02

Aplique modificadores de acesso

Os modificadores de acesso, ou modificadores de visibilidade, servem para definir quais partes de cada classe (ou se uma classe inteira) estão visíveis para serem utilizadas por outras classes do sistema. Só é permitido usar um único modificador de acesso por vez:

```
private public int x; // não compila
```

COPIAR CÓDIGO

O Java possui os seguintes modificadores de acesso:

- `public`
- `protected`
- Nenhum modificador, chamado de `default`
- `private`

Classes e interfaces só aceitam os modificadores `public` ou `default` .

Membros (construtores, métodos e variáveis) podem receber qualquer um dos quatro modificadores.

Variáveis locais (declaradas dentro do corpo de um método ou construtor) e parâmetros não podem receber nenhum modificador de acesso, mas podem receber outros modificadores.

Top Level Classes e Inner Classes

Classes internas (*nested classes* ou *inner classes*) são classes que são declaradas dentro de outras classes. Esse tipo de classe pode receber qualquer modificador de acesso, já que são consideradas membros da classe onde foram declaradas (*top level class*).

Nesta certificação não são cobradas classes internas, apenas *top level classes*.

Para entender como os modificadores funcionam, vamos imaginar as seguintes classes:

```
package forma;

class Forma{
    double lado;
    double getArea(){
        return 0;
    }
}
```

COPIAR CÓDIGO

```
package forma;

class Quadrado extends Forma{}
```

COPIAR CÓDIGO

```
package forma.outro;
import forma.*;

class Triangulo extends Forma{}
```

COPIAR CÓDIGO

Public

O modificador `public` é o menos restritivo de todos. Classes, interfaces e membros marcados com esse modificador podem ser acessados de qualquer componente, em qualquer pacote. Vamos alterar nossa classe `Forma`, marcando-a e todos seus membros com o modificador `public`:

```
package forma;

public class Forma{
    public double lado;
    public double getArea(){
        return 0;
    }
}
```

COPIAR CÓDIGO

Agora vamos fazer um teste:

```
package forma.outro;
import forma.*;

public class TesteOutroPacote{

    public static void main(String... args){
        Forma f = new Forma(); //acesso a classe forma
        f.lado = 5.5; //acesso ao atributo lado
        f.getArea(); //acesso ao método getArea()
```

```
    }
}
```

COPIAR CÓDIGO

Repare que, mesmo nossa classe `TesteOutroPacote` estando em um pacote diferente da classe `Forma`, é possível acessar a classe e todos os membros declarados como `public`.

Protected

Membros definidos com o modificador `protected` podem ser acessados por classes e interfaces no mesmo pacote, e por qualquer classe que estenda aquela onde o membro foi definido, independente do pacote.

Vamos modificar nossa classe `Forma` para entendermos melhor:

```
package forma;

public class Forma{
    protected double lado; // agora protected
    public double getArea(){}
}
```

COPIAR CÓDIGO

Com o modificador `protected`, nossa classe de testes em outro pacote não compila mais:

```
package forma.outro;
import forma.*;
```

```
public class TesteOutroPacote{

    public static void main(String... args){
        Forma f = new Forma();
        f.lado = 5.5; // erro de compilação
        f.getArea();
    }
}
```

COPIAR CÓDIGO

Se criarmos uma nova classe de teste no pacote `forma`, conseguimos acessar novamente o atributo:

```
package forma;

public class Teste{

    public static void main(String... args){
        Forma f = new Forma();
        f.lado = 5.5; // compila normal, mesmo pacote
    }
}
```

COPIAR CÓDIGO

Embora esteja em um pacote diferente, a classe `Triangulo` consegue acessar o atributo `lado`, já que ela estende da classe `Forma`:

```
package forma.outro;
import forma.*;
```

```
class Triangulo extends Forma{

    public void imprimeLado(){
        //Como é uma classe filha, acessa
        //normalmente os membros protected da classe mãe.
        System.out.println("O Lado é " + lado);
    }
}
```

COPIAR CÓDIGO

Agora repare que, se efetuarmos o casting do objeto atual para uma `Forma`, não podemos acessar seu lado:

```
package outro;
import forma.*;

class Triangulo extends Forma{

    public void imprimeLado(){
        // compila
        System.out.println("O Lado é " + lado);

        // não compila
        System.out.println("O Lado é " + ((Forma) this).lado);
    }
}
```

COPIAR CÓDIGO

Isso ocorre porque estamos dizendo que queremos acessar a variável membro `lado` de um objeto através de uma referência para este objeto, e não diretamente. Diretamente seria o uso puro do `this` ou nada. Nesse caso, após usar o `this`, usamos um casting, o que deixa o compilador perdido.

Default

Se não definirmos explicitamente qual o modificador de acesso, podemos dizer que aquele membro está usando o modificador `default`, também chamado de `package private`. Neste caso, os membros da classe só serão visíveis dentro do mesmo pacote:

```
package forma;

public class Forma{
    protected double lado;
    public double getArea(){
        return 0;
    }
    double getPerimetro(){ //default access
        return 0;
    }
}
```

COPIAR CÓDIGO

O método `getPerimetro()` só será visível para todas as classes do pacote `forma`. Nem mesmo a classe `Triangulo` - que, apesar de herdar de `Forma`, está em outro pacote - consegue ver o método.

```
package outro;
import forma.*;

class Triangulo extends Forma{

    public void imprimePerimetro(){
        //Erro de compilação na linha abaixo
```

```
            System.out.println("O Perímetro é " + getPerimetro());
        }
    }
```

COPIAR CÓDIGO

Palavra-chave %%default%%

Lembre-se! A palavra-chave `default` é usada para definir a opção padrão em um bloco `switch`, ou para definir um valor inicial em uma *Annotation*. Usá-la em uma declaração de classe ou membro é inválido e causa um erro de compilação:

```
default class Bola{ //ERRO
    default String cor; // ERRO
}
```

COPIAR CÓDIGO

A partir do Java 8, a palavra `default` também pode ser usada para definir uma implementação inicial de um método.

Mas e se declararmos uma classe com o modificador `default`? Isso vai fazer com que aquela classe só seja visível dentro do pacote onde foi declarada. Não importa quais modificadores os membros dessa classe tenham, se a própria classe não é visível fora de seu pacote, nenhum de seus membros é visível também.

Veja a classe `Quadrado`, que está definida com o modificador `default`:

```
package forma;

class Quadrado extends Forma{}
```

COPIAR CÓDIGO

Veja o seguinte código, usando a classe `TesteOutroPacote`. Perceba que não é possível usar a classe `Quadrado`, mesmo importando todas as classes do pacote `forma`:

```
package outro;
import forma.*;

public class TesteOutroPacote{

    public static void main(String... args){
        Quadrado q = new Quadrado(); // erro, esta classe não
é
                                     //visível
    }
}
```

COPIAR CÓDIGO

Linha com erro de compilação

Eventualmente, na prova, é perguntado em quais linhas ocorreram os erros de compilação. É bem importante prestar atenção nesse detalhe.

Por exemplo, neste caso, o erro sempre acontecerá quando tentarmos acessar a classe `Quadrado`, que não é visível fora de seu pacote:

```
package outro;
```

```
//import de todas as classes PÚBLICAS do pacote, nenhum erro
import forma.*;


public class TesteOutroPacote{

    public static void main(String... args){
        // erro na linha 8, Quadrado não é visível, pois não
        // é pública
        Quadrado q = new Quadrado();
    }
}
```

COPIAR CÓDIGO

O mesmo código pode apresentar erro em uma linha diferente, apenas mudando o `import`. Repare que o código a seguir dá erro nas duas linhas, tanto do `import` quanto na tentativa de uso:

```
package outro;
// erro na linha 3, não podemos importar classes não públicas
import forma.Quadrado;

public class TesteOutroPacote{

    public static void main(String... args){

        //Erro, pois Quadrado não é acessível.
        Quadrado q = new Quadrado();
    }
}
```

COPIAR CÓDIGO

É muito importante testar vários trechos de código, para ver exatamente em quais linhas de código o erro de compilação aparecerá.

Private

`private` é o mais restritivo de todos os modificadores de acesso. Membros definidos como `private` só podem ser acessados de dentro da classe e de nenhum outro lugar, independente de pacote ou herança:

```
package forma;

public class Forma{
    protected double lado;
    public double getArea(){}

    //cor só pode ser acessada dentro da classe Forma,
    //nem as classe Quadrado e Triangulo conseguem acessar
    private String cor;

}
```

COPIAR CÓDIGO

Private e classes aninhadas ou anônimas

Classes aninhadas ou anônimas podem acessar membros privados da classe onde estão contidas. Na certificação tais classes não são cobradas.

Métodos privados e padrão não podem ser sobrescritos. Se uma classe o "sobrescreve", ele simplesmente é um método novo, portanto não podemos dizer que é sobrescrita. Veremos isso mais a fundo na seção sobre sobrescrita.

Resumo das regras de visibilidade

Todos os membros da classe com o modificador de `private` só podem ser acessados de dentro dela mesma.

Todos os membros da classe sem nenhum modificador de visibilidade, ou seja, com visibilidade **package-private**, podem ser acessados de dentro da própria classe ou de dentro de qualquer outra classe, interface ou enum do mesmo pacote.

Todos os membros da classe com o modificador `protected` podem ser acessados:

- de dentro da classe, ou de dentro de qualquer outra classe, interface ou enum do mesmo pacote;
- de dentro de alguma classe que deriva direta ou indiretamente da classe, independente do pacote. O membro `protected` só pode ser chamado através da referência `this`, ou por uma referência que seja dessa classe filha.

Todos os membros da classe com o modificador `public` podem ser acessados de qualquer lugar da aplicação.

E não podemos ter classes/interfaces/enums top-level como `private` ou `protected`.

Uma classe é dita *top-level* se ela não foi definida dentro de outra classe, interface ou enum. Analogamente, são definidas as interfaces top-level e os enums top-level.