

Curso Programação Orientada a Objetos com Java

Capítulo: Interfaces

<http://educandoweb.com.br>

Prof. Dr. Nélcio Alves

Interfaces

<http://educandoweb.com.br>

Prof. Dr. Nélcio Alves

Aviso

- A partir do Java 8, interfaces podem ter "default methods" ou "defender methods"
- Isso possui implicações conceituais e práticas, que serão discutidas mais à frente neste capítulo
- Primeiro vamos trabalhar com a definição "clássica" de interfaces. Depois vamos acrescentar o conceito de default methods.

Interface

Interface é um tipo que define um conjunto de operações que uma classe deve implementar.

A interface estabelece um **contrato** que a classe deve cumprir.

```
interface Shape {  
    double area();  
    double perimeter();  
}
```

Pra quê interfaces?

- Para criar sistemas com **baixo acoplamento** e **flexíveis**.

Problema exemplo

Uma locadora brasileira de carros cobra um valor por hora para locações de até 12 horas. Porém, se a duração da locação ultrapassar 12 horas, a locação será cobrada com base em um valor diário. Além do valor da locação, é acrescido no preço o valor do imposto conforme regras do país que, no caso do Brasil, é 20% para valores até 100.00, ou 15% para valores acima de 100.00. Fazer um programa que lê os dados da locação (modelo do carro, instante inicial e final da locação), bem como o valor por hora e o valor diário de locação. O programa deve então gerar a nota de pagamento (contendo valor da locação, valor do imposto e valor total do pagamento) e informar os dados na tela. Veja os exemplos.

Example 1:

```
Enter rental data  
Car model: Civic  
Pickup (dd/MM/yyyy hh:mm): 25/06/2018 10:30  
Return (dd/MM/yyyy hh:mm): 25/06/2018 14:40  
Enter price per hour: 10.00  
Enter price per day: 130.00  
INVOICE:  
Basic payment: 50.00  
Tax: 10.00  
Total payment: 60.00
```

Calculations:

$Duration = (25/06/2018\ 14:40) - (25/06/2018\ 10:30) = 4:10 = 5\ hours$
 $Basic\ payment = 5 * 10 = 50$

$Tax = 50 * 20\% = 50 * 0.2 = 10$

Example 2:

Enter rental data
 Car model: **Civic**
 Pickup (dd/MM/yyyy hh:mm): **25/06/2018 10:30**
 Return (dd/MM/yyyy hh:mm): **27/06/2018 11:40**
 Enter price per hour: **10.00**
 Enter price per day: **130.00**
 INVOICE:
 Basic payment: 390.00
 Tax: 58.50
 Total payment: 448.50

Calculations:

$Duration = (27/06/2018\ 11:40) - (25/06/2018\ 10:30) = 2\ days + 1:10 = 3\ days$
 $Basic\ payment = 3 * 130 = 390$

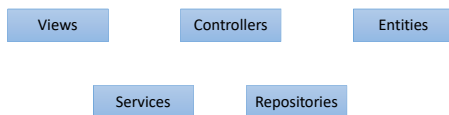
$Tax = 390 * 15\% = 390 * 0.15 = 58.50$

Solução do problema

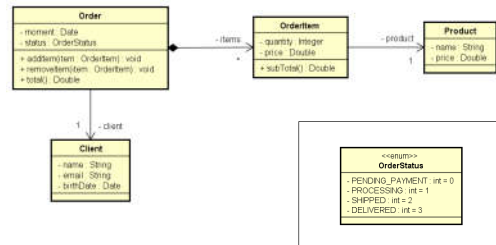
<http://educandoweb.com.br>

Prof. Dr. Nélcio Alves

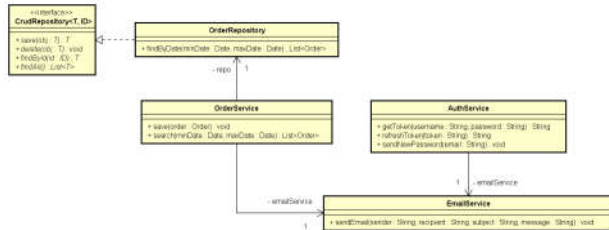
(recordando - cap. Composição)



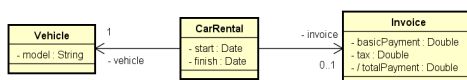
Entities



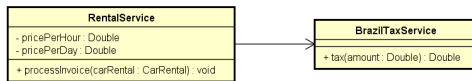
Services



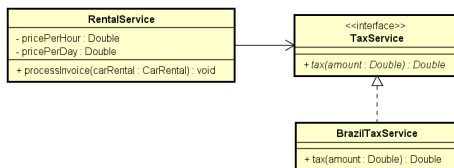
Domain layer design



Service layer design (no interface)



Service layer design



Projeto no Github

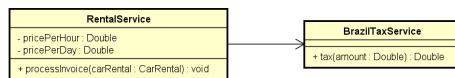
<https://github.com/acenelio/interfaces1-java>

Inversão de controle, Injeção de dependência

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

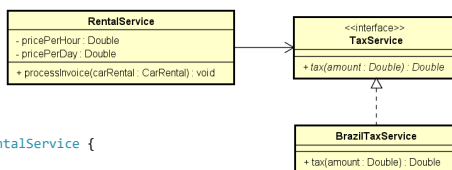
- Acoplamento forte
- A classe RentalService conhece a dependência concreta
- Se a classe concreta mudar, é preciso mudar a classe RentalService



```

class RentalService {
    (...)
    private BrazilTaxService taxService;
}
  
```

- Acoplamento fraco
- A classe RentalService não conhece a dependência concreta
- Se a classe concreta mudar, a classe RentalService não muda nada



```

class RentalService {
    (...)
    private TaxService taxService;
}
  
```

Injeção de dependência por meio de construtor

```
class Program {
    static void Main(string[] args) {
        (...)
        RentalService rentalService = new RentalService(pricePerHour, pricePerDay, new BrazilTaxService());
    }
}

class RentalService {
    private TaxService taxService;

    public RentalService(double pricePerHour, double pricePerDay, TaxService taxService) {
        this.pricePerHour = pricePerHour;
        this.pricePerDay = pricePerDay;
        this.taxService = taxService;
    }
}
```

upcasting

Inversão de controle

• Inversão de controle

Padrão de desenvolvimento que consiste em retirar da classe a responsabilidade de instanciar suas dependências.

• Injeção de dependência

É uma forma de realizar a inversão de controle: um componente externo instancia a dependência, que é então injetada no objeto "pai". Pode ser implementada de várias formas:

- Construtor
- Classe de instanciação (builder / factory)
- Container / framework

Exercício de fixação

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

Uma empresa deseja automatizar o processamento de seus contratos. O processamento de um contrato consiste em gerar as parcelas a serem pagas para aquele contrato, com base no número de meses desejado.

A empresa utiliza um serviço de pagamento online para realizar o pagamento das parcelas. Os serviços de pagamento online tipicamente cobram um juro mensal, bem como uma taxa por pagamento. Por enquanto, o serviço contratado pela empresa é o do Paypal, que aplica **juros simples** de 1% a cada parcela, mais uma **taxa** de pagamento de 2%.

Fazer um programa para ler os dados de um contrato (número do contrato, data do contrato, e valor total do contrato). Em seguida, o programa deve ler o número de meses para parcelamento do contrato, e daí gerar os registros de parcelas a serem pagas (data e valor), sendo a primeira parcela a ser paga um mês após a data do contrato, a segunda parcela dois meses após o contrato e assim por diante. Mostrar os dados das parcelas na tela.

Veja exemplo na próxima página.

Example:

```
Enter contract data
Number: 8028
Date (dd/MM/yyyy): 25/06/2018
Contract value: 600.00
Enter number of installments: 3
Installments:
25/07/2018 - 206.04
25/08/2018 - 208.08
25/09/2018 - 210.12
```

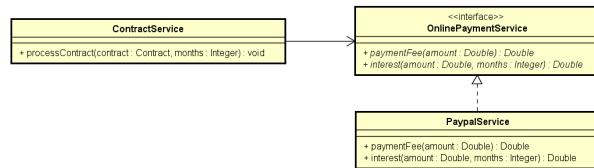
Calculations (1% monthly simple interest + 2% payment fee):

Quota #1:	Quota #2:	Quota #3:
$200 + 1\% * 1 = 202$	$200 + 1\% * 2 = 204$	$200 + 1\% * 3 = 206$
$202 + 2\% = 206.04$	$204 + 2\% = 208.08$	$206 + 2\% = 210.12$

Domain layer design (entities)



Service layer design



Repositório Github

<https://github.com/acenelio/interfaces4-java>

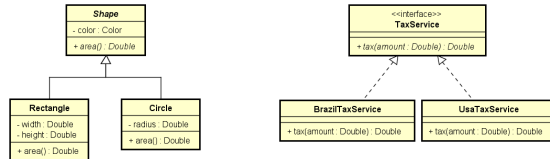
Herdar vs. cumprir contrato

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

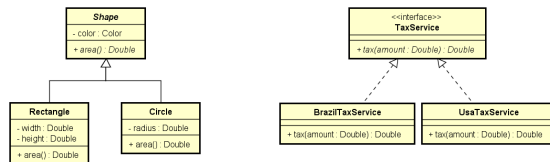
Aspectos em comum entre herança e interfaces

- Relação é-um
- Generalização/especialização
- Polimorfismo

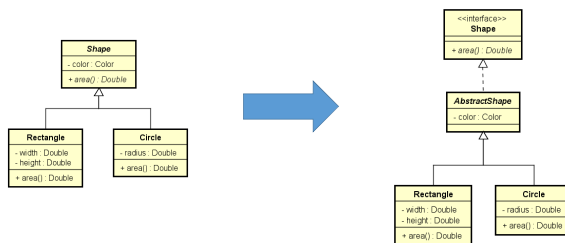


Diferença fundamental

- Herança => reuso
- Interface => contrato a ser cumprido

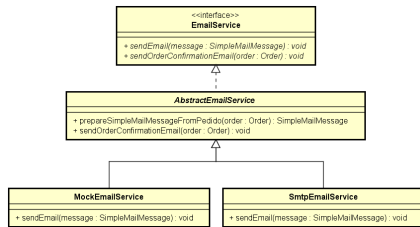


E se eu precisar implementar Shape como interface, porém também quiser definir uma estrutura comum reutilizável para todas figuras?



<https://github.com/acenelio/interfaces2-java>

Outro exemplo



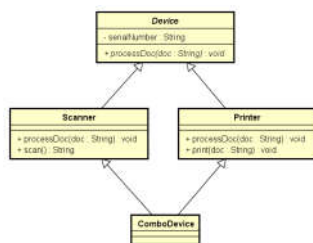
<https://github.com/acenello/springboot2-ionic-backend/tree/master/src/main/java/com/nelloalves/cursomc/services>

Herança múltipla e o problema do diamante

<http://educandoweb.com.br>

Prof. Dr. Nélcio Alves

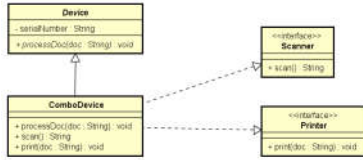
Problema do diamante



A herança múltipla pode gerar o problema do diamante: uma ambiguidade causada pela existência do mesmo método em mais de uma superclasse.

Herança múltipla não é permitida na maioria das linguagens!

Porém, uma classe pode implementar mais de uma interface



ATENÇÃO:

Isso NÃO é herança múltipla, pois NÃO HÁ REUSO na relação entre ComboDevice e as interfaces Scanner e Printer.

ComboDevide não herda, mas sim implementa as interfaces (cumpre o contrato).

<https://github.com/acenelio/interfaces3-java>

Interface Comparable

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

Interface Comparable

<https://docs.oracle.com/javase/10/docs/api/java/lang/Comparable.html>

```

public interface Comparable<T> {
    int compareTo(T o);
}
  
```

Problema motivador

Faça um programa para ler um arquivo contendo nomes de pessoas (um nome por linha), armazenando-os em uma lista. Depois, ordenar os dados dessa lista e mostra-los ordenadamente na tela. Nota: o caminho do arquivo pode ser informado "*hardcode*".

Maria Brown
Alex Green
Bob Grey
Anna White
Alex Black
Eduardo Rose
William Red
Marta Blue
Alex Brown

```
package application;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Program {

    public static void main(String[] args) {

        List<String> list = new ArrayList<>();
        String path = "C:\\temp\\lin.txt";

        try (BufferedReader br = new BufferedReader(new FileReader(path))) {

            String name = br.readLine();
            while (name != null) {
                list.add(name);
                name = br.readLine();
            }
            Collections.sort(list);
            for (String s : list) {
                System.out.println(s);
            }
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Outro problema

Faça um programa para ler um arquivo contendo funcionários (nome e salário) no formato .csv, armazenando-os em uma lista. Depois, ordenar a lista por nome e mostrar o resultado na tela. Nota: o caminho do arquivo pode ser informado "*hardcode*".

Maria Brown,4300.00
Alex Green,3100.00
Bob Grey,3100.00
Anna White,3500.00
Alex Black,2450.00
Eduardo Rose,4300.00
William Red,2900.00
Marta Blue,6100.00
Alex Brown,5000.00

Interface Comparable

```
public interface Comparable<T> {
    int compareTo(T o);
}
```

```
System.out.println("maria".compareTo("alex"));
System.out.println("alex".compareTo("maria"));
System.out.println("maria".compareTo("maria"));
```

Output:

```
12
-12
0
```

<https://docs.oracle.com/javase/10/docs/api/java/lang/Comparable.html>

Method compareTo:

Parameters:

o - the object to be compared.

Returns:

a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

```
package application;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import entities.Employee;

public class Program {

    public static void main(String[] args) {

        List<Employee> list = new ArrayList<>();
        String path = "%\\temp\\lab.txt";

        try (BufferedReader br = new BufferedReader(new FileReader(path))) {

            String employeeCsv = br.readLine();
            while (employeeCsv != null) {
                String[] fields = employeeCsv.split(",");
                list.add(new Employee(fields[0], Double.parseDouble(fields[1])));
                employeeCsv = br.readLine();
            }
            Collections.sort(list);
            for (Employee emp : list) {
                System.out.println(emp.getName() + ", " + emp.getSalary());
            }
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

```
package entities;

public class Employee implements Comparable<Employee> {

    private String name;
    private Double salary;

    public Employee(String name, Double salary) {
        this.name = name;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Double getSalary() {
        return salary;
    }

    public void setSalary(Double salary) {
        this.salary = salary;
    }

    @Override
    public int compareTo(Employee other) {
        return name.compareTo(other.getName());
    }
}
```

Default methods

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

Default methods (defender methods)

- A partir do Java 8, interfaces podem conter métodos concretos.
- A intenção básica é prover implementação padrão para métodos, de modo a evitar:
 - 1) repetição de implementação em toda classe que implemente a interface
 - 2) a necessidade de se criar classes abstratas para prover reuso da implementação
- Outras vantagens:
 - Manter a retrocompatibilidade com sistemas existentes
 - Permitir que "interfaces funcionais" (que devem conter apenas um método) possam prover outras operações padrão reutilizáveis

Problema exemplo

Fazer um programa para ler uma quantia e a duração em meses de um empréstimo. Informar o valor a ser pago depois de decorrido o prazo do empréstimo, conforme regras de juros do Brasil. A regra de cálculo de juros do Brasil é juro composto padrão de 2% ao mês.

Veja o exemplo.

ExampleAmount: **200.00**Months: **3**Payment after 3 months:
212.24

BrazilInterestService
- interestRate : double
+ payment(amount : double, months : int) : double

Calculations: $\text{Payment} = 200 * 1.02 * 1.02 * 1.02 = 200 * 1.02^3 = 212.2416$
 $\text{Payment} = \text{amount} * (1 + \text{interestRate} / 100)^N$

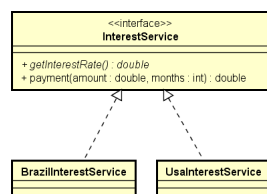
<https://github.com/acenelio/interfaces5-java>

What if there was another interest service from another country?Amount: **200.00**Months: **3**Payment after 3 months:
206.06

UsaInterestService
- interestRate : double
+ payment(amount : double, months : int) : double

Calculations: $\text{Payment} = 200 * 1.01 * 1.01 * 1.01 = 200 * 1.01^3 = 206.0602$
 $\text{Payment} = \text{amount} * (1 + \text{interestRate} / 100)^N$

<https://github.com/acenelio/interfaces5-java>



Considerações importantes

- Sim: agora as interfaces podem prover reuso
- Sim: agora temos uma forma de herança múltipla
 - Mas o compilador reclama se houver mais de um método com a mesma assinatura, obrigando a sobrescreve-lo
- Interfaces ainda são bem diferentes de classes abstratas. Interfaces não possuem recursos tais como construtores e atributos.
