Classes internas

Helder da Rocha
www.argonavis.com.br

8. Classes internas

- *Classes dentro de classes*
- *Classes anônimas*
- *Classes dentro de métodos*
- *Classes internas estáticas*

Classes internas

- *Classes dentro de classes*

```
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*
- *Podem ser estáticas:*
  - *Externa.Interna*
- *Podem ser "de instância":*
  - *Externa e = new Externa();*
    Externa.Interna ei = e.new Externa.Interna();
- *Podem ser locais (dentro de métodos)*
  - *As locais podem ter ou não, um nome (anônimas)*

Classes estáticas

- *Declaradas como static*
  - *idênticas às classes externas*
  - *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();
   }
}
```

Classes "de instância"

- 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
- Referências 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

- *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)*

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

Classes anônimas

- *Classes usadas dentro de métodos freqüentemente servem apenas para criar um objeto uma única vez*
  - *Há uma sintaxe que dispensa o nome da classe, nesses casos*
    - `Interna i = new SuperClasse() { implementação };`
    - *SuperClasse, acima, pode ser Object, alguma superclasse de Interna ou uma interface (mais comum)*
  - *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!

Para que servem classes internas?

- Mais reutilização
  - Recurso poderoso quando combinado com interfaces e herança
  - Tipo de herança de implementação pode ser obtida
    - classes internas podem estender qualquer classe e os métodos internos podem ser chamados pelos métodos externos
  - "Ponteiros seguros" apontando para métodos localizados em classes internas
  - Flexibilidade para desenvolver objetos descartáveis