PROGRAMAÇÃO ORIENTADA A OBJETOS

PROF. EDSON MORENO

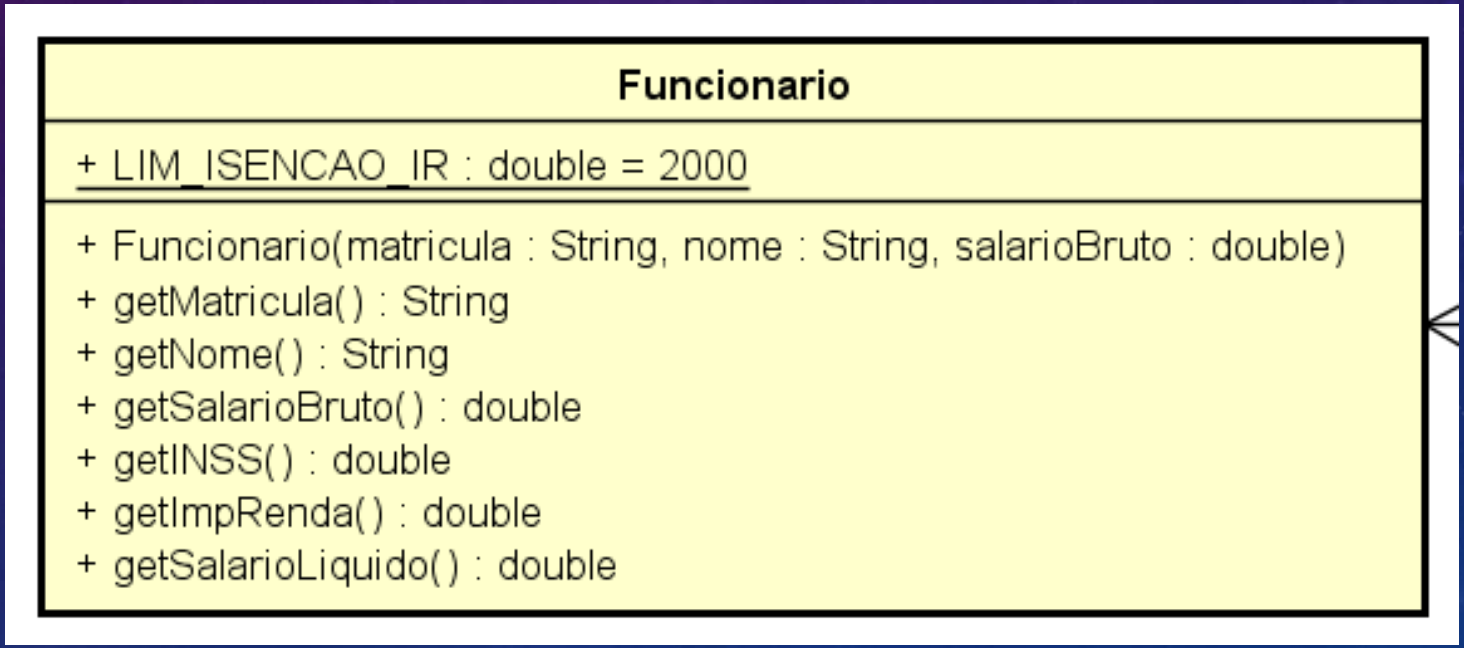
BASEADO NO MATERIAL GERADO PELO PROF. JULIO MACHADO

HERANÇA

CONCEITOS ESSENCIAIS

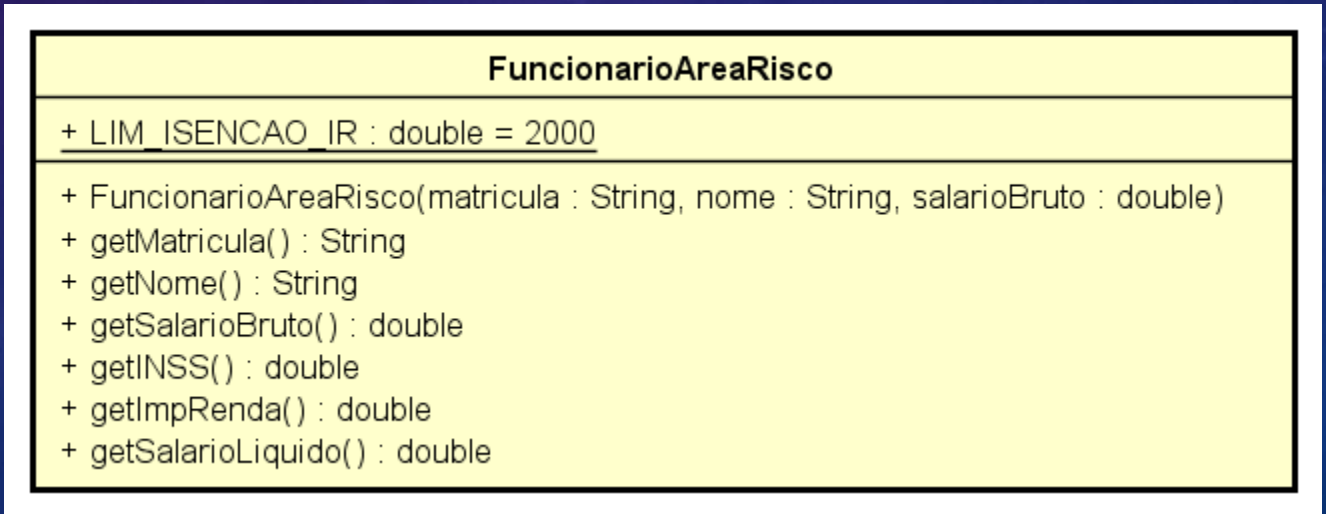
EXERCÍCIO: CLASSE FUNCIONARIO

- Modela um funcionário padrão da empresa



EXERCÍCIO: CLASSE FUNCIONARIOAREARISCO

- Possui as mesmas características que Funcionário
- Modela uma “categoria” diferente de funcionário
- Seu ***comportamento*** é diferente (o cálculo do salário não é o mesmo)
- Implica em duplicação de código → difícil de manter



EXERCÍCIO: SOLUÇÃO HERANÇA

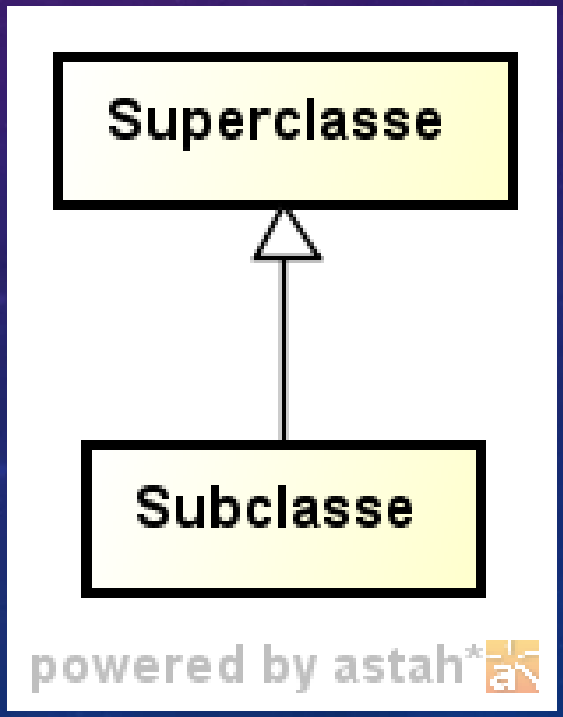
- Permite definir uma relação “é um tipo de”
- *FuncionarioAreaRisco* **é um tipo de** *Funcionario*
- Simplifica o reuso de código no caso do exercício
  - Permite reescrever apenas o comportamento que muda

HERANÇA

- Herança é uma relação de generalização/especialização entre classes
- A ideia central de herança é que novas classes são criadas a partir de classes já existentes
  - Superclasse: classe já existente
  - Subclasse: classe criada por especialização a partir da superclasse

DIAGRAMA DE CLASSES UML

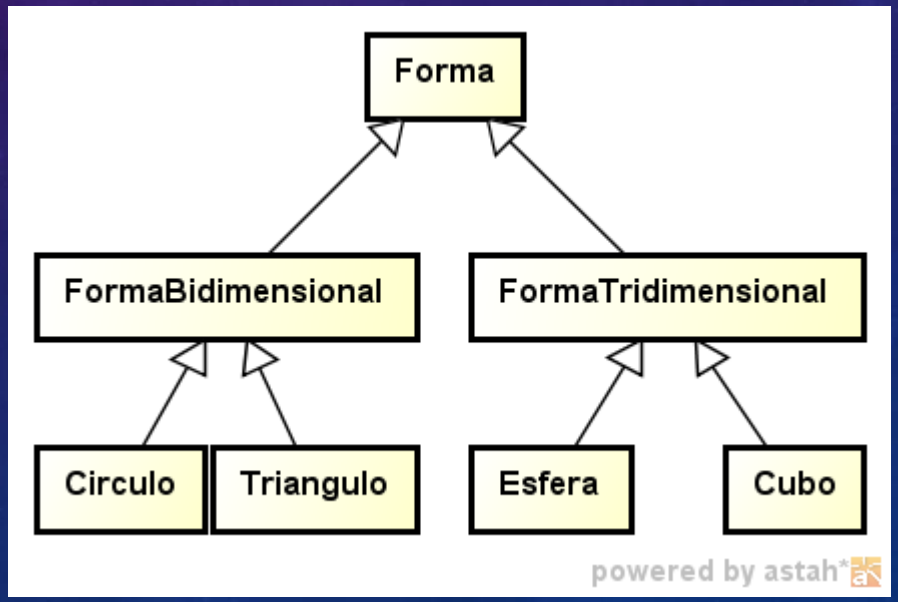
- Relacionamento de herança:



HERANÇA

- Herança cria uma estrutura hierárquica
- Ex.: uma hierarquia de classes para formas geométricas
  - Uma forma geométrica pode ser especializada em dois tipos: bidimensional e tridimensional

especialização



generalização

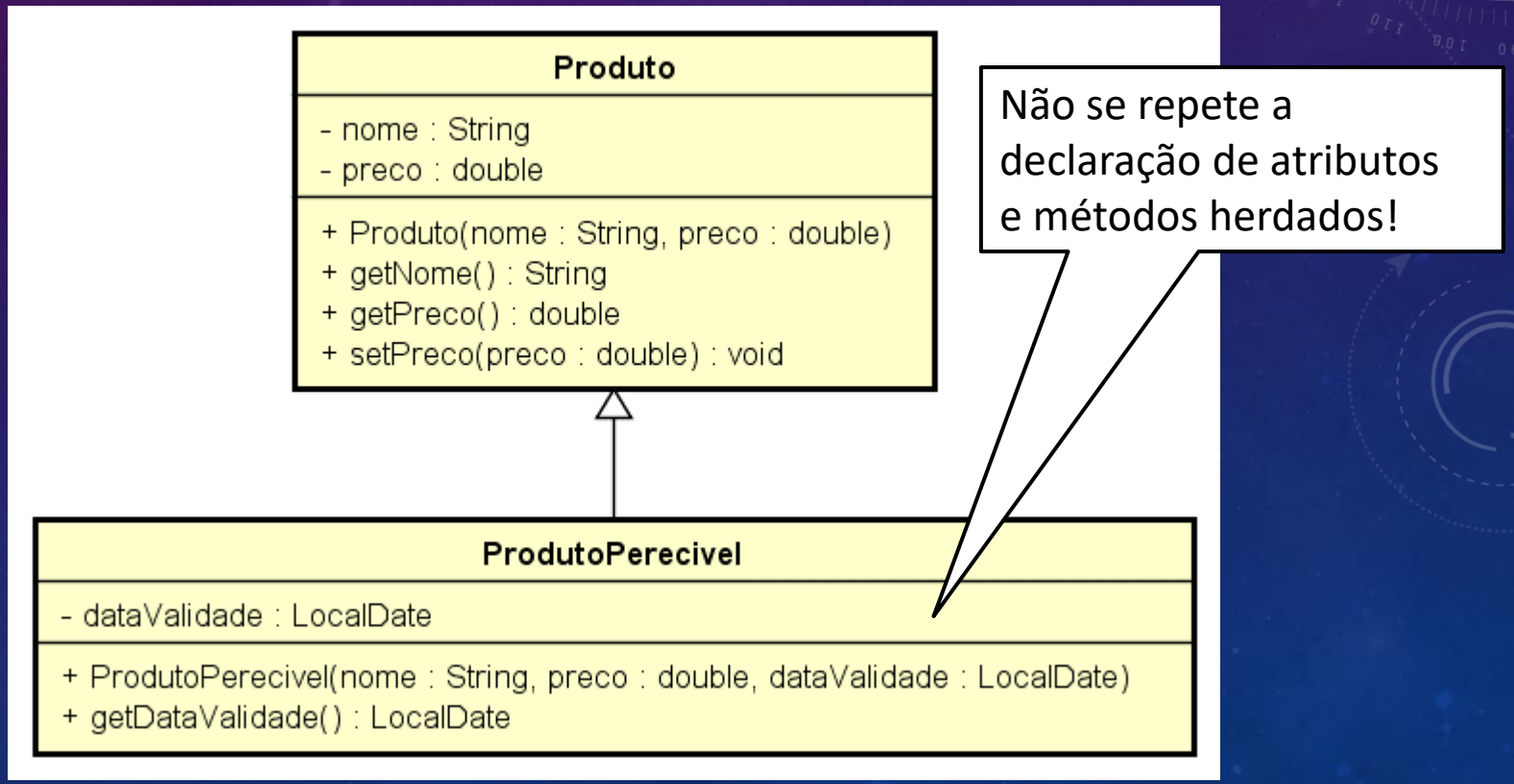
HERANÇA

- Como implementar herança em Java?
  - Utiliza-se a palavra-chave `extends` para definir herança de classes
  - Somente é possível herdar de uma única superclasse!

```
class Subclasse extends Superclasse {
   ...
   }
```

HERANÇA

- Exemplo:



HERANÇA

- Exemplo:

```
public class Produto{...}

public class ProdutoPerecivel extends
  Produto{...}
```

HERANÇA

- Ao modelar os atributos da subclasse:
  - Herdamos os atributos da superclasse
    - Todos os atributos da superclasse são herdados automaticamente
      - Ex.: atributos nome e preço de Produto
  - Podemos definir novos atributos
    - Evitar criar atributos com o mesmo nome de atributos herdados
      - Ex.: atributo validade de ProdutoPerecivel

HERANÇA

- Exemplo:

```
public class Produto{
  private String nome;
  private double preco;
  public Produto(String n, double p){
    nome = n;
    preco = p;
  }
  ...
}
```

HERANÇA

- Exemplo:

```
public class ProdutoPerecivel extends Produto{
  private LocalDate validade;
  public ProdutoPerecivel(String n, double p, LocalDate data){
    nome = n;
    preco = p;
    validade = data;
  }
  ...
}
```

Não é a forma correta de inicializar os atributos herdados!
Qual o erro indicado pelo compilador Java?

HERANÇA

- Subclasse
  - Tem acesso a todos os métodos públicos da superclasse
- Logo...
  - Devemos utilizar o construtor da superclasse para inicializar os atributos herdados
  - Utiliza-se `super()`
    - Deve ser o primeiro comando do construtor da subclasse!
    - Sempre é utilizado por padrão se não for explicitamente utilizado!

HERANÇA

- Exemplo:

```
public class ProdutoPerecivel extends Produto{
  private Date validade;
  public ProdutoPerecivel(String n, double p, LocalDate data){
    super(n,p);
    validade = data;
  }
  ...
}
```

HERANÇA

- Modificadores de acesso:
  - *public*:
    - acessível em qualquer classe
  - *private*:
    - acessível somente dentro da própria classe
  - *protected*:
    - acessível dentro da própria classe ou de uma subclasse;
    - em Java permite também acesso de mesmo *package*

HERANÇA

- Ao modelar métodos da subclasse:
  - Herdamos os métodos da superclasse
    - Os métodos são herdados automaticamente
      - Ex.: métodos getNome() e getPreco() de Produto
  - Podemos definir novos métodos
      - Ex.: método getDataValidade() de ProdutoPerecivel
  - Podemos sobrescrever métodos da superclasse

SOBRESCRITA DE MÉTODOS

- Uma subclasse pode sobrescrever métodos da superclasse
  - Sobrescrita permite modificar um comportamento herdado
  - Quando um método é referenciado em uma subclasse
    - a versão da subclasse é utilizada, ao invés do método na superclasse
  - É possível acessar o método original da superclasse:

```
super.nomeDoMetodo()
```

SOBRESCRITA DE MÉTODOS

- Um exemplo de sobrescrita
  - Métodos herdados da classe *Object*
  - Em Java
    - Todas as classes herdam da classe `Object`
    - `Object` é o topo da hierarquia de classes em Java
    - Toda classe criada sem explicitar uma superclasse, herda implicitamente da superclasse `Object`

SOBRESCRITA DE MÉTODOS

- Alguns métodos herdados de `Object`:
  - `String toString()` retorna uma representação de string do objeto
    - Usualmente utilizado para realizar a depuração de programas
    - Também é chamado implicitamente quando um objeto é utilizado em um contexto que uma string era esperada
    - Implementação original retorna o nome da classe e o código *hash* do objeto
  - `boolean equals(Object outro)` testa se o objeto possui o mesmo estado que outro objeto

- Estes métodos são usualmente sobrescritos se forem utilizados em uma subclasse!

SOBRESCRITA DE MÉTODOS

- A classe Produto pode sobrescrever o método `toString()` de `Object`:

```
public String toString(){
   return super.toString()
          + "[nome=" + nome + ","
          + "preco=" + preco + "]";
}
```

Método herdado de Object

SOBRESCRITA DE MÉTODOS

- A classe ProdutoPerecivel pode sobrescrever o método `toString()` de Produto:

Método herdado de Produto

```
public String toString(){
   return super.toString()
    + "[validade=" + validade + "]";
}
```

CONTROLE DA HERANÇA

- Modificador `final`
  - Um método pode ser marcado como final para impedir que seja sobrescrito
    - `public final void meuMetodo(){...}`
  - Uma classe pode ser marcada como final para impedir que possa ser estendida com subclasses
    - `public final class MinhaClasse{...}`

HERANÇA E POLIMORFISMO

- “Polimorfismo é a característica de linguagens orientadas a objetos que permite que diferentes objetos respondam a mesma mensagem cada um a sua maneira.”

HERANÇA E POLIMORFISMO

- A linguagem Java permite a utilização de variáveis com polimorfismo
  - Uma mesma variável permite referência a objetos de tipos diferentes
  - Os tipos permitidos são de uma determinada classe e todas as suas subclasses

HERANÇA E POLIMORFISMO

- Exemplo:

```
Produto p1 = new ProdutoPerecivel("a",LocalDate.now());          correto
ProdutoPerecivel p2 = new Produto("a",1.9); erro compilação

Produto psuper;
ProdutoPerecivel psub;
ProdutoPerecivel p3 = new ProdutoPerecivel("a",LocalDate.now
psuper = p3;    correto
psub = psuper;  erro compilação
psub = (ProdutoPerecivel) psuper;   correto
```

HERANÇA E POLIMORFISMO

- Java possui o operador *instanceof* que permitir verificar a compatibilidade com um tipo de uma instância
  - Retorna *true* se a expressão da esquerda é um objeto que possui compatibilidade de atribuição com o tipo a sua direita
  - Retorna *false* caso contrário
- Ex.:

```
if (p1 instanceof Produto) {
   ...
}
```

HERANÇA E POLIMORFISMO

- Em Java podemos utilizar métodos com polimorfismo
  - Significa que um mesmo método pode ser definido em diversas classes, cada uma implementando o método de uma maneira própria
  - Utiliza como base a sobrescrita de métodos

HERANÇA E POLIMORFISMO

- Exemplo:
  - Qual a saída no console?

```
Produto p = new
  ProdutoPerecivel(“a”,LocalDate.now());
System.out.println(p);
```