

Técnicas de Orientação a Objeto

- Sobrecarga e sobrescrita;
- Casting de Referência
- Classe Object



Programação Orientada a Objeto

Sobrecarga e Sobrescrita

- **Sobrecarga** (*Overload*) e **Sobrescrita** (*Override*) criam métodos com os **mesmos** nomes;
- **Sobrecarga** acontece **na mesma** classe (onde já vimos sobrecarga?);
- **Sobrescrita** acontece na estrutura de **herança**;
- É preciso entender claramente a diferença de cada uma para prever o comportamento do compilador.



Sobrecarga:

- Acontece na mesma classe;
- Métodos se **diferenciam pelos parâmetros** de entrada;
- A distinção não pode ser feita somente pelo retorno do método pois ela deve ser **determinística**;
 - Se houvesse dois métodos com entradas idênticas e saídas diferentes, como compilador conseguiria definir qual método utilizar?
- O compilador faz uma **comparação de parâmetros** e escolhe o que combina com a chamada do método.



Programação Orientada a Objeto

Sobrecarga

Exemplo de **Sobrecarga**:

- Na classe Pessoa temos o método `setNome()` que recebe uma `String`.
- Imagine que queremos dar uma **opção** para o usuário de armazenar o nome como a concatenação invertida:
- Dado "Reinaldo" e "Castro", armazenaremos "Castro, Reinaldo".
- Mas é preciso **manter** o `setNome()` já definido também.
- Mesma classe, parâmetros de entrada diferentes:
Sobrecarga.



Programação Orientada a Objeto

Sobrecarga

Exemplo de **Sobrecarga** no **Construtor**:

```
1  public class Pessoa{
2      private String nome;
3
4      public Pessoa() {}
5      public Pessoa(String $nome) {
6          this.nome = $nome;
7      }
8
9      public String getNome() {
10         return this.nome;
11     }
```



Programação Orientada a Objeto

Sobrecarga

Exemplo de **Sobrecarga** no **setNome()**:

```
13     public void setNome(String $nome) {  
14         this.nome = $nome;  
15     }  
16  
17     public void setNome(String $primeiroNome, String $segundoNome){  
18         nome = $segundoNome +", " + $primeiroNome;  
19     }  
20 }
```



Programação Orientada a Objeto

Sobrecarga

Utilizando a **Sobrecarga** do método **setNome()**:

```
1  public class TestaSobrecarga{
2      public static void main (String[] args) {
3          PessoaFisica p1 = new PessoaFisica();
4          p1.setNome("Marcia");
5          System.out.println("Nome: " + p1.getNome());
6          p1.setNome("Marcia","Aguena");
7          System.out.println("Nome: " + p1.getNome());
8      }
9  }
```



Programação Orientada a Objeto

Exercícios : Sobrecarga

Recordando os métodos implementados como exercício da classe `String`, utilize o método:

```
String substring(int inicio, int fim);
```

para construir um novo método `setNome()` em `Pessoa` que, dado 3 nomes de parâmetro de entrada do tipo:

`"Marcia", "Aguena", "Castro"`

armazene no atributo `nome`:

`"Castro, Marcia A."`



Sobrescrita:

- Acontece entre classes na estrutura de **herança**;
- A assinatura dos métodos devem ser as **mesmas**;
- A implementação na **subclasse** deve ser mais **específica**;
- A implementação na **superclasse** deve ser mais **genérica**;
- Quando encontra a chamada de um método, o compilador procura a implementação **primeiro na classe do objeto que o chamou**. Se não encontra, ele então procura na superclasse.



Programação Orientada a Objeto

Sobrescrita

Exemplo de Sobrescrita:

- No exemplo anterior, vamos reescrever o método `setNome()` na classe `PessoaFisica` para colocar os nomes passados como parâmetro em letras Maiúsculas;

```
1 public class PessoaFisica extends Pessoa{
2     private String cpf;
3
4     public PessoaFisica() {}
5
6     public PessoaFisica(String $nome, String $cpf) {
7         super.setNome($nome);
8         this.cpf = $cpf;
9     }
10
11     public String getCpf() {
12         return cpf;
13     }
14
15     public void setCpf(String $cpf) {
16         this.cpf = $cpf;
17     }
18
19     public void setNome(String $nome){
20         super.setNome($nome.toUpperCase());
21     }
22 }
```



Programação Orientada a Objeto

Sobrescrita

Exemplo de **Sobrescrita**:

- Testando o método `getNome()` com a classe `TestaSobrescrita`:

```
1  public class TestaSobrescrita{  
2      public static void main (String[] args) {  
3          PessoaFisica p1 = new PessoaFisica();  
4          PessoaFisica p2 = new PessoaFisica();  
5          p1.setNome("Marcia Luciana Aguená Castro");  
6          System.out.println("Nome: " + p1.getNome());  
7          p2.setNome("Reinaldo", "Castro");  
8          System.out.println("Nome: "+ p2.getNome());  
9      }  
10 }
```



Exercícios : Sobrecarga e Sobrescrita

Exercício 1 : Utilize sobrecarga ou sobrescrita no seguinte problema:

- **Todo** professor de pós-graduação **deve ser um** professor de graduação.
- Os professores de graduação possuem os seguintes atributos: matrícula, nome, salário bruto e quantidade de disciplinas.
- **Além** dos atributos de um professor de graduação, os professores de pós-graduação também armazenam a quantidade de anos de doutorado e quantidade de artigos científicos escritos.



Programação Orientada a Objeto

Exercícios : Sobrecarga e Sobrescrita

O salário de um professor de graduação é calculado da seguinte forma:

$$\begin{aligned} &= \text{salário bruto} - (\text{salário bruto} * 0.2) \\ &\quad + \text{quantidade de disciplinas} * 50 \end{aligned}$$

O salário de um professor de mestrado é calculado da seguinte forma:

$$\begin{aligned} &= \text{salário do professor de graduação} \\ &\quad + \text{quantidade de artigos científicos} * 150 \end{aligned}$$

- Crie classes `professorGraduacao` e `professorPos` e crie o método `calculaSalario()` baseado nas informações dadas;
- Na classe do método `main`, instancie um objeto de cada classe;
- Exiba o salário líquido de cada objeto instaciado.



Programação Orientada a Objeto

Exercícios : Sobrecarga e Sobrescrita

Exercício 2 : Utilize sobrecarga ou sobrescrita no seguinte problema:

- Crie um subdiretório chamado Fornecedor;
- Nele crie uma classe Contato, contendo os atributos encapsulados, com seus respectivos métodos get's e set's, e ainda o construtor padrão e mais um construtor que inicialize todos os atributos. Os atributos dessa classe são
- "nome" do tipo String;
- "idade" do tipo int;
- "telefone" do tipo String.
- Crie uma classe TesteContato com um método main. Crie um objeto da classe Contato com os seus dados. Mostre no console uma linha como a seguir:

Olá, [SEU NOME]. Sua idade em nosso sistema é [SUA IDADE] e seu número de telefone é [SEU TELEFONE].



Exercícios : Sobrecarga e Sobrescrita

Exercício 3 :

- Considere, como subclasse da classe Contato a classe Fornecedor. Cada instância da classe Fornecedor tem, além dos que caracterizam a classe Contato, os atributos:
- `valorCredito` (correspondente ao crédito máximo atribuído ao fornecedor);
- `valorDivida` (montante da dívida para com o fornecedor).
- Implemente na classe Fornecedor os métodos `get's` e `set's`, um construtor padrão e um que inicialize todos os atributos do objeto e um método `saldo()` que devolve a diferença entre os valores de `valorCredito` e `valorDivida`.



Exercícios : Sobrecarga e Sobrescrita

- Crie uma classe TesteFornecedor com um método main. Nesse método, crie um fornecedor com os seus dados e que tenha valor de crédito e de dívida que você escolher.
- Crie uma variável String que receba a frase abaixo:

Olá, [SEU NOME]. Seu saldo atual com nossa empresa é [SALDO].

- Caso o saldo seja positivo, concatene com a mensagem acima a palavra "**Parabéns.**", caso contrário, concatene com "**Procure nosso serviço de crédito o quanto antes.**".
- Imprima a variável com a mensagem final no console.



Programação Orientada a Objeto

Exercícios : Sobrecarga e Sobrescrita

Exercício 4 :

- Crie duas subclasses da classe `Fornecedor`: `FornecedorNacional` e `FornecedorInternacional`.
- Para um fornecedor nacional, garanta que ele tenha sempre 20% a mais de crédito. Ou seja, se o valor de crédito inicial é R\$ 1.000,00, automaticamente ele tem R\$ 1.200,00. Se o valor do crédito for alterado posteriormente para R\$ 800,00, com o bônus vai para R\$ 960,00.
- Já para o fornecedor internacional, seu valor da dívida deve ser sempre inicializado com 0 e também deve ser garantido que só sejam aceitas atualizações da dívida que não ultrapassem o valor total do crédito que ele tem. Se ultrapassar, o valor da dívida não pode ser alterado.
- Crie uma classe `TesteFornecedores` que valide as regras explicadas acima.



Exercícios : Sobrecarga e Sobrescrita

Atenção: Não copie ou cole nenhum exercício. A repetição é intencional para criar fluência na linguagem.

1. Crie a classe `Endereco`, com os seguintes atributos privados:

1. `"rua"` do tipo `String`;
2. `"numero"` do tipo `String`;
3. `"complemento"` do tipo `String`;
4. `"bairro"` do tipo `String`;
5. `"cep"` do tipo `String`;

2. Crie a classe `Cidade`, com o seguinte atributo privado:

1. `"nome"` do tipo `String`;
2. `"estado"` do tipo `String`;



Exercícios : Sobrecarga e Sobrescrita

3. Crie construtores vazios e completos para as classes `Endereco` e `Cidade`;
4. Na classe `Conta`, crie os métodos `setTaxa()` e `getTaxa()`, deixe os métodos vazios (coloque um `return 0.0;` em `getTaxa()`).
5. Na classe `ContaCorrente`, sobrescreva os métodos `setTaxa()` e `getTaxa()` para atualizar e retornar o atributo `taxaDeJuro`;
6. Na classe `ContaPoupanca`, sobrescreva os métodos `setTaxa()` e `getTaxa()` para atualizar e retornar o atributo `taxaDeCorrecao`;



Exercícios : Sobrecarga e Sobrescrita

7. Na classe `Cliente`, crie os métodos `setIdentificador()` e `getIdentificador()`, deixe os métodos vazios (coloque um `return ""`; em `getIdentificador()`);
8. Na classe `ClientePessoaFisica`, sobrescreva os métodos `setIdentificador()` e `getIdentificador()` para atualizar e retornar o atributo `cpf`;
9. Na classe `ClientePessoaJuridica`, sobrescreva os métodos `setIdentificador()` e `getIdentificador()` para atualizar e retornar o atributo `cnpj`;



Exercícios : Sobrecarga e Sobrescrita

10. Na classe `AplicacaoFinanceira`:

1. Crie um objeto `endereco1` da classe `Endereco` e atribua valor ao objeto utilizando seu construtor;
2. Crie um objeto `cidade1` da classe `Cidade` e atribua valor ao objeto utilizando seu construtor;
3. Substitua o método `getTaxaDeJuro()` do objeto `contaCorrente1` por `getTaxa()`;
4. Substitua o método `getTaxaCorrecao()` do objeto `contaPoupanca1` por `getTaxa()`;

11. Compile a classe `AplicacaoFinanceira` e corrija os possíveis erros.



Programação Orientada a Objeto

Casting de Referência

- Já vimos como funciona o upcasting e downcasting de tipos primitivos.
- De maneira análoga, um objeto de uma **subclasse** pode ser **atribuído** a um objeto de uma **superclasse**.
- Lembre-se que um objeto da subclasse "**é um**" objeto da superclasse;
- No exemplo Pessoa e PessoaFisica, poderíamos criar um objeto p da forma:

Pessoa p =

```
new PessoaFisica("Marcia", "123.456.789-00");
```



Programação Orientada a Objeto

Casting de Referência

```
Pessoa p =  
new PessoaFisica("Marcia", "123.456.789-00");
```

- A alocação de memória é feita para um objeto da classe PessoaFisica (subclasse) e seu construtor é chamado, mas a referência à ela será feita como uma classe Pessoa(superclasse).
- Assim p pode chamar os métodos de Pessoa, mas **não pode chamar os métodos de PessoaFisica diretamente.**



Programação Orientada a Objeto

Casting de Referência

- Para recuperar a parte de `PessoaFisica` do objeto `p`, é preciso fazer um **casting de referência** (downcasting no caso).

```
PessoaFisica pf = (PessoaFisica)p;  
((PessoaFisica)p).getCpf();
```




Programação Orientada a Objeto

Casting de Referência

```
1 public class PessoaFisica extends Pessoa{
2     private String cpf;
3
4     public PessoaFisica() {}
5
6     public PessoaFisica(String $nome, String $cpf) {
7         super.setNome($nome);
8         this.cpf = $cpf;
9     }
10
11     public String getCpf() {
12         return cpf;
13     }
14
15     public void setCpf(String $cpf) {
16         this.cpf = $cpf;
17     }
18
19     public void setNome(String $nome){
20         super.setNome($nome.toUpperCase());
21     }
22 }
```

```
1 public class Pessoa{
2     private String nome;
3
4     public Pessoa() {}
5     public Pessoa(String $nome) {
6         this.nome = $nome;
7     }
8
9     public String getNome() {
10         return this.nome;
11     }
```



Programação Orientada a Objeto

Casting de Referência

```
1  public class TestaCasting{
2      public static void main (String[] args) {
3          Pessoa p = new PessoaFisica("Marcia", "123.456.789-00");
4
5          PessoaFisica pF = (PessoaFisica) p;
6
7          System.out.println("Nome: " + p.getNome());
8          System.out.println(" CPF: " + ((PessoaFisica)p).getCpf());
9          System.out.println("Nome: " + pF.getNome());
10         System.out.println(" CPF: " + pF.getCpf());
11     }
12 }
```



Programação Orientada a Objeto

Exercícios : Casting de Referência

Exercício 1 : No exercício de sobrecarga e sobrescrita de professor de graduação/professor de pós-graduação:

- Crie uma nova classe de teste com o mesmo conteúdo da classe de teste do exercício anterior;
- Renomeie a classe `ProfessorGraduacao` para `Professor`;
- Na classe `ProfessorPos`, altere o nome do método `calculaSalario()` para `calculaSalarioLiquido()`;
- No método `main` da classe de teste nova, altere a instância dos objetos das classes e `professorPos` com referências da classe `professor`.
- Para ambos os objetos exiba o cálculo do salário e para o objeto da classe `professorPos` exiba também o salário líquido.



Exercícios : Casting de Referência

Exercício 2 : No exercício de Sobrecarga e Sobrescrita de Fornecedor, `FornecedorNacional` e `FornecedorInternacional`:

- Crie uma nova classe de teste com o mesmo conteúdo da classe de teste do exercício anterior.
- Na nova classe de teste, mude a referência da instância dos objetos das classes `FornecedorNacional` e `FornecedorInternacional` para **Fornecedor**.
- Quais as modificações nos métodos do programa principal devem ser feitas?



Exercícios : Casting de Referência

Exercício 3 : No exercício anterior:

- Crie uma nova classe de teste com o mesmo conteúdo da classe de teste do exercício anterior.
- Na nova classe de teste, mude a referência da instância dos objetos das classes `FornecedorNacional` e `FornecedorInternacional` de `Fornecedor` para **Contato**.
- Quais as modificações nos métodos do programa principal devem ser feitas?



Programação Orientada a Objeto

Classe Object

- A classe **Object** é a **superclasse** de todas as classes do Java;
- Toda classe criada em Java é **subclasse** de **Object**: todas herdam seus métodos e atributos;
- Quando uma classe não usa o `extends`, existe um `extends Object` implícito nela durante a compilação;
- Qualquer objeto pode ser referenciado como da classe `Object`:

```
Object o1 = new Pessoa("Qualquer");
```



Programação Orientada a Objeto

Classe Object

- A Classe `Object` possui vários métodos implementados que todas as classes podem utilizar, entre eles temos:
 - `equals`
 - `toString`



Programação Orientada a Objeto

Classe Object - Método equals

equals:

- Quando `==` e `!=` são utilizado para comparar objetos em Java, as referências desses objetos são comparadas;
- `==` só retorna `true` (`!=` só retorna `false`) se as referências comparadas referenciam o mesmo objeto.
- Exemplo:

```
Pessoa p1 = new Pessoa("Gêmeo");
Pessoa p2 = new Pessoa("Gêmeo");
if (p1 != p2) // ou !(p1 == p2)
{
    System.out.println("Diferente");
    // O resultado é sempre diferente
}
```




equals:

- O método `equals` padroniza a comparação de igualdade de objetos: cada classe pode reescrever o que é igualdade entre seus objetos;
- A assinatura do método definido na classe `Object` é:

```
public boolean equals (Object obj)
```
- Se o método não for sobrescrito, ele funcionará exatamente igual a operação `==`;



Programação Orientada a Objeto

Classe Object - Método equals

- Exemplo de utilização do método equals:

```
1  public class TesteDeIgualdade{
2
3      public static void main(String[] args){
4
5          Integer int1a = new Integer(1);
6          Integer int1b = new Integer(1);
7
8          System.out.println("\nint1a==int1b?      -> " + (int1a==int1b));
9          System.out.println("\nint1a.equals(int1b)? -> " + int1a.equals(int1b));
10
11         System.out.println("\n =====");
12
13         int1b=int1a;
14         System.out.println("\nint1a==int1b?      -> " + (int1a==int1b));
15         System.out.println("\nint1a.equals(int1b)? -> " + int1a.equals(int1b));
16     }
17 }
```



Programação Orientada a Objeto

Classe Object - Método equals

equals:

- Para verificar se um objeto é igual ao outro, na maioria das vezes **é necessário comparar atributos;**
- O método **equals** pode ser sobrescrito para comparar um objeto com outro, retornando `true` se forem iguais de acordo com um novo critério estabelecido.



Programação Orientada a Objeto

Classe Object - Método equals

```
1 public class Pessoa{
2     private String nome;
3     public Pessoa() {}
4     public Pessoa(String n) {
5         this.nome = n;
6     }
7     public String getNome() {
8         return nome;
9     }
10    public void setNome(String n) {
11        nome = n;
12    }
13    public boolean equals(Object obj){
14        if (!(obj instanceof Pessoa))
15            return false;
16        Pessoa p = (Pessoa)obj;
17        return(p.nome==this.nome);
18    }
19 }
```

- Exemplo de sobrescrita do método equals:



Classe Object - Método toString

toString:

- O método `toString` retorna a representação de um objeto em `String`;
- Quando não é sobrescrito em uma classe, ele retorna um descritivo do objeto no formato:

`path.dos.pacotes.NomeDaClasse@objectId`

- O método pode ser sobrescrito para tornar-se mais significativo para uma classe mostrando o valor de seus atributos.



Programação Orientada a Objeto

Classe Object - Método toString

- Exemplo de utilização do método toString:

```
1  public class Pessoa{
2      private String nome;
3      public Pessoa() {}
4      public Pessoa(String n) {
5          this.setNome(n);
6      }
7      public String getNome() {
8          return nome;
9      }
10     public void setNome(String n) {
11         nome = n;
12     }
13     public String toString(){
14         return "Nome: " + this.getNome() + "\n" +
15             "Id   : " + super.toString();
16     }
17 }
```



Programação Orientada a Objeto

Exercícios : Classe Object

Exercício 1 : No exercício de sobrecarga e sobrescrita de professor de graduação/professor de pós-graduação:

- Crie uma nova classe de teste com o mesmo conteúdo da classe de teste do exercício anterior;
- No método main da classe de teste nova, altere a instância dos objetos das classes professorGraduacao e professorPos com referências da classe Object.
- Para ambos os objetos exiba o salário bruto e para o objeto da classe professorPos exiba também o salário líquido, fazendo os downcasting necessários.



Programação Orientada a Objeto

Exercícios : Classe Object

Exercício 2 : No exercício de Sobrecarga e Sobrescrita de Fornecedor, `FornecedorNacional` e `FornecedorInternacional`:

- Crie uma nova classe de teste com o mesmo conteúdo da classe de teste do exercício anterior.
- Na nova classe de teste, mude a referência da instância dos objetos das classes `FornecedorNacional` e `FornecedorInternacional` para **Object**.
- Quais as modificações nos métodos do programa principal devem ser feitas?