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)*

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)
  ```

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() {...}
}
```

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 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!

- *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)*