TI-220 Java Orientado a Objetos

ANTONIO CARVALHO - TREINAMENTOS

Modificadores

Outros modificadores

Outros Modificadores

Existem outros modificadores que não são de acesso:

- **abstract**
- **final**
- **static**
- **strictfp**
- **synchronized**
- **volatile**
- **transient**
- **native**

Eles podem ser combinados com os modificadores de acesso

E podem ser combinados entre eles evitando apenas a combinação ***final*** e ***abstract***, pois um elemento não pode ser incompleto e completo ao mesmo tempo.

Outros Modificadores (abstract)

O modificador **abstract** identifica o elemento como sendo incompleto

Uma classe abstract não pode ser instanciada

Ela pode ser herdada por alguma subclasse a qual pode implementar os métodos que faltam, deixando de ser abstrata e permitindo a instanciação de objetos do seu tipo

- Pode ser usado apenas em **classes** e **métodos**

Nota : Os métodos marcados com o modificador ***abstract*** possuem apenas a assinatura e terminam com (;) no final da assinatura, não há o par de chaves { } que delimita o bloco de código .

Outros Modificadores (abstract)

Uma classe abstrata pode conter métodos **não abstratos**

A classe abstrata **pode** ou **não** conter métodos **abstratos**

Se um método for declarado como abstrato, então a classe inteira deve ser declarada como abstrata.

Se uma classe herda de uma classe abstrata é preciso que ela implemente todos os métodos abstratos para que se torne uma classe completa, porém é possível não implementar todos os métodos, tornando a subclasse abstrata também.

Nota : Não é possível criar objetos de uma classe abstrata. Para possibilitar instanciação de objetos é primeiro necessário extender a classe abstrata implementando os métodos abstratos, nesse momento haverá uma classe concreta que pode ser instanciada.

Outros Modificadores (abstract)

Classe abstrata

```
package ocjp.java.certification.abstrato;
public abstract class Reptil {
      public abstract void comer();
      public void andar() {
            System.out.println("Andando");
      }
}
```

```
package ocjp.java.certification.abstrato;
public class Jacare extends Reptil {
      @Override
      public void comer() {
            System.out.println("Como carne");
      }
}
```

Classe concreta

```
package ocjp.java.certification.abstrato;
public class Lagarto extends Reptil {
      @Override
      public void comer() {
            System.out.println("Como ovos");
      }
}
```

Classe concreta

Outros Modificadores (final)

Modificador ***final*** identifica que a classe não pode ser estendida (herdada)

Quando aplicada a um **método**, indica que este não pode ser **sobrescrito**, e quando aplicada em uma **variável**, indica que o seu valor **não pode ser alterado**.

Tornar uma classe final assegura que ninguém pode criar outra classe a partir dela, evitando o uso de uma subclasse modificada.

- Pode ser usado em **classes, métodos** e **variáveis**

Nota : O uso do modificador **final** deve ser planejado cuidadosamente, pois os métodos marcados com **final** não podem ser reescritos perdendo esta vantagem da Orientação a Objetos, assim como as classes marcadas com **final** não podem ser extendidas.

Outros Modificadores (final)

```
package ocjp.java.certification;
public class Funcionario {
    final public float pisoSalarial = 600.0f;
    protected void trabalhar() {
        System.out.println("trabalhando");
    }
}
```

```
package ocjp.java.certification;
public class Gerente extends Funcionario {
    private void liderar() {
        pisoSalarial = 750.0f;
        trabalhar();
        System.out.println("liderando");
    }
}
```

The final field Funcionario.pisoSalarial cannot be assigned

A variável pisoSalarial é **final** e não pode ter seu conteúdo **modificado**

Outros Modificadores (final)

O modificador **final** pode ser utilizado também em variáveis que são recebidas como parâmetros pelos métodos.

```
package ocjp.java.certification;
public class Funcionario {
    final public float pisoSalarial = 600.0f;
    protected void trabalhar() {
        System.out.println("trabalhando");
    }

    public void calculaSalario( final float valorHora ) {
    }

}
```

- Isto assegura que mesmo que a função seja **sobrescrita** em uma **herança**, o valor da variável não poderá ser modificado ao longo do código.

Outros Modificadores (static)

static, determina que o membro pertence a classe e não há vinculo deste membro com futuras instâncias

Sintaxe :

```
[modificador de acesso] static <tipo> <nome> ([parametros])
```

Exemplo:

```
public static double calcularAreaRetangulo (double base, double
altura) {
    return base * altura
}
```

- Pode ser usado em **métodos** ou em **variáveis de instância**.

Outros Modificadores (strictfp)

strictfp, força o método a utilizar números flutuantes e operações com números flutuantes aderentes ao padrão IEEE 754.

- Algumas plataformas tem vantagens em precisão do que outras, ao usar strictfp o método não usará estas vantagens em precisão.

Sintaxe :

[modificador de acesso] strictfp <tipo> <nome> ([parametros])

Exemplo:

```
public strictfp float calculaPagamento() {  }
```

- Pode ser usado em **métodos** ou em **classes** indicando que todos os métodos são **strictfp**

Mais informações sobre Float Numbers ➔ https://ciechanow.ski/exposing-floating-point/

Outros Modificadores (synchronized)

synchronized, é aplicável a métodos e indica que o método somente pode ser acessado por uma *Thread* por vez

Sintaxe :

[modificador de acesso] synchronized <tipo> <nome> ([parametros])

Exemplo:

public synchronized boolean recebimento() { }

- Pode ser usado apenas em **métodos**

Outros Modificadores (volatile)

volatile, é aplicável a variáveis indicando que elas são acessadas diretamente da memória principal, sem passarem por *caches* locais das *Threads*.

Sintaxe :

[modificador de acesso] volatile <tipo> <nome>[= <valor inicial>];

Exemplo:

public volatile int counter = 10;

- Pode ser usado apenas em **variáveis de instância ou estáticas**

Outros Modificadores (transient)

transient, é aplicável na variável de instância, indicando que ela não fará parte de um processo de serialização.

Sintaxe :

[modificador de acesso] transient <tipo> <nome>[= <valor inicial>];

Exemplo:

public transient int idade;

```
class Pessoa implements Serializable {
        private transient int idade;
        private String nome;
        private LocalDate nascimento;
}
```

- Pode ser usado apenas em **variáveis de instância**

Outros Modificadores (native)

native, é aplicável a métodos e indica que o método é dependente da plataforma.

- O corpo do método não deve ser implementado, portanto a assinatura deve terminar com (;)

Sintaxe :

[modificador de acesso] native <tipo> <nome> ([parametros]);

Exemplo:

public native boolean recebimento();

- Pode ser usado apenas em **métodos**

Uso dos modificadores

Modificador	Classes	Métodos	Variáveis de instância	Variáveis (locais)
final	Sim	Sim	Sim	Sim
default (package)	Sim	Sim	Sim	Não
public	Sim	Sim	Sim	Não
abstract	Sim	Sim	Não	Não
strictfp	Sim	Sim	Não	Não
private	Não	Sim	Sim	Não
protected	Não	Sim	Sim	Não
static	Não	Sim	Sim	Não
native	Não	Sim	Não	Não
syncronized	Não	Sim	Não	Não
transient	Não	Não	Sim	Não
volatile	Não	Não	Sim	Não

Dúvidas

Modificadores - Exercícios

(Fonte : Sierra, adaptador Autor)
Dados dois arquivos:

```
1. package pkgA;
2. public class Foo {
3.   int a = 5;
4.   protected int b = 6;
5.   public int c = 7;
6. }
3. package pkgB;
4. import pkgA.*;
5. public class Baz {
6.   public static void main(String[] args) {
7.     Foo f = new Foo();
8.     System.out.print(" " + f.a);
9.     System.out.print(" " + f.b);
10.    System.out.print(" " + f.c);
11.  }
12. }
```

Qual será o resultado ? (Escolha todos que se aplicam)

A) 5 6 7

B) 5 seguido por uma exceção

C) Falha de compilação com erro na linha 7

D) Falha de compilação com erro na linha 8

E) Falha de compilação com erro na linha 9

F) Falha de compilação com erro na linha 10

Modificadores - Exercícios

(Fonte : Sierra, adaptador Autor)
Dado o seguinte código:

```
 4. public class Frodo extends Hobbit {
 5.   public static void main(String[] args) {
 6.     Short myGold = 7;
 7.     System.out.println(countGold(myGold, 6));
 8.   }
 9. }
10. class Hobbit {
11.   int countGold(int x, int y) { return x + y; }
12. }
```

Qual é o resultado ?

A. 13

B. Falha de compilação devido a múltiplos erros

C. Falha de compilação devido a um erro na linha 6

D. Falha de compilação devido a um erro na linha 7

E. Falha de compilação devido a um erro na linha 1