



Membros de Classes

Eduardo Figueiredo

<http://www.dcc.ufmg.br/~figueiredo>

[Membros de Uma Classe]

- Construtor
- Métodos
 - De classe
 - De objeto
- Variáveis
 - De classe
 - De Objeto
- Constantes



Construtores de Objetos

[Construtor Padrão]

- Toda classe deve ter pelo menos um construtor
 - Responsável pela criação do objeto
 - A chamada do construtor deve ser precedida pela palavra reservada *new*
- Se nenhum construtor for declarado, o compilador adiciona automaticamente o construtor padrão
 - Construtor vazio e sem parâmetros

[Exemplo: Time1]

```
public class Time1 {
```

```
    private int hour; // 0 - 23
```

```
    private int minute; // 0 - 59
```

```
    private int second; // 0 - 59
```

```
    public void setTime( int h, int m, int s ) {
```

```
        hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
```

```
        minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
```

```
        second = ( ( s >= 0 && s < 60 ) ? s : 0 );
```

```
    }
```

```
    public String toString() {
```

```
        return ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ) + ":" +  
                minute + ":" + second + " " + ( hour < 12 ? "AM" : "PM" );
```

```
    }
```

```
}
```

**Construtor vazio da classe
não precisa ser declarado.**

```
public Time1 ( ) {  
}
```

[Exemplo: Time1Test]

```
public class Time1Test {  
    public static void main( String[] args ) {  
        Time1 time = new Time1();  
  
        System.out.print( "Hora inicial: " );  
        System.out.println( time.toString() );  
        System.out.println();  
  
        time.setTime( 13, 27, 6 );  
        System.out.print( "Hora apos setTime: " );  
        System.out.println( time.toString() );  
        System.out.println();  
  
        time.setTime( 99, 99, 99 );  
        System.out.println( "Apos definir hora invalida." );  
        System.out.print( "Hora: " );  
        System.out.println( time.toString() );  
    }  
}
```

**Chamada ao
construtor padrão
cria um objeto Time1.**

[Construtores Sobrecarregados]

- Pode-se definir vários construtores para uma classe
 - Construtores devem ter assinaturas diferentes (parâmetros)
- Se for declarado algum construtor, o vazio só existe quando declarado
 - Construtor vazio só não precisa ser declarado se não existe outro construtor

[Exemplo: Time2]

```
public class Time2 {  
    public Time2() {  
        this( 0, 0, 0 );  
    }  
    public Time2( int h ) {  
        this( h, 0, 0 );  
    }  
    public Time2( int h, int m ) {  
        this( h, m, 0 );  
    }  
    public Time2( int h, int m, int s ) {  
        setTime( h, m, s );  
    }  
    public Time2( Time2 time ) {  
        this( time.getHour(), time.getMinute(), time.getSecond() );  
    }  
    ...  
}
```

**Construtores da
classe Time2.**


[Exemplo: Time2]

```
public class Time2 {  
    public Time2() {  
        this( 0, 0, 0 );  
    }  
    public Time2( int h ) {  
        this( h, 0, 0 );  
    }  
    public Time2( int h, int m ) {  
        this( h, m, 0 );  
    }  
    public Time2( int h, int m, int s ) {  
        setTime( h, m, s );  
    }  
    public Time2( Time2 time ) {  
        this( time.getHour(), time.getMinute(), time.getSecond() );  
    }  
    ...  
}
```

Se outros construtores forem declarados, o construtor padrão (sem argumentos) não é fornecido automaticamente.

[Exemplo: Time2]

```
public class Time2 {  
    public Time2() {  
        this( 0, 0, 0 );  
    }  
    public Time2( int h ) {  
        this( h, 0, 0 );  
    }  
    public Time2( int h, int m ) {  
        this( h, m, 0 );  
    }  
    public Time2( int h, int m, int s ) {  
        setTime( h, m, s );  
    }  
    public Time2( Time2 time ) {  
        this( time.getHour(), time.getMinute(), time.getSecond() );  
    }  
    ...  
}
```



A referência *this* indica a chamada do construtor correspondente.

Exemplo: Time2Test

```
public class Time2Test {  
    public static void main( String[] args ) {  
        Time2 t1 = new Time2();  
        Time2 t2 = new Time2( 2 );  
        Time2 t3 = new Time2( 21, 34 );  
        Time2 t4 = new Time2( 12, 25, 42 );  
        Time2 t5 = new Time2( 27, 74, 99 );  
        Time2 t6 = new Time2( t4 );  
    }  
}
```

**Criação de 6 objetos
usando diferentes
construtores.**

```
System.out.print( "t1: " ); System.out.println( t1.toString() );  
System.out.print( "t2: " ); System.out.println( t2.toString() );  
System.out.print( "t3: " ); System.out.println( t3.toString() );  
System.out.print( "t4: " ); System.out.println( t4.toString() );  
System.out.print( "t5: " ); System.out.println( t5.toString() );  
System.out.print( "t6: " ); System.out.println( t6.toString() );
```



Membros Estáticos e Constantes

[Membros Estáticos]

- Variável de Classe
 - Usada quando apenas uma variável é compartilhada pelos objetos da classe
- Método de Classe
 - Funções que podem ser executadas independente de um objeto
- Variáveis e métodos estáticos podem ser usados mesmo sem nenhum objeto

[Exemplo: Employee]

```
public class Employee {  
    private String firstName;  
    private String lastName;  
    protected static int count = 0;
```

Variável de classe conta o número de objetos Employee.

```
    public Employee( String first, String last ) {  
        firstName = first;  
        lastName = last;  
        ++count;  
    }
```

Um método estático só pode acessar diretamente membros estáticos.

```
    public String getFirstName() { return firstName; }  
    public String getLastName() { return lastName; }
```

```
    public static int getCount() {  
        return count;  
    }  
}
```

Exemplo: EmployeeTest

```
public class EmployeeTest {  
    public static void main( String[] args ) {  
        System.out.println( "Inicio do programa, count: " + Employee.count);  
        Employee e1 = new Employee( "Susan", "Baker" );  
        System.out.println( "Criou Susan" );  
        System.out.println( "e1.count: " + e1.count);  
  
        Employee e2 = new Employee( "Bob", "Blue" );  
        System.out.println( "Criou Bob" );  
        System.out.println( "e2.count: " + e2.count);  
  
        System.out.println( "" );  
        System.out.println( "Employee.count: " + Employee.count);  
        System.out.println( "e1.count: " + e1.count);  
        System.out.println( "e2.count: " + e2.count);  
    }  
}
```

Cria Employee Susan.

Cria Employee Bob.

Exemplo: EmployeeTest

```
public class EmployeeTest {  
    public static void main( String[] args ) {  
        System.out.println( "Inicio do programa, count: " + Employee.count);  
        Employee e1 = new Employee( "Susan", "Baker" );  
        System.out.println( "Criou Susan" );  
        System.out.println( "e1.count: " + e1.count);
```

Os valores são o mesmos (2) porque os objetos compartilham a mesma variável *count*

```
        System.out.println( "" );  
        System.out.println( "Employee.count: " + Employee.count);  
        System.out.println( "e1.count: " + e1.count);  
        System.out.println( "e2.count: " + e2.count);
```

```
    }
```

```
}
```


[Constantes]

- O modificador *final* é usado para indicar constantes do programa
 - Os valores não podem ser modificados
- O uso do modificador *final* torna o programa mais robusto
 - Evita que o valor seja acidentalmente modificado

[Exemplo: Increment]

```
public class Increment {  
    private int total = 0;  
    private final int INCREMENT;
```

```
    public Increment( int incrementValue ) {  
        INCREMENT = incrementValue;  
    }
```

```
    public void addIncrementToTotal() {  
        total += INCREMENT;  
    }
```

```
    public String toString() {  
        return "total = " + total;  
    }  
}
```

A padronização do nome de constantes é MAIUSCULO.

O valor de uma constante só pode ser alterado no construtor.

Constantes podem ser acessadas da mesma forma que variáveis.



[Exemplo: IncrementTest

```
public class IncrementTest {  
    public static void main( String[] args ) {  
        Increment value = new Increment( 5 );  
  
        System.out.println( "Before incrementing: " + value );  
  
        for ( int i = 1; i <= 3; i++ ) {  
            value.addIncrementToTotal();  
            System.out.println( "After increment " + i + ": " + value );  
        }  
    }  
}
```

[Método *finalize*]

- O método *finalize* está disponível em todo objeto
 - Ele pertence a classe *Object*, que toda classe estende direta ou indiretamente
- Antes de um objeto ser descartado pelo coletor de lixo, o método *finalize* é chamado
 - Este método pode ser sobrescrito por qualquer objeto (**o que deve ser evitado**)

[Bibliografia da Aula]

- DEITEL, H. M.; DEITEL P. J. **Java: Como Programar**, 8a. Edição. Pearson, 2010.
 - Capítulo 8 Classes e Objetos
 - Capítulo 9 Herança