UNIP
UNIVERSIDADE PAULISTA

Linguagem de Programação Orientada a Objeto

Herança

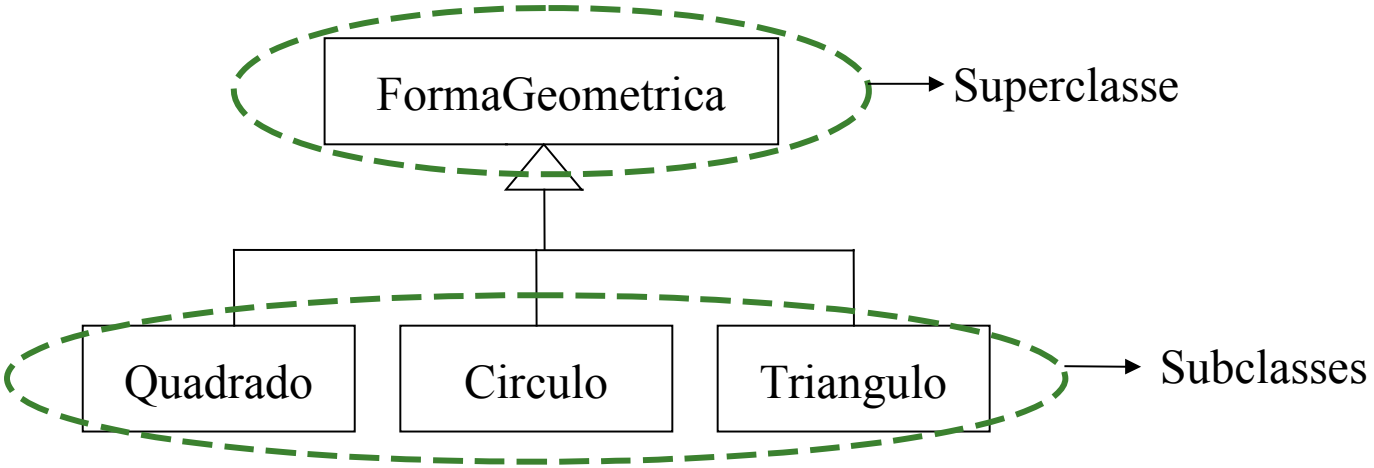
Herança

- Mecanismo simples e poderoso do paradigma OO que permite que uma nova classe seja descrita a partir de uma classe já existente.
- Herança é um mecanismo que permite a uma classe herdar todo o comportamento e os atributos de outra classe.
- Pode-se adicionar novas operações, estender a representação dos dados ou redefinir a implementação de operações existentes.

Conceitos da Herança

- **Classe Base ou Superclasse ou Classe Pai ou Classe mãe:** é uma classe a partir da qual classes novas podem ser derivadas (classe mais geral).
- **Classe Derivada ou Subclasse ou Classe Filha:** é uma classe que herda os atributos e métodos da classe base (classe mais específica). Possui atributos e métodos próprios

Herança Exemplo

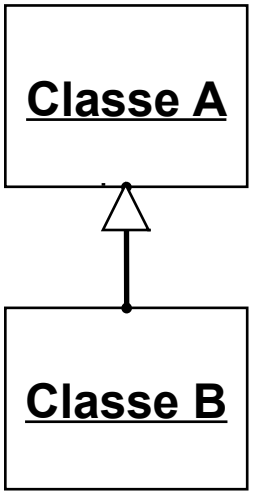


Herança - Exemplos

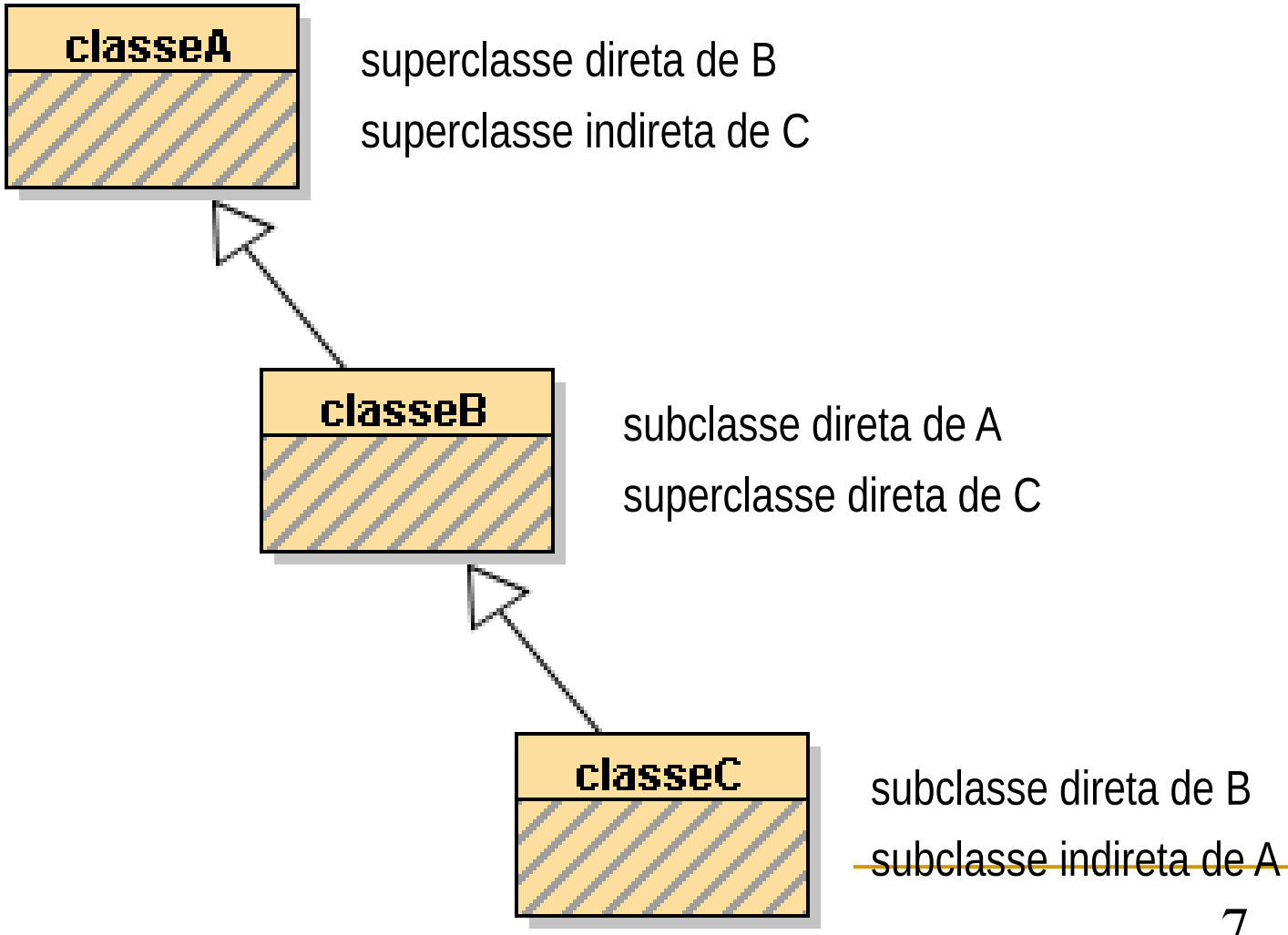
Superclasse	**Subclasses**
Aluno	AlunoDeGraduacao, AlunoDePosGraduacao
Financiamento	FinanciamentoDeCasa, FinanciamentoDeCarro
Empregado	Engenheiro, Contador, Gerente
ContaBancaria	ContaCorrente, ContaPoupanca

Mecanismo de Herança

- **B** herda de **A** todas as variáveis e métodos (public e protected)
- **B** pode definir novas variáveis e novos métodos próprios.
- **B** pode redefinir variáveis e métodos **herdados**.



Herança



superclasse direta de B

superclasse indireta de C

subclasse direta de A

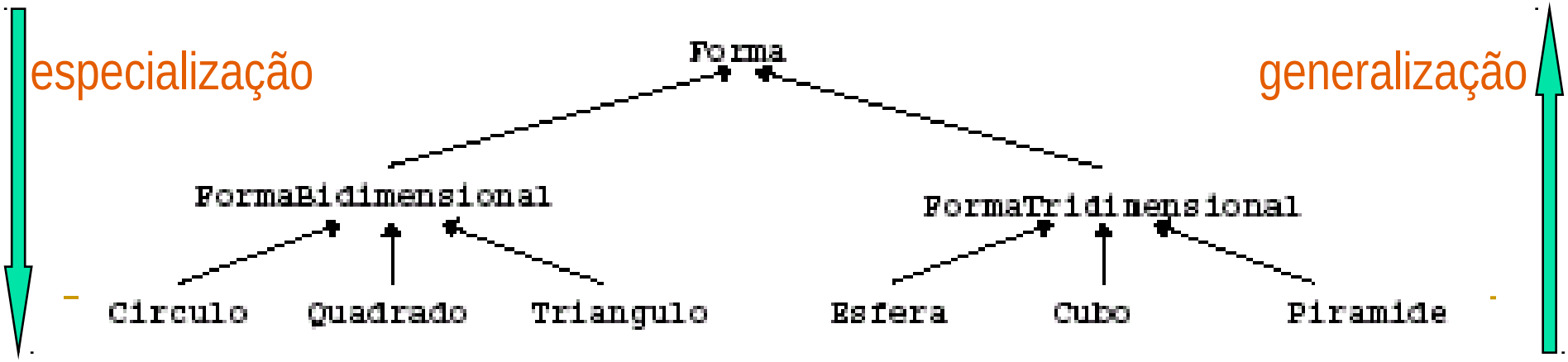
superclasse direta de C

subclasse direta de B

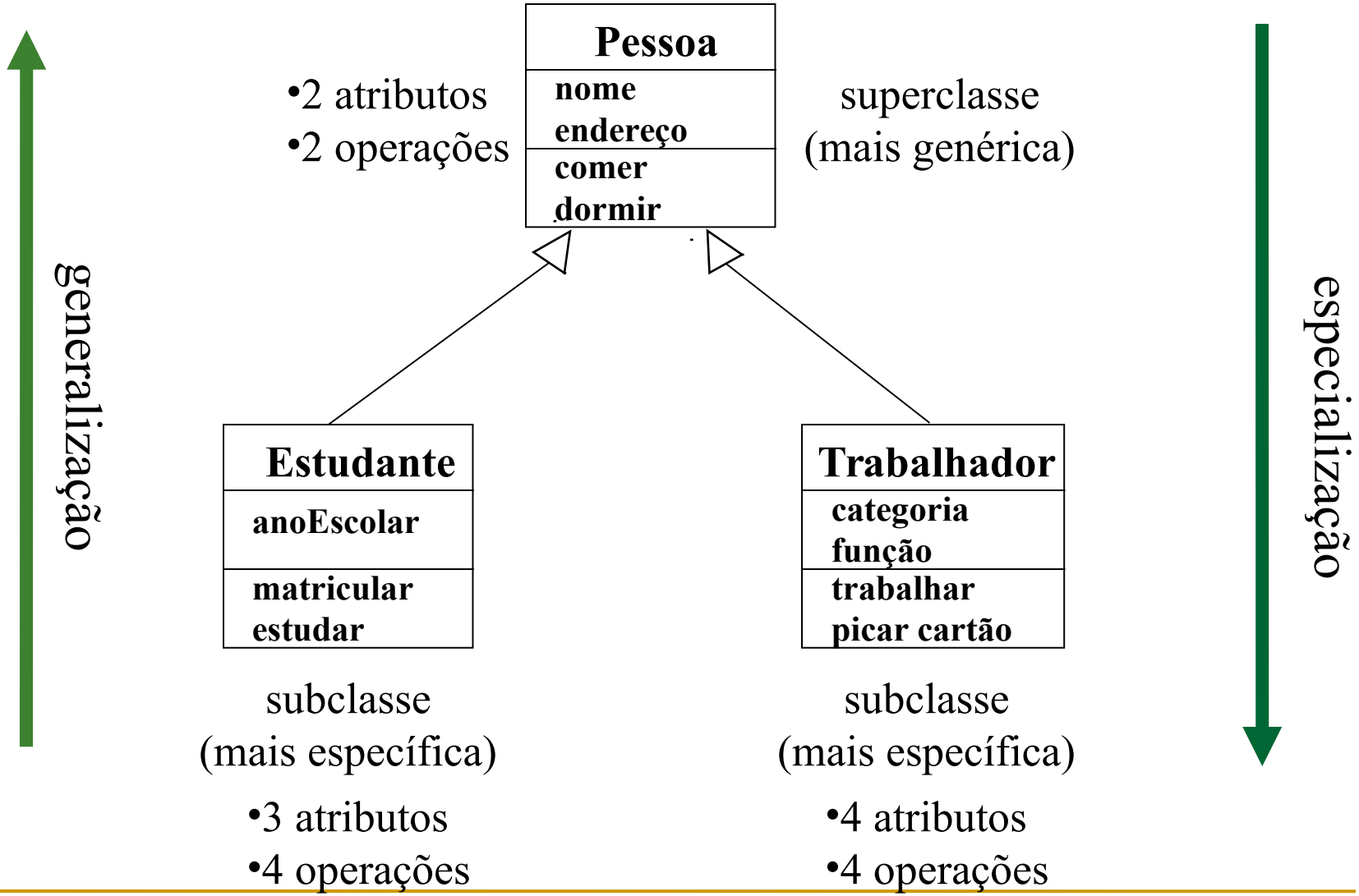
subclasse indireta de A

Herança

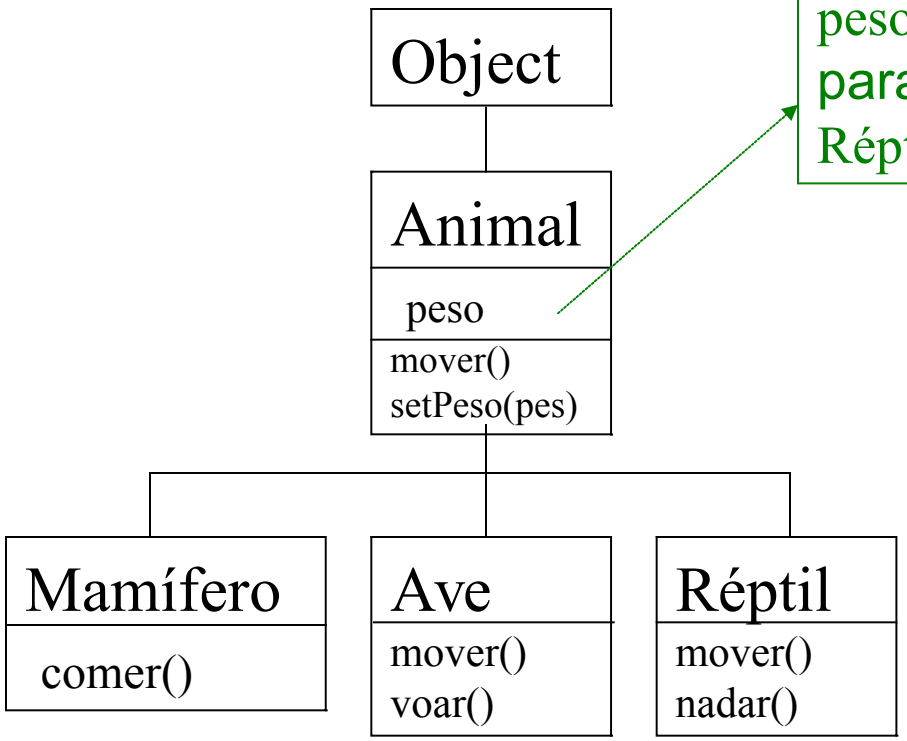
- Herança cria uma estrutura hierárquica (árvore)
  - Exemplo
    - Hierarquia de classes para formas geométricas
    - Uma forma geométrica pode ser especializada em dois tipos: bidimensional e tridimensional.



Exemplo 1


Pessoa
nome
endereço
comer
dormir
•2 atributos
•2 operações
superclasse
(mais genérica)
generalização
especialização
Estudante
anoEscolar
matricular
estudar
Trabalhador
categoria
função
trabalhar
picar cartão
subclasse
(mais específica)
•3 atributos
•4 operações
subclasse
(mais específica)
•4 atributos
•4 operações

Herança ⇒ Hierarquia de Classes



peso é uma variáveis objectos criadas para as classes Animal, Mamífero, Ave e Réptil.

```
Ave pardal = new Ave();
pardal.setPeso(700);

Mamifero boi = new Mamifero();
boi.setPeso(30000);
boi.mover();
boi.voar(); //Erro! não existe
```

Herança em Java

- **Forma geral**:

```
public class <nome-da-subclasse> extends <superClass>{
}
```

- é permitido apenas uma superclasse
  - **não há herança múltipla em Java**
- cada classe possui exatamente uma superclasse
  - **excepção: java.lang.Object**
- caso não exista a cláusula **`extends`**, então, assume-se que a superclasse é **Object**.
- A classe Object é uma classe que serve de superclasse para todas as classes existentes em Java.

Exemplo de Herança

```
class Animal extends Object
{  int peso;
   void mover()
   {  /* movimentação do animal */ }
}

class Mamifero extends Animal
{  void comer()
}

class Ave extends Animal
{  void mover()  {…}
   void voar()  {…}
}
```

Sobreposição de método!

Construtores de Subclasses

```
class Animal
{
  double velocidade;

  Animal()
  {
   velocidade = 0.0;
  }

  void mover(double vel)
  {
   velocidade = vel;
  }
}
```

```
class Ave extends Animal {
  int altura;

 Ave()
 {
    super(); // velocidade = 0.0;
    altura = 0;
 }

void mover (double vel)
 {
  if(altura >0) velocidade = vel;
 }

void voar (int alt)
  {
     altura = alt;  }
}
```

- super() deve ser o primeiro comando do construtor da subclasse

O Construtor Padrão

```
public NomeClasse() { super(); }
```

- As chamadas aos construtores são encadeadas
  - sempre que um **objecto** for criado, uma sequência de métodos construtores serão invocados, da subclasse para a **superclasse**, e assim sucessivamente até atingir a classe `Object`.
- Métodos construtores NUNCA são herdados mas podem ser chamados para inicializar os atributos herdados (membros da superclasse), utilizando-se `super()`

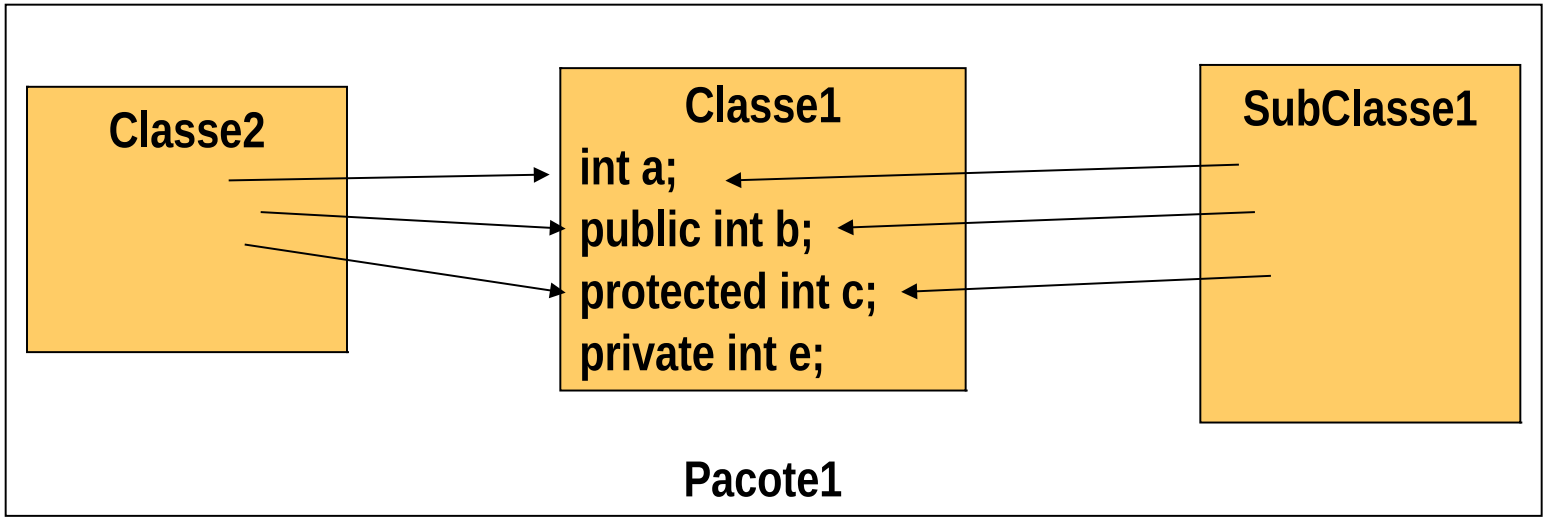
Modificadores de acesso

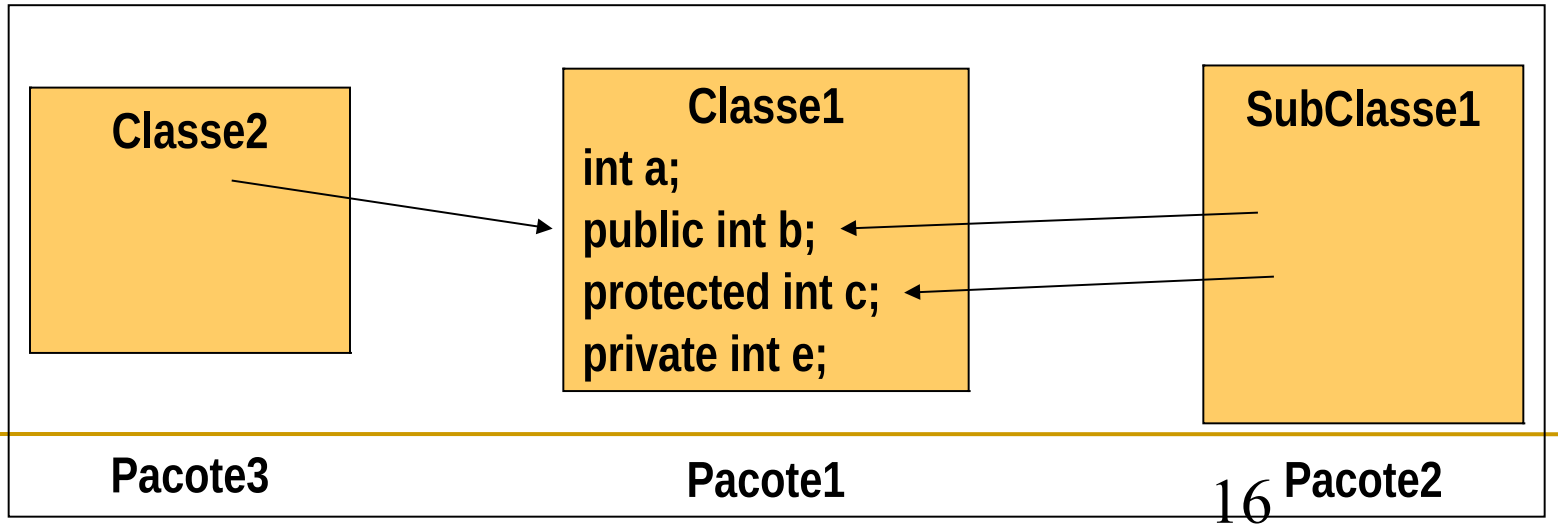
- Nem todos os membros da superclasse são acessíveis na classe derivada. Dependerá dos modificadores de acesso.

Modificador	Acesso permitido
`sem atributo`	de todas as classes no mesmo pacote
`public`	de qualquer classe em qualquer pacote
`private`	sem acesso de fora da classe
`protected`	de todas as classes no mesmo pacote e a partir de qualquer subclasse em qualquer pacote

- Note que: Atributos **private** da superclasse NÃO estão acessíveis diretamente na subclasse! Para acessá-los diretamente na subclasse se pode utilizar o modificador de acesso **protected.**
- **ATENÇÃO:** Para derivar uma classe de fora do pacote contendo a superclasse, a superclasse deve ser declarada como `public`

Modificadores de Acesso





Modificadores de Acesso

- Recomendações
  - Métodos que fazem a interface externa de uma classe devem ser declarados como `public` (sendo portanto herdados pelas subclasses)
  - Atributos devem ser `private`
  - Utiliza-se o atributo `protected` quando classes são definidas dentro de um pacote e se deseja dar ao usuário do pacote (desenvolvedor de classes em outro pacote) acesso apenas às subclasses
  - Classes, em geral, são declaradas como `public`

Herança Exemplo

```
public class Funcionario {
  private String nome;
  private int cpf;

  public Funcionario(String n, int c) {
    nome = n;
    cpf = c;
  }

  public String getNome() {
    return nome;
  }

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

Herança

```
public class Motorista extends Funcionario {
  private int numCarteira;
  private String dataExpiracao;
  public int getNumCarteira() {
    return numCarteira;
  }
  public String getDataExpiracao() {
    return dataExpiracao;
  }
  public Motorista(String n, int c, int nc, String d) {
    nome = n;
    cpf = c;
    numCarteira = nc;
    dataExpiracao = d;
  }
}
```

Error! Não podem ser acessados diretamente, pois são *private*!

Usar:

```
public Motorista(String n, int c, int nc, String d) {
    super(n,c);
    numCarteira = nc;
    dataExpiracao = d;
  }
```

Vantagens da Herança

- Modificação de uma classe (inserção de novos métodos e variáveis) sem mudanças na classe original.
- Reutilização do código.
- Alteração do comportamento de uma classe.
- Não é necessário reinventar a roda a cada nova aplicação, podemos herdar de classes prontas para estender funcionalidades.

Bibliografia

Vários ex xemplos foram adaptados e/ou extraidos das fontes a seguir:

- Escola superior de tecnologia setubal, Algoritmos e tipos abstractos de informação 2005/2006
- Slides da profa. Isabel Harb Manssour, d009.