A INDÚSTRIA DO CONHECIMENTO

www.sc.senai.br | 0800 48 1212

Modificadores final e abstract

Aula 08 – 18/05

Modificador final

O modificador final pode ser utilizado na declaração de atributos, métodos e classes tendo o comportamento diferente em cada uma das situações.

Modificador final (atributos)

O modificador final em atributos, significa que somente uma atribuição poderá ser feita ao atributo, podendo ser feita na declaração ou no construtor. Se o atributo for do tipo primitivo, depois de feita a inicialização, não podemos mais alterar o seu conteúdo.

Exemplo

```
class FinalPrimitivos {

   final int i = 0;//isso é uma constante em
java.

}
```

Modificador final (atributos)

Se tentarmos alterar o valor da variável i teremos um erro de compilação.

```
class TesteFinalPrimitivos {
    public static void main(String[] args) {
        FinalPrimitivos p =
        new FinalPrimitivos();
        t.i = 0; //Erro de compilação
    }
}
```

```
Javac TesteFinalPrimitivos.java
Cannot assign a value to final variable i
```

Modificador final (atributos)

Se o atributo for do tipo reference, já sabemos que a variável armazena uma referência. No entanto, podemos chamar métodos e atributos do objeto referenciado.

Modificador final (atributos)

```
class Livro {

        String nome;

}


class TesteLivroFinal {

        public static void main(String[] args){

                final Livro l = new Livro();

                l.nome = “Java Core7”; //Ok podemos alterar atributos.

                l = new Livro(); // isso nos da um erro de compilação.

        }

}
```

Modificador final (classes)

O modificador final pode ser utilizado na declaração de classe, indicando que a classe não poderá ser estendida por outra classe, ou seja, não poderá ser herdada.

Modificador final (classes)

```
final class Conta {}


class ContaCorrente extends Conta{
}
```

A primeira classe compila, mas a segunda nos da um erro.

```
javac ContaCorrente.java
Cannot inherit from final Conta
```

Modificador final (classes)

Em uma linguagem orientada a objetos porque não gostaríamos que uma classe fosse herdada? Imagine se você estendesse a classe String e alterasse o comportamento de suas Strings.

Isso não seria nada agradável, prevendo isso quando desenvolveram o java criaram a classe String como final.

Modificador final (métodos)

Podemos utilizar o modificador final na declaração de um método, para que este não possa ser sobrescrito em uma classe filhas.

Modificador final (métodos)

```
class Conta {

     double saldo

     final void saque(double v) {

          saldo = saldo + v;

     }

}

class ContaPoupanca extends Conta {

     //não será permitido sobrescrevermos o método saque.

     void saque(double v) {

          saldo = saldo + v;

     }

}
```

Modificador abstract

O modificador **abstract** pode ser usado para métodos e classes.

Classes abstract

Utilizamos o modificador abstract para indicar que uma classe não está pronta para ser utilizada, não pendendo ser instanciada, somente estendida (utilizada em herança).

- ✓ uma classe pode ser declarada como **abstract** sem conter nenhum método abstrato, indicando apenas que a classe não poderá ser instanciada;
- ✓ uma classe **abstract** pode conter métodos não **abstract**;
- ✓ se a classe contiver pelo menos um método **abstract** será obrigatoriamente declarada como **abstract**.

Classes abstract

Exemplo:

```
public abstract class FormatadorRecibo {
	private String nomeEmpresa;

	public String getNomeEmpresa() {
		return nomeEmpresa;
	}

	public void setNomeEmpresa(String nomeEmpresa) {
		this.nomeEmpresa = nomeEmpresa;
	}

	public String gerarRecibo(String nomeCliente, double valor, String data, String motivo) {
		return "método em construção";
	}
}
```

Classes abstract

Exemplo:

```
public class TesteRecibo {

    public static void main(String[] args) {
        //Essa linha não irá compilar
        FormatadorRecibo fr = new FormatadorRecibo();
        fr.setNomeEmpresa("Senai/SC");
        String recibo = fr.gerarRecibo(
            "Catia", 110.0, "18/5/2012",  "telefone");
    }

}
```

```
javac FormatadorRecibo .java
FormatadorRecibo is abstract cannot be instantiated;
```

Classes abstract

Como a classe FormatadorRecibo não poderá ser instanciada diretamente, será necessário criar uma subclasse que a estenda, ou seja, a classe FormatadorRecibo serve como um padrão para outras classes.

Vejamos então um exemplo de uma classe FormtadorModeloA, que estende a classe FormatadorRecibo e sobrescreve seu método gerarRecibo;

Classes abstract

```
public class FormatadorModeloA extends FormatadorRecibo {

	@Override
	public String gerarRecibo(String cliente, double v, String dt,String m) {
		StringBuilder sb = new StringBuilder();
		sb.append("Nós da ").append( super.getNomeEmpresa() );
		sb.append(" recebemos de ").append( cliente);
		sb.append(" em ").append(dt);
		sb.append(" o valor de R$ ").append(v);
		sb.append(" referente a").append(m);

		return sb.toString();
	}

}
```

Classes abstract

Agora ser voltarmos a classe TesteRecibo

```
public class TesteRecibo {

    public static void main(String[] args) {
        FormatadorRecibo fr;

        //Ok o objeto fr do tipo FormatadorRecibo está recebendo
        // uma classes filha
        fr = new FormatadorModeloA();
        fr.setNomeEmpresa("Senai/SC");
        String recibo = fr.gerarRecibo("Catia", 110.0, "18/5/2012", "telefone");

        System.out.println(recibo);
    }

}
```

Classes abstract

Verificamos que não é possível instanciar classe marcadas como o modificador abstract. Na prática uma classe abstrata serve para acomodar métodos abstratos, como veremos a seguir.

Métodos abstract

Muitas vezes nos deparamos com essa frase: “Existe mais de uma forma de resolver esse problema.” Muitas vezes uma rotina de software pode ser escrita de várias formas para atingir um mesmo resultado:

- ✓ Utilizando racionalmente a memória em um ambiente restrito;
- ✓ Minimizando as operações de I/O em um ambiente de resposta rápida;
- ✓ Visando a simplicidade e a clareza do código para futuras manutenções;
- ✓ Utilizando bibliotecas ou API’s para integrar com outros sistemas.

Métodos abstract

A orientação a objetos facilita a convivência com os dilemas que vimos anteriormente com o conceito de métodos abstratos.

Um método abstrato é utilizado em um a classe quando o desenvolvedor ou projetista de software percebe que:

- ✓ É possível implementar um método de várias maneiras interessantes.
- ✓ Ainda não existe requisitos bem definidos para decidir por uma implementação do método.

Métodos abstract

Um método abstrato define um ponto de diversificação, indicando que mais de uma solução técnica pode ser adotada.

Podemos ter métodos abstratos somente dentro de classes abstratas. Isso fará com que estes métodos tenham que ser obrigatoriamente implementados nas subclasses não abstratas.

Métodos abstract

Um método abstrato tem a seguinte semântica:

- ✓ declaramos apenas a sua assinatura (tipo de retorno, nome, parâmetros de entrada e exceções) incluindo o modificador abstract;
- ✓ não declaramos corpo de instruções;
- ✓ finalizamos a declaração com ponto e virgula.

Métodos abstract

Métodos abstratos são utilizados para definir um contrato de comportamento, garantindo que serão implementados nas classes filhas.

- ✓ sempre que declararmos **um método abstract**, a **classe** também deverá ser declarada como **abstract**;
- ✓ todas as **classes filhas** deverão obrigatoriamente **implementar os métodos abstract** ou então serem declaradas como **abstract**.

Métodos abstract

Exemplo: se voltarmos para a classe FormatadorRecibo e declararmos o método como abstract.

```
public abstract class FormatadorRecibo {
    private String nomeEmpresa;

    public String getNomeEmpresa() {
        return nomeEmpresa;
    }

    public void setNomeEmpresa(String nomeEmpresa) {
        this.nomeEmpresa = nomeEmpresa;
    }

    //agora declaração de método abstract
    public abstract String gerarRecibo(String c, double v, String d, String m);

}
```

Repare que não temos corpo no método nem chaves { } abrindo e fechando só a assinatura e terminado com ";"

Utilizamos a palavra reservada abstract

Métodos abstract

Agora todas as classes não abstratas que herdem de FormatadorRecibo serão **obrigadas** a implementar o método gerarRecibo.

Métodos abstract

Exemplo:

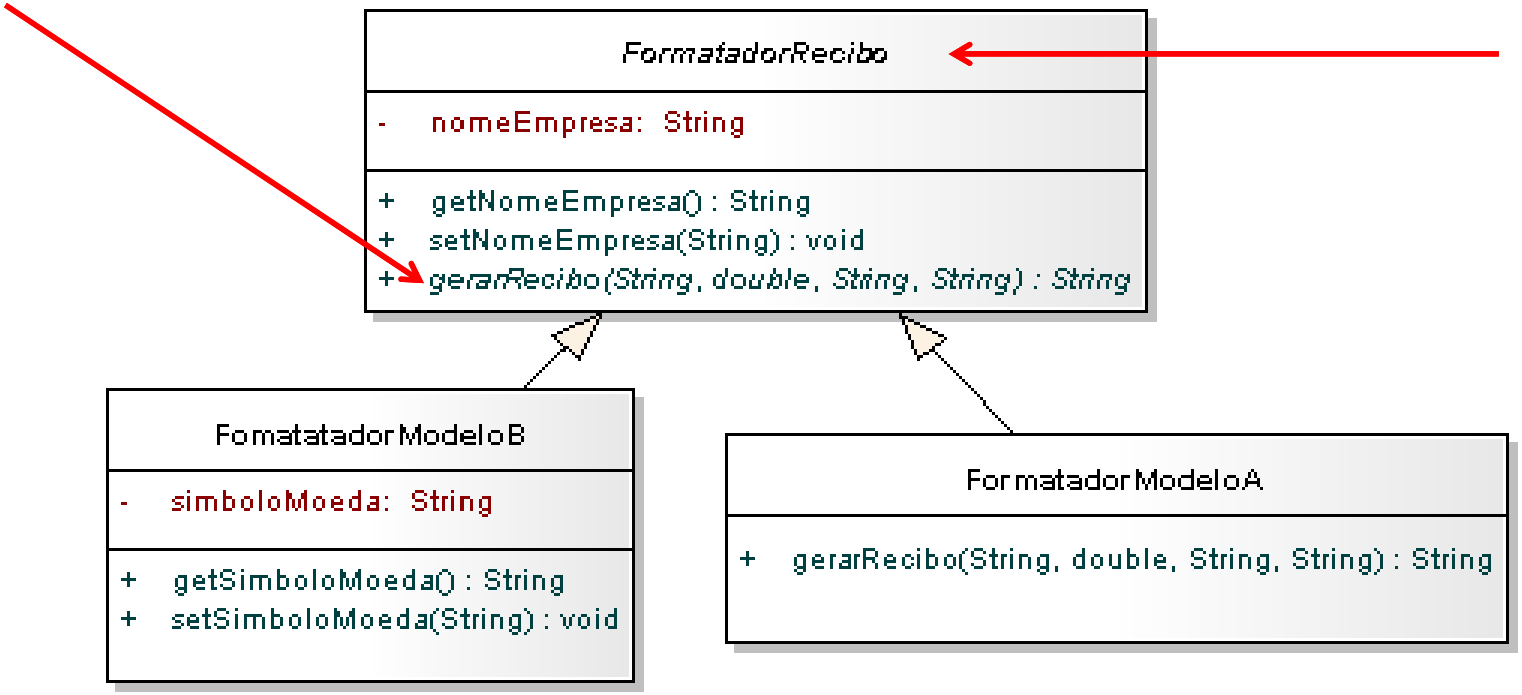
```
public class FomatatadorModeloB extends FormatadorRecibo {

	private String simboloMoeda;

	public String getSimboloMoeda() {
		return simboloMoeda;
	}

	public void setSimboloMoeda(String simboloMoeda) {
		this.simboloMoeda = simboloMoeda;
	}
}
```

javac FomatatadorModeloB .java

FomatatadorModeloB is not abstract and does not override abstract method gerarRecibo(String, double,String,String);

Métodos abstract

Representação UML de métodos e classes abstract são em *itálico*.



Métodos abstract

Erros comuns:

- ✓ declarar métodos abstract e não declarar a classe como abstract;
- ✓ declarar métodos abstract com a implementação vazia com {}
- ✓ Implementar o método na subclasse sem respeitar a assinatura definida na superclasse.

Laboratório final e abstract