

CARACTERÍSTICAS AVANÇADAS DAS CLASSES

OPERADOR STATIC

- Pode definir um membro de classe para ser usado independentemente de qualquer objeto dessa classe.
- Normalmente, o membro de uma classe deve ser acessado por intermédio de um objeto de sua classe.
 - É possível criar um membro para ser usado por conta própria, sem referência a uma instância específica.
- Pode-se declarar tanto métodos quanto variáveis como estáticos.
- O exemplo mais comum de um membro static é main().
 - O método main() é declarado como static, porque deve ser chamado pela JVM quando o programa inicia.
 - Fora da classe, para usar um membro static, deve-se especificar o nome de sua classe seguido pelo operador ponto.
 - Nenhum objeto precisa ser criado.

OPERADOR STATIC

- Especificador static
 - Pode ser aplicado a atributos, métodos e inicializadores.
 - As classes podem ter tanto atributos quanto métodos estáticos.
 - Apenas classes internas podem ser estáticas.
 - Quando aplicado o operador static a um atributo, o mesmo passa a pertencer a classe, passando a ser conhecido como “atributo da classe”.
 - Em outras palavras, todos os objetos instanciados da classe passam a compartilhar o mesmo atributo, similarmente ao que ocorre com variáveis globais em linguagens de programação estrutura

OPERADOR STATIC

- Exemplos de atributos estáticos
 - Vários atributos constantes são definidos em Java como `public static final`.
 - `E` (2.71828...) e `PI` (3.14159...) da classe `Math`.
 - Outro exemplo de variável `public static final` é a variável `out` da classe `System`.

OPERADOR STATIC

- System.out
 - O atributo out está associado à apresentação de caracteres na saída padrão, ou seja, na tela do monitor.
 - `public static final PrintStream out`
 - Uma simples análise dessa declaração mostra que out é um atributo de classe e pode ser acessado na forma System.out.
 - Esse atributo é público.
 - O atributo é final.
 - O Atributo é do tipo PrintStream.

OPERADOR STATIC

- Métodos e atributos estáticos
 - Os métodos estáticos pertencem a classe e não operam sobre instâncias dessa classe.
 - Como os métodos estáticos não funcionam com uma instância da classe, eles só podem acessar membros estáticos (atributos e métodos) da classe.
 - Um atributo estático pode ser inicializado com uma atribuição no momento de sua declaração ou usando um bloco de inicialização estático.
 - Atribuição inicial:
 - `public static double fatorreducao = 1.3;`

OPERADOR STATIC

- Blocos estáticos
 - Uma classe pode precisar de algum tipo de inicialização antes de estar pronta para criar objetos.
 - Por exemplo, ela pode ter que estabelecer uma conexão com um site remoto. Também pode ter que inicializar certas variáveis static antes de seus métodos static serem usados.
 - Para tratar esses tipos de situações, Java permite que você declare um bloco static.
 - Um bloco static é executado quando a classe é carregada pela primeira vez. Portanto, ele é executado antes da classe poder ser usada para qualquer outro fim.

OPERADOR STATIC

- Blocos estáticos - Exemplo

```
// Usa um bloco estático
class StaticBlock {
    static double rootOf2;
    static double rootOf3;

    static { ←————— Esse bloco é
        System.out.println("Inside static block.");
        rootOf2 = Math.sqrt(2.0);
        rootOf3 = Math.sqrt(3.0);
    }
    StaticBlock(String msg) {
        System.out.println(msg);
    }
}
```

```
class SDemo3 {
    public static void main(String args[]) {
        StaticBlock ob = new StaticBlock("Inside Constructor");

        System.out.println("Square root of 2 is " +
            StaticBlock.rootOf2);
        System.out.println("Square root of 3 is " +
            StaticBlock.rootOf3);
    }
}
```

A saída é mostrada abaixo:

```
Inside static block.
Inside Constructor
Square root of 2 is 1.4142135623730951
Square root of 3 is 1.7320508075688772
```


OPERADOR STATIC

- Considerações
 - Normalmente os atributos estáticos não são inicializados no método construtor.
 - Os atributos estáticos normalmente são utilizados na padronização de valores (constantes) dentro do projeto/sistema.
 - Métodos estáticos não podem ser abstratos.
 - Os métodos estáticos normalmente são usados em rotinas isoladas dentro do projeto/sistema.

OPERADOR FINAL

- Se uma classe for declarada final, ela não pode ser uma superclasse (não possuirá herança)
- Todos os métodos de uma classe final são implicitamente finais.
- Mesmo com a sobreposição de métodos e a herança sendo tão poderosas e úteis, podemos querer evitar que ocorram.
- Por exemplo, podemos ter uma classe que encapsule o controle de algum dispositivo de hardware. Além disso, essa classe pode dar ao usuário a oportunidade de inicializar o dispositivo, fazendo uso de informações privadas. Nesse caso, não vamos querer que os usuários de nossa classe possam sobrepor o método de inicialização. Qualquer que seja a razão, em Java, com o uso da palavra-chave final, é fácil impedir que um método seja sobreposto ou uma classe seja herdada.

OPERADOR FINAL

- Características
 - Quando aplicado em classes, não permite estendê-las.
 - Quando aplicado em métodos, impede que o mesmo seja sobrescrito (overriding) na subclasse.
 - Quando aplicado nos atributos, os valores não podem ser alterados depois que já tenham sido atribuídos.

OPERADOR FINAL

- Métodos finais
 - A palavra-chave final impede a sobreposição. Métodos private são implicitamente finais.
 - Para impedir que um método seja sobreposto, especifique final como modificador no início de sua declaração. Métodos declarados como final não podem ser sobrepostos. Observe o código a seguir:

```
class A {  
    final void meth() {  
        System.out.println("This is a final method.");  
    }  
}  
  
class B extends A {  
    void meth() { // ERRO! não pode sobrepor.  
        System.out.println("Illegal!");  
    }  
}
```

Já que meth() é declarado como final, não pode ser sobreposto em B. Se você tentar fazê-lo, ocorrerá um erro de tempo de compilação.

OPERADOR FINAL

- Atributos finais
 - A atribuição de valor pode ocorrer diretamente na declaração do atributo.
 - É possível ter atributos de uma classe que sejam finais mas não recebem valor na declaração.
 - Esses atributos finais recebem seus valores iniciais nos construtores da classe, são os chamados “blank final variable”.
 - maior grau de flexibilidade na definição de constantes para atributos de uma classe
 - A definição deve obrigatoriamente ocorrer em uma das duas formas.
 - A utilização de final para uma referência a objetos/vetores é permitida.
 - Os objetos/vetores em geral podem ser modificados, apenas a referência é fixa.
 - O operador final também pode ser aplicado aos parâmetros de um método que não devem ser modificados.

OPERADOR FINAL

- Atributos finais - Exemplo

```
// Retorna um objeto String.
class ErrorMsg {
    // Códigos de erro.
    final int OUTERR    = 0;
    final int INERR     = 1; ← Declara constantes final.
    final int DISKERR   = 2;
    final int INDEXERR  = 3;


    String msgs[] = {
        "Output Error",
        "Input Error",
        "Disk Full",
        "Index Out-Of-Bounds"
    };

    // Retorna a mensagem de erro.
    String getErrorMsg(int i) {
        if(i >=0 & i < msgs.length)
            return msgs[i];
        else
            return "Invalid Error Code";
    }
}
```

```
class FinalD {
    public static void main(String args[]) {
        ErrorMsg err = new ErrorMsg();

        System.out.println(err.getErrorMsg(err.OUTERR));
        System.out.println(err.getErrorMsg(err.DISKERR));
    }
}
```

Usa constantes **final**.



Observe como as constantes final são usadas em main(). Uma vez que são membros da classe ErrorMsg, devem ser acessadas via um objeto dessa classe. É claro que também podem ser herdadas pelas subclasses e acessadas diretamente dentro delas.

Por uma questão estilística, muitos programadores de Java usam identificadores maiúsculos em constantes final, como no exemplo anterior, mas essa não é uma regra fixa.

OPERADOR FINAL

- Classes finais
 - Se uma classe for declarada final, ela não pode ser uma superclasse (não possuirá herança)
 - Todos os métodos de uma classe final são implicitamente finais.
 - Você pode impedir que uma classe seja herdada precedendo sua declaração com final. A declaração de uma classe como final também declara implicitamente todos os seus métodos como final. Como era de se esperar, é inválido declarar uma classe como abstract e final, uma vez que uma classe abstrata é individualmente incompleta e depende de suas subclasses para fornecer implementações completas. Aqui está um exemplo de uma classe final:

```
final class A {  
    // ...  
}  
  
// A classe seguinte é inválida.  
class B extends A { // ERRO! não pode criar uma subclasse de A  
    // ...  
}
```

Como os comentários sugerem, é inválido B herdar A, já que A é declarada como final.

OPERADOR FINAL

- Considerações finais
 - O uso do modificador final inibe o uso dos recursos OO: herança e polimorfismo.
 - Importante por razões de segurança e/ou desempenho, sendo que as chamadas a métodos final são substituídos por suas definições (técnica de inclusão de código inline).
 - `const` é uma palavra chave reservada da linguagem Java, mas não é usada para nada. Deve ser usada a palavra final para definir uma constante.

QUESTÃO CONCLUSIVA

- Variáveis membros final podem ser transformadas em static? A palavra-chave final pode ser usada em parâmetros de métodos e variáveis locais?
- A resposta às duas perguntas é sim. Transformar uma variável membro final em static permite que você referencie a constante pelo nome de sua classe em vez de por um objeto. Por exemplo, se as constantes de `ErrorMsg` fossem modificadas por static, as instruções `println()` de `main()` teriam esta aparência:

```
System.out.println(err.getErrorMsg(ErrorMsg.OUTERR));  
System.out.println(err.getErrorMsg(ErrorMsg.DISKERR));
```

- A declaração de um parâmetro final impede que ele seja alterado dentro do método. A declaração de uma variável local final impede que ela receba um valor mais de uma vez

REFERÊNCIAS

- SCHILDT, H. Java para iniciantes. [s. l.], 2015. Disponível em: <http://search.ebscohost.com/login.aspx?direct=true&db=edsmib&AN=edsmib.00007154&lang=pt-br&site=eds-live&scope=site>.
- TURINI, Rodrigo. **Desbravando Java e orientação a objetos**: um guia para o iniciante da linguagem. São Paulo, SP: Casa do Código, [2017]. 222 p.
- DEITEL, Paul J.; DEITEL, Harvey M. **Java, como programar**. 10. ed. São Paulo, SP: Pearson Prentice Hall, 2017. xxix, 1144 p. ISBN 9788543004792.