Membros Estáticos

Prof. Ms. Peter Jandl Junior

Prof. Ms. Télvio Orru

Prof. Nathan Silva

J12B

Linguagem de Programação Orientada a Objetos

Ciência da Computação - UNIP – Jundiaí

Static::conceitos e aplicação

- Esta apresentação contém um detalhamento dos membros estáticos, isto é, daqueles declarados com o modificador *static*, incluindo uma conceituação mais precisa, restrições de uso e suas aplicações.
- Também aproveita para introduzir a noção de padrões de projeto (*design patterns*) com a discussão do **Singleton**.

Modificador ***static***

Usos e restrições

Importação estática

Padrões de Projeto

MODIFICADOR **static**

Estrutura dos Programas

- Sabemos que os programas Java são compostos de unidades de compilação (arquivos com código Java).
- Cada arquivo pode conter:
  - **[0 | 1]** declaração de pacote;
  - **[0 | m]** diretivas de importação;
  - **[0 | nc]** declaração de classes;
  - **[0 | ni]** declaração de interfaces;
  - **MAS apenas UM elemento PÚBLICO!**
    public(ni) +public(nc) = 1
- Os arquivos fonte, de extensão *.java*, **devem** possuir o mesmo nome do elemento público (classe ou interface ou enumeração).

Programas Java

- Para que os programas possam ser iniciados autonomamente, uma de suas classes deve conter o método **main**:
  - **public static void main(String [])**
  - que é o início *convencionado* dos programas (seu ponto de entrada ou *entry-point*).
- É possível que várias classes de um programa contenham um *main* (ou seja, vários *entry-points* ou inícios para a aplicação).

É como na linguagem C/C++!

Método main

- É convencionado (obrigatório) que seja:
  - acesso: public
  - tipo: **static**
  - retorno: void
  - nome: main
  - argumentos: (String args[])

Para possibilitar acesso externo à classe.

Para evitar que JVM necessite instanciar objetos.

Não retorna valor (para JVM).

Nome convencional

Para receber 0, 1 ou N argumentos da linha de comando.

Especificadores e Modificadores

- A linguagem Java possui **especificadores** e **modificadores** que afetam como os membros de uma classe são utilizados.
- Especificadores:
  - public, protected, private.
- Modificadores:
  - abstract, final, synchronized, transient, volatile, strictfp, native e static.

Estabelecem a visibilidade do membro.

Determinam o funcionamento do membro e como se dá a geração do seu código.

Modificador	Descrição	Aplicação
`abstract`	Indica que classe não é concreta (não pode ser instanciada), pois contém membros não implementados, mas cuja interface já está definida.	Herança
`final`	Indica que classe ou membro particular de uma classe não pode ser modificado por meio da herança.	Herança
`native`	Indica ao compilador que método é implementado em outra linguagem de programação (C, C++, assembly) de maneira específica da plataforma.	JNI
`static`	Define quando um membro de uma classe passa a pertencer a classe em si, não precisando ser acessado por variáveis de instância.	Geral
`strictfp`	Indica ao compilador que código gerado deve aderir estritamente à especificação para cálculos numéricos, garantindo resultados idênticos mesmo em plataformas diferentes.	Geral
`synchronized`	Indica que método só pode ser acessado por uma thread de cada vez, exigindo o uso de um monitor por parte da JVM.	Threads
`transient`	Indica que campo referido não deve ser serializado.	Serialização
`volatile`	Indica campos que podem ser modificados por várias threads simultâneas, habilitando a manutenção de cópias de trabalho em cada thread e evitando otimizações inadequadas.	Threads

Membros de instância

- Uma classe pode declarar vários membros:
  - Variáveis-membro (campos ou atributos) e
  - Funções-membro (métodos).
- Estes membros, comuns, só podem ser utilizados pelas instâncias da classes, ou seja:
  - Cada objeto instanciado tem cópias exclusivas das variáveis membro para seu uso individual;
  - As funções-membro acionadas pelos objetos instanciados utilizam apenas as variáveis-membro de suas respectivas instâncias.
- Assim, tais membros são denominados ***membros de instância***.

Membros de instância

```
public class Point {

    public double x;
    public double y;

    public void reflectX() {
        y = - y;
    }

    public void reflectY() {
        x = - x;
    }

}
```

- Instanciação:

  Point p1 = new Point();
  Point p2 = new Point();

- Uso:
  - p1.x = 5.5; p1.y = 3.0;
    p2.x = -1.5; p2.y = 6.7;
  - p1.reflectY();
    p2.reflectX();

Cada instância armazena seu próprios valores.

Operações numa instância não afeta a outra.

Modificador static

- Quando um membro de uma classe (variável ou método) é declarado como **static**, ele deixa de pertencer às instâncias e passa a pertencer a própria classe.
  - Variável-membro ou campo → variável de classe ou campo estático
  - Método de instância ou operação → Método de classe ou método estático
- Não afetam os especificadores de acesso.
- Construtores não podem ser declarados estáticos.

Membros estáticos

```
public class Cruise {

    public static double MAX;
    public double velocity;

    public boolean overMax () {
        return velocity > MAX;
    }

    public static double
            maxRate(double v) {
        return v/MAX;
    }

}
```

Uso de campo e método estático é feito qualificando nome da classe!

- Instanciação:

  Cruise c1 = new Cruise();
  Cruise c2 = new Cruise();

- Uso:
  - Cruise.MAX = 100;
  - c1.velocity = 81.5;
    c2.velocity = 105.7;
  - c1.overMax();
    c2.overMax();
  - Cruise.maxRate(120);

Campo estático é comum a todas as instância!

Cada instância armazena seu próprios valores.

RESTRIÇÕES E USOS DE MEMBROS **static**

Restrições de membros **static**

- O uso do modificador **static** acarreta em um conjunto de implicações que determinam a forma com que pode ser explorado.
- As principais restrições são:
  - Acesso qualificado pela classe.
  - Membros estáticos estão atrelados à suas classes.
  - Dependência junto a outros membros estáticos.
  - Construtores não podem ser estáticos.

Modificador **static**::acesso

- **Primeira restrição**:
  - **Não são necessárias instâncias da classe (i.e., objetos) para permitir o acesso à membros estáticos**.
  - Acesso passa a ser qualificado pela **classe**.
    Math.pow(double, double)
    String.format(String, Object ...)
  - Instâncias têm acesso aos membros estáticos sem restrições adicionais (ou seja, respeitando os especificadores de acesso combinados).

Modificador **static**::compartilhamento

- **Segunda restrição**:
  - **Os membros estáticos pertencem à classe, ou seja, eles são únicos** e não dependem da existência de instâncias da classe (i.e., objetos).
  - Assim todos as instâncias tem acesso aos membros estáticos únicos associados às suas próprias classes.
  - Torna-se possível compartilhar valores entre instâncias por meio de variáveis estáticas (é como se fossem variáveis "*globais*" de um tipo específico).
  - Métodos estáticos podem oferecer operações baseadas em valores comuns (compartilhados) entre as instâncias de uma classe.

Modificador **static**::dependência

- **Terceira restrição**:
  - **Métodos estáticos não podem depender de membros que não sejam estáticos.**
  - Um atributo estático só pode ser inicializado com:
    - Valor literal;
    - Constante (valor estático);
    - Valor extraído de instância imediata.
  - Um método estático só pode depender de:
    - Outros métodos estáticos;
    - Valores literais e constantes;
    - Valores e métodos extraídos de instâncias imediatas.

Modificador **static**::construtrores

- **Quarta restrição**:
  - **Construtores não podem ser declarados estáticos.**
  - Isto ocorre pela natureza de um construtor, que é criar uma nova instância de objetos.
  - Além disso, os construtores públicos são acessíveis diretamente, sem necessidade de qualificação ou de instâncias de qualquer tipo, o que, sob certos aspectos, torna redundante a declaração *static*.

Usos comuns de membros **static**

- O uso do modificador **static** é, em muitas situações, conveniente e útil.
- Os principais usos são:
  - **Compartilhamento de informação** entre instâncias de uma mesma classe.
  - Criação simplificada de **biblioteca de funções**.
  - **Definição de constantes**.

Modificador **static**::acesso

Classe

```
public class Estatica {
    public static int val = 0;
    public static void reset() {
        val = 0;
    }
}
```

Aplicação

- Uso de campo:
  - Estatica.val = 33;
- Uso de método:
  - Estatica.reset();

Modificador **static**::compartilhamento

Classe

```
public class Estatica2 {
   private static int n = 0;

   public Estatica2 () {
      n++;
   }
   protected void finalize() {
      n--;
   }
   public static int instances() {
      return n;
   }
}
```

Aplicação

- Uso da classe:

```
System.out.println(
        Estatica2.instances() );
Estatica2 obj1 =
        new Estatica2();
System.out.println(
        Estatica2.instances() );
Estatica2 obj2 =
        new Estatica2();
System.out.println(
        Estatica2.instances() );
obj1 = null; // GC pode atuar!
System.out.println(
        Estatica2.instances() );
```

Exibe 0

Exibe 1

Exibe 2

Exibe 1 ou2, conforme atuação GC.

Modificador **static**::serviços

- Um conjunto de métodos estáticos cujas operações pertençam a um mesmo domínio podem ser agrupados em uma única classe, que comporta-se como uma biblioteca de funções.
- Exemplo:
  - Classe java.lang.**Math**
    Só contém métodos estáticos que oferecem funções matemáticas independentes.
  - Classe javax.swing.**BorderFactory**
    Contém vários métodos-fábrica que podem ser acionados sem uma instância da fábrica, retornando novos objetos de maneira conveniente.

Modificador **static**::bibliotecas

Classe

```
public class Conversao {
// conversão C → F
   public static double
          C2F (double c) {
      return 9*c/5 + 32;
   }
// conversão F → C
   public static double
          F2C (double f) {
      return 5*(f-32)/9;
   }
}
```

Aplicação

- Uso da classe:

```
double x = 100;
double y= Conversao.C2F(x);
System.out.println( "C = " + x +
        "=> F = " + y);
y = 32;
x = Conversao.F2C(y);
System.out.println("F = " + y +
        " => C = " + x);
System.out.println( "C = 24" +
        "=> F = " +
        Conversao.C2F(24) );
```

Modificador **static**::constantes

- Java não contém um modificador próprio para definição de constantes (como ***const*** em linguagem C++).
- Efeito pode ser obtido pela combinação de modificadores na declaração de variáveis:
  - **static** → indica que membro pertence à classe
  - **final** → indica que membro não pode ser alterado, nem mesmo em subclasses.

Modificador **static**::constantes

- Exemplos:
  - public **static final** double PI = 3.1415926;
  - public **final static** int MAX = 123;
  - **static** public **final** byte MASK = 0b01001001;

Inicialização obrigatória na declaração!

A ordem dos modificadores e especificadores não altera o efeito produzido!

- A convenção de código Java solicita que os nomes das constantes seja grafado exclusivamente com letras maiúsculas e uso de _ (*underline*) para separação de palavras.

IMPORTAÇÃO ESTÁTICA

Importação estática

- Introduzida na versão 5, permite uso de constantes declaradas em classes externas sem necessidade de sua qualificação (indicação do nome da classe).
- Exemplo:
  - Classe Math contém constantes usualmente empregadas (de forma qualificada) como:
    - **Math**.PI
    - **Math**.E
- Esta característica não tem impacto expressivo no código.

Importação estática::aplicação

Sem importação estática

```
import java.lang.Math;

public class SemImportStatic {

public static void main(String args[]) {

   double raio =
            Double.parseDouble(args[0]);

   // uso da constante qualificada

   double perim = 2 * Math.PI * raio;

   System.out.println("raio = " + raio +
            ", perim = " + perim);

}}
```

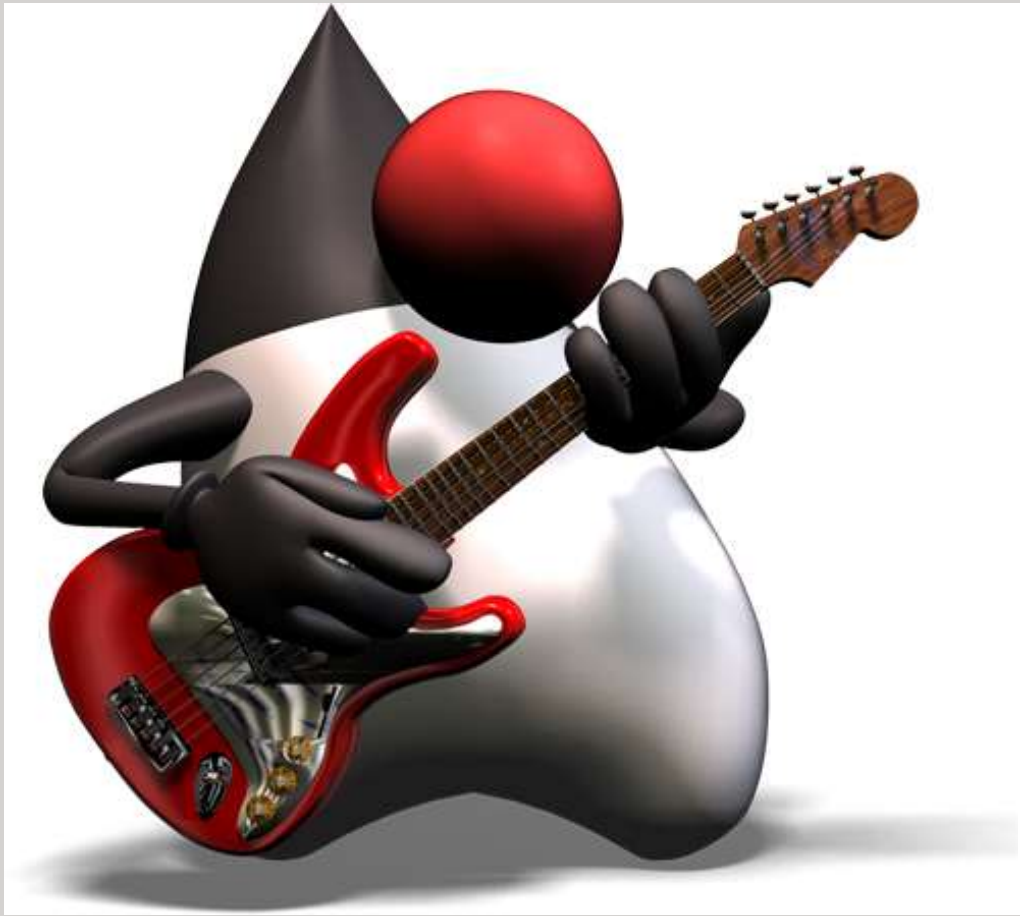
Com importação estática

```
import static java.lang.Math.*;

public class ComImportStatic {

public static void main(String args[]) {

   double raio =
            Double.parseDouble(args[0]);

   // uso abreviado da constante

   double perim = 2 * PI * raio;

   System.out.println("raio = " + raio +
            ", perim = " + perim);

}}
```

RECOMENDAÇÕES DE ESTUDO

A prática
(leva primeiro ao cansaço, mas depois)
conduz à perfeição!

Recomendações de Estudo



- **Estudar** com livros:
  - Java – Guia do Programador, 3ª Ed, P. JANDL Jr, Novatec, 2015.
  - Java: Como Programar, 6ª Ed, DEITEL, H. & DEITEL, P., Pearson, 2007.
  - Mais Java, P. JANDL Jr., Futura, 2003.