UFMS DIGITAL

Programação Orientada a Objetos

Prof. Dr. Anderson V. de Araujo

UFMS DIGITAL

Módulo III - Pilares da programação orientada a objetos

Unidade II - Encapsulamento e Modificadores de acesso/visibilidade

Declarações de Classe e Modificadores de Acesso

- Declaração de classe simples:

```
modificador_acesso class MinhaClasse { }
```

- Modificadores de acesso de classe possíveis:
  - `public`: acessada de qualquer classe;
  - Sem Modificador (package-private ou padrão): acessada dentro do mesmo pacote.

Resumindo...

Modificador	Pacote	Qualquer lugar
public	Y	Y
Sem modificador (package-private)	Y	N

Declarações de Membros Internos e Modificadores de Acesso

- Declaração de membros internos:

```
modificador_acesso String nome;

modificador_acesso void executar(){}
```

- Modificadores de acesso:
  - `public`: acessado de qualquer classe
  - `protected`: acessado por classes filhas e do mesmo pacote
  - Sem Modificador (package-private ou padrão): acessado dentro do mesmo pacote (filha ou não)
  - `private`: acessado dentro da mesma classe

Modificadores de Acesso

- Existem duas questões relativas a acesso que devem ser entendidas:
  - O código de um método em uma classe pode acessar um membro de outra classe?
  - Uma subclasse pode herdar um membro da sua superclasse?

Modificadores de Acesso

- O que significa o fato do código em uma classe ter acesso a um membro de outra classe?
  - Se a classe A tiver acesso a um membro da classe B, significa que o membro da classe B está visível para a classe A

Modificadores de Acesso

- Uma subclasse herda quais membros da sua superclasse?
  - A SUBCLASSE herda todos os membros da SUPERCLASSE
    - E se ele estiver visível é exatamente como se a SUBCLASSE tivesse declarado o membro nela mesma
  - Não é uma questão de se herdar ou não, mas se é visível ou não.
    - Ou seja, quais membros de uma SUPERCLASSE podem ser acessados por uma SUBCLASSE por meio da herança

Membros `public`

- Quando um membro é declarado **`public`**, significa que todas as outras classes, independente do pacote ao qual pertençam, podem acessar o membro em questão
  - Assumindo que a própria classe esteja visível.

Membros `protected`

- Pode ser acessado (através de herança) por uma subclasse, mesmo se a subclasse estiver em um pacote diferente:
  - Depois que a subclasse de fora do pacote herda o membro `protected`, ele torna-se `private` para qualquer código de fora da subclasse, com exceção das subclasses da subclasse.
- Pode ser acessado por classes do mesmo pacote também:
  - Diferentemente de C++, C# e outras linguagens OO.

Membros *package-private*

- Sem modificador (*package-private*);
- Só pode ser acessado se a classe que o estiver acessando pertencer ao mesmo pacote.

Membros `private`

- Membros marcados como **`private`** não podem ser acessados por nenhuma outra classe, a não ser aquela na qual o membro private foi declarado.
  - Obs.: Isso caracteriza o conceito OO chamado **Encapsulamento.**

Encapsulamento

- Técnica para fazer com que os campos de uma classe sejam escondidos e acessá-los via métodos públicos
  - Se um campo é declarado privado, ele não pode ser acessado fora da classe, escondendo, assim, os campos dentro da classe
- Pode ser descrito como uma barreira protetora que impede que o código e os dados sejam acessados por outro código definido fora da classe
- Provê fácil manutenção, flexibilidade e extensibilidade para o código
- Também conhecido como Data Hiding
- A forma mais comum de uso do Encapsulamento é através dos getters e setters

Encapsulamento - Exemplo 1

```
public class Car{
    public float speed = 0;
    public boolean started = false;
    public boolean reverse = false;
}
public class Test{

    public static void main(String[] args){
          Car car = new Car();
          car.started = true;
          car.speed = 100;
          car.speed = 0;
          car.speed = 200;
          car.reverse = true;
          car.started = false;
    }
}
```

```java
class Car {
 private float speed = 0;
 private boolean reverse = false;
 private boolean started = false;
 public void start() {
     if (started) {// the car is already started
     } else {
        started = true;
     }
 }
 public void accelerate() {
     if (started) {
        speed += 10;
     } else {
        // car is not started yet
     }
 }
 public void break() {
     if (started) {
       speed -= 10;
     } else {
       // car is not started yet
     }
 }
 public void off() {
    if (started) {
      started = false;
     } else {
       // car is already off
     }
 }
}
```

```java
class Test{
   public static void main(String[] a){
      Car car = new Car();
      car.off(); //car is already off
      car.start();
      car.accelerate();
      car.accelerate();
      car.accelerate();
      car.break();
      car.break();
      car.break();
      car.off();
    }
}
```

Qualquer classe que queira acessar os atributos "escondidos" deve fazê-lo através dos **métodos públicos**

Combinações de Acesso e Visibilidade

Visibilidade	PUBLIC	PROTECTED	Sem Modificador (package- private)	PRIVATE
A partir da mesma classe	Sim	Sim	Sim	Sim
A partir de qualquer classe do mesmo pacote	Sim	Sim*	Sim	Não
A partir de uma subclasse do mesmo pacote	Sim	Sim	Sim	Não
A partir de uma subclasse de fora do mesmo pacote	Sim	Sim, através da herança	Não	Não
A partir de qualquer classe que não seja uma subclasse e esteja fora do pacote	Sim	Não	Não	Não

*Diferente de C++, C# ...

Outros Modificadores de Atributos

- **`final`**: o valor do atributo não pode ser alterado;
  - `private final String nome = "João";`
- **`static`**: só vai ter uma instância/valor do atributo para todas as instâncias da classe. Se torna um atributo da classe e não da instância;
- Outros que vamos ver mais tarde...

Constantes

- São definidas através do uso de duas palavras reservadas da linguagem Java: **`static`** e **`final`**
  - **`static`**: indica que a variável é da classe, ou seja, existe um só valor para classe e todos os objetos criados a partir dela;
  - **`final`**: indica que o valor de uma variável não pode ser alterado.
- Exemplo:
  - `static final double PI = 3.141592653589793;`

Outros Modificadores de Métodos

- **`final`**: nenhum outro método pode sobrescrever o método;
- **`static`**: se torna um método da classe e não da instância;
- **`abstract`**: não vai ser implementado na classe corrente, somente em um de seus filhos;
- Outros que vamos ver mais tarde...

Licenciamento