

1	Introdução à criação de classes	7
1.1	Classes & Objetos	7
1.1.1	Classes	7
1.1.2	Objetos	8
1.1.3	Abstração	8
1.2	Atributos	10
1.2.1	Declaração de atributos	10
1.2.2	Convenções para nomes de atributos	11
1.2.3	Acessando atributos a partir de outras classes	11
1.2.4	Inicialização de atributos	12
1.2.5	Laboratório	14
1.3	Métodos	15
1.3.1	Declaração de métodos	15
1.3.2	Convenções para nomes de métodos	15
1.3.3	Retorno de métodos	16
1.3.4	Passagem de parâmetros	17
1.3.5	Acessando métodos a partir de outras classes	18
1.3.6	Exemplos	19
1.3.7	Erros comuns	21
1.3.8	Laboratório	23
1.3.9	Passagem de parâmetros de tipos primitivos	24
1.3.10	Passagem de parâmetros de tipos reference (Objetos e arrays)	25
1.4	Certificação Sun Certified Java Programmer (SCJP)	26
2	Introdução a UML	27
2.1	UML e Metodologias de desenvolvimento de software	28
2.2	Principais diagramas da UML	29
2.2.1	Diagrama de casos de uso	29
2.2.2	Diagrama de classes	29
2.2.3	Diagrama de estados	29
2.2.4	Diagrama de sequência	29
2.2.5	Diagrama de Colaboração	30
2.2.6	Diagrama de pacotes	30
2.2.7	Diagrama de componentes	30
2.2.8	Diagrama de implantação / deployment	30
2.2.9	Diagrama de atividades	30
2.3	Introdução ao diagrama de classes	31
2.3.1	Representando atributos	31
2.3.2	Representando métodos	31
3	Encapsulamento	33
3.1	Getters e Setters	34
3.1.1	Métodos get	34
3.1.2	Métodos set	34
3.1.3	Métodos is	34
3.2	Modificadores de acesso	38
3.2.1	Modificador public	38
3.2.2	Modificador private	39
3.2.3	Quando devemos utilizar private ou public?	40

3.2.4	Exemplos	41
3.3	Objeto this	43
3.4	Encapsulamento de atributos compostos	45
3.5	Laboratório	51
3.6	Certificação Sun Certified Java Programmer (SCJP)	52
4	Sobrecarga de métodos	53
4.1	Laboratório	57
5	Construtores e método finalize	58
5.1	Introdução	58
5.2	Declarando construtores	61
5.2.1	Erros comuns na declaração de construtores	64
5.3	Sobrecarga de construtores	66
5.4	Método finalize	69
5.5	Laboratório	73
5.6	Certificação Sun Certified Java Programmer (SCJP)	74
6	Modificador static	75
6.1	Atributo static	75
6.1.1	Quando devemos utilizar atributos estáticos?	75
6.1.2	Quando não devemos utilizar atributos estáticos?	78
6.2	Métodos static	80
6.3	Bloco de código static	81
6.4	Representação do modificador static na UML	83
6.5	Laboratório	84
6.6	Certificação Sun Certified Java Programmer (SCJP)	85
7	Relacionamento entre classes	86
7.1	Associação	86
7.1.1	Representação de associações na UML	86
7.1.2	Cardinalidade / Multiplicidade	87
7.1.3	Navegabilidade	89
7.1.4	Restrições	91
7.1.5	Associação reflexiva	91
7.1.6	Agregação	92
7.1.7	Composição	93
7.1.8	Dependência	94
7.1.9	Estudo de caso: Modelagem de uma empresa	95
7.1.10	Laboratório	97
7.1.11	Laboratório	99
7.2	Herança	101
7.2.1	Representação de herança e UML	102
7.2.2	Exemplos	102
7.2.3	Modificador private	109
7.2.4	Modificador protected	111
7.2.5	Referência implícita super	113
7.2.6	Construtores x Herança	115
7.2.7	Sobrescrita de métodos	120
7.2.8	Sobrescrita do método toString()	122
7.2.9	Laboratório	124

7.3	Certificação Sun Certified Java Programmer (SCJP)	126
7.4	Modificador final	127
7.4.1	Modificador final na declaração de atributos	127
7.4.2	Modificador final na declaração de classes	129
7.4.3	Modificador final na declaração de métodos	130
7.5	Modificador abstract	131
7.5.1	Classes abstratas	131
7.5.2	Métodos Abstratos	133
7.5.3	Representando o modificador abstract em UML	135
7.5.4	Erros comuns	135
7.5.5	Laboratório	136
7.6	Interfaces	137
7.6.1	Interfaces em Java	138
7.6.2	Definindo uma Interface	139
7.6.3	Implementando uma Interface	140
7.6.4	Estendendo uma interface	142
7.6.5	Representando interfaces na UML	143
7.6.6	Interface vs. Abstract	144
7.6.7	Erros comuns	144
7.6.8	Laboratório	145
7.7	Cast de objetos e polimorfismo	147
7.7.1	Cast up (Widening)	148
7.7.2	Cast down (Narrowing)	149
7.7.3	Polimorfismo	150
7.7.4	Parâmetros polimórficos	150
7.8	Método equals() e hashCode()	153
7.8.1	equals()	153
7.8.2	hashCode()	156
7.9	Estudo de caso: Sistema de reservas de passagens aéreas	157
7.10	Laboratório de modelagem	158
7.11	Certificação Sun Certified Java Programmer (SCJP)	159
8	Pacotes	165
8.1	Utilizando classes de outros pacotes	167
8.1.1	Modificador default / package	169
8.1.2	Importando duas classes com o mesmo nome em pacotes diferentes	171
8.1.3	Erros comuns	173
8.2	Declarando o pacote das classes	175
8.2.1	Compilando classes que estão em pacotes	176
8.2.2	Executando classes que estão em pacotes	176
8.2.3	Erros comuns	177
8.3	Trabalhando com classes que estão em pacotes diferentes	179
8.3.1	Erros comuns	180
8.4	Diagrama de pacotes	182
8.4.1	Relacionamento de dependência	182
8.4.2	Representando pacotes no diagrama de classes	182
8.4.3	Dicas para utilização de pacotes	184
8.5	Componentes: JAR (Java ARchive)	186
8.5.1	Criação de um JAR simples	187

8.5.2	Extração das classes de um JAR	188
8.5.3	Criação de um JAR executável	189
8.5.4	Execução de um JAR	190
8.5.5	Disponibilizando um JAR para muitas aplicações	190
8.6	Diagrama de componentes	193
8.7	Laboratório	195
8.8	Certificação Sun Certified Java Programmer (SCJP)	196
9	Tratamento de erros	197
9.1	Introdução	197
9.2	Exceptions	198
9.2.1	Hierarquia das Exceptions	198
9.2.2	A classe Exception	199
9.2.3	RuntimeExceptions	200
9.2.4	Laboratório	202
9.2.5	Checked Exceptions	203
9.2.6	A cláusula throws	204
9.2.7	As cláusulas try / catch	207
9.2.8	A cláusula finally	211
9.2.9	Capturando múltiplas Exceptions	213
9.2.10	Laboratório	216
9.2.11	Criando suas próprias Exceptions	217
9.2.12	A cláusula throw	219
9.2.13	Relançando Exceptions	221
9.3	Considerações sobre override, abstract e interfaces	223
9.4	Laboratório	224
9.5	Certificação Sun Certified Java Programmer (SCJP)	225
10	Outros diagramas UML	228
10.1	Diagrama de Caso de Uso	228
10.1.1	Objetivos	228
10.1.2	Principais elementos do Diagrama de Caso de uso	228
10.1.3	Dicas para criação de use cases	229
10.1.4	Exemplo de cenário: Conta bancária	229
10.1.5	Inclusão (include)	230
10.1.6	Extensão (extend)	230
10.1.7	Herança (generalization)	230
10.1.8	Estudo de caso: Sistema de reservas de passagens aéreas	231
10.1.9	Documento de fluxo de eventos	232
10.2	Diagrama de Sequência	234
10.2.1	Representação de objetos	234
10.2.2	Mensagens	234
10.2.3	Exemplo: Sistema de reservas de passagens aéreas	237
10.2.4	Laboratório	239
11	Apêndice	241
11.1	Solução das questões preparatórias para certificação	241
12	Apêndice II	246
12.1	Introdução	246
12.1.1	Doclets	246

12.2	Principais Tags que podem ser utilizadas no Javadoc	247
12.3	Descrição das principais tags	248
12.4	Executando o utilitário Javadoc	254

Globalcode®
THE DEVELOPERS COMPANY

Modelo de Apresentação

Atenção: A Cópia integral ou parcial
deste material não está autorizada.

Globalcode®
THE DEVELOPERS COMPANY

Modelo de Apresentação

Atenção: A Cópia integral ou parcial
deste material não está autorizada.

1 Introdução à criação de classes

1.1 Classes & Objetos

1.1.1 Classes

Uma classe é um tipo definido pelo usuário que possui especificações (características e comportamentos) que o identifiquem. De uma maneira mais objetiva, podemos dizer que a classe é um molde que será usado para construir objetos que representam elementos da vida real.

Classe = Características + Comportamentos

Tal molde é gerado através da observação e agrupamento de elementos que possuam as mesmas características e comportamentos sob uma mesma denominação. Exemplificando, em uma loja online (ambiente), identificamos um elemento chamado Produto (classe), que possui um conjunto de características tal como a Identificação e o Preço (atributos).

Produto	
Identificação	Preço

Cada Produto poderia ser materializado com as informações abaixo:

Produto	
Identificação	Preço
Core Java	R\$ 500,00

Produto	
Identificação	Preço
Camiseta	R\$ 15,00

No entanto, os **Atributos** nem sempre são úteis individualmente. Convém definirmos algumas ações e comportamentos que esse elemento possa executar dentro do ambiente modelado usando os atributos existentes na Classe.

Anotações

Esses comportamentos são denominados **Métodos** e definem as ações previstas para essa classe. Ainda no exemplo da loja, poderíamos atribuir aos Produtos algumas funcionalidades tais como aumentar/diminuir o preço e alterar a Identificação. Note que normalmente as ações estão associadas aos atributos da classe (Preço, Identificação).

Produto	
Identificação	Preço
Aumentar preço Diminuir preço Alterar identificação	

1.1.2 Objetos

Note que nos exemplos acima (Core Java, R\$ 150,00; Camiseta, R\$ 15,00) representamos casos particulares da classe Produto. Eles são denominados instâncias das classes, ou objetos, e eles têm vida independente entre si, apesar de compartilharem o mesmo “molde” (Classe). Estaremos detalhando este tópico adiante.

A criação de objetos é feita através da utilização do operador `new` e então o espaço necessário de memória para o objeto é alocado. O Garbage Collector libera o espaço de memória alocado para o objeto quando ele não está mais sendo utilizado.

Sintaxe

```
<NomeDaClasse> <nomeVariavel> = new <NomeDaClasse>()
```

Para criarmos uma instância da classe `Pessoa` utilizamos o seguinte comando:

```
Pessoa p = new Pessoa();
```

Para criarmos uma instância da classe `Data` utilizamos o seguinte comando:

```
Data d = new Data();
```

1.1.3 Abstração

Como uma classe pode representar qualquer coisa em qualquer ambiente, é sugerido que a modelagem foque nos objetivos principais do negócio. Isso evita que o sistema seja muito grande e conseqüentemente de difícil manutenção e compreensão. A análise focada é denominada abstração.

No mundo real, podemos utilizar um produto que possui informações de garantia, matéria-prima, etc. Porém, conforme o tipo de aplicativo, pode ser ou não interessante colocarmos tais informações na definição do objeto.

No mundo real podemos trocar produtos entre pessoas, entretanto, num cenário de e-business, não precisamos implementar este comportamento ao definirmos o Produto.

Exemplo: abstração de Data

Qual é a estrutura básica de uma Data?

- dia
- mês
- ano

Qual o comportamento ou operações relativos a esta entidade?

- Transformar o número do mês em um texto com o nome do mês (Ex: 1 : Janeiro)
- Transformar o número do dia em nome do dia (Ex: 24:Segunda, 25:Terça)..
- Devemos saber se o ano é bissexto.

Exemplo: abstração de Produto

Quais são as características básicas (estrutura) de um produto?

- id
- preço
- nome

Qual o comportamento desta entidade no mundo real?

- Aumentar o preço
- Aplicar desconto
- Alterar o nome

Anotações

1.2 Atributos

As definições das características de um objeto são feitas através de atributos da classe.

Atributos armazenam os dados do objeto.

1.2.1 Declaração de atributos

Atributos são declarados dentro da declaração da classe, ou seja, dentro do bloco delimitado pela abertura e fechamento das chaves ({ e }). Não podemos declarar atributos dentro de métodos, pois variáveis declaradas dentro de métodos são chamadas variáveis locais e tem escopo menor do que um atributo.

Quando declaramos um atributo em uma classe indicamos o tipo de dado que será armazenado nele.

Atributos podem ser de dois tipos:

- ✚ tipos primitivos (Ex: long, int, boolean, etc..)
- ✚ tipos reference (Ex: Arrays ou String, Integer, Date, Cliente, Pessoa, etc)

Sintaxe para declaração

```
*<tipo do atributo> identificador;
```

Observação: Caso o atributo seja um array a declaração é feita da mesma forma que declaramos uma variável local do tipo array.

Exemplos: `int[] array` ou `int array[]`

*Opcionalmente podem ser declarados alguns modificadores ou palavras reservadas que podem ser aplicadas a atributos e métodos alterando características e a forma de acesso ao método. Veremos os modificadores em capítulos posteriores.

Exemplo: Pessoa.java

```
class Pessoa {  
    long rg;  
    String nome;  
    String sobrenome;  
    String dataNasc;  
    String[] telefones;  
}
```

Exemplo: Data.java

```
class Data {  
    int dia;  
    int mes;  
    int ano;  
}
```

1.2.2 Convenções para nomes de atributos

As seguintes convenções são utilizadas para declaração de atributos pois desta forma visualiza-se com maior facilidade o que são métodos e o que são atributos.

Além disto, vamos imaginar um cenário onde um desenvolvedor pergunta para o outro, como é o nome dos atributos da classe `Cliente`? E o outro responde nome, telefone e endereço.

Se as convenções estiverem sendo utilizadas sabemos exatamente como utilizar os atributos... senão teremos que consultar a documentação para saber como os atributos foram declarados, com maiúsculas, minúsculas, underscore.



Identificadores de atributos devem ser declarados com letras minúsculas;

Exemplo: `nome`, `telefone`, `i`, `item`, `email`

Quando o nome do atributo for composto por duas ou mais palavras, a separação deve ser feita com um caracter maiúsculo e não underscore.

Exemplo: `telefoneComercial`, `enderecoDeEntrega`, `contratoPessoaJuridica`, `curriculoVitae`, `itemDeVenda`

1.2.3 Acessando atributos a partir de outras classes

A partir de uma instancia de classe, podemos acessar os atributos de uma instância com dois objetivos, ler o valor do atributo ou alterar o valor do atributo.

Para tal, utilizamos ponto e mais o nome do atributo.

Sintaxe

`nomeVariavel.nomeAtributo`

No exemplo a seguir, criamos uma classe chamada `TestePessoa`, que no método `main` acessa os atributos da classe `Pessoa` definida anteriormente com dois objetivos: escrita e leitura.

Anotações

Exemplo: TestePessoa.java

```

class TestePessoa {
    public static void main(String[] args) {
        // Criamos uma instancia da classe Pessoa
        Pessoa p = new Pessoa();
        // Acessando os atributos da classe Pessoa para definir seus valores
        p.nome = "Rodrigo";
        p.sobrenome = "Monteiro";
        p.dataNasc = "25/12/1975";
        p.rg = 11111;
        String osTelefones[] = { "1234567", "7654321"};
        p.telefones = osTelefones;
        // Acessando os atributos para leitura
        System.out.println("Nome : " + p.nome + " " + p.sobrenome);
        System.out.println("Data Nasc : " + p.dataNasc);
        System.out.println("RG : " + p.rg);
        System.out.println("Telefones:");
        for (int i=0; i<p.telefones.length; i++){
            System.out.println(p.telefones[i]);
        }
    }
}

```

Saída gerada a partir da execução da classe TestePessoa:

```

C:\WINDOWS\system32\cmd.exe
C:\java\eclipse\workspace\ExemplosAJ2>java TestePessoa
Nome : Rodrigo Monteiro
Data Nasc : 25/12/1975
RG : 11111
Telefones:
1234567
7654321
C:\java\eclipse\workspace\ExemplosAJ2>

```

1.2.4 Inicialização de atributos

Como já vimos, variáveis locais não são inicializadas; no entanto, atributos são inicializados com valores default. Isto significa que se, não inicializamos os atributos de classe eles o serão conforme com a seguinte tabela:

Tipo	Valor de Inicialização
byte	0
short	0
int	0
float	0.0f
long	0L
double	0.0d
char	'\u0000'
boolean	false
refer.objeto	null



É preciso atenção especial com atributos do tipo array, pois arrays são inicializados com `null`, e portanto qualquer acesso a elementos levará a um erro em tempo de execução chamado `NullPointerException`, lembrando que não é possível acessar atributos ou métodos de objetos que estão com o valor `null`, ou acessar elementos de arrays que estão nulos.

Exemplo

```
class Pessoa {
    long rg;
    String nome;
    String sobrenome;
    String dataNasc;
    String[] telefones;
}

class TestePessoa {
    public static void main(String[] args) {
        // Criamos uma instancia da classe Pessoa
        Pessoa p = new Pessoa();
        // Acessando os atributos para leitura
        System.out.println("Nome : " + p.nome + " " + p.sobrenome);
        System.out.println("Data Nasc : " + p.dataNasc);
        System.out.println("RG : " + p.rg);
        System.out.println("RG : " + p.telefones);
    }
}
```

Saída gerada pela execução da classe `TestePessoa`:

```
C:\WINDOWS\system32\cmd.exe

C:\java\eclipse\workspace\ExemplosAJ2>java TestePessoa
Nome : null null
Data Nasc : null
RG : 0
RG : null

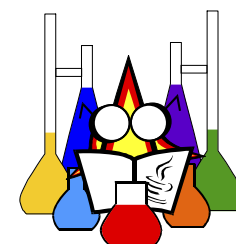
C:\java\eclipse\workspace\ExemplosAJ2>
```

Anotações


1.2.5 Laboratório

Objetivo:

Praticar a criação de classes e utilização de atributos.

Tabela de atividades

LABORATÓRIO

Atividade	OK
1. Faça o download do arquivo aj2lab01_01.zip na URL indicada pelo instrutor(a).	
 <p>Descompacte o arquivo no seu diretório de trabalho. Para descompactar o arquivo no Linux utilize o seguinte comando no mesmo diretório onde está o arquivo que você quer descompactar: <code>unzip <nome arquivo></code>. Os arquivos serão descompactados no diretório corrente.</p> <p>DICAS</p>	
2. Descompacte o arquivo em seu diretório de trabalho.	
<p>3. Adicione os seguintes atributos na classe <code>Agencia</code>.</p> <ul style="list-style-type: none"> ➔ numero do tipo <code>String</code> ➔ banco do tipo <code>int</code> 	
4. Siga as instruções encontradas na classe <code>TestaAgencia</code> .	
<p>5. Adicione os seguintes atributos na classe <code>Cliente</code>.</p> <ul style="list-style-type: none"> ➔ nome do tipo <code>String</code> ➔ cpf do tipo <code>String</code> 	
6. Siga as instruções encontradas na classe <code>TestaCliente</code>	
<p>7. Adicione os seguintes atributos na classe <code>Conta</code>.</p> <ul style="list-style-type: none"> ➔ saldo do tipo <code>double</code> ➔ numero do tipo <code>String</code> ➔ titular do tipo <code>String</code> ➔ agencia do tipo <code>int</code> ➔ banco do tipo <code>int</code> 	
8. Siga as instruções encontradas na classe <code>TestaConta</code> .	

1.3 Métodos

Através dos métodos definimos as operações que podem ser executadas com ou sobre um objeto. Popularmente diz-se que os métodos definem o comportamento da classe.

1.3.1 Declaração de métodos

Métodos são declarados dentro do corpo da classe, ou seja, dentro do bloco de código da definição da classe, que é definido com abre e fecha chaves.

Um método é dividido em três partes:

- retorno do método,
- nome do método
- parâmetros do método.

Sintaxe para declaração de métodos

```
* <tipo do retorno> nomeDoMetodo (<parametrosDeMetodos>){}
```

Sintaxe para declaração de <parametrosDeMetodos>:

```
(<tipo> identificador, ..., <tipo> identificador)
```

*Opcionalmente podem ser declarados alguns modificadores ou palavras reservadas que podem ser aplicadas a atributos e métodos alterando suas características e a sua forma de acesso. Veremos os modificadores em capítulos posteriores.

1.3.2 Convenções para nomes de métodos

- Identificadores de métodos devem ser declarados com letras minúsculas;
- Quando o nome do método for composto por duas ou mais palavras, a separação deve ser feita com um caracter maiúsculo.
- Uma sugestão seria o uso de verbos no infinitivo e na voz passiva.



Anotações
