

Construtores, mais sobre métodos, APIs e bibliotecas

Maurício Linhares

Variáveis estáticas

- ▶ Variáveis estáticas são variáveis que pertencem a uma classe, em vez de a um objeto;
- ▶ Não é necessário ter um objeto da classe para acessar uma variável estática dela, o acesso é feito através da própria classe;
- ▶ Variáveis estáticas que não podem ser alteradas (estão marcadas como “**final**” funcionam como constantes em Java;



Exemplo de declaração de variável estática

```
public class Statics {
```

```
    public static final float PI = 3.14F;
```

```
    public static SimpleDateFormat format =  
        new SimpleDateFormat("dd/MM/yyyy");
```

```
}
```



Usando variáveis estáticas

```
public class ExemploDeStatics {  
  
    public static void main(String[] args) {  
        System.out.println( Statics.PI );  
        Statics.PI = 3.12F; // ?  
    }  
  
}
```



Métodos estáticos

- ▶ São os métodos que pertencem a uma classe e não a um objeto em específico;
- ▶ Métodos estáticos são definidos através da adição do modificador “static” após a definição do nível de visibilidade do método;
- ▶ Os métodos estáticos são invocados através do nome da classe aonde eles foram definidos, como por exemplo em “System.currentTimeMillis()”;



Exemplo de método estático

```
public class Statics {  
  
    public static SimpleDateFormat format =  
        new SimpleDateFormat("dd/MM/yyyy");  
  
    public static void imprimirData( Date data ) {  
        System.out.println( format.format( data ) );  
    }  
  
}
```



Usando um método estático

```
public class ExemploDeStatics {  
  
    public static void main(String[] args) {  
  
        System.out.println( Statics.PI );  
        Statics.imprimirData( new Date() );  
  
    }  
  
}
```



Dando nomes aos bois (ou como escrever código Java em Java)

- ▶ Todos os nomes de métodos, variáveis de instância, locais ou estáticas devem ser escritos com a primeira letra em minúsculo, se o nome for composto por mais do que uma palavra, a primeira letra da nova palavra deve ser escrita com maiúsculo;
 - ▶ String nome
 - ▶ public void meuMetodo()
 - ▶ boolean casoEncerrado
 - ▶ **int variavel_com_nome_bizarro**
 - ▶ **String str_nome**
-



Dando nomes aos bois

- ▶ Constantes (variáveis **static** e **final**) devem ser escritas com o nome todo em maiúsculas e se o nome for formado por diversas palavras, cada palavra deve ser separada por um sublinhado (“_”);
 - ▶ `public static final float PI`
 - ▶ `public static final int DAY_OF_MONTH`
 - ▶ `public static final int DAY_OF_YEAR`



Dando nomes aos bois

- ▶ Nomes de classes devem sempre começar com uma letra maiúscula e se eles forem compostos por mais do que uma palavra, a primeira letra de cada palavra também deve ser escrita em maiúsculo;
- ▶ String
- ▶ DateFormat
- ▶ LinkedHashSet
- ▶ CopyOnWriteArrayList



Operadores de comparação

- ▶ São os operadores da linguagem que geram como resultado os valores TRUE ou FALSE;
- ▶ Em Java existem diversos operadores que podem ser utilizados para comparações booleanas;
- ▶ Todos esses operadores funcionam para referências e tipos primitivos, eles nem sempre funcionam da mesma forma para objetos;



Quais são os operadores de comparação?

- ▶ ==

- ▶ Compara se dois primitivos são iguais ou se duas referências apontam para o mesmo objeto;

- ▶ !=

- ▶ O contrário do operador ==

- ▶ >, >=, < e <=

- ▶ Fazem a comparação de maior, maior ou igual, menor e menor ou igual, respectivamente, apenas para os tipos primitivos numéricos;

- ▶ !

- ▶ Operador de negação, inverte qualquer resultado booleano;



Exemplo

```
System.out.println( 1 == 2 );
```

```
System.out.println( 'z' != 'a' );
```

```
System.out.println( 8 > 10 );
```

```
System.out.println( 5 >= 5 );
```

```
System.out.println( "abc" == "bcd" );
```



Operadores booleanos

- ▶ São operadores que funcionam apenas para comparar valores booleanos;
- ▶ Eles são utilizados normalmente em operações condicionais (como ifs e whiles);
- ▶ Existem em versões simples (“|” e “&”) e em versões curto-circuitadas (“||” e “&&”);



Exemplo

```
String vazia = null;
```

```
if ( vazia != null && vazia.length() > 0 ) {  
    System.out.println("A string não é  
    vazia");  
}
```

```
vazia = null;
```

```
if ( vazia != null & vazia.length() > 0 ) {  
    System.out.println("A string não é  
    vazia");  
}
```



Qual a diferença entre os simples e os curto circuitados?

- ▶ Os operadores curto-circuitados podem parar uma comparação ainda na primeira avaliação:
 - ▶ Se em um `&&` o lado esquerdo for “false” toda a expressão torna-se falsa automaticamente;
 - ▶ Se em um `||` o lado esquerdo for “true”, toda a expressão torna-se verdadeira automaticamente;
- ▶ Os operadores que não são curto-circuitados sempre avaliam os dois lados da operação;



As coleções (ou **como fazer um array que aumenta e diminui**)

- ▶ As Coleções são estruturas de dados que podem ser utilizadas no lugar dos Arrays para guardar conjuntos de objetos;
- ▶ Existem diversos tipos diferentes de coleções para os mais diversos tipos de necessidade, desde **Listas** (que funcionam como um array dinâmico) até conjuntos que não permitem duplicatas;
- ▶ Todas as coleções ficam dentro do pacote “java.util” e quase todas elas implementam a interface Collection;



Quais são os objetos que implementam as listas?

- ▶ **java.util.ArrayList**

- ▶ Objeto que implementa a lista através do uso de um Array interno (e invisível) para os seus usuários;

- ▶ **Java.util.LinkedList**

- ▶ Objeto que implementa a lista na forma de uma lista encadeada (como uma fila de pessoas, primeiro a entrar, é o primeiro a sair)



Usando os objetos Lista

```
List<String> strings =  
    new ArrayList<String>() ;  
strings.add("Maurício") ;  
strings.add("José") ;  
strings.add("Carol") ;  
  
System.out.println( strings ) ;
```



Métodos comuns em objetos lista

- ▶ **add(int, Object)**
 - ▶ Adiciona um objeto no índice especificado
- ▶ **add(Object)**
 - ▶ Adiciona um objeto na última posição da lista
- ▶ **get(int)**
 - ▶ Pega o objeto que estiver na posição passada como parâmetro da lista
- ▶ **size()**
 - ▶ Diz a quantidade de itens que existem na lista
- ▶ **remove(Object)**
 - ▶ Remove o objeto passado como parâmetro da lista;
- ▶ **indexOf(Object)**
 - ▶ Diz o índice no qual o objeto passado como parâmetro se encontra, ou -1 se ele não estiver na lista;



Pausa para os comerciais

- ▶ Se você comparar dois objetos usando o operador `==` a única coisa que você vai saber é se as duas referências apontam pro mesmo objeto, não se os objetos tem os mesmos atributos;
- ▶ Cada objeto tem a sua própria lógica para definir quando é que ele é igual a um outro objeto qualquer;
- ▶ Para definir qual é essa lógica de equivalência, os objetos em Java implementam o método `equals()`;



Como implementar o método equals()?

- ▶ Primeiro, definir quais as características definem um objeto como único:
- ▶ Em um cliente, o CPF deve ser único, não pode haver mais do que um cliente com um CPF, outras características como o nome podem ser repetidas, mas você deve avaliar cada caso;
- ▶ Métodos equals devem ser seguros quanto a passagem de valores nulos ou incorretos;
- ▶ Ao implementar equals, você está definindo as regras de equivalência entre dois objetos;



Exemplo de um método equals

```
public boolean equals(Object obj) {  
    boolean result = false;  
  
    if ( obj instanceof Cliente ) {  
        Cliente cliente = (Cliente) obj;  
        if ( this.cpf == null ) {  
            result = cliente.cpf == null;  
        } else {  
            result = this.cpf.equals(cliente.cpf);  
        }  
    }  
  
    return result;  
}
```



Algumas linhas especiais

▶ **if (obj instanceof Cliente)**

- ▶ O operador `instanceof` compara se o objeto que está a esquerda é da classe ou de uma subclasse da classe (ou tipo) a direita;
- ▶ Ele retorna `true` se a comparação for verdadeira e `false` se ela não for verdadeira ou se o valor da referência for `null`;
- ▶ Nós usamos esse operador para ter certeza dos tipos com os quais estamos lidando;



Algumas linhas especiais

- ▶ `Cliente cliente = (Cliente) obj;`
 - ▶ O nome da classe entre parênteses “(Cliente)” denota uma operação chamada de “cast”;
 - ▶ Um cast acontece quando você tem uma referência de uma classe mais genérica e quer colocar esse objeto em uma referência mais específica (como colocar um `Object` em um `cliente`);
 - ▶ Se a operação de cast não for possível, uma exceção vai ser lançada quando o código estiver executando;
 - ▶ Sempre proteja operações de cast com um `instanceof` quando você não tiver certeza do que está vindo;



Usando o nosso equals() em uma coleção

```
Cliente cliente = new  
    Cliente( "Maurício",  
        "000.000.000-00" );
```

```
Cliente outroCliente = new  
    Cliente( "José", "000.000.000-00" );
```

```
List<Cliente> clientes = new  
    ArrayList<Cliente>();
```

```
clientes.add( cliente );
```

```
System.out.println( clientes.contains( o  
    outroCliente ) );
```



Mas ainda não estamos terminados...

- ▶ Além de implementar o método `equals` nos nossos objetos, também é necessário implementar o método `hashCode()`;
- ▶ O método `hashCode` serve para definir um identificador para os nossos objetos de forma que eles possam ser utilizados em tabelas de espalhamento (???)
- ▶ A documentação do método `hashCode` diz que sempre que dois objetos são `equals` eles devem ter, obrigatoriamente, o mesmo `hashCode`, mas dois objetos que tem o mesmo `hashCode` não precisam ser `equals`;



Exemplo de uma implementação de hashCode

```
public int hashCode() {  
    return  
        this.cpf == null ?  
            super.hashCode() :  
            this.cpf.hashCode();  
}
```



Ainda sobre hashCode

- ▶ Normalmente não é necessário implementar o método hashCode diretamente, as classes base do Java já tem implementações que podem ser utilizadas;
- ▶ Não implementar o método hashCode pode levar a problemas na hora de se utilizar coleções especiais como Maps (mapas) e Sets (conjuntos);



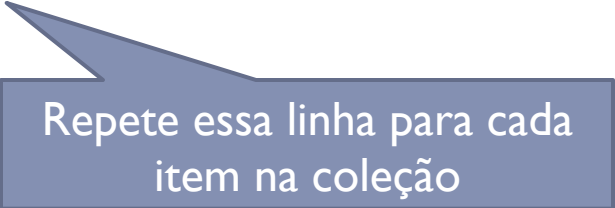
O laço for

- ▶ Em java existem dois tipos de laço for, um especial para coleções e arrays (comumente referenciado como for-each) e o for comum vindo da linguagem c;
- ▶ O for-each só é utilizado para navegar em uma coleção de objetos;
- ▶ O for comum pode ser utilizado para qualquer caso de laço, mas ele deve ser utilizado apenas quando for um caso de “contagem” e não de condição, se o laço é apenas condicional, um while pode ser a melhor opção;



Exemplo de for-each

```
ArrayList<String> strings =  
    new ArrayList<String>();  
strings.add("Maurício");  
strings.add("José");  
strings.add("Carol");  
  
for ( String nome : strings ) {  
    System.out.println( nome );  
}
```



Repete essa linha para cada
item na coleção



Exemplo de um for comum

```
ArrayList<String> strings = new ArrayList<String>();  
strings.add("Maurício");  
strings.add("José");  
strings.add("Carol");
```

Inicialização

Condição

Incremento

```
for ( int x = 0; x < strings.size(); x ++ ) {  
    String nome = strings.get(x)  
    System.out.println( nome );  
}
```

Métodos construtores

- ▶ São os métodos chamados para inicializar um objeto;
- ▶ Sempre que há um "**new**" há um método construtor sendo chamado;
- ▶ Os métodos construtores são definidos com o mesmo nome da sua classe;
- ▶ Uma classe pode ter diversos métodos construtores;



Exemplo de métodos construtores

```
public class Cliente {  
    private String nome;  
    private String cpf;  
    private Date nascidoEm;  
  
    public Cliente() {}  
  
    public Cliente( String nome ) {  
        this.nome = nome;  
    }  
  
    public Cliente( String nome, String cpf ) {  
        this.nome = nome;  
        this.cpf = cpf;  
    }  
  
    public Cliente( String nome, String cpf, Date nascidoEm ) {  
        this.nome = nome;  
        this.cpf = cpf;  
        this.nascidoEm = nascidoEm;  
    }  
}
```



Exemplo de uso de construtores

```
Cliente umCliente = new Cliente();
```

```
Cliente outroCliente = new  
    Cliente( "Maurício" );
```

```
Cliente cliente = new  
    Cliente( "Maurício",  
    "000.000.000-00" );
```

```
Cliente aindaCliente = new  
    Cliente( "Maurício",  
    "000.000.000-00", new Date() );
```



Informações especiais

- ▶ Quando você não define um construtor na sua classe, o compilador adiciona um construtor default, que é o construtor sem parâmetros;
- ▶ Se você definir um construtor qualquer na sua classe, o compilador não vai adicionar um construtor default, se você o quiser vai ter que adicioná-lo manualmente;
- ▶ Os construtores tem níveis de visibilidade assim como métodos;
- ▶ O construtor a ser chamado é selecionado pelos parâmetros que estão sendo passados;



Implementando um simples “Afunde um Político”

Um Batalha Naval aonde você afunda políticos, em vez de
barcos 😊

Idéia básica

- ▶ Você deve montar um quadro (ou matriz) com 8 linhas e 8 colunas (veja que essa matriz não precisa existir de verdade...);
- ▶ Os políticos devem ser espalhados nesse quadro, sempre na posição horizontal ou vertical (nunca em diagonal), ocupando 1 quadrado;
- ▶ O programa deve receber uma entrada do cliente, perguntando em qual linha e em qual coluna ele deve tentar atirar;



Continuando...

- ▶ Se o usuário acertar em um político, deve receber uma mensagem avisando do acerto e se o político afundou ou não (ele só afunda quando os 3 quadrados forem acertados);
- ▶ Se o usuário não acertar, o programa deve também avisar que o tiro errou;
- ▶ O programa não deve ficar todo em uma única classe 😊

