Programação Orientada a Objetos

Aula 07 – Sobrecarga de Métodos e Herança

Prof. Luiz Mário Lustosa Pascoal

Herança, sobrecarga e ligação dinâmica

- Sobrecarga de métodos
- Herança
- Referência super
- Sobreposição
- Ligação dinâmica de métodos
- final

Sobrecarga de métodos(overloading)

- A sobrecarga de métodos é usada para criar vários métodos com o mesmo nome que realizam tarefas semelhantes, mas sobre tipo de dados diferentes

- Métodos sobrecarregados são distinguidos por sua assinatura (nome do método + número de parâmetros + tipo dos parâmetros)

- Diferentes tipos de retorno dos métodos não podem ser considerados para sobrecarga

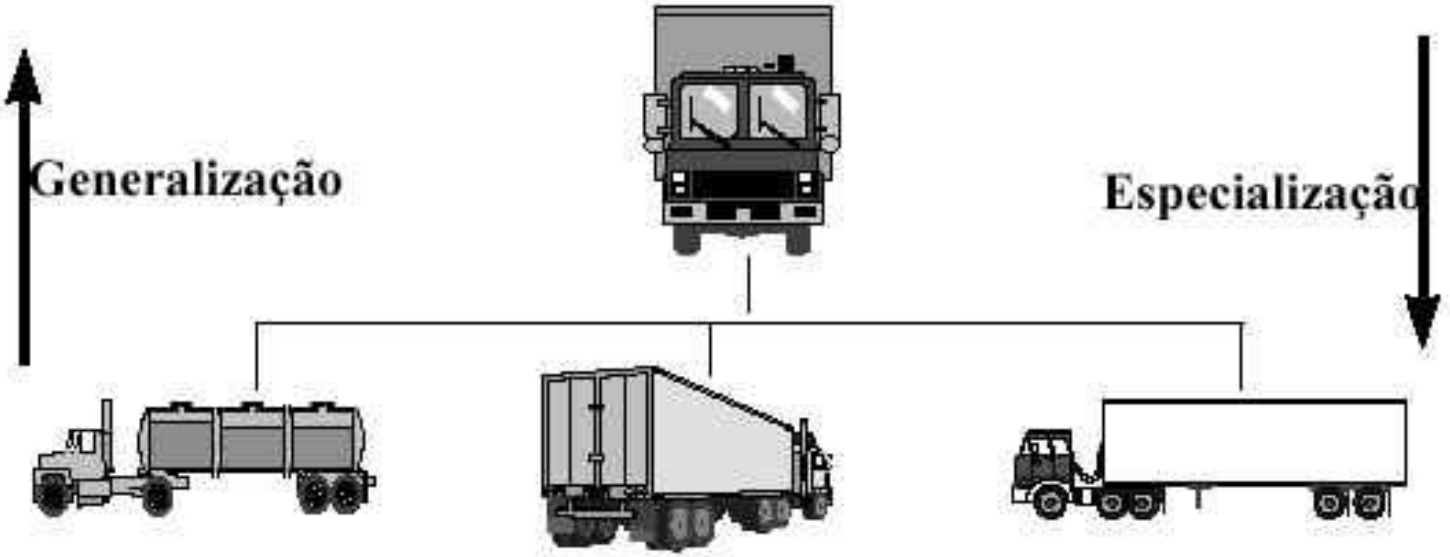
Sobrecarga de métodos(overloading)

```
public class Ponto {
 int x, y;

 Ponto(int x, int y){
  this.x = x;
  this.y = y;
 }

 Ponto( ){
  x = -1;
  y = -1;
 }

}
```

```
public class CriaPonto {
 public static void main(String args[ ]) {
  Ponto p = new Ponto( );
  Ponto q = new Ponto(2,3);
  System.out.println("x= "+ p.x + "y = "+ p.y);
  System.out.println("x=" + q.x + "y = "+ q.y);
 }
}
```

Generalização / Especialização

Generalização é o relacionamento entre uma classe e uma ou mais **versões refinadas** dessa classe



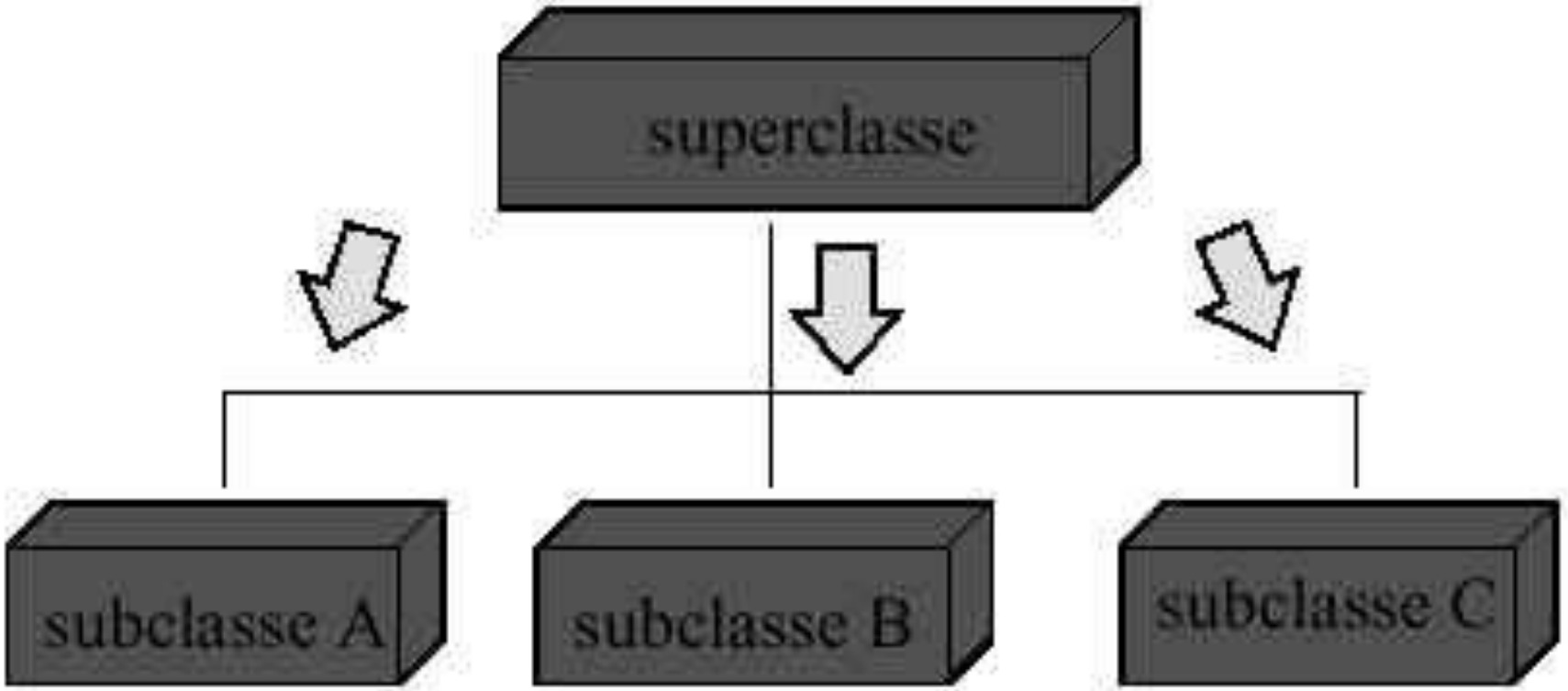
Generalização é a abstração que permite **compartilhar** semelhanças entre classes, preservando suas diferenças

34

Herança

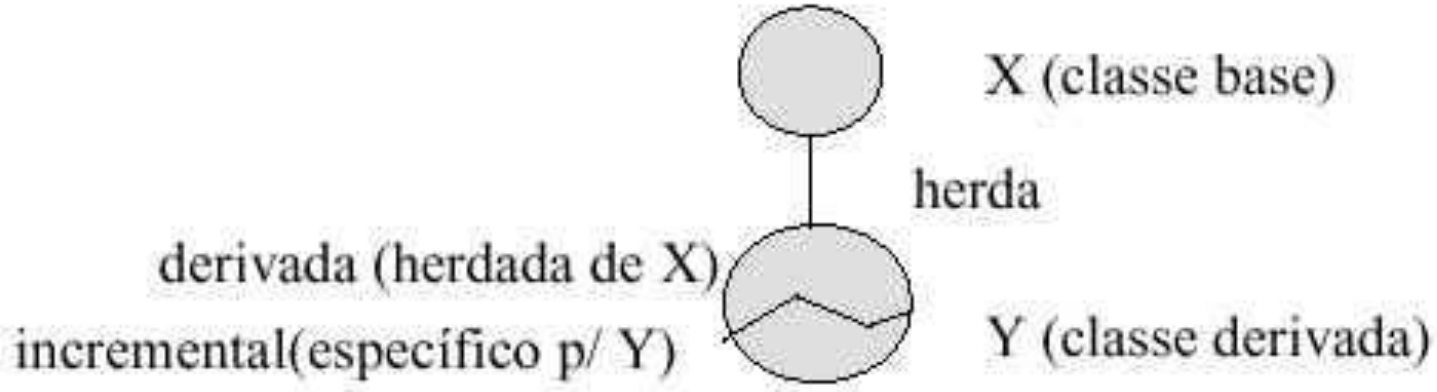
- Herança é um conceito que mapeia as classes relacionadas entre si de maneira hierárquica
- Os descendentes de uma classe herdem todas as suas variáveis e métodos de seus ancentrais bem como criem seus próprios
- Os descendentes são subclasses e o seu ascendente imediato é chamado de sua superclasse

Hierarquia de Classes



Hierarquia de Classes

- Uma classe derivada herda as propriedades e métodos da classe, podendo:
  - Adicionar novos métodos
  - Adicionar novos atributos
  - Redefinir a implementação de métodos existentes (override)



Herança e Modificadores de Acesso

Métodos e Atributos da Classe	Implementação da Classe	Instância da Classe	SubClasse da Classe	Instância da SubClasse
privados (*private*)	sim	não	não	não
protegidos (*protected*)	sim	não	sim	não
pacote (*package*)	sim	sim (no mesmo pacote)	sim (no mesmo pacote)	sim (no mesmo pacote)
públicos (*public*)	sim	sim	sim	sim

Herança

- Em Java a palavra-chave ***extends*** é usada como mecanismo para definição de herança e subtipos
- Java oferece suporte somente a herança simples de classes

- **Restrições**
  - Atributos e métodos privados são herdados, mas não podem ser acessados
  - Construtores não são herdados
  - O construtor padrão somente é disponível na subclasse se estiver presente na superclasse

Herança

```
public class Ponto {
 int x, y;

 Ponto(int x, int y){
  this.x = x;
  this.y = y;
 }

 Ponto( ){
  x = -1;
  y = -1;
 }

}
```

```
public class Ponto3D extends Ponto {
 int z;

 Ponto3D(int x, int y, int z){
  this.x = x;
  this.y = y;
  this.z = z;
 }

 Ponto3D( ){
  x = -1;
  y = -1;
  z = -1;
 }

}
```

super

- Quando um método da subclasse redefinir um método da superclasse, pode-se acessar o método da superclasse através da palavra-chave super seguida de um ponto(.)
- Por Exemplo:

```
super.codTurma;
```

- Somente a palavra super pode ser utilizada para ativar o construtor da superclasse
- Por Exemplo:

```
super(); //se o construtor da superclase não possuir parâmetros
```

super

```
public class Ponto3D extends Ponto {
 int z;

 Ponto3D(int x, int y, int z){
  super(x, y); //chama o construtor Ponto(x, y).
  this.z = z;
 }

 Ponto3D( ){
  x = -1;
  y = -1;
  z = -1;
 }

}
```

Sobreposição

- Uma subclasse pode redefinir um método da superclasse utililizando a mesma assinatura isto é chamado de anular ou sobrescrever *(override)* um método da supeclasse

- Quando a redefinição do método da superclasse não possuir a mesma assinatura na subclasse, isto não é anulação, mas um exemplo de sobrecarga de método

Sobreposição

Superclasse

```
class Ponto{
 int x, y;
 Ponto(int x, int y){
  this.x=x;
  this.y=y;
 }
 double distancia(int x, int y){
   double dx = this.x - x;
   double dy = this.y -y;
   return Math.sqrt(dx*dx + dy*dy);
 }
}
```

Subclasse

```
class Ponto3D extends Ponto{
 int z;
 Ponto3D(int x, int y, int z){
  super(x, y);
  this.z=z;
 }
 double distancia(int x, int y){
   double dx = (this.x/z) - x;
   double dy = (this.y/z) -y;
   return Math.sqrt(dx*dx + dy*dy);
 }
}
```

Ligação dinâmica

```
public class A {
 void chameme( ) {
   System.out.println(“Dentro do método chameme de A”);
 }
}
public class B extends A{
 void chameme( ) {
   System.out.println(“Dentro do método chameme de B”);
 }
}
public class DespachoDinamico{
 public static void main (String args[ ]){
  A obj1 = new B( );
  obj1.chameme( );
 }
}
```

final

- Todos os métodos e variáveis de instância herados podem ser sobrepostos

- Para não permitir que as subclasses sobreponham suas variáveis ou seus métodos, pode declará-las como *final*

- Por exemplo:

```
final int ARQNOVO = 1;
public final void imprime( );
```

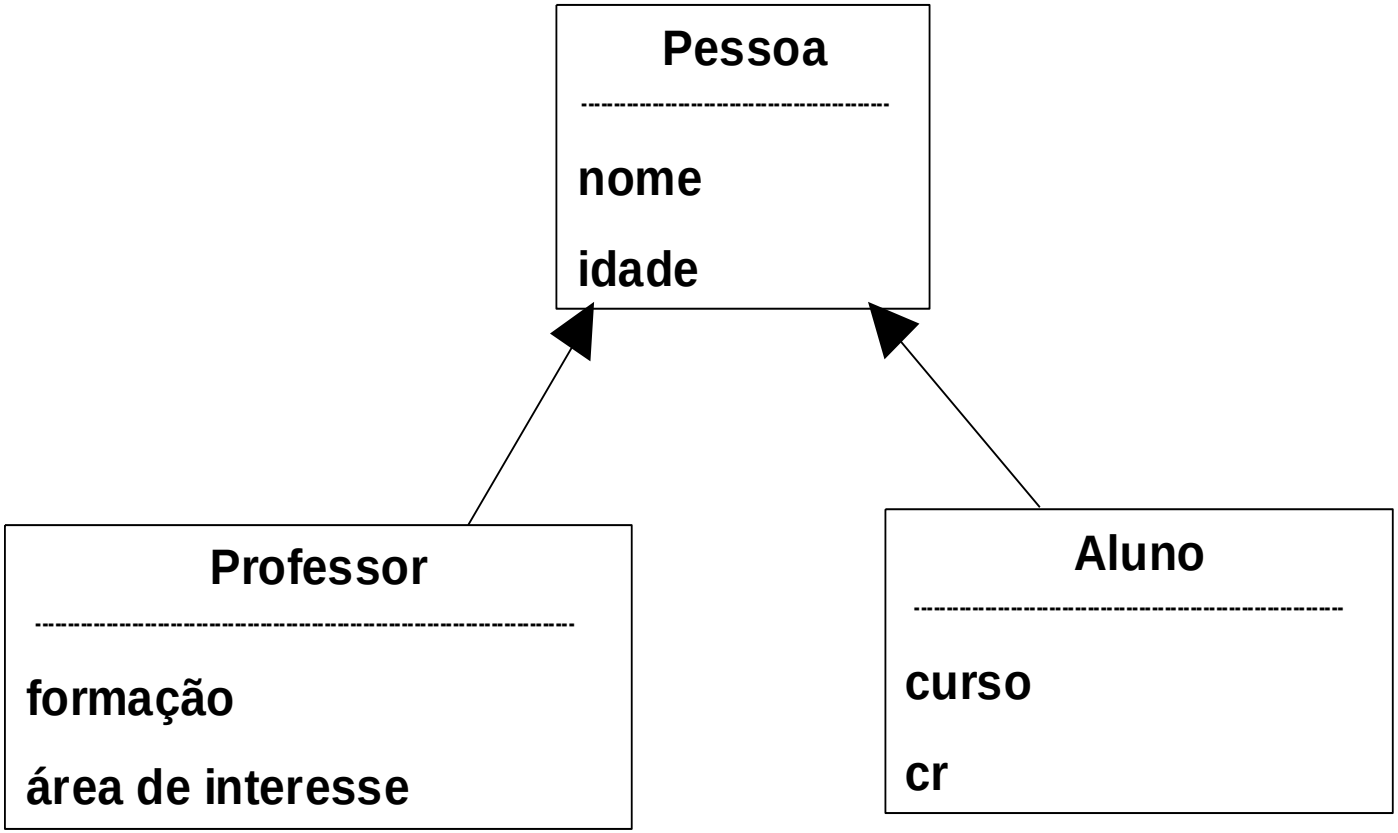
Resumindo Herança…

- Permite a uma classe **herdar o estado (atributos) e o comportamento (métodos) de outra classe.**

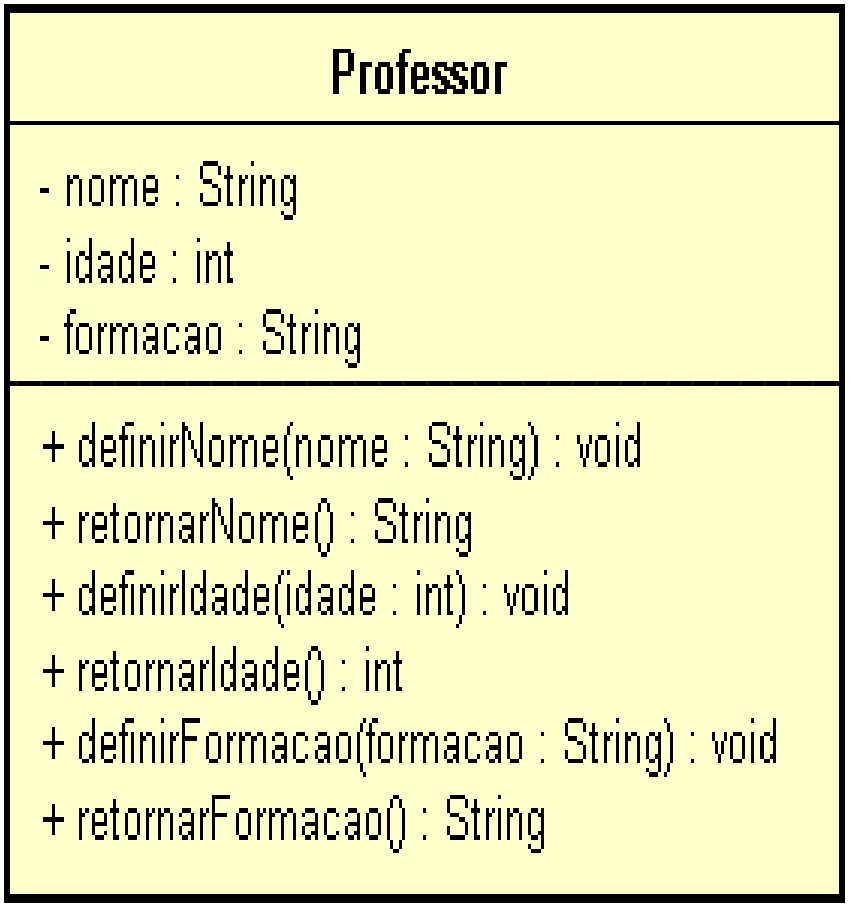
- **Herança** : entre diferentes classes podem existir diversas semelhanças, ou seja, duas ou mais classes poderão compartilhar os mesmos atributos e/ou os mesmos métodos

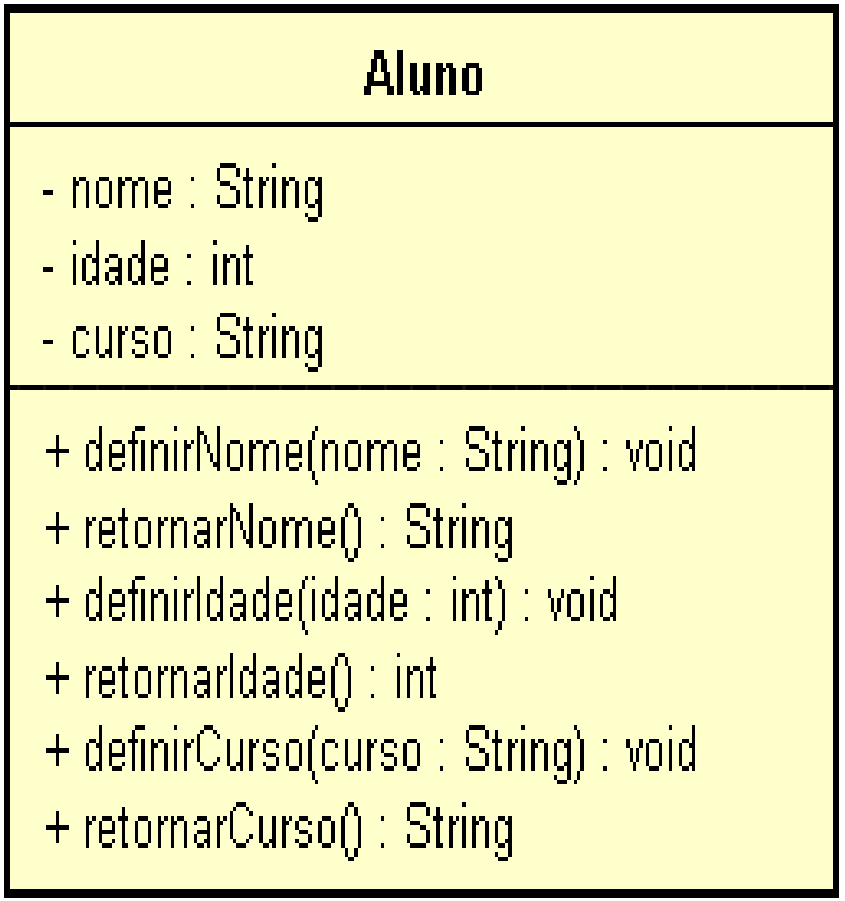
  - Superclasse (Ancestral)
  - Subclasse (Descendente)

Resumindo Herança… (Representação)

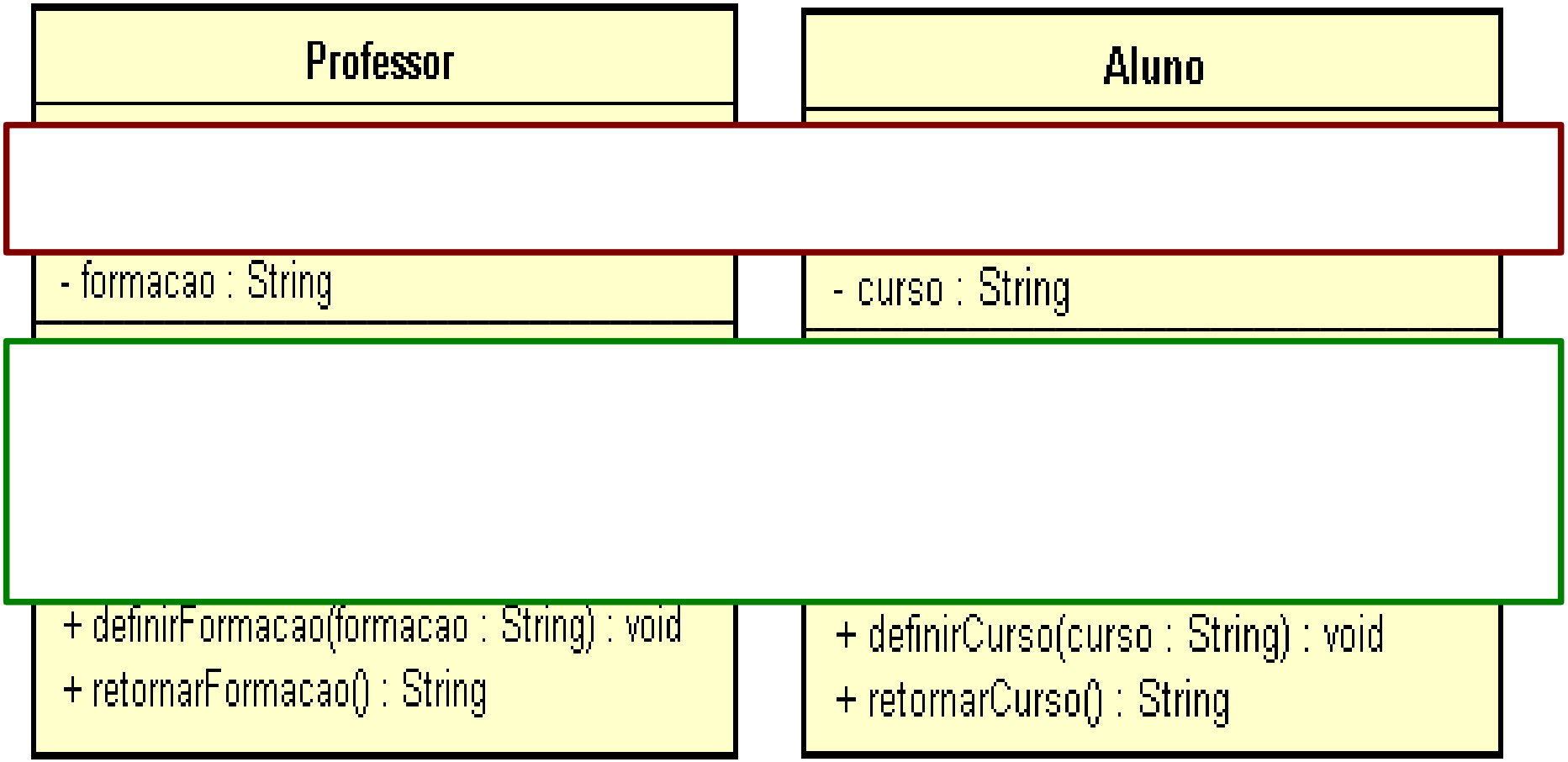


Resumindo Herança… (Representação UML)

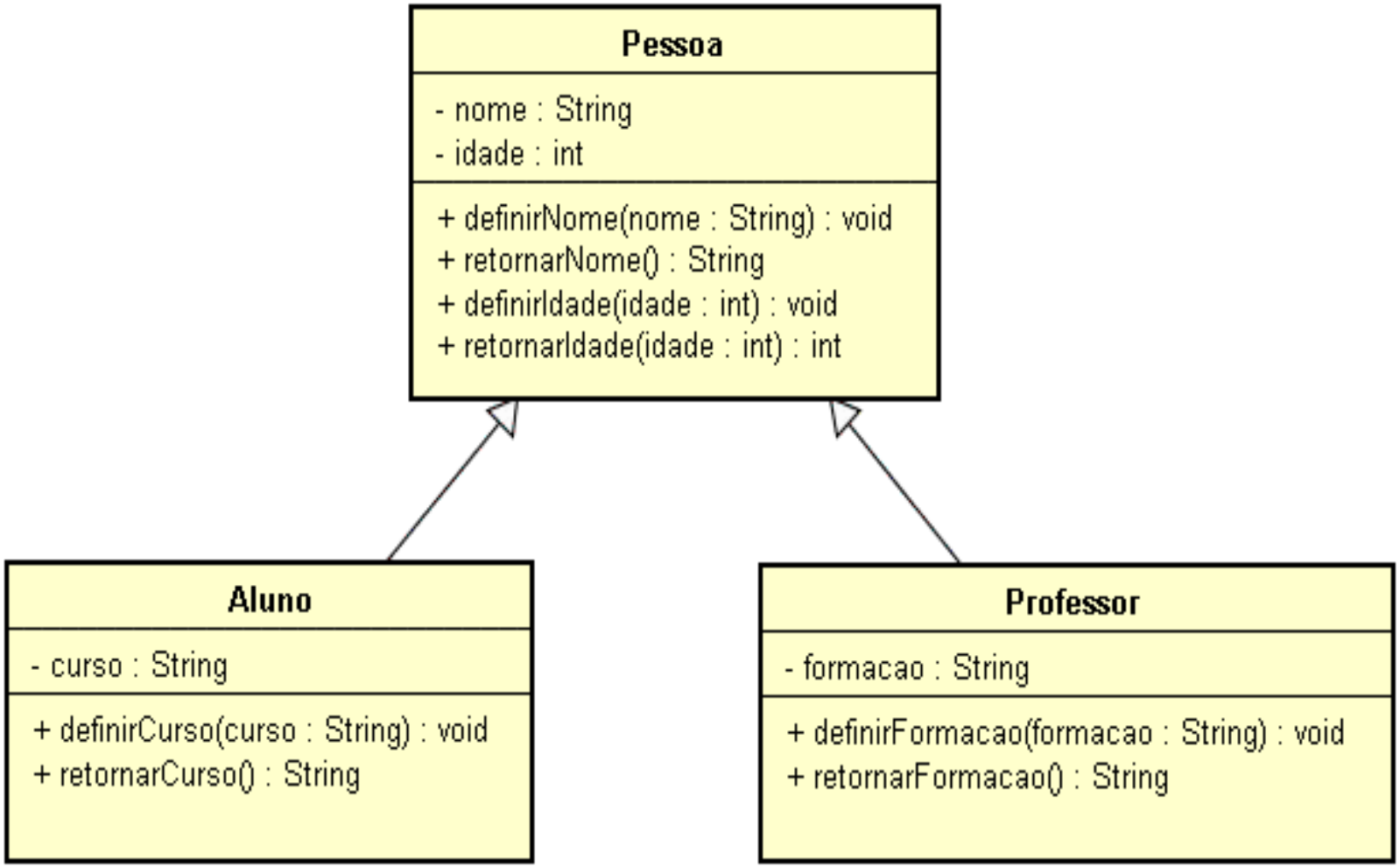




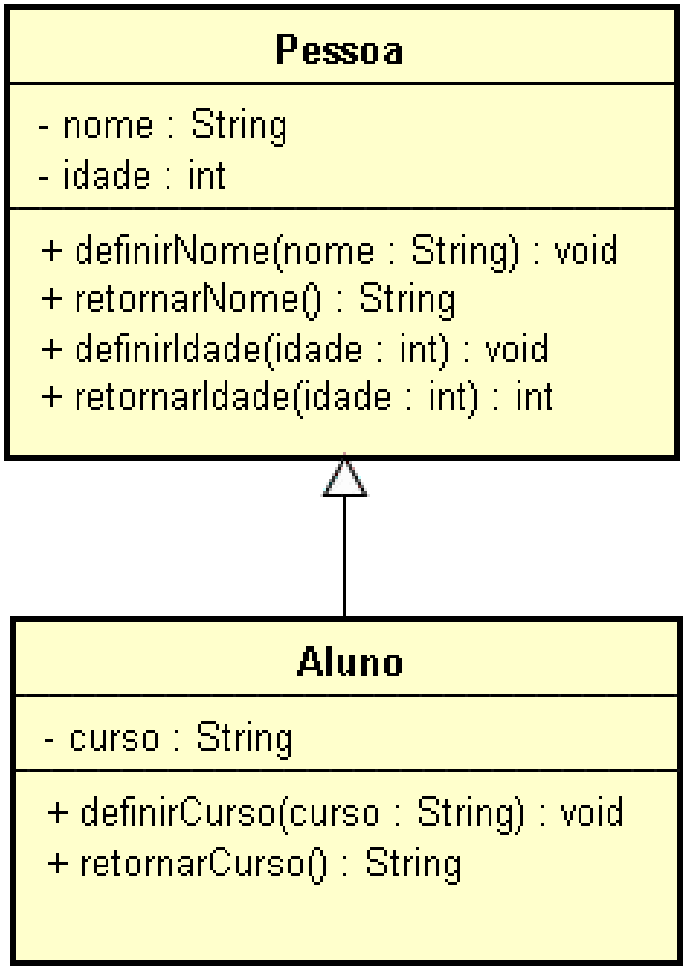
Resumindo Herança… (Representação UML)



Resumindo Herança… (Representação UML)



Resumindo Herança… (Representação UML)



Instâncias de Aluno

João
25
Sistemas de Informação

Maria
20
Sistemas de Informação

Resumindo Herança… (Aparência em código)

```
// SuperClass.java
        public class SuperClass
        {
           // Atributos e métodos
        }
//  SubClass.java
        public class SubClass EXTENDS SuperClass
        {
           // Atributos e métodos
        }
```

```
class Aluno extends Pessoa {
...
}
```

Resumindo Herança… (Aparência em código)

```
class Pessoa {
    String          nome;
    int             idade;

    void definirNome(String valor) {
        nome = valor;
    }

    String retornarNome() {
        return nome;
    }

    void definirIdade(int valor) {
        idade = valor;
    }

    int retornarIdade() {
        return idade;
    }
}
```

```
class Aluno extends Pessoa {
    String    curso;

    void definirCurso(String valor) {
        curso = valor;
    }

    String retornarCurso() {
        return curso;
    }

}
```

Resumindo Herança… (Aparência em código)

```
Aluno joao = new Aluno();
joao.definirNome("João");
joao.definirIdade(25);
joao.definirCurso("Sistemas de Informação");
```

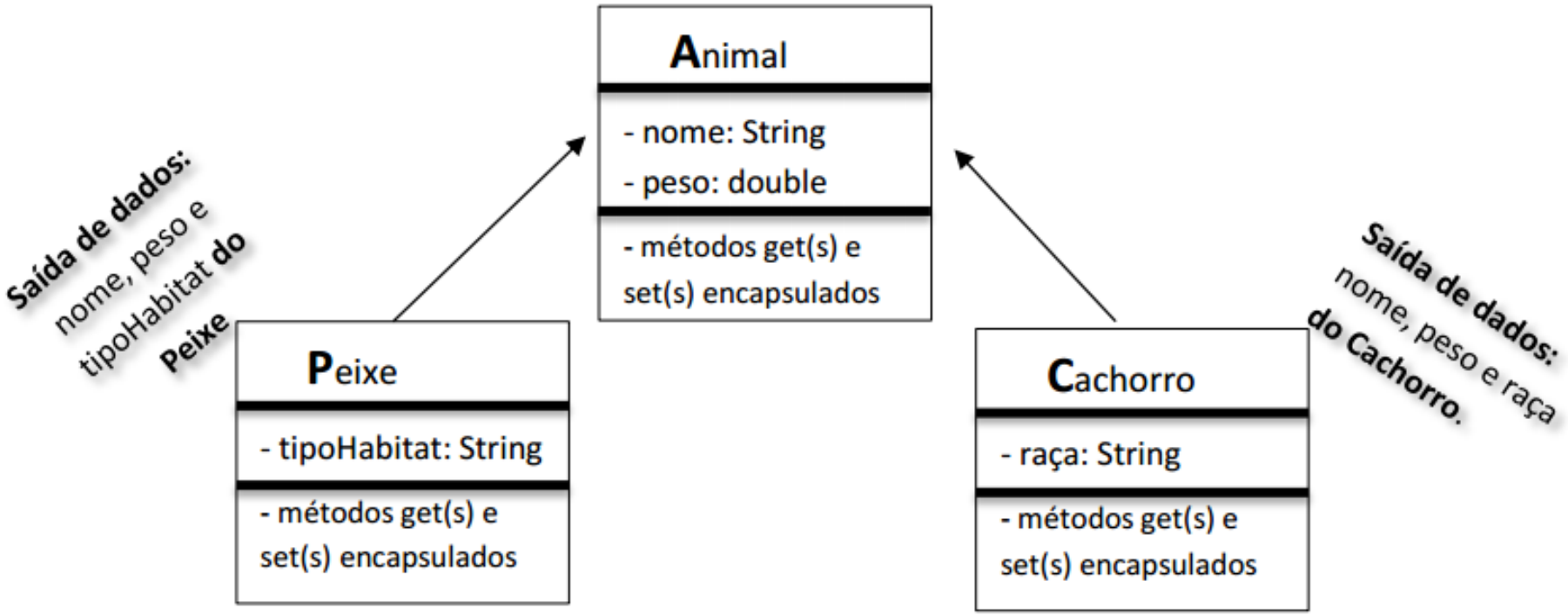
```
João
25
Sistemas de Informação
```

```
Aluno maria = new Aluno();
maria.definirNome("Maria");
maria.definirIdade(20);
maria.definirCurso("Sistemas de Informação");
```

```
Maria
20
Sistemas de Informação
```

Exercício 1.

a. Crie as classes solicitadas.
b. Faça o relacionamento (herança) entre as classes.
c. Defina a saída dos dados (toString()) nas classes indicadas. A classe onde tem a indicação é onde estará a saída – toString().
d. Faça a classe de teste e execute.



* tipoHabitat = agua doce, salgada, ornamental, etc.