Análise e Desenvolvimento de Software

Unified Modeling Language - Classes

Pedro Emanuel Cardoso de Sousa

pesousa@ipca.pt

UML - CLASSES

- Porque existem:
  - Modelar/apresentar os termos que existem no negócio
  - Modelar as relações entre classes que implementam os casos de uso
  - Modelar a estrutura das classes e os interfaces do sistema
  - Define a razão do sistema e a forma de operação
  - Constrói a implementação base de um sistema em Orientação por Objetos

Classes – Modelar termos de negócio

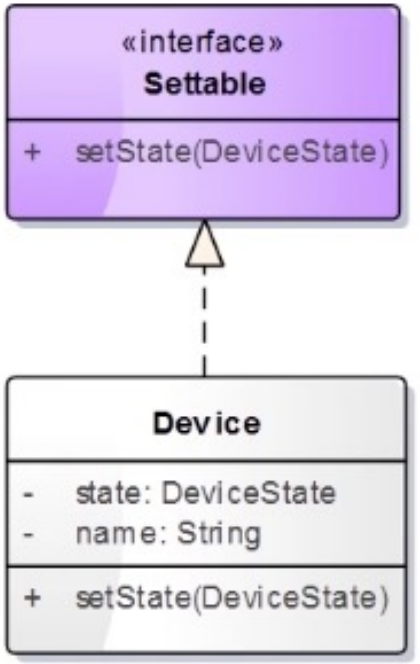
- Modelo conceptual sobre os tópicos do negócio ou problema (diagrama);
- Documenta os pontos relevantes: entidades, caraterísticas, papeis, relações e restrições;
- Ajuda a clarificar o vocabulário do negócio ou problema;
- É um canal de comunicação partilhado para entendimento do domínio ou problema;
- Um diagrama de classes é conhecido como modelo do domínio;

Classes - porquê

- Permite dividir o problema em elementos e propõe um desenho sintetizado da solução para o problema;
- Raramente existe apenas uma solução;
- Permite ao analista visualizar diferentes soluções, sem ter que produzir software na realidade;

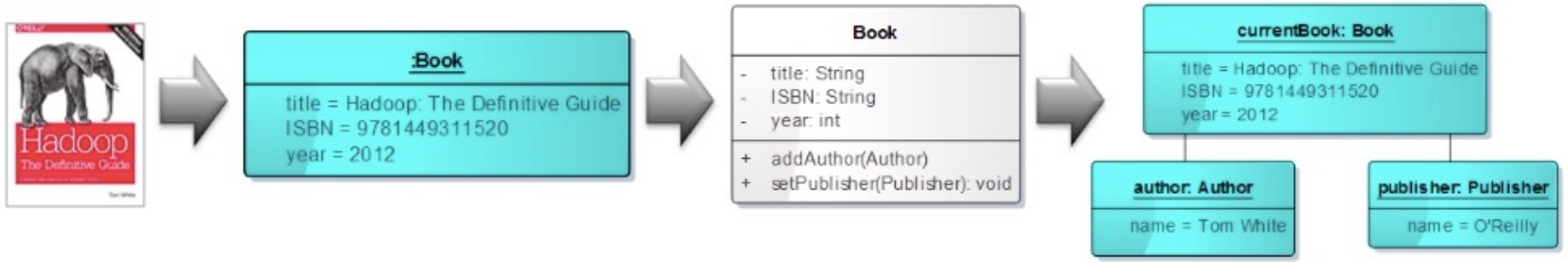
Classes – base da implementação

```
package com.armadillosh.ha.domain;

/**
 * An electrical device that is controlled by
 * the home automation system.
 * @author Simon Bennett
 * @version 1.0
 */
public class Device implements Settable {

      private DeviceState state;
      private String name;

      /**
       *
       * @param state
       */
      public setState(DeviceState state){
            this.state = state;
      }
```



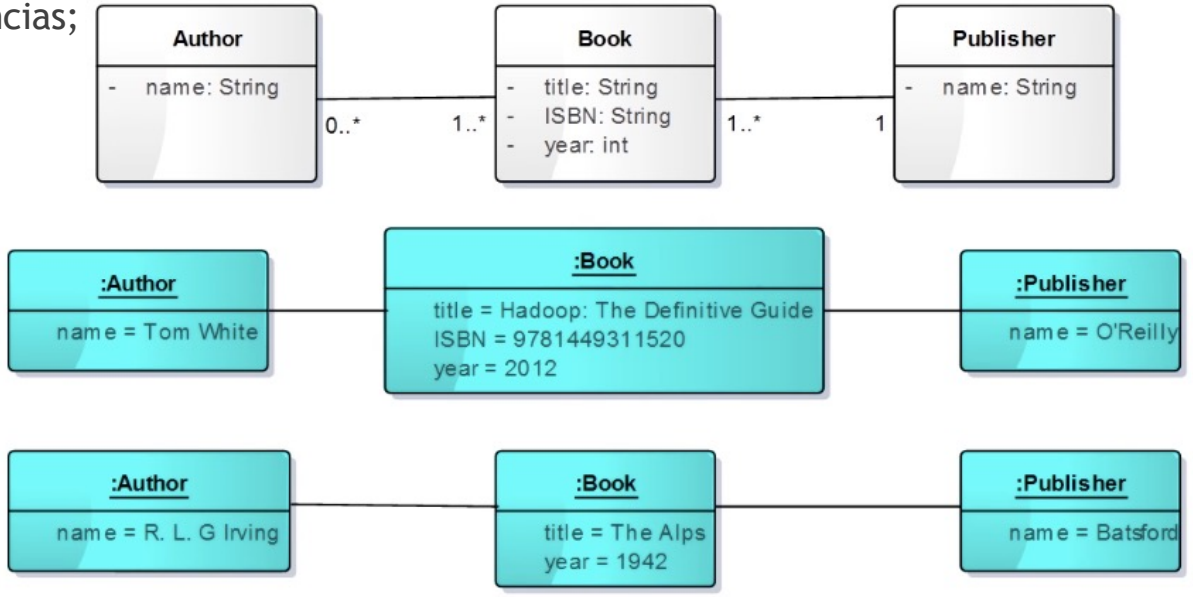
Objetos vs Classes

- Programação Orientada por Objetos
- Objetos representam instâncias individuais (da sua abstração) de coisas relevantes para o domínio;
- Classes representam definições de tipos de coisas relevantes para o domínio;
- Instâncias de objetos colaboram em tempo de execução;



Classes - definições

- As classes definem os tipos de:
  - Instâncias a que pertencem os objetos;
  - As propriedades comuns dessas instâncias;

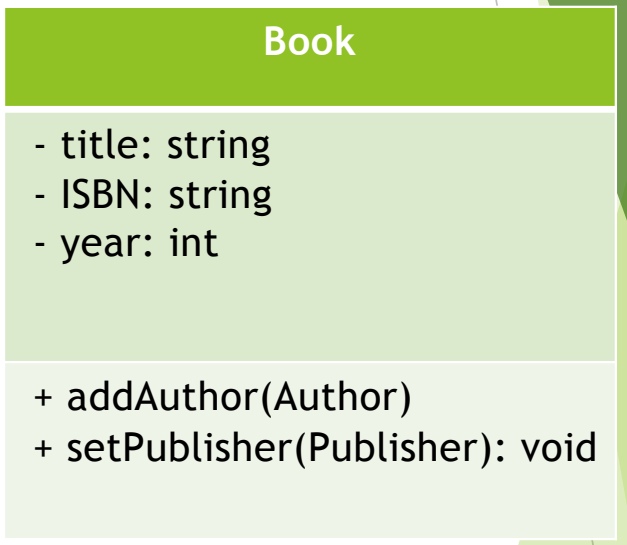


Classes - Notation

- Classes representam categorias de pessoas, coisas, eventos;
- A notação simples é um retângulo com o nome da classe no topo;
- Nome da classe – inicia-se com a letra maiúscula;
- Pascal → para múltiplas palavras juntar as mesmas com as iniciais em maiúsculas: BookEdition;

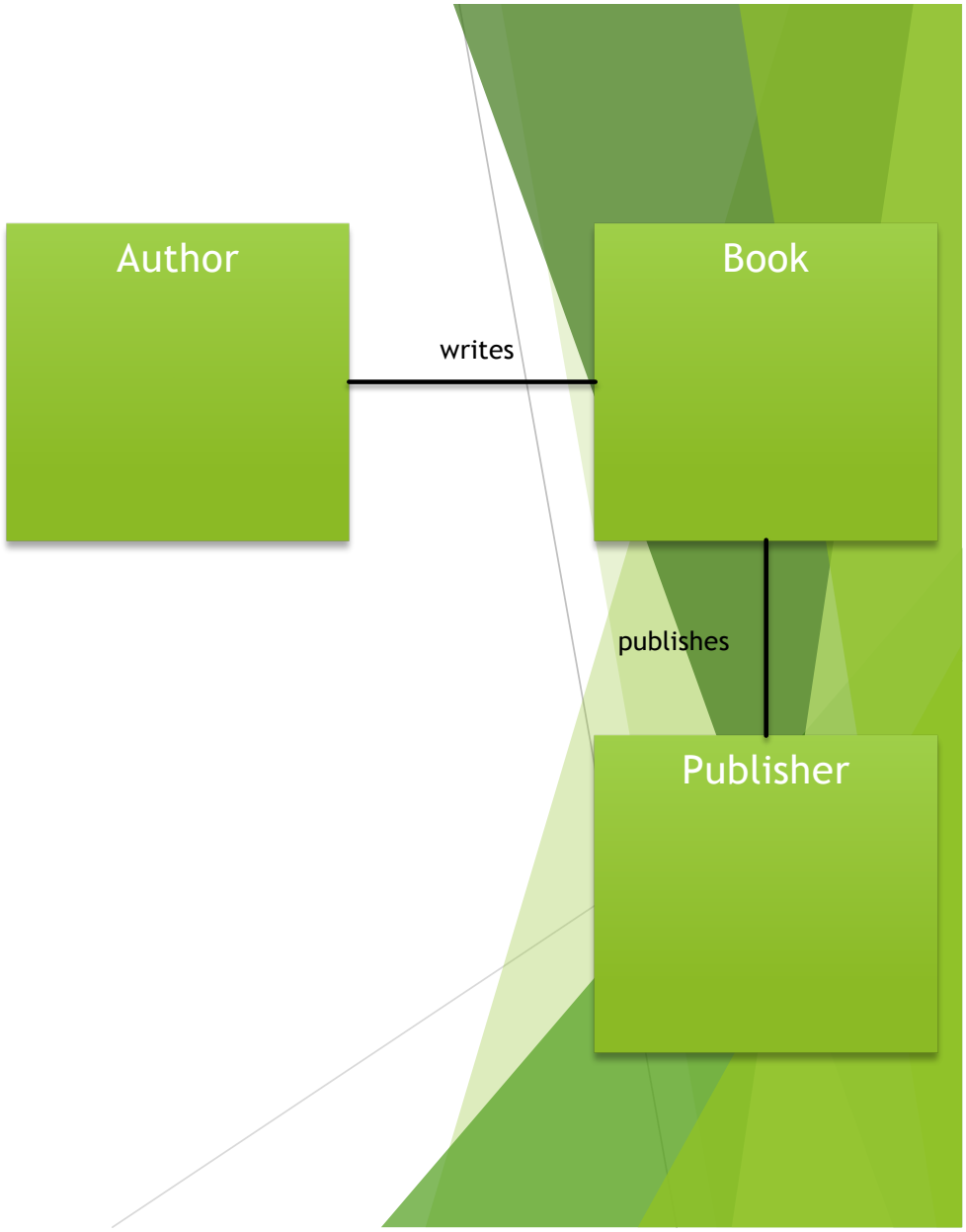
Classes - divisões

- Num diagrama, uma classe pode ter múltiplos compartimentos;
- Normalmente temos 2 para atributos e operações;



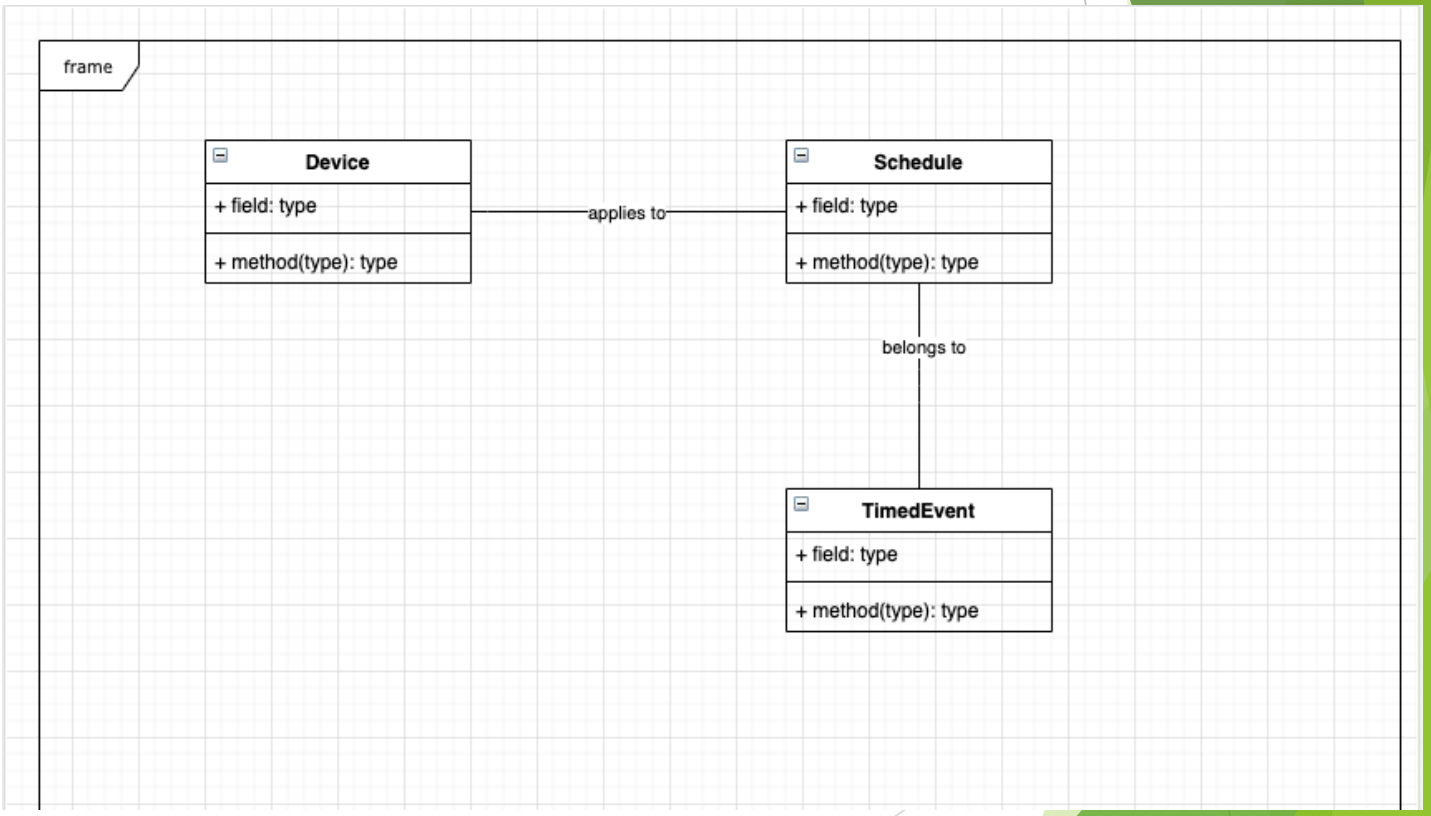
Classes - associações

- Associações: são relações entre 2 classes que representam relações no mundo real;
- Notação simples: linha sólida com a descrição da relação;
- Nome da associação: é um verbo que está relacionado com a relação:
  - “Na Author writes a Book”;



Exercício

- Abrir o DRAW.IO
- Template de UML
- Construir 3 classes
- Estabelecer ligações



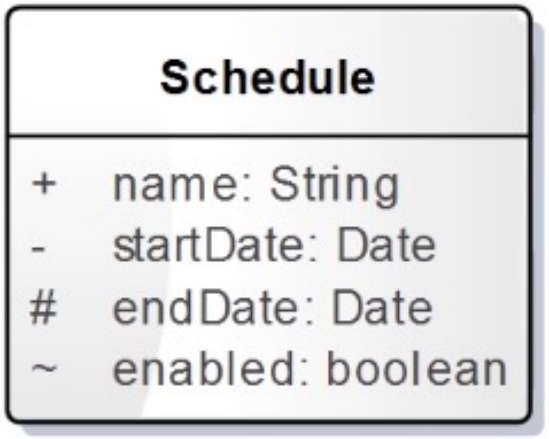
Classes - Atributos e Operações

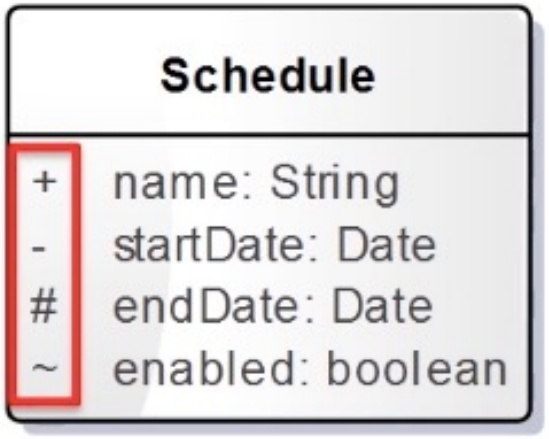
- Atributos:
  - Visibilidade
  - Tipo
  - Valor por defeito
  - Atributos derivados
- Operações:
  - Visibilidade
  - Valores de retorno
  - Parâmetros
  - ”Getters and Setters”

Classes - Atributos

- São apresentados no primeiro compartimento, depois do nome da classe;
- Também são conhecimentos como:
  - propriedades
- Nomes dos atributos utilizam o formato “camel case”

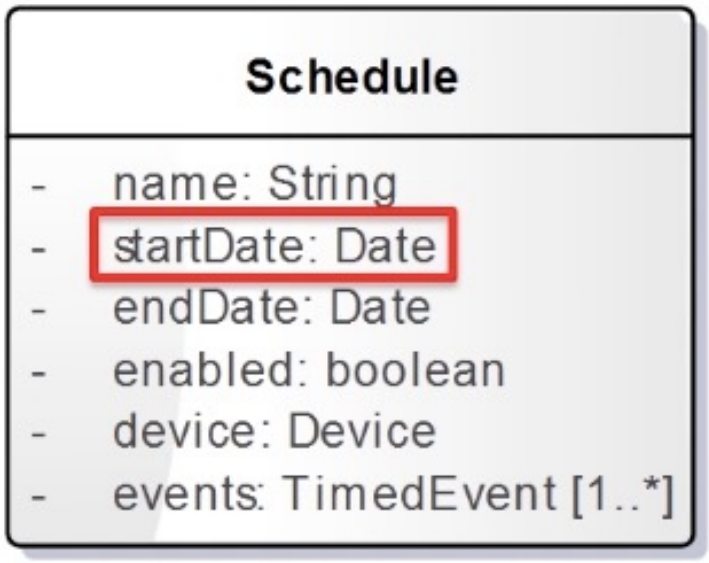
- Visibilidade:
  - + public
  - - private
  - # protected
  - ~ package





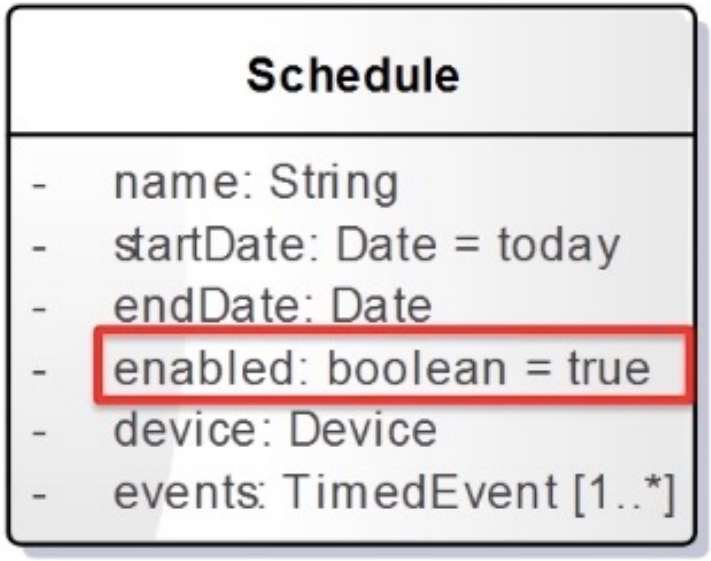
Classes – atributos - tipos

- O tipo de atributo é separado por :
- Os tipos podem ser:
  - Primitivos;
  - Classes:
    - Da libraria
    - Do modelo
  - Podem receber múltiplos valores, com limite ou sem limite:
    - Exemplo
      - Mínimo: 1 TimedEvent;
      - Máximo: ilimitado;



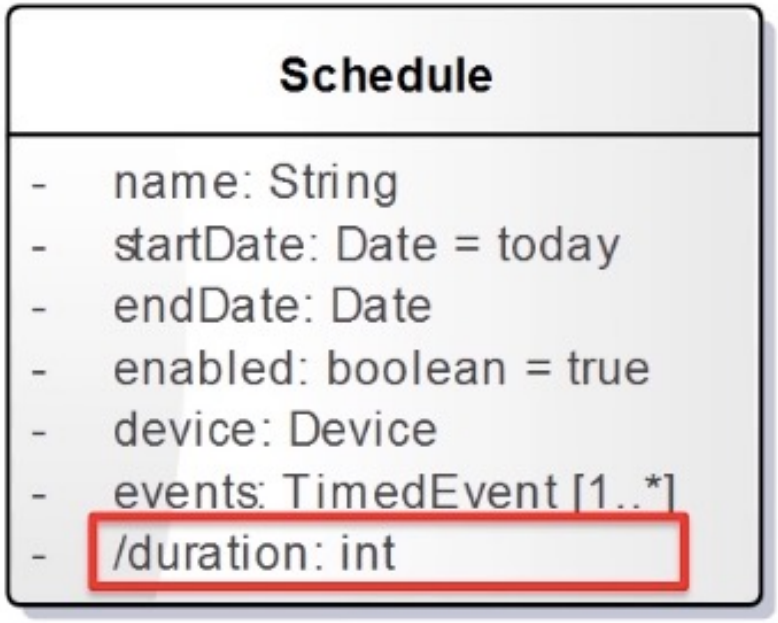
Classes – atributos – valor inicial

- Valor por defeito ou valor inicial
- Definido com um sinal de =
- Quando um objeto é criado, este valor é definido



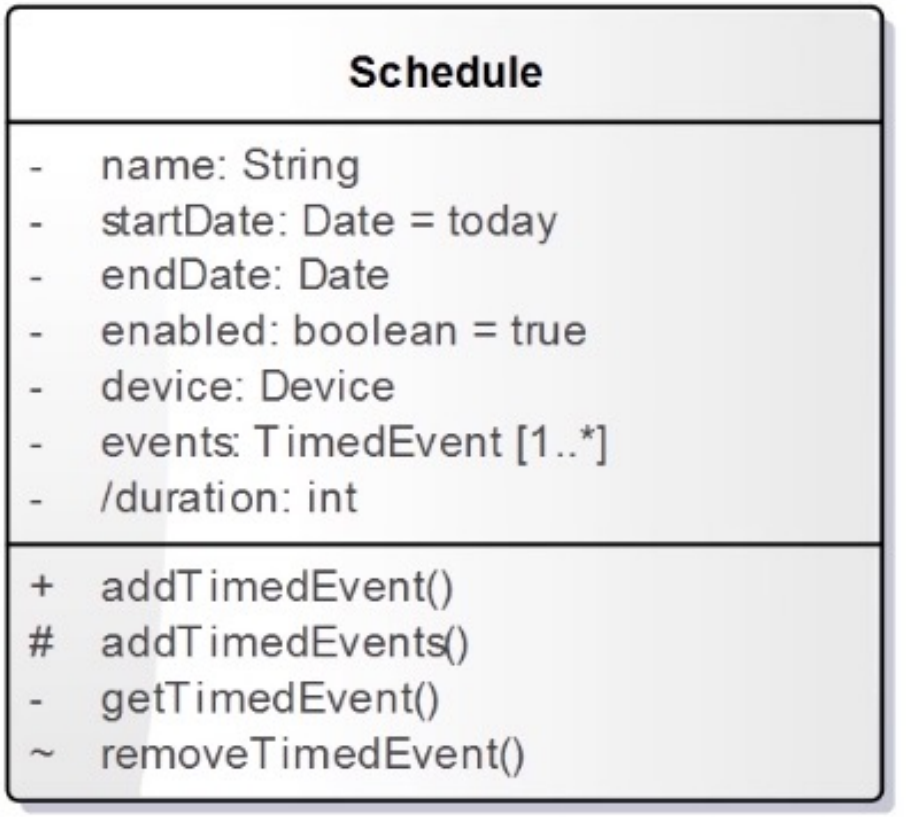
Classes – Atributos derivados

- Atributos em que os valores são derivados de outros;
- Exemplo: a idade de uma pessoa, deriva da sua data de nascimento;
- É apresentado com uma barra antes do atributos;



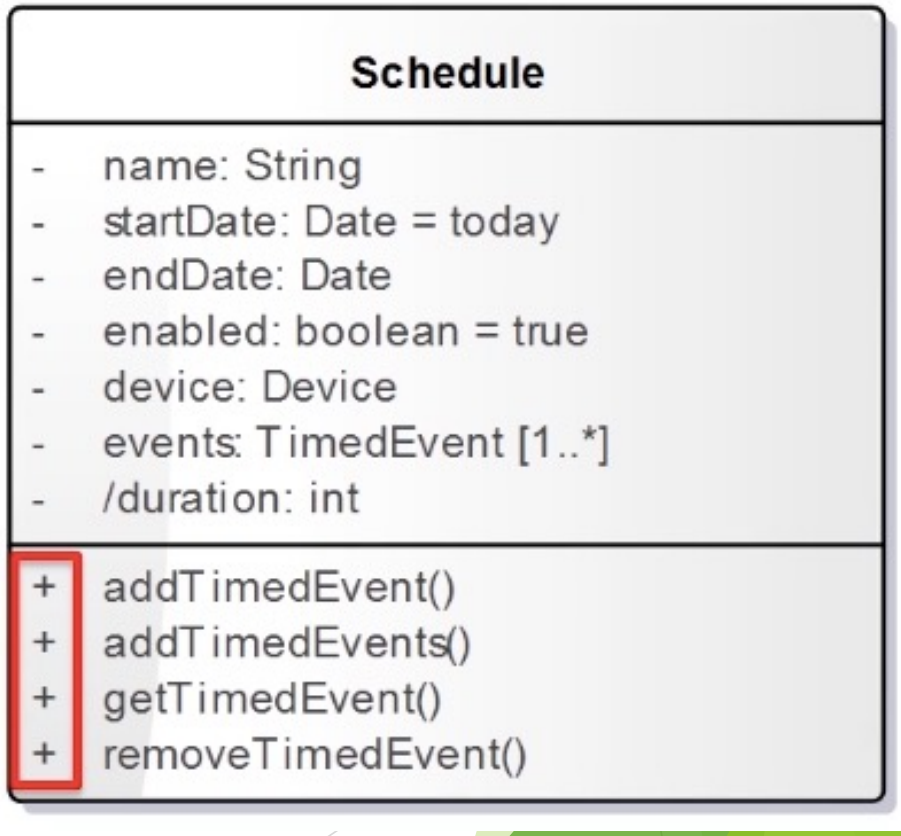
Classes - Operações

- A operações são apresentadas na segunda caixa;
- Os nomes são definidos no formato “camel case”;



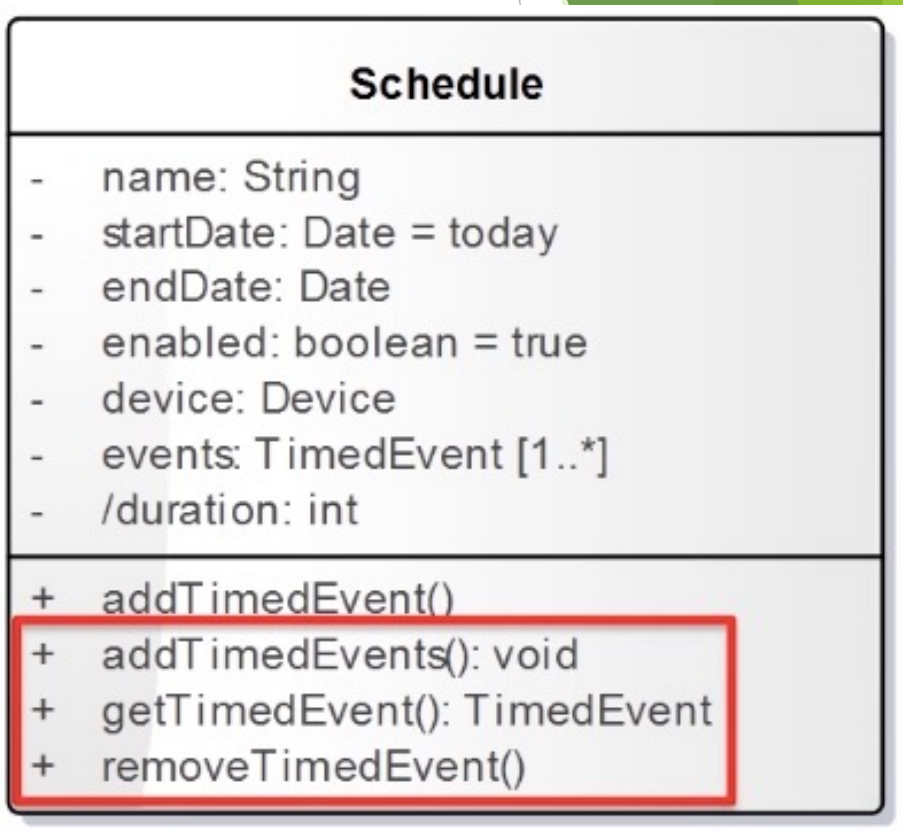
Classes – Operações - Visibilidade

- Tal como os atributos as operações também tem visibilidade;
- Visibilidade:
  - + public
  - - private
  - # protected
  - ~package
- Normalmente as operações tem a visibilidade como public



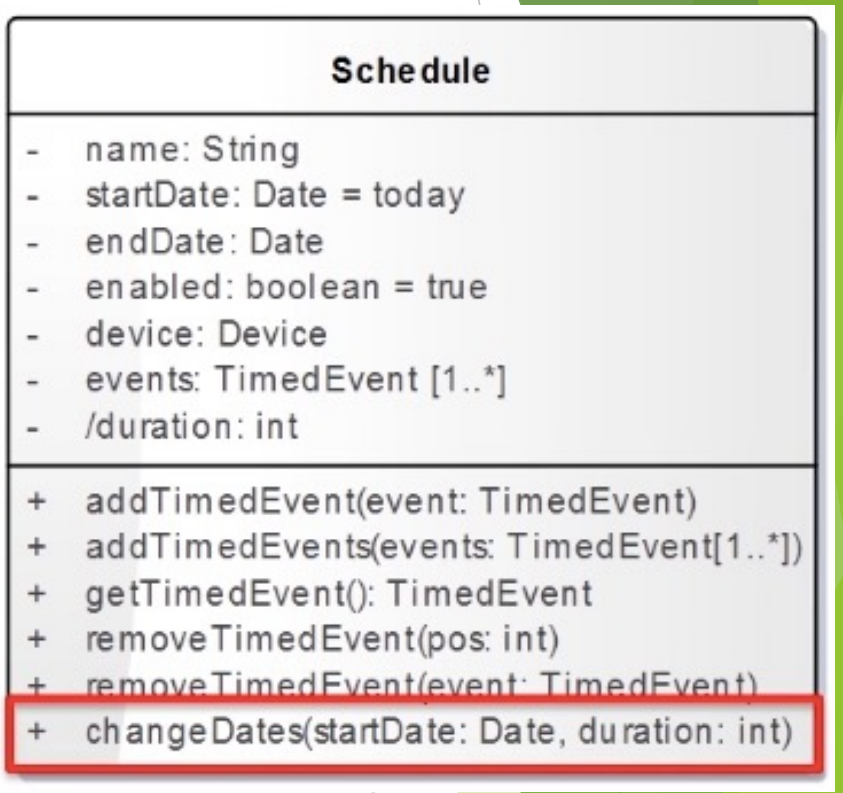
Classes – Operações – Valor de Retorno

- O valor de retorno de uma operação aparece separado por :, onde colocamos o tipo de valor;
- Quando não temos valor de retorno podemos definir como “void” ou deixar em branco;



Classes – Operações - Parâmetros

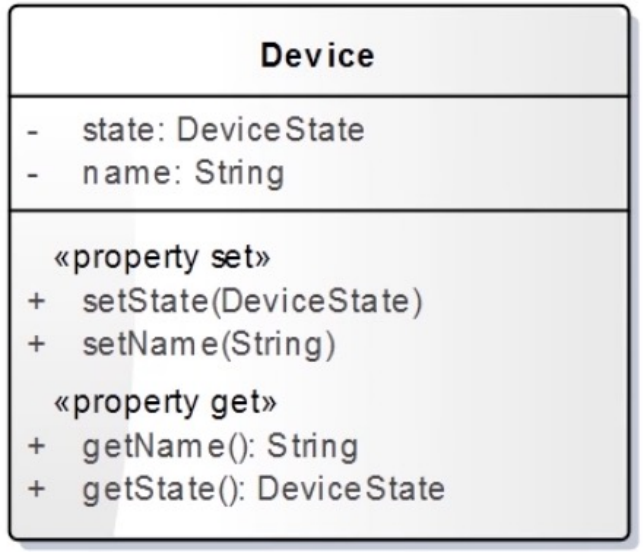
- O nome de cada parâmetro é definido no modelo;
- Nome e tipo estão separados por :
- Mais do que 1 parâmetro estão separados por ,
- É opcional:
  - Apresentar apenas o nome;
  - Apresentar apenas o tipo;
  - Não apresentar nada;



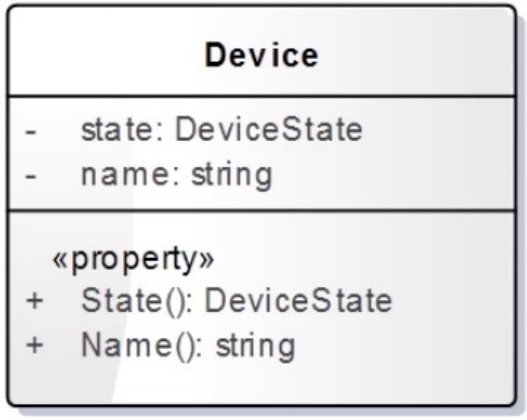
Classes – Getters and Setters

- Operações para definir um valor ou retornar um valor são conhecidas como “Setters and Getters”;
- Para classes muito grandes, estas operações podem ficar por descrever;
- Alguns softwares fazem essa implementação pelo programador

Java



C#



Exercício – Adicionar atributos e operações

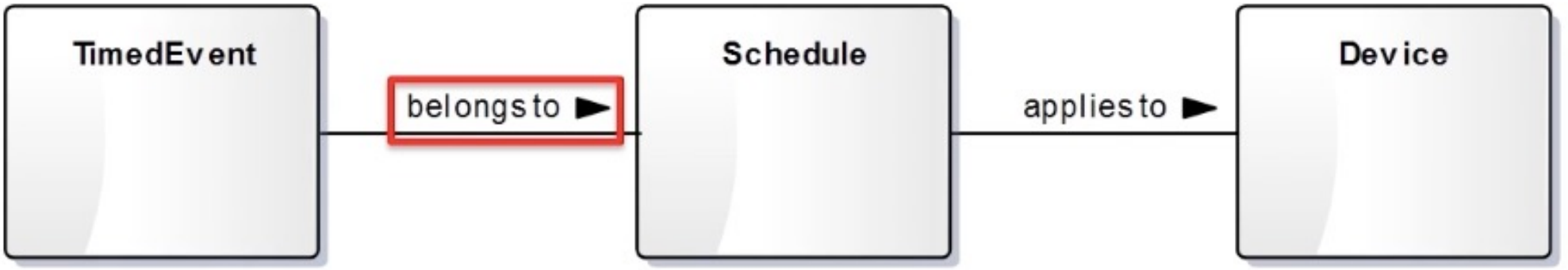
- Entrar no Draw.IO;
- Replicar os processos apresentados nos slides anteriores;

Classes - Associações

- Nome e direção da associação
- Direção de navegação
- Multiplicidade das associações
- Papeis nas associações

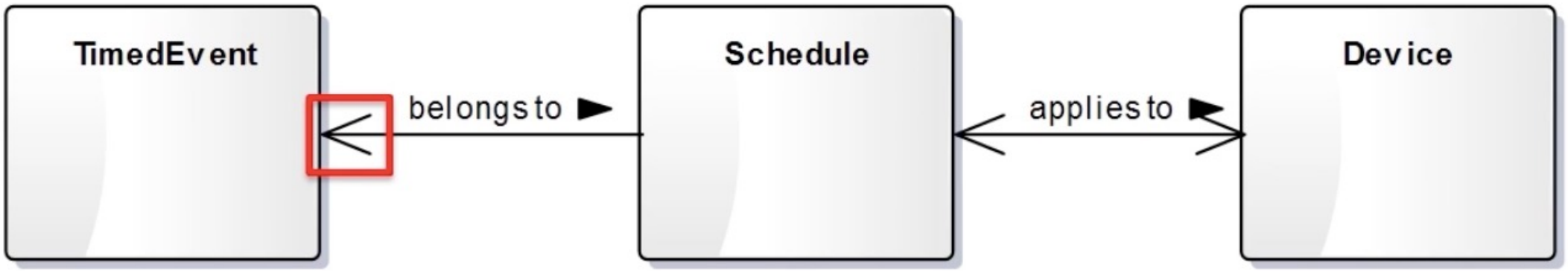
Classes – Associações – Nomes e direções

- Nome da associação é apresentado junto da mesma
- A seta mostra a direção da associação



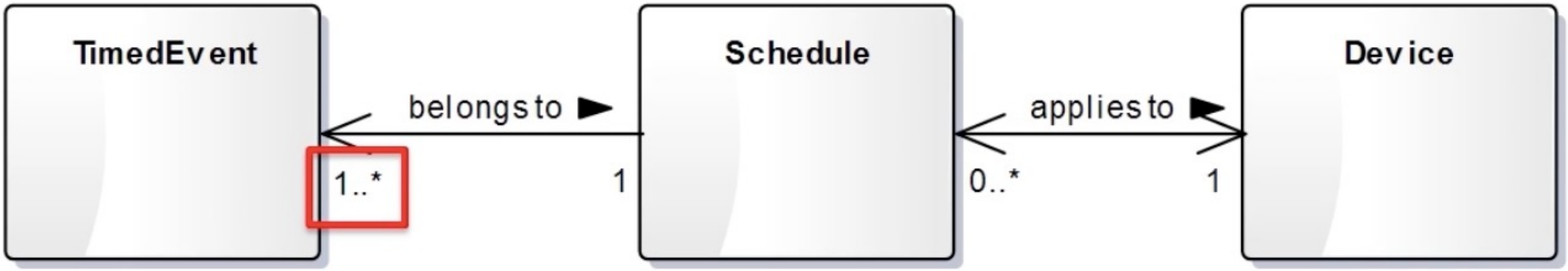
Classes – Associações - Navegabilidade

- A seta indica se uma associação é navegável a partir da classe da outra ponta
- Significa que existe uma referência a objetos nesta ponta



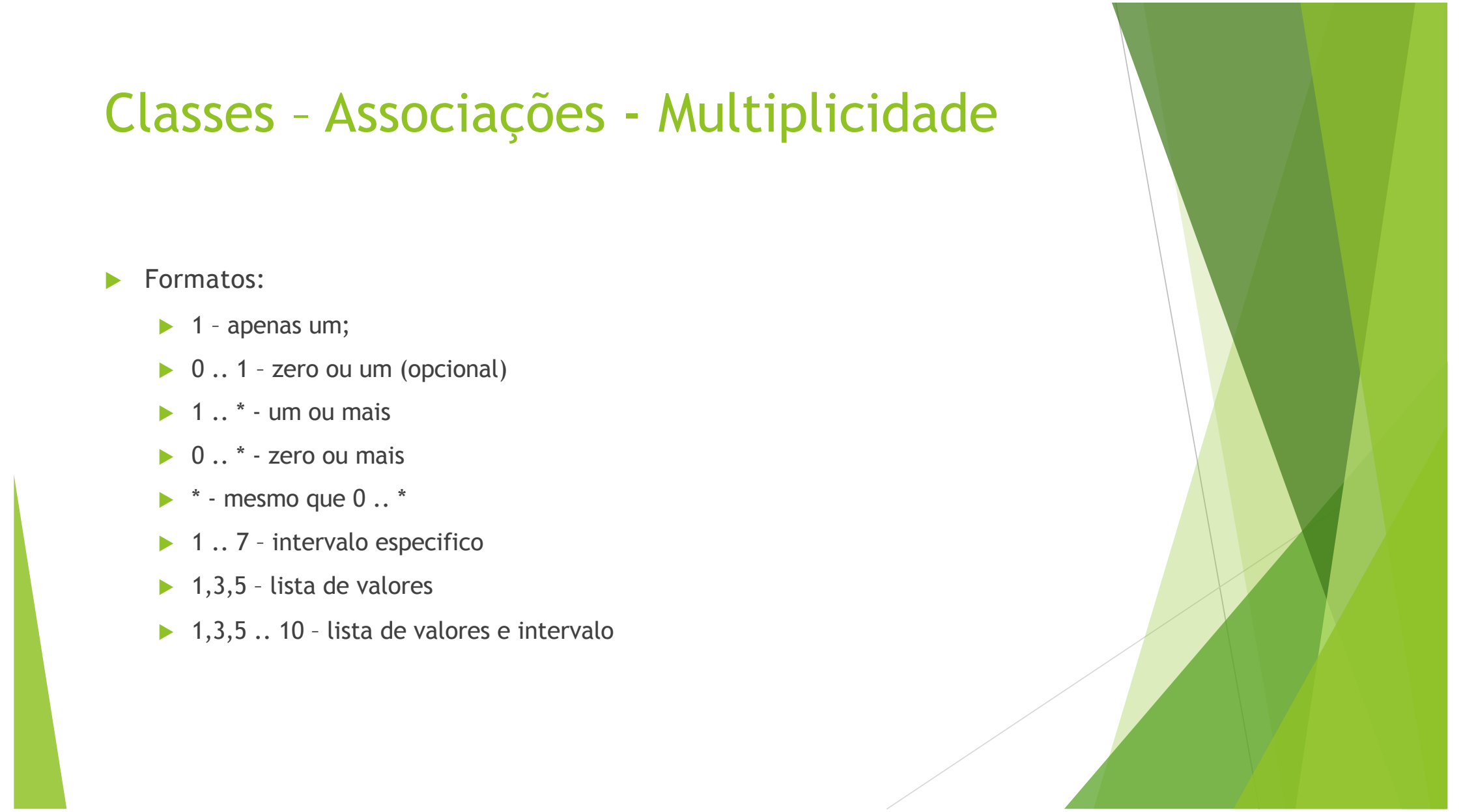
Classes – Associações - Multiplicidade

- Na etiqueta conseguimos perceber a multiplicidade da relação



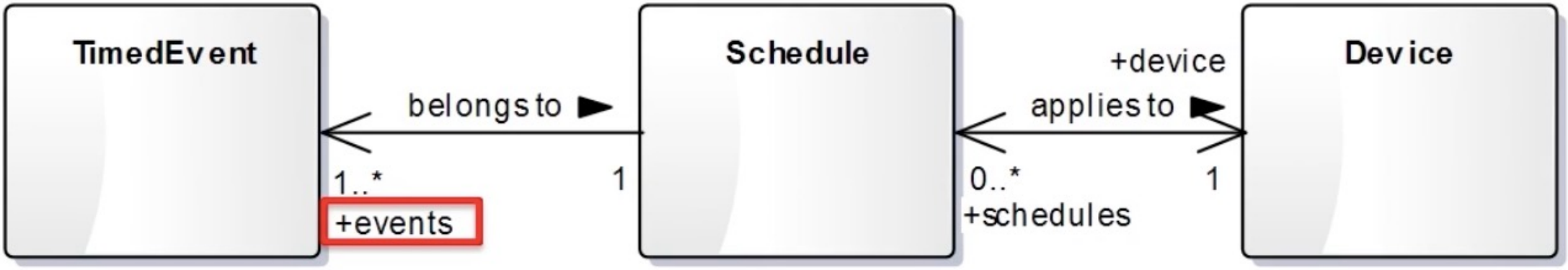
Classes – Associações - Multiplicidade

- Formatos:
  - 1 – apenas um;
  - 0 .. 1 – zero ou um (opcional)
  - 1 .. * - um ou mais
  - 0 .. * - zero ou mais
  - * - mesmo que 0 .. *
  - 1 .. 7 – intervalo especifico
  - 1,3,5 – lista de valores
  - 1,3,5 .. 10 – lista de valores e intervalo



Classes – Associações - Papeis

- No final de cada associação temos o papel;
- Geralmente coincide com o nome da classe no final da relação, mas com o inicio com letra minúscula;
- Tipicamente os nomes são atribuídos no plural;



Exercício

- Entrar no Draw.IO
- Replicar as associações apresentadas nos slides anteriores

Classes – Composição e Agregação

- Relações inteiras
- Composição
- Agregação

Classes – Relações inteiras

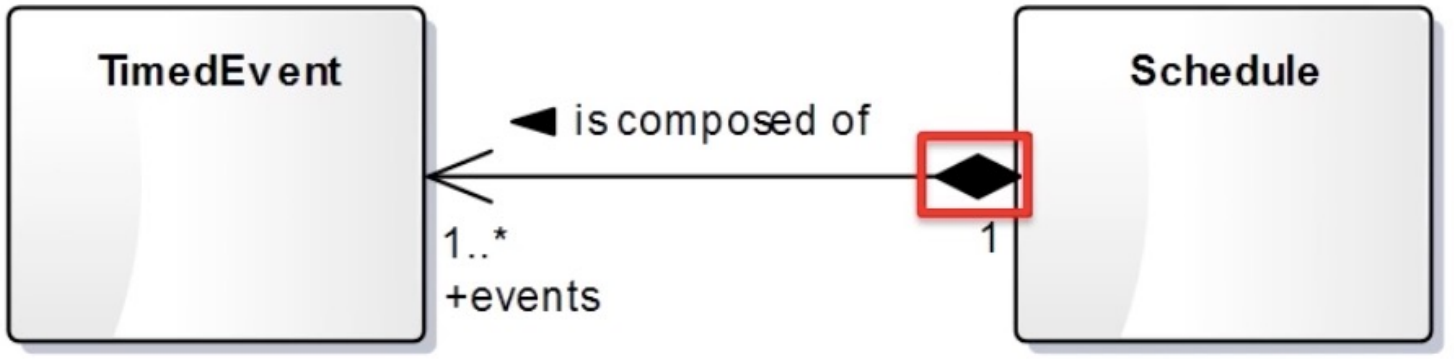
- Em muitos sistemas, as relações entre classes representam a relação entre o todo e as partes, para que o sistema esteja completo.
- Por exemplo:
  - Fatura e FaturaLinha
  - Interface Gráfico, butão, etiqueta, campo, etc.

Classes – Relações inteiras

- Relação forte:
  - Fatura → Fatura Linha: apagar fatura significa apagar as linhas;
- Em outros casos a parte continua a existir:
  - Flight Route → Flight Leg: Mesmo que a rota seja removida, a ligação pode continuar;

Classes - Composição

- Composição é a parte forte de uma relação inteira
- Se uma classe composta por outras for removida, resulta da remoção das restantes;
- É representada pelo diamante com preenchimento;



Classes – Agregação

- Agregação representa a parte fraca de uma relação inteira
- Se uma classe é um agregação de outras, se esta for removida, as restantes ficam intactas;
- É representada pelo diamante sem preenchimento;

Exercício

- Entrar no Draw.IO
- Representar as relações efetuadas anteriormente;

Classes – Generalização e Especialização

- Herança em sistemas orientados por objetos;
- Notação para generalização e especialização;
- Atributos e operações;

Classes – Herança em Orientação por Objetos

- Um aspeto chave das linguagens orientadas por objetos é a capacidade de definir classes que herdam capacidades de outras classes;

Java – `public class OnOffDevice extends Device`

C# – `class OnOffDevice : Device`

Python – `class OnOffDevice(Device)`

Classes – Herança em Orientação por Objetos

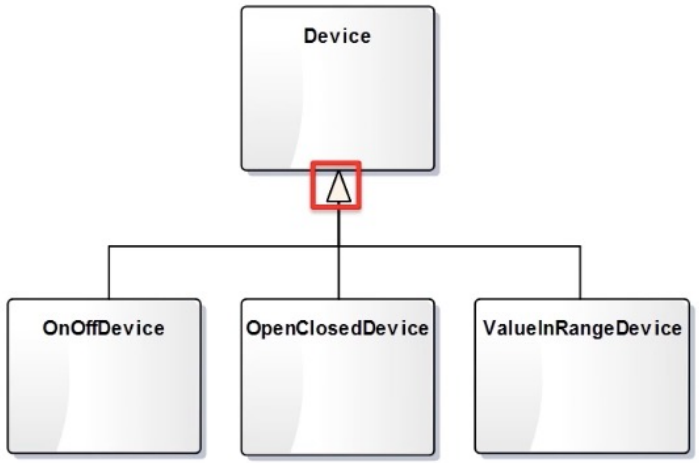
- A classe que herda é conhecida como subclasse ou classe filha;
- A classe que é herdada é conhecida como superclasse ou classe mãe;
- Uma subclasse é um especialização da superclasse;
- Uma superclasse é uma generalização de subclasses;

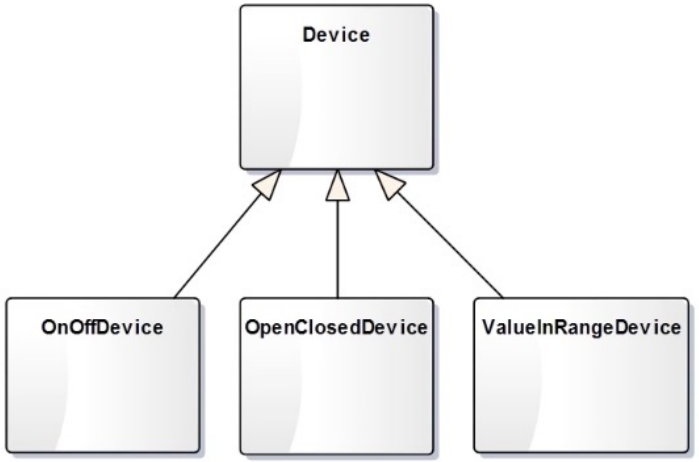
Classes – Herança em Orientação por Objetos

- Uma subclasse herda atributos, operações e relações da superclasse;
- As subclasses acrescentam atributos, operações ou associações às herdadas da superclasse;
- Uma subclasse pode também reescrever definições das operações vindas da superclasse;

Classes – Herança - Notação

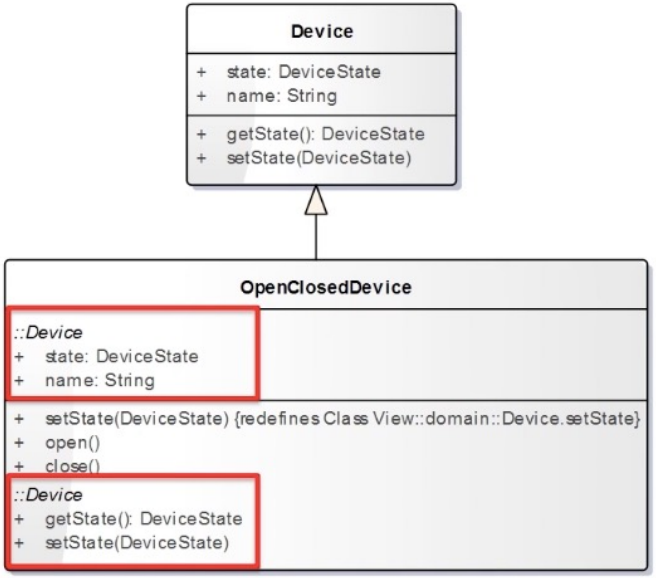
- É definido por um triangulo;
- Pode ser apresentado em árvore ou de forma individual;
- A superclasse fica sempre por cima das subclasses;

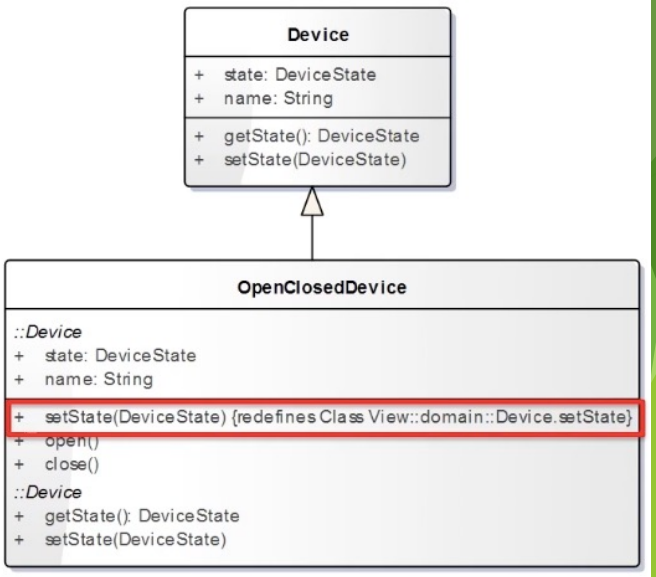




Classes – Herança - Notação

- Herda os atributos e operações;
- Pode adicionar novas operações;
- Pode reescrever operações existentes;





Exercício

- Entrar no Draw.IO;
- Representar o processo de herança dos slides anteriores;

Obrigado !!!