)
    - sintaxe: objeto instanceof Classe

```
public static void main(String args[]){
   ClasseA obj;
   obj = new ClasseC();
    if (obj instanceof ClasseB)
          System.out.println("obj é
ClasseB");
    if (obj instanceof ClasseA)
     System.out.println("obj é ClasseA");
    if (obj instanceof ClasseC)
     System.out.println("obj é ClasseC");
}
```



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

Classes Abstratas

- Algumas vezes classes no topo da hierarquia são muito gerais:
- Exemplo:
  - Classe Pessoa
    - PessoaFísica extends Pessoa
    - PessoaJuridica extends Pessoa
  - Em um array de objetos do tipo Pessoa, faz sentido que os objetos ou sejam PessoaFísica ou sejam PessoaJuridica
  - A classe Pessoa, nesse caso, estaria sendo usada apenas para ganhar o polimorfismo e herdar algumas coisas.
  - Não faz sentido permitir instanciar a classe Pessoa!

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

Classes Abstratas

- Exemplo Funcionário
  - Em geral, nas empresas, os funcionários possuem alguma função: Diretor, Gerente, Secretária, etc.
  - A classe Funcionário idealiza um tipo. Define um rascunho para outras classes.
- Usamos a palavra chave abstract para impedir que ela possa ser instanciada.
  - Esse é o efeito direto de se usar o modificador abstract na declaração de uma classe

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

Classes Abstratas

```
abstract class Funcionario {

  protected double salario;

  public double getBonificacao() {
    return this.salario * 1.2;
  }

  // outros atributos e métodos comuns a todos Funcionarios

}
```

- E no meio do código

```
Funcionario f = new Funcionario(); // não compila!!!
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Cannot instantiate the type Funcionario

        at br.com.caelum.empresa.TestaFuncionario.main(TestaFuncionario.java:5)
```

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

Classes Abstratas

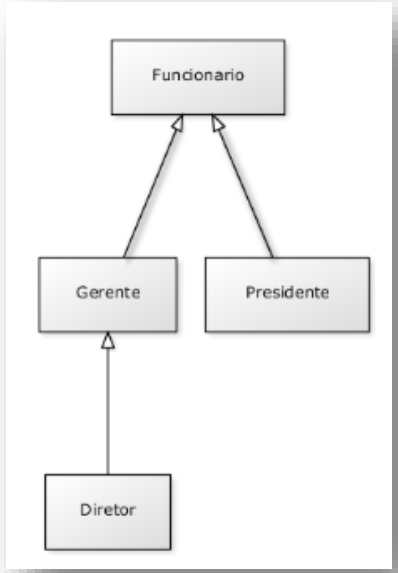
- Mas se a classe não pode ser instanciada, pra que ela serve?
  - Serve para o polimorfismo e herança dos atributos e métodos, que são recursos muito poderosos, como já vimos

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

Classes Abstratas

- Se o método bonificação da classe Gerente não fosse reescrito, ele herdaria o método da classe Funcionário.

```
public class Gerente extends Funcionario{

    public double bonificacao(){
        return getSalario() * 0.15;
    }
```

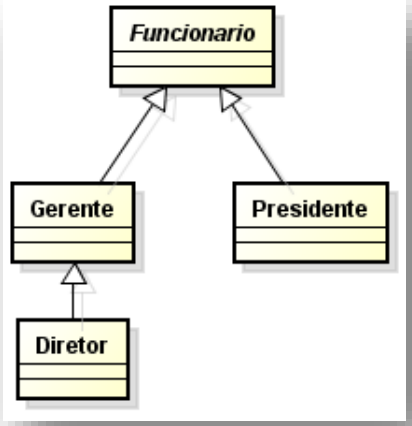


- Porém, levando em consideração que cada funcionário do sistema (Gerente, Presidente, Diretor, etc) tem uma regra totalmente diferente para ser bonificado, faz algum sentido ter esse método na classe Funcionario?

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

Classes Abstratas

- Porém, levando em consideração que cada funcionário do sistema (Gerente, Presidente, Diretor, etc) tem uma regra totalmente diferente para ser bonificado, faz algum sentido ter esse método na classe Funcionario?
  - A princípio, não! Cada classe filha terá um método diferente de bonificação.
- Então o mehor a fazer é jogar fora o método bonificação da classe Pai?
  - O problema é que, se ele não existir, não será possível chamar o método apenas com uma referência a um Funcionario, pois ninguém garante que essa referência aponta para um objeto que possui esse método.



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

Classes Abstratas

- SOLUÇÃO:
  - Existe um recurso em Java que, em uma classe abstrata, podemos escrever que determinado método será sempre escrito pelas classes filhas. Isto é, um método abstrato.
  - Um método abstrato não tem implementação. É representado apenas pela assinatura do método.
- Toda classe que possua pelo menos um método abstrato deverá ser, obrigatoriamente, abstrata!

```
public abstract class Funcionario {
    /*... */
    public abstract double bonificacao();

    /*... */
}
```

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

Classes Abstratas

- O método abstrato da classe “Pai” nunca vai ser chamado, pois a classe abstrata nunca será instanciada.
  - Qualquer classe que estender a classe Pai (ex: Gerente extends Funcionario) será obrigada a reescrever este método, tornando-o "concreto". Se não reescreverem esse método, um erro de compilação ocorrerá.

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

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

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

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

Classes Abstratas

- Na classe ControleDeGastos, o método bonificacaoPorFuncionario utiliza o método bonificacao, da classe Funcionario.
  - Porém, o método bonificacao não está implementado na classe Funcionario. Só que o abstract garante que todas as classes filha obrigatoriamente o implementarão.

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

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

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

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

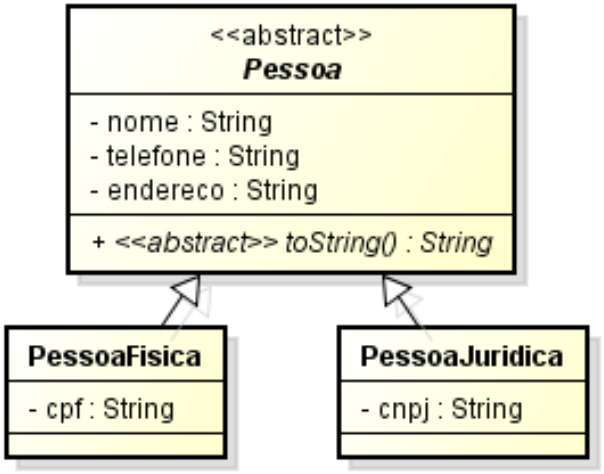
Classes Abstratas

- Algumas regras:
  - Métodos estáticos não podem ser abstratos;
  - Construtores não podem ser abstratos;
  - Classes abstratas podem ter construtores (lembre-se que são chamados pelas subclasses!);
  - Métodos abstratos não podem ser privados.
  - Uma classe abstrata pode ter métodos abstract ou não;
  - Nunca usa abstract com: final e private;
  - Não se instancia algo abstract (new class()) – isso nem compila;
  - Um método abstract na classe obriga a classe se tornar abstrata;

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

Exercícios

1. Implemente as classes conforme o diagrama abaixo:

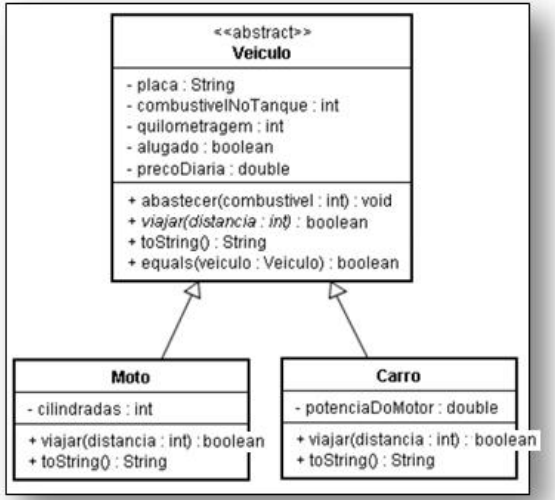


Crie uma classe **TestarPessoa** que instancie objetos dos tipos criados.

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

Exercícios

2. Implemente as classes conforme o diagrama:
   - O método abastecer deve adicionar o valor passado por parâmetro ao atributo combustivelNoTanque
   - O método equals deve retornar true se o valor do atributo placa for o mesmo para os dois objetos.



- Método viajar - Moto: o método deve considerar que uma moto faz 30km com 1 litro de combustível. Logo, deve verificar se o combustível no tanque é suficiente para percorrer a distância passada como parâmetro do método. Caso seja, deverá reduzir essa quantidade do atributo combustívelNoTanque e adicionar o valor da distância ao valor do atributo quilometragem. Retorne o valor true caso seja possível realizar a viagem. Caso contrário, retorne false
- Método viajar - Carro: Deve considerar que um carro faz 10km com um litro de combustível. Fazer as mesmas operações que as descritas no método viajar da classe Moto.
- Crie a classe TestarVeiculo e instancie objetos das classes implementadas.

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

Atributos e métodos final

- Variáveis com o modificador final
  - Não podem ser modificadas depois de inicializadas (constantes)
  - Podem ser de instância ou de classe
- Classes com o modificador final
  - Não podem ser herdadas
- Pode ser usado junto com static

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

Palavra chave - final

- Não podem ser herdadas
  - Não possuem subclasses
  - Não podem ser abstratas
- Por conseqüência, não possuem métodos abstratos
- Classe que não pode ser herdada

```
public final class Imutavel(){
   //Implementação da Classe
}
```

- Método que não pode ser redefinido

```
public class Teste(){
   public final void metodoImutavel(){}
}
```

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

Interfaces

- Na linguagem de programação java, uma interface não é uma classe, mas um conjunto de requisitos para classes que precisam adequar-se a ela. (CORE Java)
- É um tipo especial de classe 100% abstrata “pura”;
- Você nunca pode utilizar operador new para instanciar uma interface;
- Ao conversar com outros programadores, cuidado para não confundir com “interface com o usuário”.

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

Interfaces

- Estabelece a interface (contrato) de um conjunto de classes;
- Classes utilizam implements ao invés de extends para implementar uma interface;
- Classes podem implementar mais de uma interface.
  - Alternativa para herança múltipla

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

Interfaces

- Exemplo:
  - O Sistema de Controle do Banco pode ser acessado, além de pelos Gerentes, pelos Diretores do Banco. Então, teríamos uma classe Diretor:

```
class Diretor extends Funcionario {

  public boolean autentica(int senha) {
    // verifica aqui se a senha confere com a recebida como parametro
  }

}
```

```
class Gerente extends Funcionario {

  public boolean autentica(int senha) {
    // verifica aqui se a senha confere com a recebida como parametro
    // no caso do gerente verifica também se o departamento dele
    // tem acesso
  }

}
```

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

Interfaces

- Considere o SistemaInterno e seu controle: o método login precisa receber um Diretor ou Gerente como argumento, verificar se ele se autentica e colocá-lo dentro do sistema.

```
class SistemaInterno {

  void login(Funcionario funcionario) {
    // invocar o método autentica?
    // não da! Nem todo Funcionario tem
  }
}
```

- PROBLEMA: Nem todo Funcionario tem o método autentica, o que acarreta erro de compilação!

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

Interfaces

- SOLUÇÃO 1: Criar 2 métodos

```
class SistemaInterno {

  // design problemático
  void login(Diretor funcionario) {
    funcionario.autentica(...);
  }

  // design problemático
  void login(Gerente funcionario) {
    funcionario.autentica(...);
  }

}
```

- PROBLEMA: ter vários métodos, um para cada tipo de funcionário. Acarreta dificuldade de manutenção. Cada vez que uma nova classe de funcionário autenticável no sistema é criada, é preciso criar um método na classe SistemaInterno

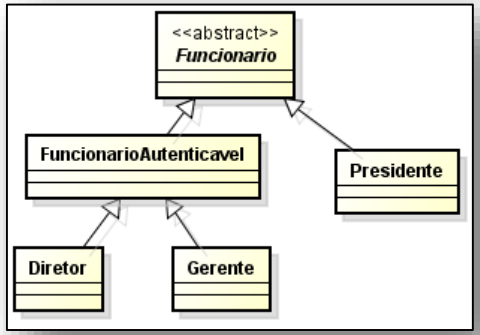
Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Interfaces

- SOLUÇÃO 2: Criar uma classe intermediária (FuncionarioAutenticavel) que possua o método login. As classes Gerente e Diretor herdarão dessa classe intermediária.

```
class SistemaInterno {

  void login(FuncionarioAutenticavel fa) {

    int senha = //pega senha de um lugar, ou de um scanner de polegar

    // aqui eu posso chamar o autentica!
    // Pois todo FuncionarioAutenticavel tem
    boolean ok = fa.autentica(senha);

  }
}
```

- PROBLEMA: Ok. Mas se o cliente do banco também
- puder fazer login no sistema? Não faz sentido
- fazer o cliente herdar da classe
- FuncionarioAutenticavel.



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

Interfaces

- O que é preciso para resolver o problema?
  - Arranjar uma forma de poder referenciar Diretor, Gerente e Cliente de uma mesma maneira, isto é, achar um fator comum.
- Garantir a existência de um determinado método, através de um contrato.

contrato Autenticavel:

- quem quiser ser Autenticavel precisa saber fazer

1. autenticar dada uma senha do tipo inteiro, devolvendo um booleano

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

Interfaces

- Quem quiser, pode "assinar" esse contrato, sendo assim obrigado a explicar como será feita essa autenticação. A vantagem é que, se um Gerente assinar esse contrato, é possível ser referir a um Gerente como um Autenticavel.
- Esse contrato em Java seria da seguinte forma:

```
interface Autenticavel {

  boolean autentica(int senha);

}
```

- Na interface, são definidas as assinaturas dos métodos que se deve implementar.

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

Interfaces

- O Gerente pode “assinar” o contrato = implementar a interface.
- No momento que implementa a interface, é obrigado a escrever os métodos pedidos pela interface
  - Muito parecido com herdar os métodos abstratos
- Para implementar uma interface, utiliza-se a palavra implements

```
class Gerente extends Funcionario implements Autenticavel {

  private int senha;

  // outros atributos e métodos

  public boolean autentica(int senha) {
    if(this.senha != senha) {
      return false;
    }
    // pode fazer outras possíveis verificações, como saber se esse
    // departamento do gerente tem acesso ao Sistema

    return true;
  }

}
```

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

Interfaces

- A partir de então, a classe Gerente pode ser tratada como Autenticavel:

```
Autenticavel a = new Gerente();
```

- Isso permite muito mais Polimorfismo!

```
class SistemaInterno {

  void login(Autenticavel a) {
     int senha = // pega senha de um lugar, ou de um scanner de
polegar
     boolean ok =  a.autentica(senha);

     // aqui eu posso chamar o autentica!
     // não necessariamente é um Funcionario!
     // Mais ainda, eu não sei que objeto a
   // referência "a" está apontando exatamente! Flexibilidade.
  }

}
```

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

Interfaces

- Para que outro tipo de objeto tenha acesso ao sistema, basta que ele implemente também a interface Autenticavel!
- Isso permite muito mais Polimorfismo!

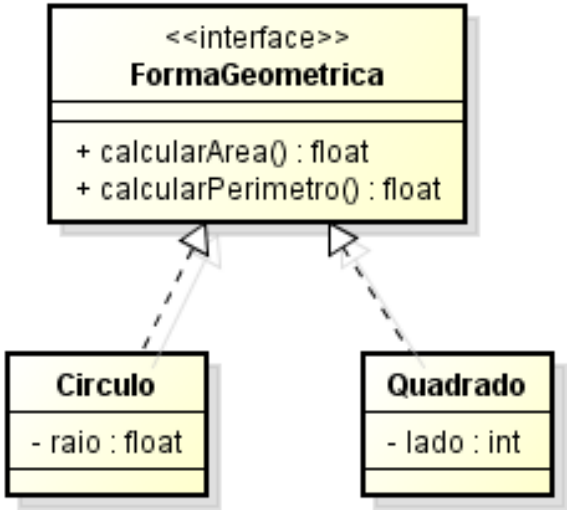
```
class SistemaInterno {

  void login(Autenticavel a) {
     int senha = // pega senha de um lugar, ou de um scanner de
polegar
     boolean ok =  a.autentica(senha);

     // aqui eu posso chamar o autentica!
     // não necessariamente é um Funcionario!
     // Mais ainda, eu não sei que objeto a
   // referência "a" está apontando exatamente! Flexibilidade.
  }

}
```

- Não faz diferença se o objeto passado para o método login é Gerente, Diretor ou Cliente, contato que implemente a interface Autenticavel com o método autentica.

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

Exercícios

3. Implemente o diagrama de classes a seguir:



- Crie uma classe de teste e instancie objetos das classes implementadas.

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

Referências

- CAELUM. Apostila de Java e Orientação a Objetos.
- SIERRA, Katy; BATES, Bert. **Use a cabeça JAVA**. Ed 2, Editora Altabooks;
- SIERRA, Katy; BATES, Bert. **SCJP - Certificação Sun para Programador Java**. Editora Editora Altabooks;
- Material do **ESJUG - Grupo de Usuários Java do Espírito Santo** – Programação Orientada Objetos. Março de 2008.
- LIGUORI, Robert; LIGUORI, Patricia. **Java Guia de bolso**. Editora Alta Books.
- Material produzido pela empresa **Argonavis** – Helder da Rocha. Programação Orientada Objetos.
- Material do Curso de Orientação a Objetos da Professora **Marília Freire - IFRN Central.**

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