LP Orientada a Objetos II

UML e Funções Lambda

Diego Addan

DS142 - UFPR - 2023

Para hoje

Funções Anônimas e Lambda

Mais de modelagem

Exercícios

Funções Lambda

Funções Lambda

São funções de primeira ordem

Podem ser criadas, armazenadas e passadas como parâmetro (como outras variáveis).

Tem visibilidade diferente das funções normais

Utilizadas em padrões de projeto e interfaces gráficas

Funções Lambda

Funções Lambda

Funções Lambda são anônimas, e sua sintaxe tem sempre:

(parametro) -> <expressão>

A expressão pode ou não retornar algo.

Ex

(Car car) -> car.getColor( ).equals(“red”);

Funções Lambda

Funções Lambda

Criamos uma lista de valores (e saída de visualização)

```
ArrayList<Integer> valores = new ArrayList<Integer>();
valores.add( e: 1);valores.add( e: 22);valores.add( e: 15);
valores.add( e: 13);valores.add( e: 4);valores.add( e: 8);
System.out.println( x: valores);
```

Para criar um processo de exibir o dobro dos valores, ou somente as entradas par da lista, poderíamos criar laços que percorrem e filtram/calculam, mas o código ficaria pouco otimizado.

Funções Lambda

Funções Lambda

Podemos percorrer nossa lista de **valores** com o comando **forEach**, incluindo em seus parâmetros a sintaxe: valores.forEach(( ) -> { });

```
ArrayList<Integer> valores = new ArrayList<Integer>();
valores.add( e: 1);valores.add( e: 22);valores.add( e: 15);
valores.add( e: 13);valores.add( e: 4);valores.add( e: 8);
System.out.println( x: valores);
ArrayList<Integer> dobro = new ArrayList<Integer>();
valores.forEach((v)->{ dobro.add(v*2);});
System.out.println( x: dobro);
```

Funções Lambda

Funções Lambda

Também podemos adicionar operações lógicas como valor de um Consumer, ex:

Consumer<Tipo do Retorno> = (parametro) -> { Operação };

```
ArrayList<Integer> dobro = new ArrayList<Integer>();
Consumer<Integer> dobrar = (v)->{dobro.add(v*2);};
valores.forEach( action: dobrar);
```

Funções Lambda

Funções Lambda

Podemos usar funções Lambda para operações mais complexas

```
ArrayList<Integer> dobro = new ArrayList<Integer>();
ArrayList<Integer> par = new ArrayList<Integer>();
valores.forEach((v)->{
    dobro.add(v*2);
    if(v%2 == 0){
        par.add( e: v);
    }
});
System.out.println( x: dobro);
System.out.println( x: par);
```

Funções Lambda

Funções Lambda

Muito comum em arquiteturas mobile e web

Javascript

```js
let fn = (x) => x * 3 + 1;
consolp.log(fn(2)) //7
```

Java

```java
Function<Integer, Integer> function = (x) -> x * 3 + 1;
System.out.print(function.apply(2));
```

Funções Lambda

Funções Lambda

Pode ser usada com Generics. Vamos ver um exemplo para imprimir qualquer lista usando (f)lambda:

```
public class Lambda01 {

    private static <T> void fEach(List<T> list, Consumer<T> consumer){
    for (T e : list){ consumer.accept( t: e);}
}

    public static void main(String[] args) {
    List<String> str = List.of( e1: "Ana",  e2: "Pedro",  e3: "Carlos",  e4: "Dante");
        fEach( list: str, (String s) -> System.out.println( x: s));
```

Para imprimir uma outra lista, de inteiros por exemplo, bastaria alterar o tipo na chamada

Funções Lambda

Funções Lambda

Uma função Lambda pode ter um parâmetro recebido e um de retorno.

Ex: Recebe uma lista de Strings e retorna o tamanho de cada uma.

```
private static <T, R> List<R> map(List<T> list, Function<T, R> function){
List<R> result = new ArrayList<>();
for(T e : list){
    R r = function.apply( t: e);
    result.add( e: r);
}
return result;
}
```

Funções Lambda

Funções Lambda

```
private static <T, R> List<R> map(List<T> list, Function<T, R> function){
List<R> result = new ArrayList<>();
for(T e : list){
    R r = function.apply( t: e);
    result.add( e: r);
}
return result;
}

public static void main(String[] args) {
List<String> str = List.of( e1: "Ana",  e2: "Pedro",  e3: "Carlos",  e4: "Dante");
List<Integer> inteiros = map( list: str, (String s)->s.length());
System.out.println( x: inteiros);
```

Funções Lambda

Funções Lambda

Neste caso poderia passar qualquer função como parâmetro:

List<String> fn = map(str, s -> s.toUpperCase( ));

List<Integer> inteiros = map(str, (String s)->s.length());

```
public static void main(String[] args) {
List<String> str = List.of( e1: "Ana",  e2: "Pedro",  e3: "Carlos",  e4: "Dante");
List<Integer> inteiros = map( list: str,  (String s)->s.length());
System.out.println( x: inteiros);
}}
```

Funções Lambda

Classes Anônimas e funções Lambda

Como uma função Lambda encaixa no conceito de uma interface anônima?

SAM (Single Abstract Method)

Se a classe so possui um método abstrato podemos utilizar uma função Lambda para implementar este recurso.

Trabalho.realizarJornada() Precisamos implementar o método assalariado

Funções Lambda

Classes Anônimas e funções Lambda

Não recebe um parâmetro e gera como saída uma interface que implementa uma String, como função de valor:

```
Trabalho.realizarJornada( ( ) -> {

        System.out.println( " Trabalhando de forma ágil" );

        } );
```

Mais compacto e limpo

Funções Lambda

Classes Anônimas e funções Lambda

```
(Integer idade) → idade > 10; //Entrada é um inteiro e devolve um
booleano

(Integer idade) → idade < 10? "Criança" : "Não é Criança"; //Entrada
é um inteiro e devolve uma String

(Integer idade) → {System.out.println(idade);}; //Entrega é
Inteiro, saída é um void

() → {return Math.random() + "Number";}; //Sem Entrada, e devolve
String
```

Funções Lambda

Classes Anônimas e funções Lambda

Também é possível

definir (f)Lambda com

Predicados compostos

```
(nome, list) → {
    return (
        list.stream()
            .filter(e → e.getNome().startsWith(nome))
            .map(Pessoa::getIdade)
            .findFirst()
    );
};
```

Modelagem de sistemas através de UML

UML

Modelagem de sistemas através de UML

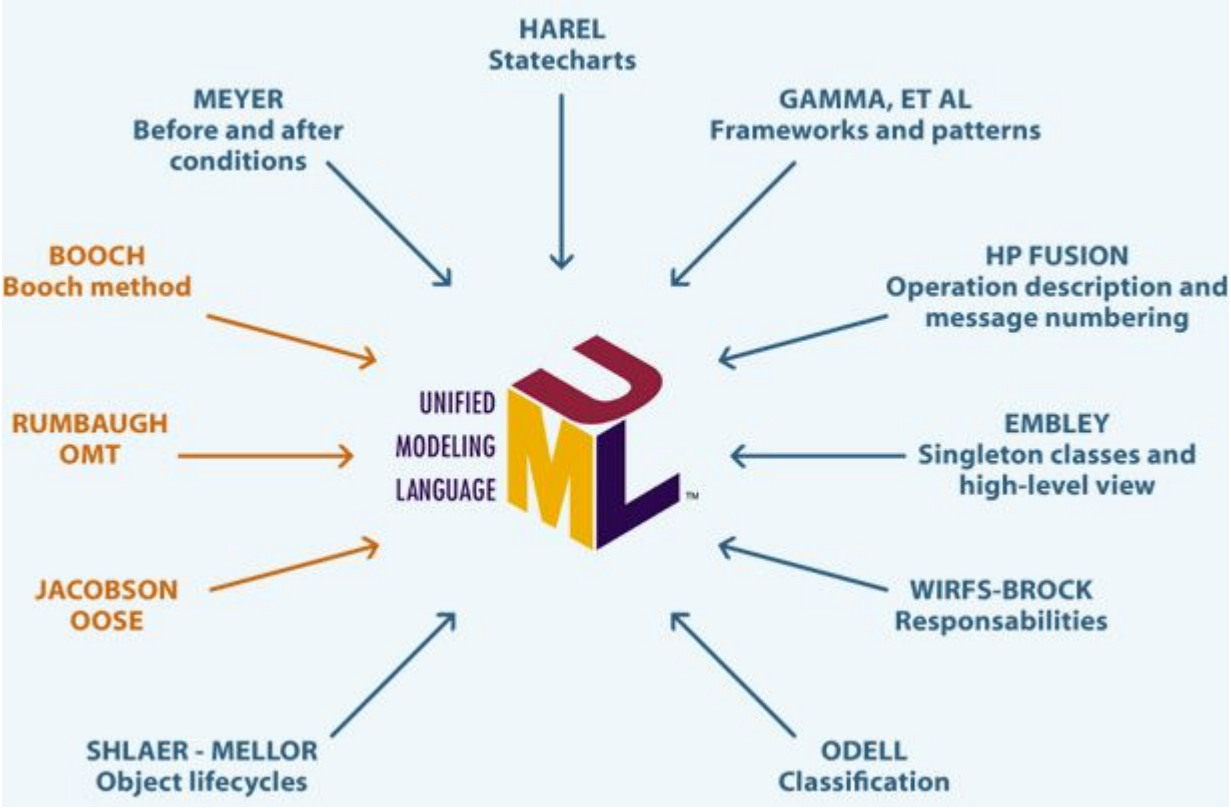
Como eu uso UML para desenvolver sistemas?

Fase de análise:

- tamanho da equipe,
- escopo orçamentário,
- Tecnologias
- Design participativo

Existem diversas formas de modelagem, formais ou não, de sistemas computacionais.

Modelagem de sistemas através de UML



Modelagem de sistemas através de UML

Diagramas Estruturais: priorizam a descrição estática de estruturas de um sistema, como classes, atributos e operações, além de prováveis relacionamentos entre tais construções;

Diagramas Comportamentais: detalha o funcionamento (comportamento) de partes de um sistema ou processos de negócio;

Diagramas de Interação: considerados um subgrupo dos diagramas comportamentais, sendo normalmente utilizados na representação de interações entre objetos de uma aplicação;

Modelagem de sistemas através de UML

Diagramas Estruturais:

- Diagrama de classes
- Diagrama de componentes: Módulo
- Diagrama de pacotes: Agrupamento lógico
- Diagrama de instalação: Hardware, implantação
- Diagrama de perfil: Estruturas customizadas

Diagramas Comportamentais: detalha o funcionamento (comportamento) de partes de um sistema ou processos de negócio;

Diagramas de Interação: considerados um subgrupo dos diagramas comportamentais, sendo normalmente utilizados na representação de interações entre objetos de uma aplicação;

Modelagem de sistemas através de UML

Diagramas Estruturais:

Diagramas Comportamentais:

- Casos de Uso: Funcionalidades
- Atividades: Tarefas
- Transição de Estados

Diagramas de Interação: considerados um subgrupo dos diagramas comportamentais, sendo normalmente utilizados na representação de interações entre objetos de uma aplicação;

Modelagem de sistemas através de UML

Diagramas Estruturais:

Diagramas Comportamentais:

Diagramas de Interação:

- Sequência
- Colaboração ou Comunicação
- Tempo

Modelagem de sistemas através de UML

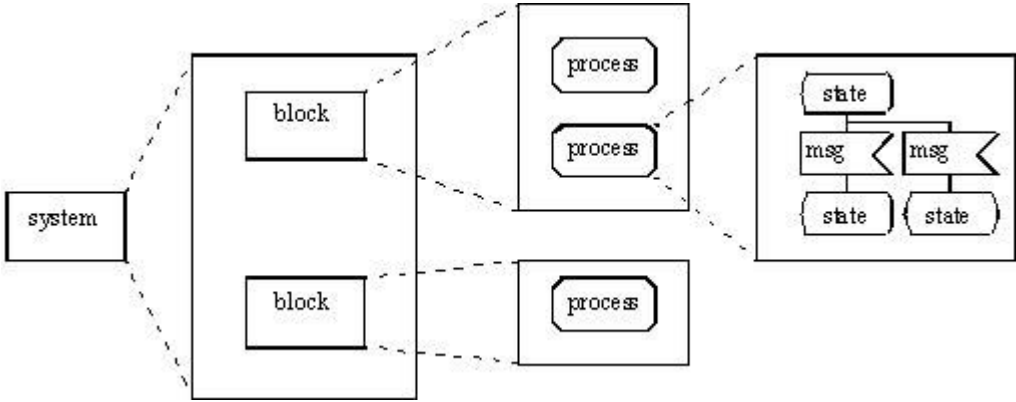
Ferramentas para a geração de diagramas UML

- Visio: Pacote office
- Enterprise UML - Sparx System: Engenharia reversa e geração de código
- Draw.io

Quando não usar UML?

Specification and Description Language

(SDL)



Modelagem de sistemas através de UML

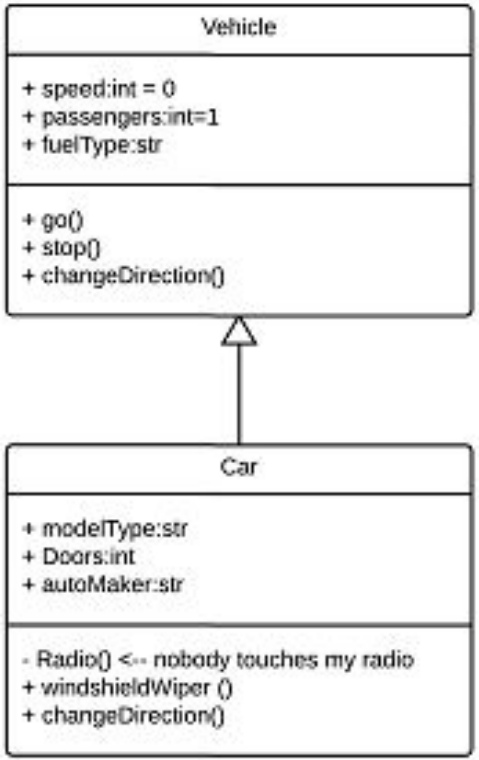
POO em UML

Componentes básicos

Parte superior: contém o nome da classe.

Parte do meio: contém os atributos da classe.

Parte inferior: inclui as operações da classe (métodos).



Modelagem de sistemas através de UML

POO em UML

Modificadores de acesso

- Público (+)
- Privado (-)
- Protegido (#)
- Pacote (~)
- Derivado (/)
- Estático (sublinhado)

Modelagem de sistemas através de UML

POO em UML

- **Pacotes**: formas projetadas para organizar classificadores relacionados em um diagrama. São simbolizados por uma grande forma de retângulo com abas.
- **Interfaces**: Interfaces são semelhantes às classes, exceto que uma classe pode ter uma instância de seu tipo, e uma interface deve ter pelo menos uma classe para implementá-la.
- **Objetos**: instâncias de uma classe ou classes. Objetos podem ser adicionados a um diagrama de classes para representar instâncias concretas ou prototípicas.
- **Artefatos**: elementos de modelo que representam as entidades concretas em um sistema de software, tais como documentos, bancos de dados, arquivos executáveis, componentes de software etc.

Modelagem de sistemas através de UML

POO em UML

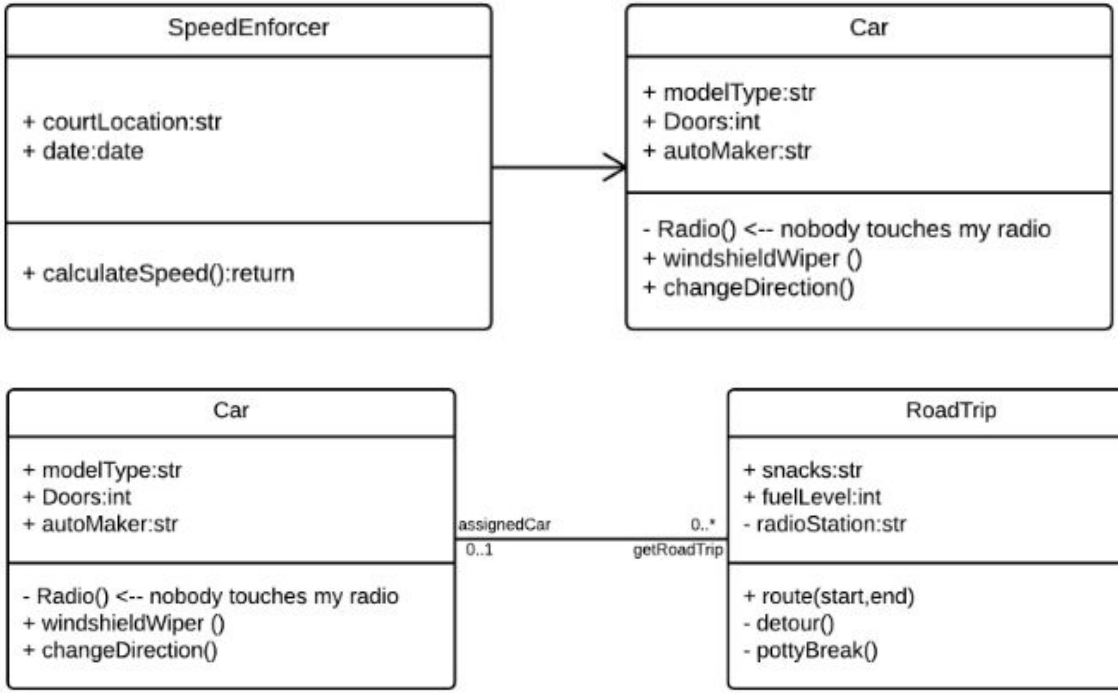
Interações

Hereditariedade: Seta

Associação bidirecional:

0..1 “assignedCar” (carro atribuído)

0..* “getRoadTrip” (obter passeio)



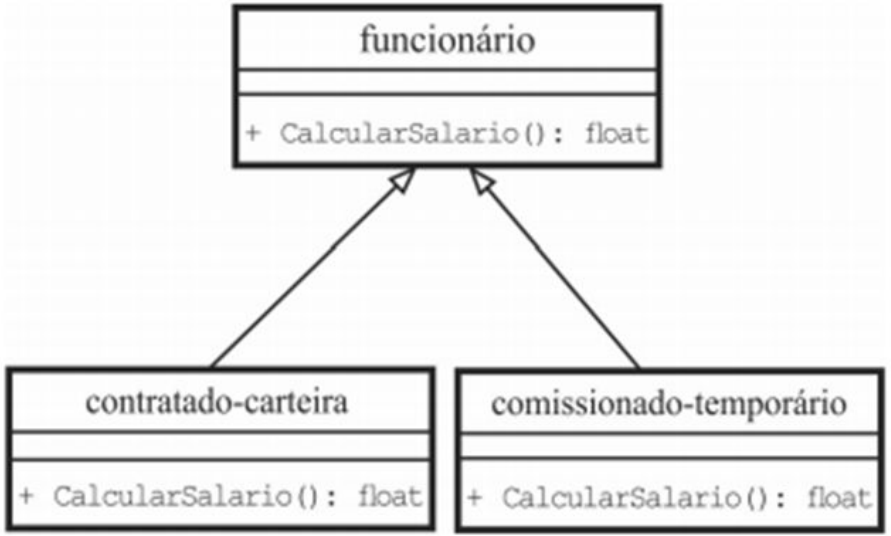
Modelagem de sistemas através de UML

POO em UML

Polimorfismo: Seta unidirecional

Classes Anônimas

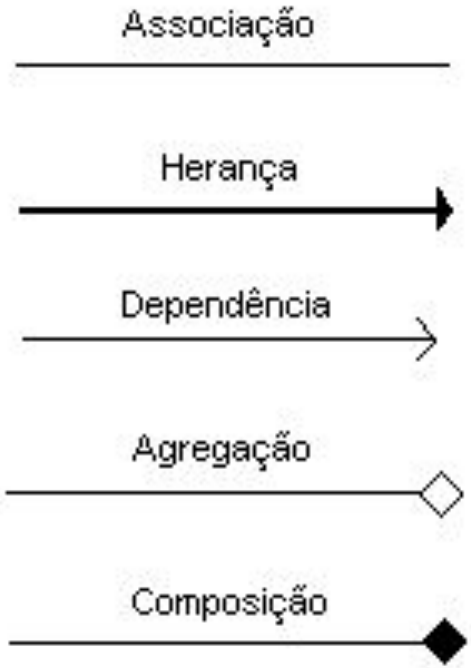
Métodos Genéricos



Modelagem de sistemas através de UML

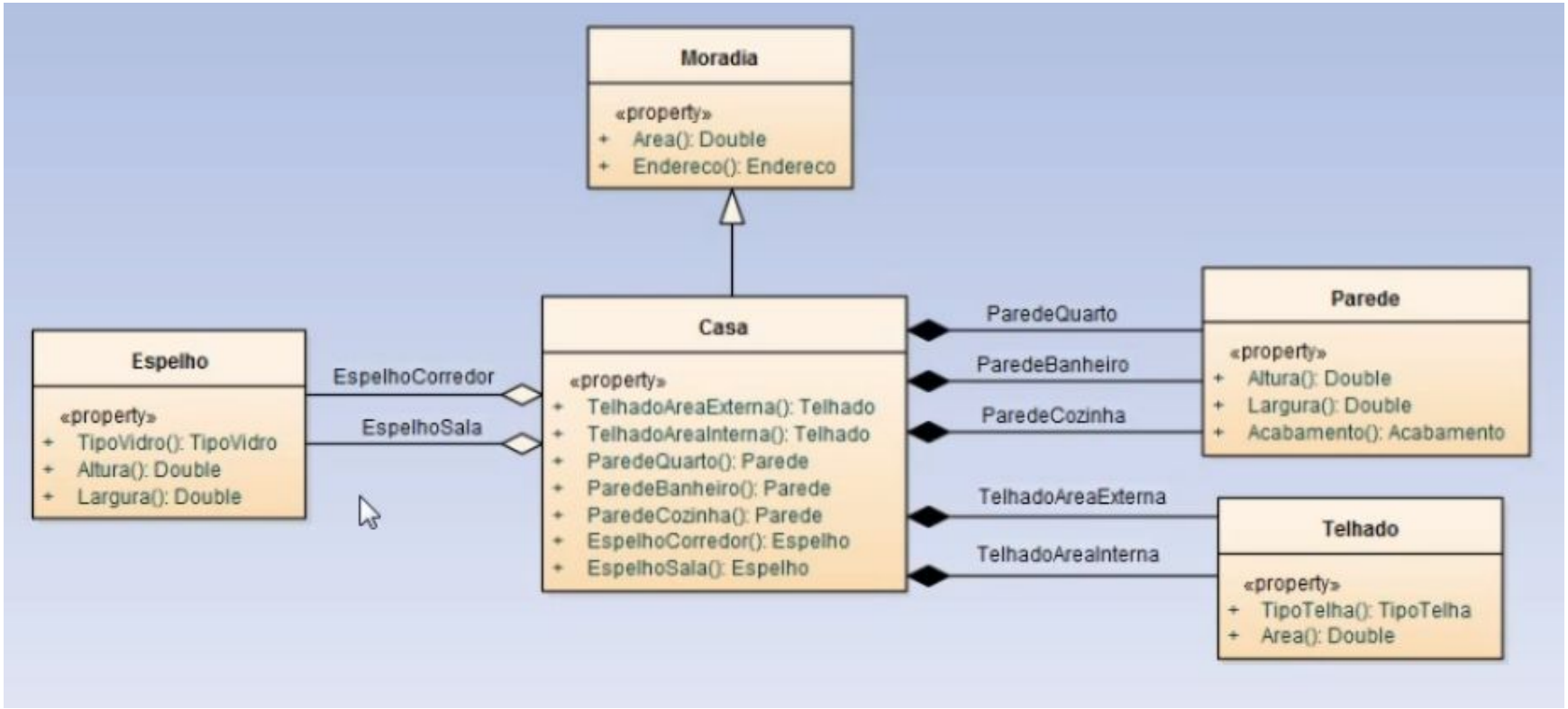
POO em UML

Setas



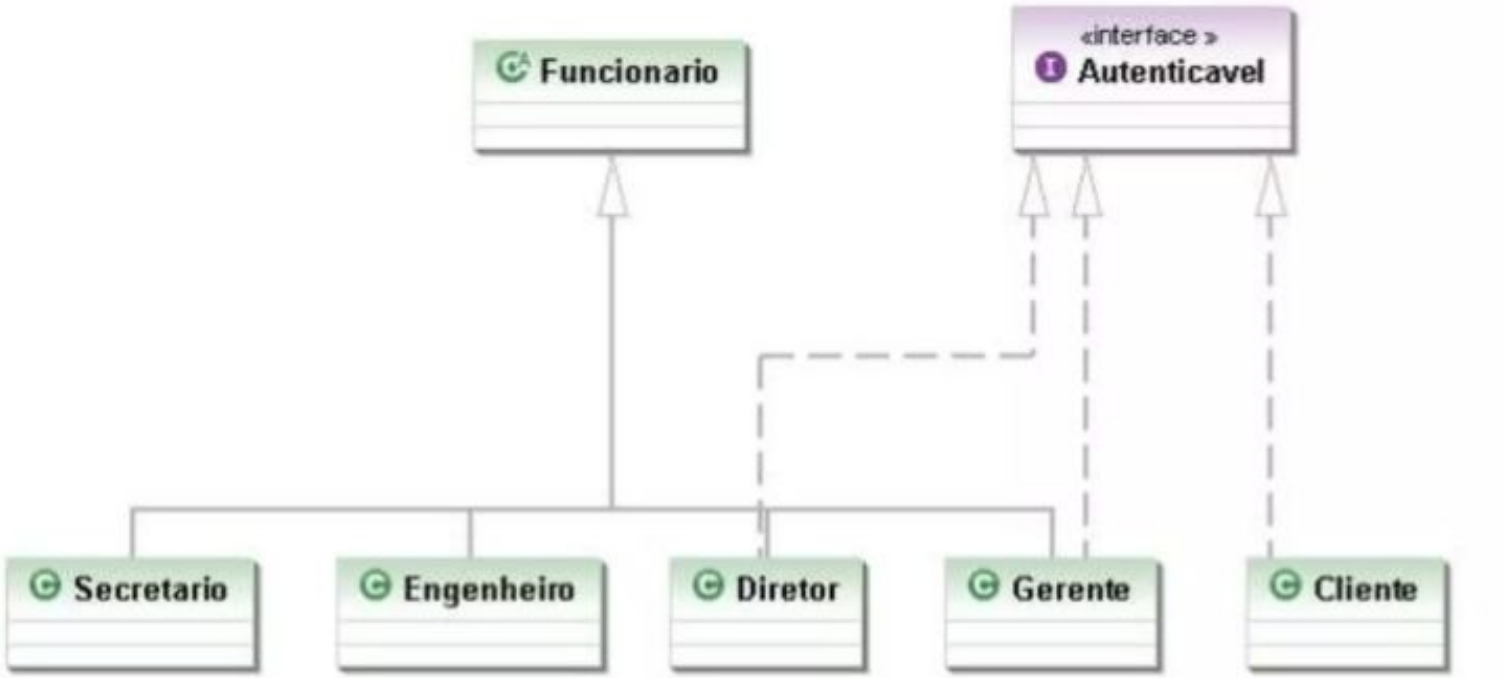
Modelagem de sistemas através de UML

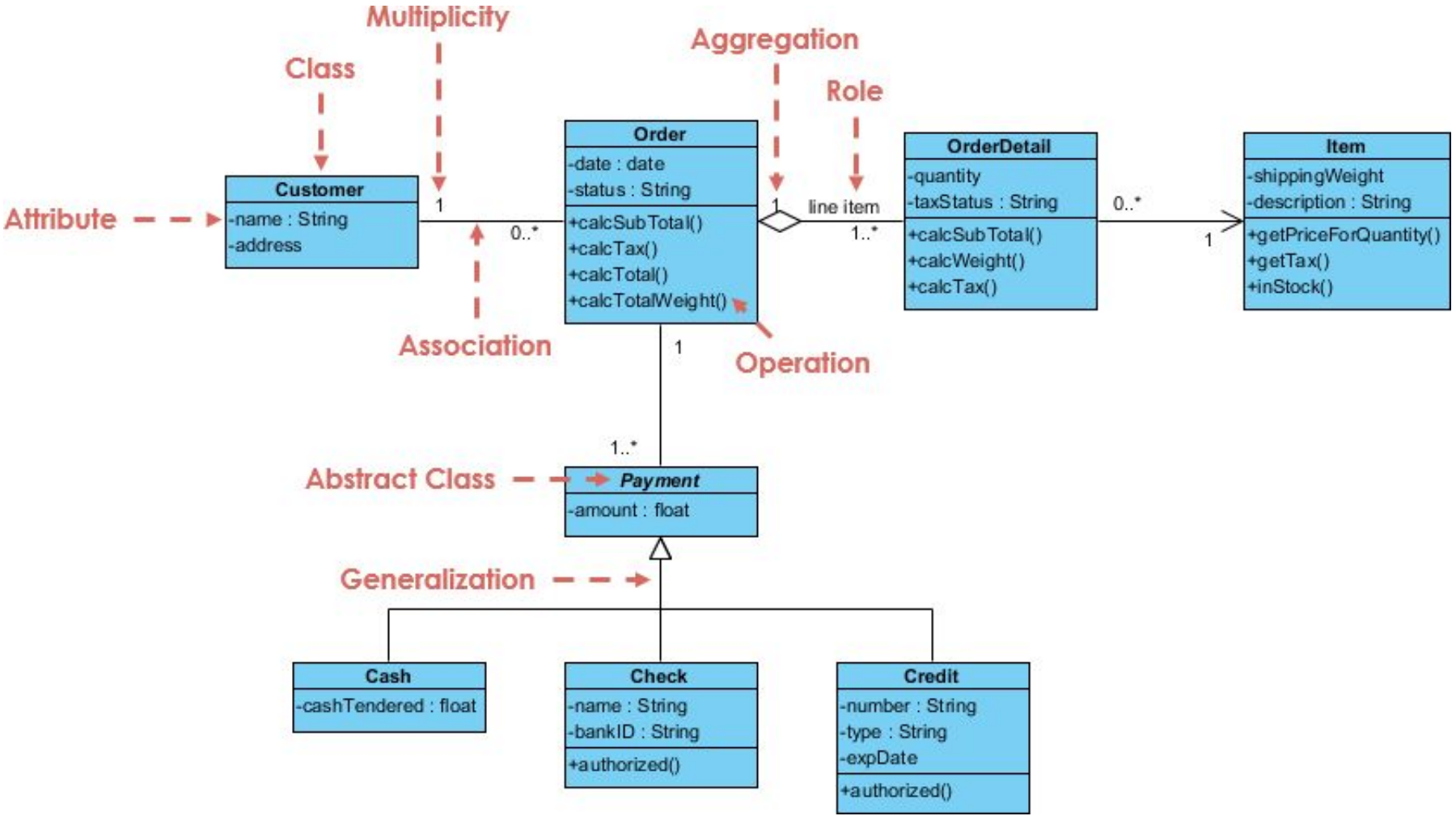
POO em UML: Agregação e Composição entre classes



Modelagem de sistemas através de UML

POO em UML: Agregação e Composição entre classes




Multiplicity
Class
Aggregation
Role
Attribute
Customer
-name : String
-address
1
0..*
Association
Order
-date : date
-status : String
+calcSubTotal()
+calcTax()
+calcTotal()
+calcTotalWeight()
1
line item
1..*
OrderDetail
-quantity
-taxStatus : String
+calcSubTotal()
+calcWeight()
+calcTax()
0..*
1
Item
-shippingWeight
-description : String
+getPriceForQuantity()
+getTax()
+inStock()
Operation
1
1..*
Abstract Class
Payment
-amount : float
Generalization
Cash
-cashTendered : float
Check
-name : String
-bankID : String
+authorized()
Credit
-number : String
-type : String
-expDate
+authorized()

Conclusão

Funções Lambda

UML

Exercício Prático

Referências

1. DEITEL. JAVA Como Programar. 8a. ed. São Paulo: Pearson Prentice Hall, 2010.

2. JANDL JUNIOR, Peter. Java Guia do Programador. São Paulo: Novatec, 2014.

3. FREEMAN, Eric. Use a cabeça: padrões e projetos. 2. ed. rev. Rio de Janeiro: Alta Books, 2009

Exercício

1) Criar a classe *Pessoa* com as seguintes características:
   - atributos: idade e dia, mês e ano de nascimento, nome da pessoa
   - métodos:
     - *calculaIdade()*, que recebe a data atual em dias, mês e anos e calcula e armazena no atributo *idade* a idade atual da pessoa
     - *informaIdade()*, que retorna o valor da idade
     - *informaNome()*, que retorna o nome da pessoa
     - *ajustaDataDeNascimento()*, que recebe dia, mês e ano de nascimento como parâmetros e preenche nos atributos correspondentes do objeto.
   - Criar dois objetos da classe *Pessoa*, um representando Albert Einstein (nascido em 14/3/1879) e o outro representando Isaac Newton (nascido em 4/1/1643)
   - Fazer uma classe principal que instancie os objetos, inicialize e mostre quais seriam as idades de Einstein e Newton caso estivessem vivos.