

# 10 Classes internas

# Classes internas

- *Classes podem ser membros de classes, de objetos ou locais a métodos. Podem até serem criadas sem nome, apenas com corpo no momento em que instanciam um objeto*
- *Há poucas situações onde classes internas podem ou devem ser usadas. Devido à complexidade do código que as utiliza, deve-se evitar usos não convencionais*
- *Usos típicos incluem tratamento de eventos em GUIs, criação de threads, manipulação de coleções e soquetes*
- *Classes internas podem ser classificadas em quatro tipos*
  - *Classes dentro de instruções (classes anônimas)*
  - *Classes dentro de métodos (classes locais)*
  - *Classes dentro de objetos (membros de instância)*
  - *Classes internas estáticas (membros de classe)*

# Tipos de classes internas

- São sempre classes dentro de classes. Exemplo:

```
class Externa {  
    private class Interna {  
        public int campo;  
        public void metodoInterno() {...}  
    }  
    public void metodoExterno() {...}  
}
```

- Podem ser *private*, *protected*, *public* ou *package-private*
  - Exceto as que aparecem dentro de métodos, que são locais
- Podem ser estáticas:
  - E chamadas usando a notação `Externa.Interna`
- Podem ser de instância, e depender da existência de objetos:
  - `Externa e = new Externa();`  
`Externa.Interna ei = e.new Externa.Interna();`
- Podem ser locais (dentro de métodos)
  - E nas suas instruções podem não ter nome (anônimas)

# Classes estáticas (internal classes)

- Declaradas como *static*
  - Idênticas às classes externas, mas não têm campos *static*
  - Classe externa age como um **pacote** para várias classes internas estáticas: `Externa.Coisa`, `Externa.InternaUm`
  - Compilador gera arquivo `Externa$InternaUm.class`

```
class Externa {
    private static class InternaUm {
        public int campo;
        public void metodoInterno() {...}
    }
    public static class InternaDois
        extends InternaUm {
        public int campo2;
        public void metodoInterno() {...}
    }
    public static interface Coisa {
        void existe();
    }
    public void metodoExterno() {...}
}
```

# Classes de instância (embedded classes)

- São membros do **objeto**, como métodos e campos de dados
- Requerem que objeto **exista** antes que possam ser usadas.
  - Externamente use **referencia.new** para criar objetos
- Variáveis de mesmo nome sempre se referem à classe externa
  - Use **NomeDaClasse.this** para acessar campos internos

```
class Externa {
    public int campoUm;
    private class Interna {
        public int campoUm;
        public int campoDois;
        public void metodoInterno() {
            this.campoUm = 10; // Externa.campoUm
            Interna.this.campoUm = 15;
        }
    }
    public static void main(String[] args) {
        Interna e = (new Externa()).new Interna();
    }
}
```

# Classes dentro de métodos (embedded)

- Servem para tarefas "descartáveis" já que deixam de existir quando o método acaba
  - Têm o escopo de **variáveis locais**. Objetos criados, porém, podem persistir além do escopo do método, se retornados
  - Se usa variáveis locais do método essas variáveis **devem ser constantes** (declaradas final), pois assim podem persistir após a conclusão do método.

```
public Multiplicavel calcular(final int a, final int b) {
    class Interna implements Multiplicavel {
        public int produto() {
            return a * b; // usa a e b, que são constantes
        }
    }
    return new Interna();
}
public static void main(String[] args){
    Multiplicavel mul = (new Externa()).calcular(3,4);
    int prod = mul.produto();
}
```

# Classes anônimas (dentro de instruções)

- *Classes usadas dentro de métodos freqüentemente servem apenas para criar um objeto uma única vez*
  - *A classe abaixo estende ou implementa SuperClasse, que pode ser uma interface ou classe abstrata (o new, neste caso, indica a criação da classe entre chaves, não da SuperClasse)*  
Object i = new SuperClasse() { implementação };
  - *Compilador gera arquivo Externa\$1.class, Externa\$2.class,*

```
public Multiplicavel calcular(final int a, final int b) {
    return new Multiplicavel() {
        public int produto() {
            return a * b;
        }
    };
}
public static void main(String[] args) {
    Multiplicavel mul = (new Externa()).calcular(3,4);
    int prod = mul.produto();
}
```

← Compare com parte em preto e vermelho do slide anterior!

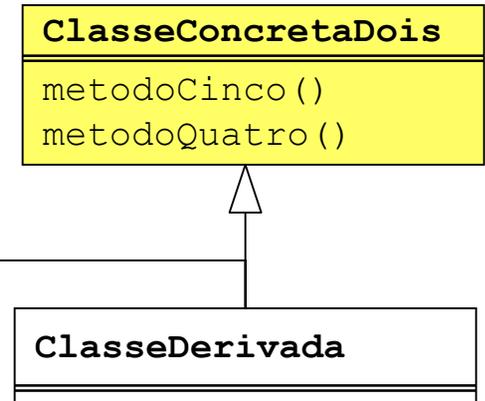
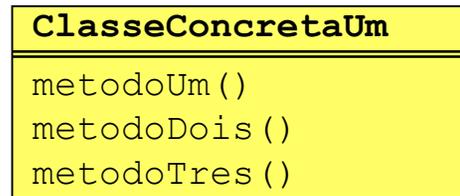
← A classe está dentro da instrução: preste atenção no ponto-e-vírgula!

# Para que servem classes internas?

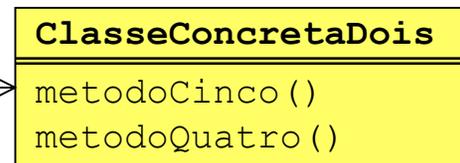
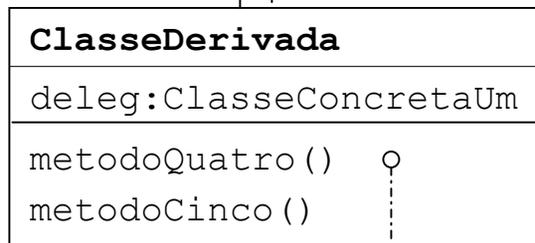
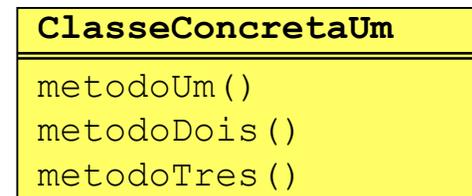
- *Mais reutilização*
  - *Recurso poderoso quando combinado com interfaces e herança - facilita implementação de delegação: tipo de herança de implementação que combinando composição com herança de interfaces (simula herança múltipla)*
  - *"Ponteiros seguros" apontando para métodos localizados em classes internas*
  - *Flexibilidade para desenvolver objetos descartáveis*
- *Riscos*
  - *Aumenta significativamente a complexidade do código*
  - *Dificulta o trabalho de depuração (erros de compilador são mais confusos em classes internas)*
- *Evite fugir do convencional ao usar classes internas*

# Como delegação simula herança múltipla

*Efeito  
Desejado  
(não permitido em Java)*



*Efeito Possível  
em Java*



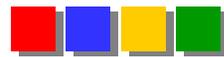
-  Classes existentes
-  Classes novas

- 1. Escreva uma aplicação que chame o método *imprimir()* de cada uma das classes do arquivo *Internas.java* (cap 15)
- 2. Implemente a classe *InMethod* de *Internas.java* como uma classe anônima.

# *Curso J100: Java 2 Standard Edition*

*Revisão 17.0*

© 1996-2003, Helder da Rocha  
(helder@acm.org)

 [argonavis.com.br](http://argonavis.com.br)