# Construção de novas Classes em Java

Classes
Atributos
Métodos
Herança
Polimorfismo
Interfaces
Classe *Object*

Prof. Marcel Hugo
DSC/FURB

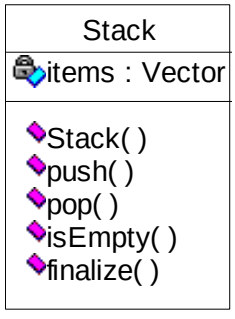


1

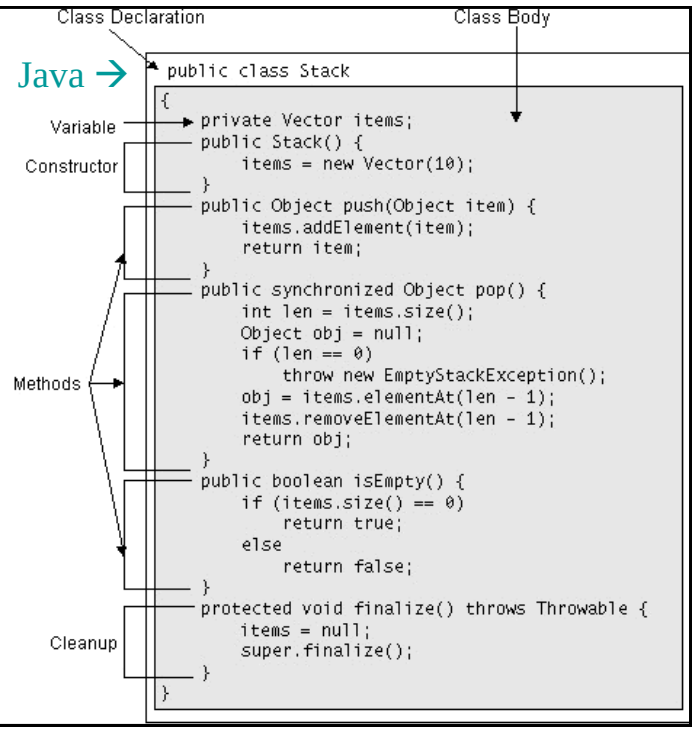
## Exemplo

Representação em UML

(Unified Modeling Language)




Java →




2



1

# Declaração de Classes

- Declaração

  [*modificadores*] **class** *NomeClasse*
  [**extends** *SuperClasse*]
  [**implements** *Interface*]
  {
  *atributos*
  *métodos*
  }

- Modificadores
  - Classe pública (**public**): a classe pode ser utilizada por objetos de fora do pacote. Por default, a classe só pode ser acessada no próprio pacote
  - Classe Abstrata (**abstract**): representa uma generalização e não pode ter objetos instanciados
  - Classe final (**final**): a classe não pode ter subclasses

3

3

# Atributos da Classe

- Declaração de Atributos

  [*acesso*] [*chaves*] *tipo nomeAtributo* [= *expressão*];

- Moderador de Acesso
  - **public:** o mundo inteiro pode acessar
  - **protected:** somente os métodos da classe e de suas subclasses podem acessar, ou ainda, estando na mesma *package*
  - **private:** somente os métodos da classe podem acessar o atributo
- Chaves
  - **static:** o atributo é da classe, não do objeto, logo, todos os objetos da classe compartilham o mesmo valor deste atributo
  - **final:** o valor do atributo não pode ser alterado (constante)
  - **transient:** o atributo não é serializado (não é persistente)

4

4

# Métodos da classe

- Declaração de Métodos

  [*acesso*] [chaves] *tipoRetorno nomeMétodo* ( [*parâmetros formais*] )
  [throws *exceptions*]
  { *corpo* }

  - chaves:
    - **static**: método da classe e não das instâncias
    - **abstract**: apenas a definição da mensagem. Utilizado somente em classes abstratas, o método não tem corpo.
    - **final**: o método não pode ser sobre-escrito
    - **synchronized**: declara o método como zona de exclusão mútua no caso de programas concorrentes
  - a passagem de parâmetros em Java sempre é por valor.
  - Um método é identificado pelo seu nome e pelos parâmetros
    - void x(int a) ≠ void x(float a)

5

5

# Sobrecarga de métodos (*overloading*)

| Pessoa |
| --- |
| -nome : String<br>-dataNascimento : Date |
| + getIdade( ) : int<br>+ getNome( ) : String<br>+ getNome(cxAlta:boolean) : String<br>+ getNome(prefix:String) : String |

```
Pessoa p = new Pessoa();
String n;

n = p.getNome();
n = p.getNome(true);
n = p.getNome("Sr.");
```

6

6

# Inicialização e finalização de objetos

- Construtor da Classe
  - Se um método da classe tem o mesmo nome que a classe este método é um construtor da classe, ou seja, um método que é chamado na criação do objeto (new).

```
class Teste {
  public Teste() { .... }
  public Teste(int i) { .....  }
  .....
}
```

- Destrutor da Classe (Finalizador)
  - Se a classe possuir um método como o nome **finalize**, este será chamado antes do objeto ser destruído.

```
protected void finalize() throws Throwable { ... }
```

7

7

# Exemplo criação de classe (1)

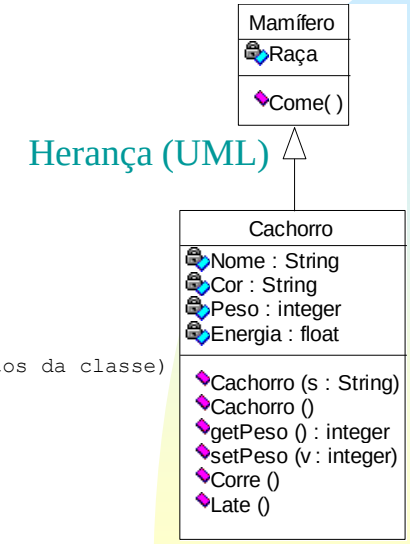
```
class Agent {
   private String nome;
   private int    valor;

   // Constructors
   Agent(String s) { nome = s; valor = 10; }
   Agent() { nome="Sem nome"; valor = 10; }

   // Servicos
   void setValor(int v) { valor = v; }

   void run(int inicial) {
         System.out.println("Agente " + nome + " rodando....");
         for (int i=inicial; i <= valor; i++)
                 System.out.println(i);
   }
   void run() {
         run(1);
}  }

class teste {
   public static void main(String[] args) {
         Agent t1 = new Agent("Exemplo Construção de classe");
         t1.setValor(3);
         t1.run();

         Agent t2 = new Agent();
         t2.run(5);
}  }
```

8

8

# Exemplo criação de classe (2)

```
class Cachorro extends Mamífero {

   // Atributos dos objetos da classe
   private String nome;
   private String cor;
   private int    peso;
   private float  energia;

   // Construtores (formas da classe)
   Cachorro(String s) { nome = s; }
   Cachorro() { nome = "Sem nome"; }

   // Métodos (comportamentos dos objetos da classe)
   void setPeso(int v) { peso = v; }
   int  getPeso() { return peso; }

   void corre() { ... }
   void late() { ... }
}
```




9

9

# Herança

- Novas classes podem ser criadas a partir de classes existentes herdando os atributos e métodos.
  - Ex.: Classe: Bicicleta
    SubClasses: Mountain Bike, Racing Bike, ....
    Se Bicicleta tem o método trocar roda, as subclasses herdam este método, sem necessidade de implementação.
- No exemplo, Bicicleta é **super classe** de Mountain Bike, Racing Bike, ... e Mountain Bike, Racing Bike são **subclasses**, ou especializações, de Bicicleta.
- Entretanto, um subclasse pode
  - acrescentar novos atributos e métodos
  - sobre-escrever métodos, especificando um comportamento mais específico

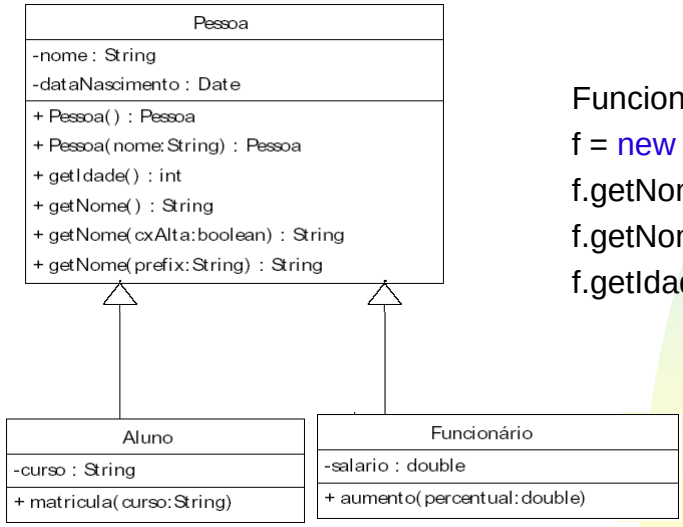
10

10



5

# Herança




```
Funcionário f;
f = new Funcionário();
f.getNome();
f.getNome(false);
f.getIdade();
```

11

11

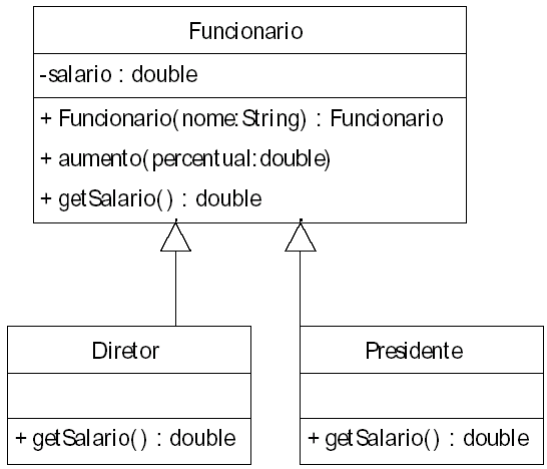
# Construtores em sub-classes

```
class Funcionário extends Pessoa {
    public Funcionário(String nome) {
        super(nome);
    }
}

...
Funcionário f1 = new Funcionário("marcos");
Funcionário f2 = new Funcionário(); // bip, bip, bip...
```

12

12

# Sobre-escrita (*overriding*)




```
Funcionario f;
f = new Presidente();
f.getSalario();

f = new Diretor();
f.getSalario();

f = new Funcionario();
f.getSalario();
```



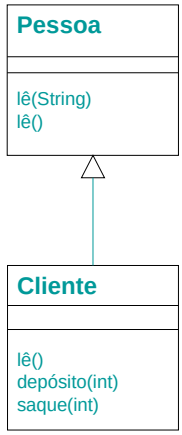


# Tipos, classes e upcast

```
Funcionario f;          // define o tipo de f
f = new Funcionario(); // atribui um valor a f
f = new Diretor();      // atribui outro valor a f , upcast
Pessoa p;               // define o tipo de p
p = f;                  // atribui um valor a p
p = new Diretor();      // atribui outro valor a f
f = p;                  // bip, bip, bip, ....
Diretor d = (Diretor)p; // é necessário um downcast
```

- Variáveis do tipo Pessoa podem receber valores de Pessoa, Funcionário, Diretor e Presidente.

# Exemplo de sobre-escrita, sobre-carga, upcast

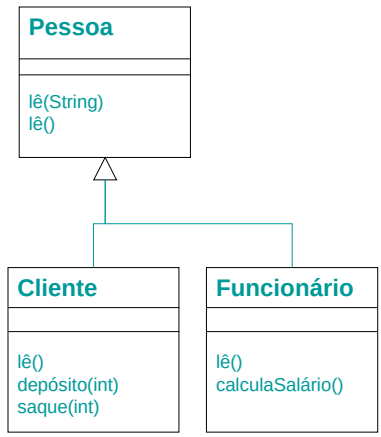



```
Cliente c = new Cliente();
c.lê();      // qual método é chamado?
c.lê("Digite:");

Pessoa p;
p = c;       // upcast

p.lê();      // qual método é chamado?
p.lê("Digite:");
```

15

15

# Exemplo de polimorfismo




```
Pessoa p;
if (opção == 1)
      p = new Cliente();
else
      p = new Funcionário();

p.lê();      // qual método é chamado?
p.lê("Digite:");
```

16

16

# Interfaces

- São “classes” com algumas restrições
  - são públicas e abstratas (não contém implementação dos métodos)
  - não tem atributos
  - são utilizadas para simular herança múltipla.
  - podem ser utilizadas na definição de referências para objetos, ou seja, é uma referência para um objeto que implementa tal interface.

- Exemplo
  - ArrayList e Vector implementam a interface List que tem métodos add, get, ...
  - Valem as atribuições:

```
ArrayList al = new ...
Vector    vt = new ...
...
List l;
l = al;
l.add("X")
...
l = vt;
l.add("Y")
```

17

17

# Exemplo de Interface

```
public interface AnimalVoa {
   void voa(int vel);
   int  envergadura();
}
class Mamifero extends SerVivo {
   String corPelo;
   ...
   int nroFilhotes() {
       ...
   }
}
class Morcego  extends Mamifero
               implements AnimalVoa {
   void voa(int vel) {
       ....
   }
   int envergadura() {
       ....
   }
   ...
}
```

18

18

# *Object* como superclasse

- Em Java há a classe ***Object***, da qual toda classe descende.
- Desta forma, qualquer objeto em Java pode ser tratado como Object e responde aos métodos desta classe:
  - •protected Object clone() throws CloneNotSupportedException
    Cria e retorna uma cópia deste objeto.
  - •public boolean equals(Object obj)
    Indica se algum outro objeto é “igual a" este objeto.
  - •protected void finalize() throws Throwable
    Destrutor
  - •public final Class getClass()
    Retorna a classe de criação do objeto.
  - •public int hashCode()
    Retorna o valor de *hash code* para o objeto.
  - •public String toString()
    Retorna uma representação em String deste objeto.
- Ainda há os métodos *notify, notifyAll* e *wait* – ligados a *threads*.
- Todos estes métodos podem (e devem) ser sobrescritos.

19

19



10