Escopo e visibilidade

Programação II – Aula 10

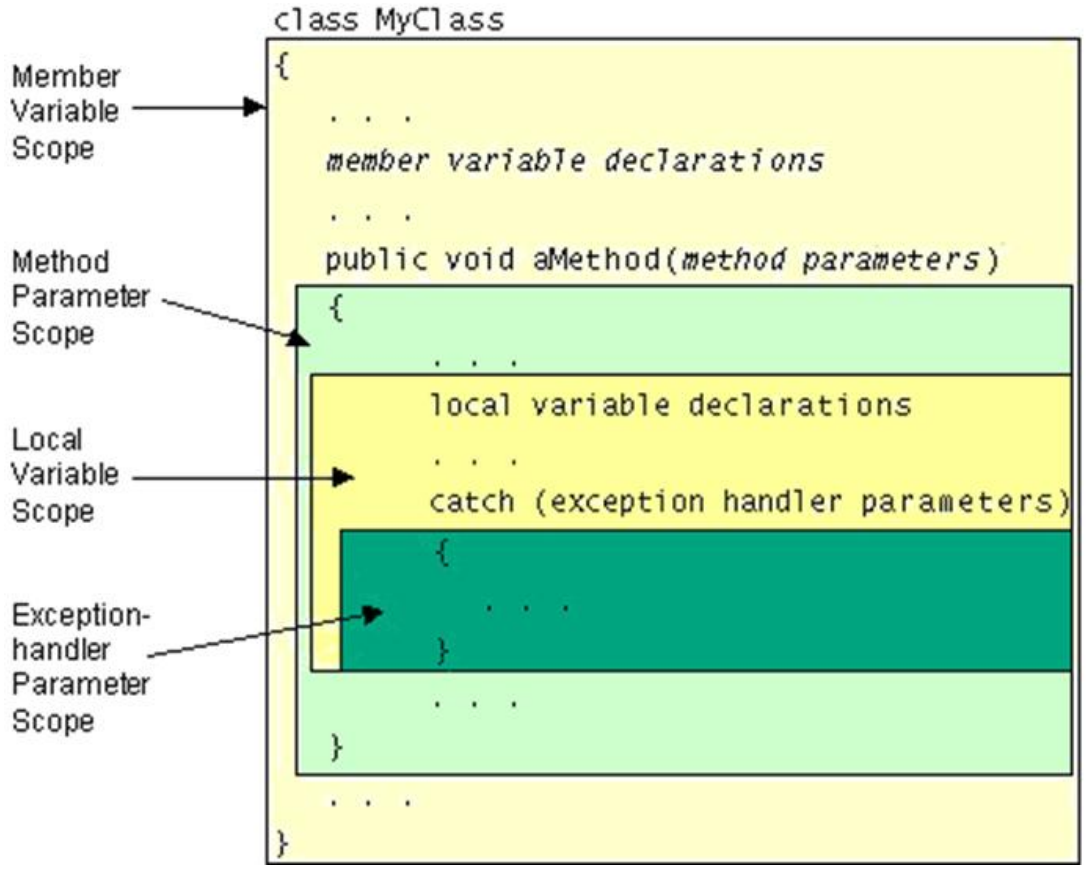
Material cedido por Jacques Sauvé (poucas alterações no original)

Objetivos da seção

- Discutir escopo de variáveis de forma mais detalhada
- Discutir ocultação de informação, encapsulamento e a visibilidade de dados de forma mais detalhada
- Discutir a organização de código em pacotes (packages)

Escopo de variáveis

- O escopo de uma variável é a região de um programa dentro do qual a variável pode ser referenciada através de seu nome
- O escopo define quando o sistema aloca e libera memória para armazenar a variável
- Escopo ≠ visibilidade (falaremos de visibilidade mais adiante)
  - Visibilidade é aplicada apenas a membros de classes (atributos, métodos) e determina se a variável pode ser usada de fora da própria classe onde ela é declarada
  - A visibilidade é definida através de modificadores de acesso
- O local onde a variável é declarada dentro do seu programa define seu escopo
- São 4 as categorias de escopo



- Exemplo

```
if (...) {
     int i = 17;
     ...
}
System.out.println("O valor de i = " + i);
```

Encapsulamento e Ocultação da informação

- O encapsulamento de dados com o código que os manipula em classes é uma das principais vantagens da Orientação a Objeto

- O ideal é que os dados não sejam acessados diretamente, mas através de uma interface (conjunto de métodos públicos) bem definida
- O encapsulamento por si só não garante essa proteção aos dados

- Por isso, é comum usarmos "**`private`**" como especificador de controle de acesso para atributos de uma classe
  - A lei é: "**Não posso quebrar o que não posso acessar**"
- Já que é freqüente querermos que métodos de uma classe sejam chamados por objetos de outras classes, não é raro usarmos "**`public`**" como especificador de controle de acesso para métodos de uma classe
- Especificadores de controle de acesso controlam a visibilidade de membros de uma classe
  - São aplicados a atributos e métodos

Ocultação da informação e métodos "accessor" get/set

- Ocultação da informação é outro conceito importante de Orientação a Objetos
  - Uma implementar esse conceito é usando métodos
- Com atributos sendo "**`private`**", é frequente usar métodos "accessor" (get/set) para manipular atributos
- Porém, devemos ter cuidado para não comprometer a ocultação da informação
  - Se uma classe faz objeto.getAtributo(), manipula o valor do atributo e depois faz objeto.setAtributo(), o atributo é essencialmente público e quebra a transparência da complexidade de manipulação do objeto muitas vezes desejada

- Mudanças de atributos devem ser conduzidas por métodos da própria classe para garantir que ela mantenha o estado do objeto consistente
- Tomemos como exemplo um semáforo
  - › Não devemos efetuar as mudanças de atributos fora do semáforo
  - › Portanto, não é bom ter um setCor()
  - › É melhor ter um método muda() que pede uma mudança de cor

  i) O próprio semáforo deve saber em que seqüência mudar as cores do semáforo

```
public class Semaforo {

   private static final String VERMELHO = "VERMELHO";
   private static final String AMARELO  = "AMARELO";
   private static final String VERDE    = "VERDE";

   private String corAtual;

   public Semaforo() {
      corAtual = VERMELHO;
   }

   public void muda() {

      // Java 7 permite usar switch para a classe String!

      if (corAtual.equals(VERMELHO)) {
         corAtual = VERDE;
      } else if (corAtual.equals(AMARELO)) {
```

```
            corAtual = VERMELHO;
        } else {
            corAtual = AMARELO;
        }
    }

    public String getCorAtual() {
        return corAtual;
    }

    public String toString() {
        return "semaforo esta' " + getCorAtual();
    }
}
```

Especificadores de controle de acesso

- Há quatro graus de visibilidade que podemos usar com membros de uma classe
- As palavras chaves usadas são: "**public**", "**private**", "**protected**", e nenhuma
- Para entender quando usar cada um desses graus de visibilidade, lembre que há vários papeis que os programadores podem assumir:

  i) Você, que está escrevendo uma classe

  ii)O programador "cliente" que só quer usar a classe que você criou (ele pode nem ter código fonte)

  iii) Outros programadores que estão trabalhando num pacote com você (várias classes de um mesmo pacote estão sendo feitas por vários programadores

iv) O programador que poderá estender sua classe no futuro (ele normalmente tem o código fonte)

A visibilidade public

- Quem tem acesso à classe tem acesso também a qualquer membro com visibilidade public
- O alvo aqui é o programador cliente que usa suas classes
- É raro ter atributos públicos, mas é comum ter métodos públicos

A visibilidade private

- O membro private (atributo ou método) não é acessível fora da classe
- A intenção aqui é permitir que apenas você que escreve a classe possa usar esse membro

A visibilidade protected

- O membro protected é acessível à classe e a suas subclasses
- A intenção é dar acesso ao programadores que estenderão sua classe

Packages

- O conceito de package (pacote) foi inventado para permitir criar um espaço de nomes grande em Java
- O que ocorre se você quiser criar uma classe DVD e outro programador já criou uma classe DVD?
  - Lembre-se que o paradigma de orientação a objetos possibilita o reuso de classes
  - Será que você vai ser impossibilitado de criar sua classe?

- Não há problema, desde que as classes assim chamadas estejam em packages diferentes
- Os nomes dos packages formam uma árvore, permitindo assim um espaço de nomes muito grande
  - Exemplo: p1.aplic.banco é um nome de package
  - Exemplo: p1.aplic.banco.Conta é uma classe deste package

A visibilidade "package"

- Um membro de classe sem especificador de controle de acesso é dito ter a visibilidade package (ou "friendly")
- É como public, mas somente dentro do package
- Todas as classes do package podem acessar um membro "friendly"
- É usado para permitir acesso mais liberal, mas somente dentro de um mundo controlado e não pelo usuários da classe
- Deve-se ter cuidado com a visibilidade friendly para atributos pois pode abrir muito o acesso, principalmente em pacotes grandes

Um exemplo

- Observe a visibilidade de Agencia.fecharConta()

```
package p1.aplic.banco;

public class Agencia {
  protected static Map contas = null;  /* mapa iniciado no construtor */

  ...
```

```
  /**
   * Fecha uma conta.
   * @param número O número da conta a fechar.
   * @throws NaoPodeFecharContaException Se a conta não existir ou tiver
   *                                     saldo
   */
  /* observe visibilidade "package": tem que fechar a partir de fechar() da
conta */
  static void fecharConta(int número) throws NaoPodeFecharContaException {
    abrirCaixa();
    Conta c = localizarConta(número);
    if(c == null) {
      throw new NaoPodeFecharContaException(c, "Conta nao existe");
    }
    if(c.getSaldo() != 0.0) {
      throw new NaoPodeFecharContaException(c, "Saldo nao esta zerado");
    }
    contas.remove(Integer.toString(número));
  }
  ...
}
```

- Fora do pacote p1.aplic.banco, nenhuma classe pode chamar Agencia.fecharConta()
- Dentro do pacote p1.aplic.banco, várias classes poderiam chamar Agencia.fecharConta()
  - Uma decisão foi tomada: apenas a classe Conta vai chamar Agencia.fecharConta()
  - Para fechar uma conta, o usuário do pacote deve usar Conta.fechar()