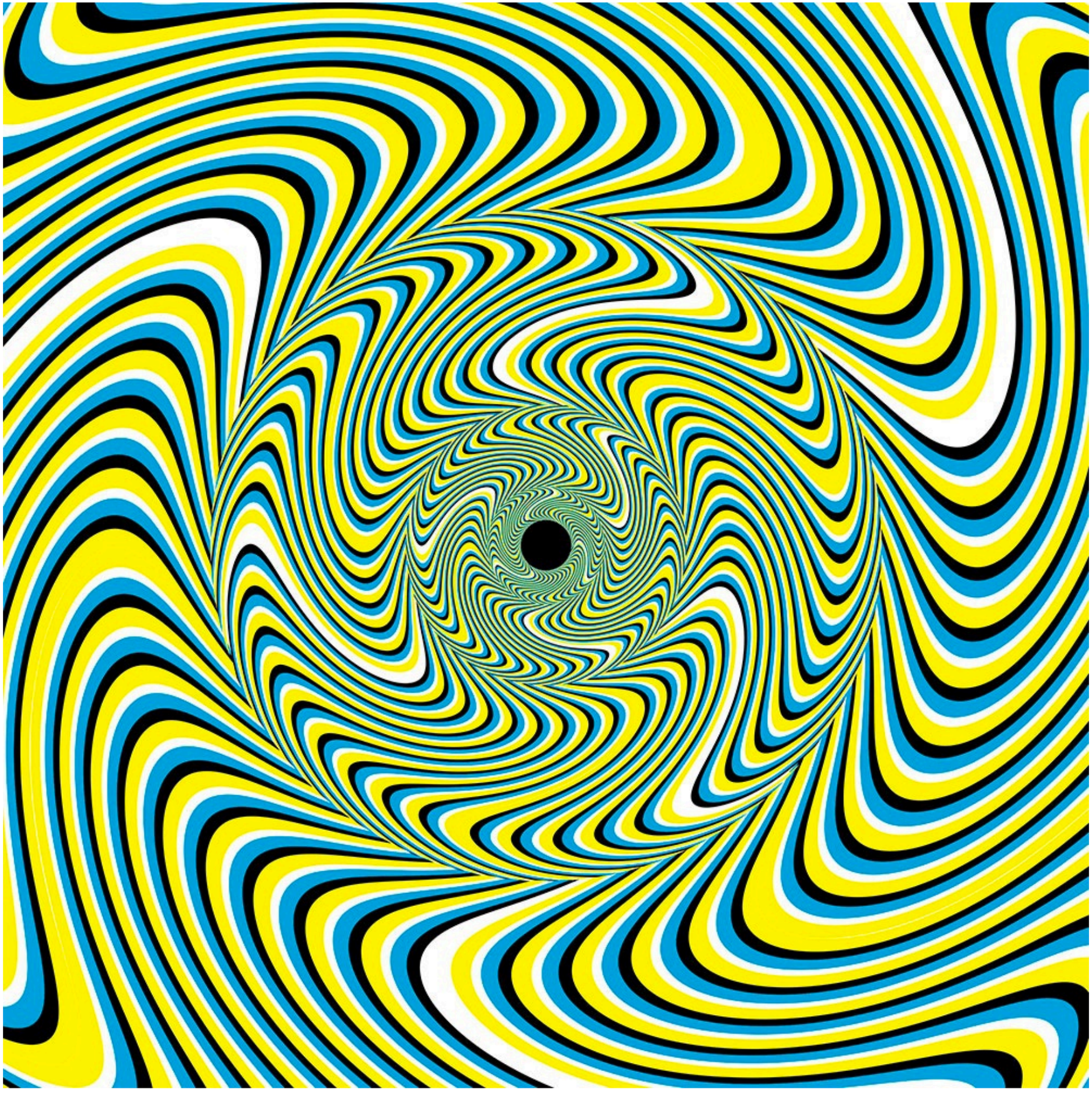


Informação importante

Mudança da data da frequência intermédia

Passa para 14 Novembro (Sábado) às 10h

Mais informações importantes

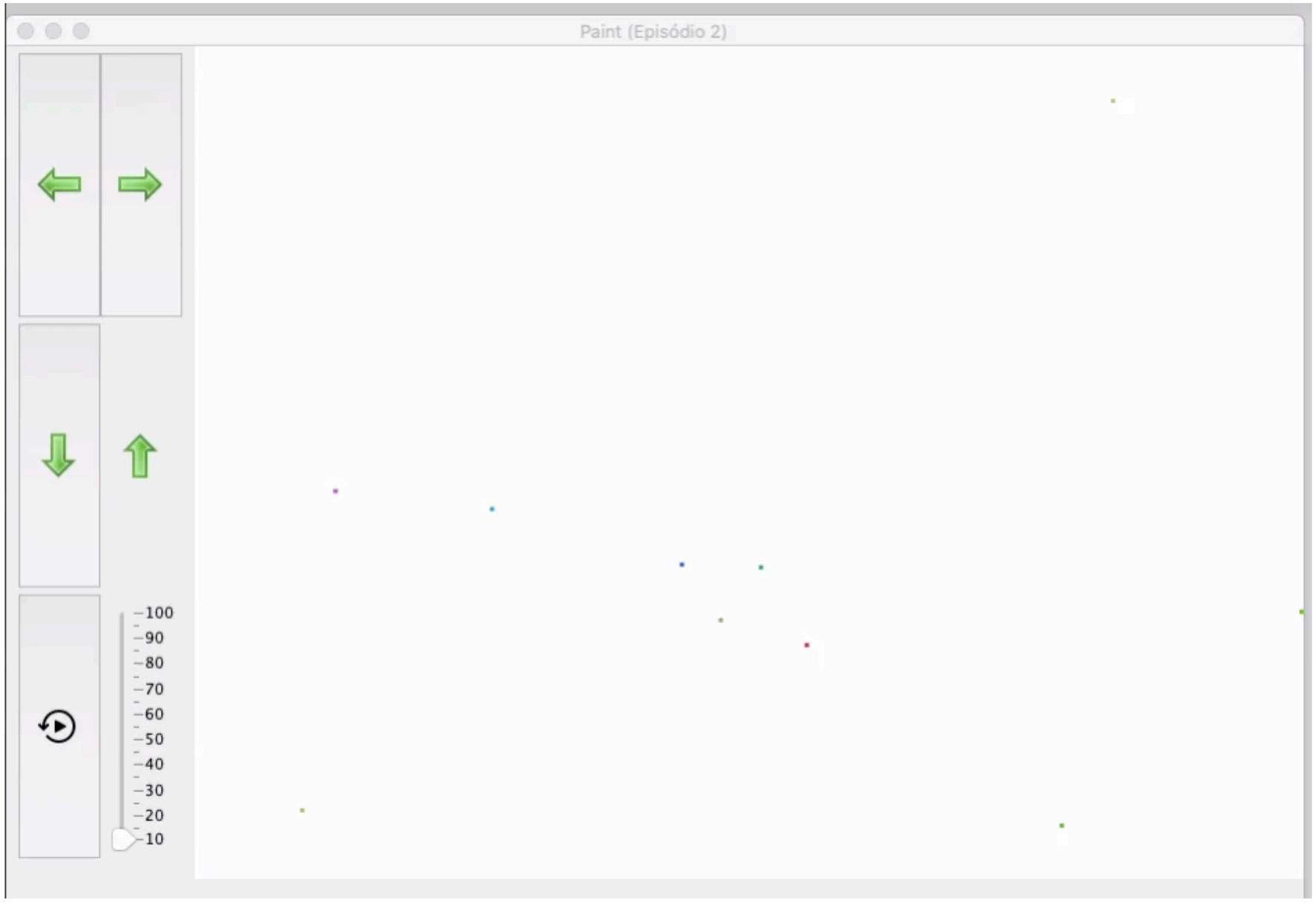
Mini-teste prático 1 na próxima semana

- Durante a aula prática na qual estão inscritos - têm que ir a essa aula
- (Preferencialmente) Levem o vosso portátil com tudo configurado (Java, Intellij)
- Exercício simples de criação de classes: variáveis, métodos, construtores, toString(), arrayList
- Entregue via Drop Project

Paint (episódio 2)

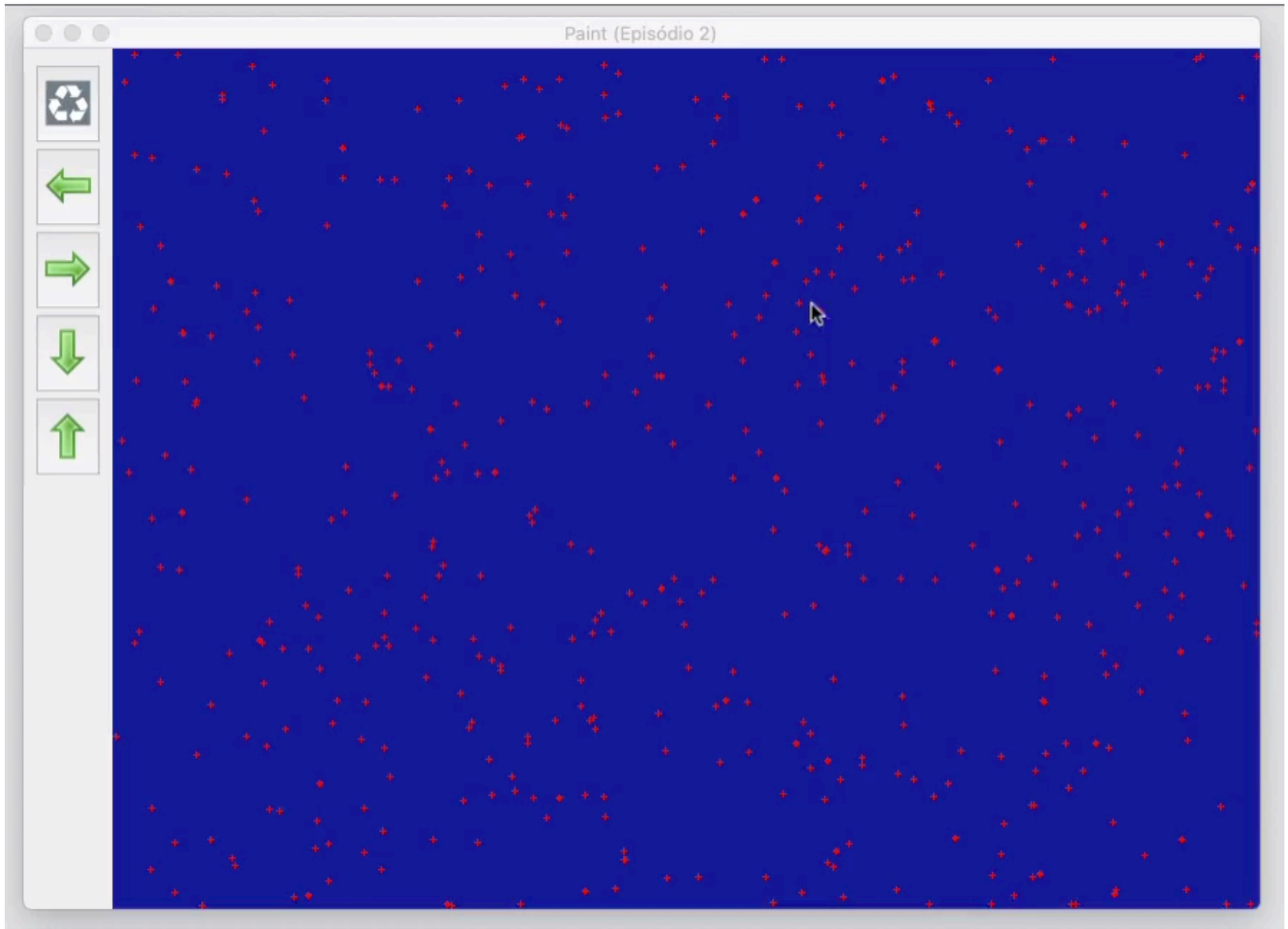
Que coelhos tiraram da cartola?

Hall of fame



Telmo Costa

Hall of fame



Paulo Pinto

Princípio do encapsulamento

Cliente: Qual a potência deste carro?

Vendedor: Pergunta ao carro!

Cada objecto sabe o seu estado interno mas não sabe o dos outros objectos

Princípio do encapsulamento

```
class ContaBancaria {
    int saldo;
}

public class Applicacao {
    public static void main(String[] args) {

        ContaBancaria conta = new ContaBancaria();

        // depositar 100 euros
        conta.saldo += 100;

        // retirar 50 euros
        conta.saldo -= 50;
    }
}
```

A classe Aplicacao sabe demasiado sobre a classe ContaBancaria!!!

Princípio do encapsulamento

```java
class ContaBancaria {
    int saldo;
}

public class Applicacao {
    public static void main(String[] args) {

        ContaBancaria conta = new ContaBancaria();

        // depositar 100 euros
        conta.saldo += 100;

        // retirar 50 euros
        conta.saldo -= 50;

        // ERRO!!! NÃO DEVIA SER PERMITIDO LEVANTAR DINHEIRO QUE NÃO SE TEM
        conta.saldo -= 100;
    }
}
```

A classe Aplicacao sabe demasiado sobre a classe ContaBancaria!!!

Princípio do encapsulamento

```java
class ContaBancaria {
    int saldo;

    void deposita(int valor) {
        saldo += valor;
    }

    // retorna false se não houver saldo suficiente para este levantamento
    boolean levanta(int valor) {
        if (valor > saldo) {
            return false;
        } else {
            saldo -= valor;
            return true;
        }
    }
}
```

```java
public class Applicacao {
    public static void main(String[] args) {

        ContaBancaria conta = new ContaBancaria();
        boolean sucesso;

        // depositar 100 euros
        conta.deposita(100);

        // retirar 50 euros
        sucesso = conta.levanta(50);

        // tenta retirar 100 euros mas não consegue
        sucesso = conta.levanta(100);
        if (!sucesso) {
            System.out.println("Não tem dinheiro suficiente para esta operação");
        }
    }
}
```

Princípio do encapsulamento

```java
public class Applicacao {
    public static void main(String[] args) {

        ContaBancaria conta = new ContaBancaria();
        boolean sucesso;

        // depositar 100 euros
        conta.deposita(100);

        // retirar 50 euros
        sucesso = conta.levanta(50);

        // tenta retirar 100 euros mas não consegue
        sucesso = conta.levanta(100);
        if (!sucesso) {
            System.out.println("Não tem dinheiro suficiente para esta operação");
        }
    }
}
```

A classe Aplicacao não sabe nem quer saber que a ContaBancaria tem uma variavel “**saldo**”

Princípio do encapsulamento

```java
class ContaBancaria {
    int saldo;
}

public class Applicacao {
    public static void main(String[] args) {

        ContaBancaria conta = new ContaBancaria();

        // depositar 100 euros
        conta.saldo += 100;
    }
}
```

Outro problema – E se mudar a representação interna do objeto?

Princípio do encapsulamento

Este número não cabe num inteiro!!

```
class ContaBancaria {
    int saldo = 10000000000000000000;    // fortuna do Mark Zuckerberg?
}

public class Applicacao {
    public static void main(String[] args) {

        ContaBancaria conta = new ContaBancaria();

        // depositar 100 euros
        conta.saldo += 100;
    }
}
```

Princípio do encapsulamento

```
class ContaBancaria {
    BigDecimal saldo = new BigDecimal("10000000000000000000");    // fortuna do Mark Zuckerberg?
}

public class Applicacao {
    public static void main(String[] args) {

        ContaBancaria conta = new ContaBancaria();

        // depositar 100 euros
        conta.saldo += 100;
    }
}
```

Passamos a guardar o saldo numa variável do tipo BigDecimal, que consegue guardar qualquer númvero

Resolvemos o problema de guardar a fortuna do Mark mas “estragámos” a classe Aplicacao!

Princípio do encapsulamento

```java
class ContaBancaria {
    BigDecimal saldo = new BigDecimal("100000000000000000000");    // fortuna do Mark Zuckerberg?

    void deposita(int valor) {
        saldo.add(new BigDecimal(valor));
    }
}

public class Applicacao {
    public static void main(String[] args) {

        ContaBancaria conta = new ContaBancaria();

        // depositar 100 euros
        conta.deposita(100);
    }
}
```

A classe Aplicacao não sabe nem quer saber que a ContaBancaria tem uma variavel “**saldo**”

Imperativo vs Orientado a Objectos

Imperativo	Orientado a Objetos
`deposita(conta, 300)`	`conta.deposita(300)`
`soma(3,4)`	`calculadora.soma(3,4)`
`abreJanela()`	`??`
`move(carro, "norte")`	`??`
`enviaMsg(pedro, cristina)`	`??`

Enviar as 3 respostas, através do teams para p4997

Dica: Pensem qual o objeto cujo estado vai mudar com a execução da função?

Imperativo vs Orientado a Objectos

Imperativo	Orientado a Objetos
`deposita(conta, 300)`	`conta.deposita(300)`
`soma(3,4)`	`calculadora.soma(3,4)`
`abreJanela()`	`janela.abre()`
`move(carro, "norte")`	`carro.move("norte")`

`enviaMsg(pedro, cristina)`	`msg.envia (pedro, cristina)`
	ou
	`pedro.enviaPara(msg, cristina)`

Problema da Biblioteca

Pretende-se desenvolver uma aplicação para gerir os livros de uma biblioteca. Os utilizadores poderão consultar os livros existentes na biblioteca (título, ano de publicação, etc.) e requisitar livros para levar para casa caso estejam disponíveis.

Análise

Entidades

- Livro, Biblioteca, Utilizador

Atributos

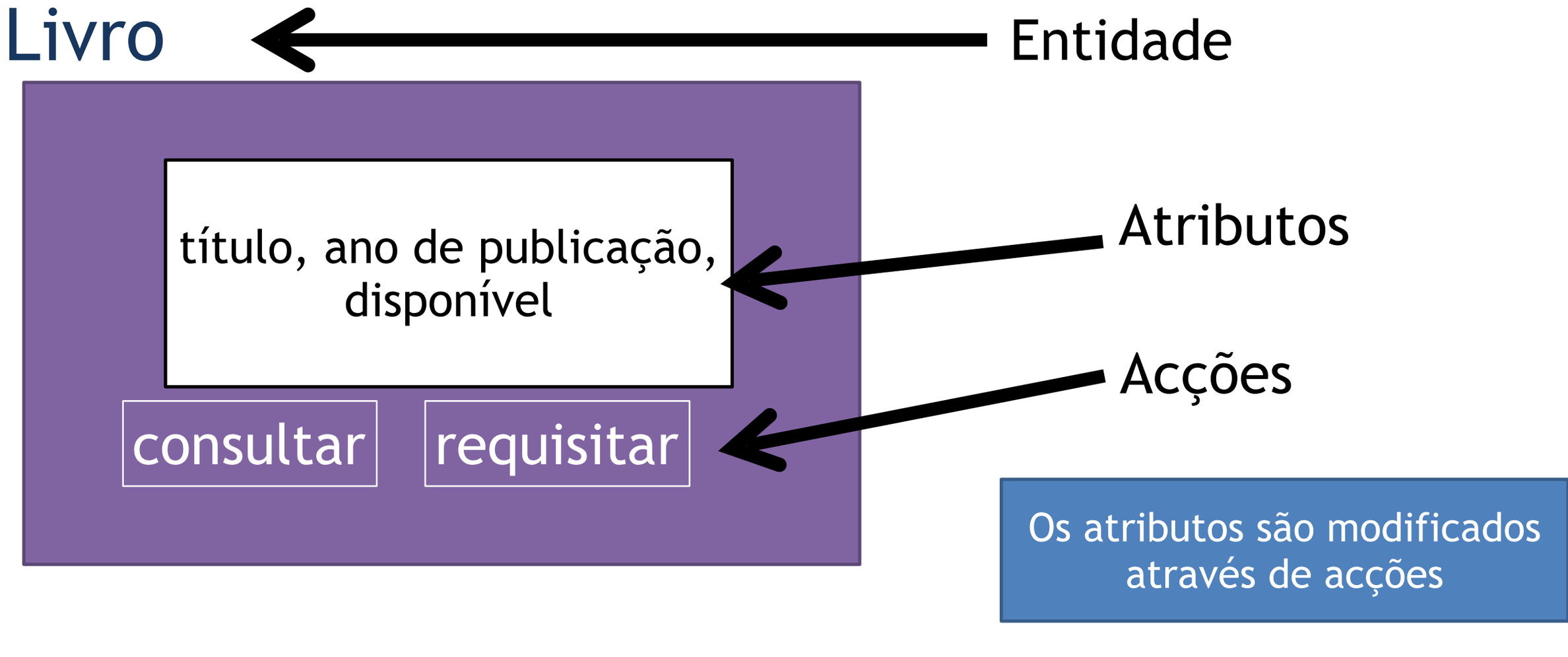
- título (Livro), ano de publicação (Livro), disponível (Livro)

Acções

- consultar (Livro), requisitar (Livro)

Livro

Entidades têm <u>dados/informação</u> (atributos) e <u>comportamento</u> (acções)



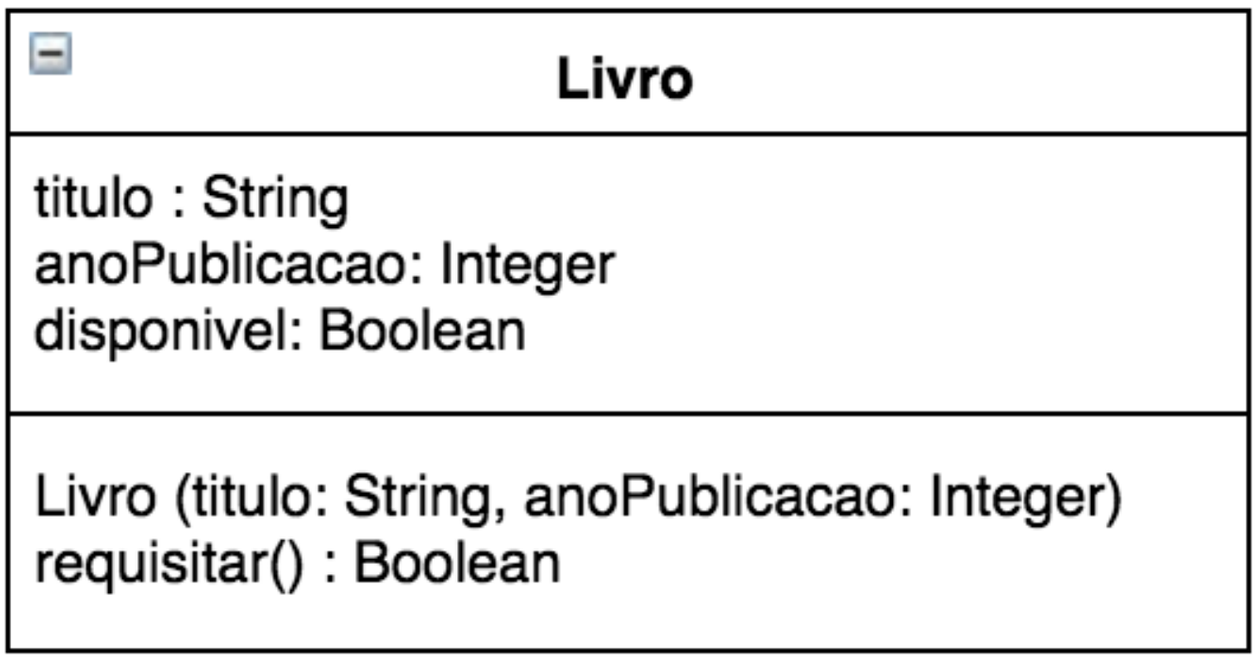
Livro

```
class Livro {

    String titulo;
    Integer anoPublicacao;
    Boolean disponivel;

    // construtores omitidos por simplificação

    Boolean requisitar() {
        if (disponivel) {
            disponivel = false;
            return true;
        } else {
            return false;
        }
    }

}
```

UML

Standard para representar diagramas de classes de forma visual

UML

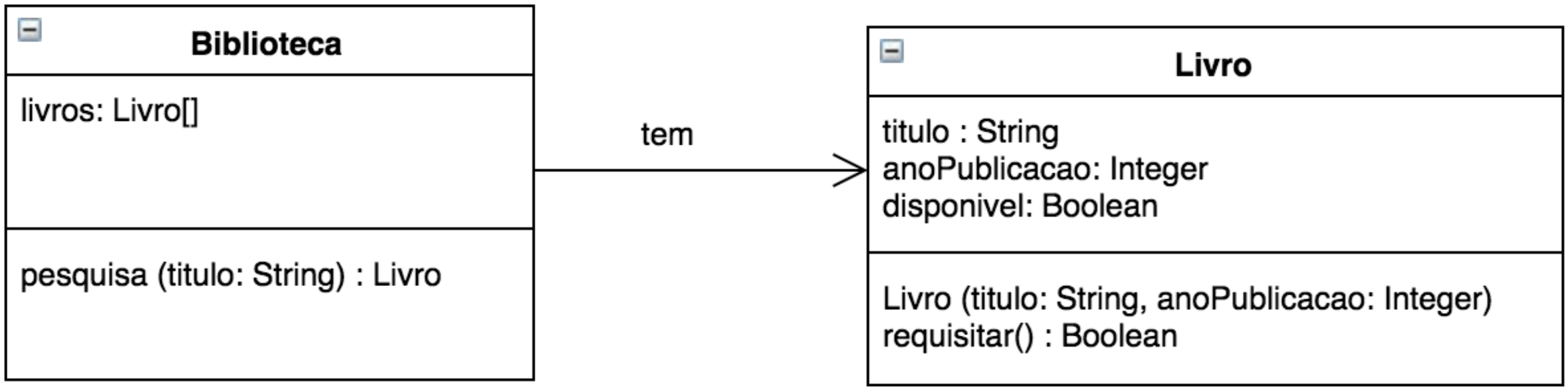


Classe

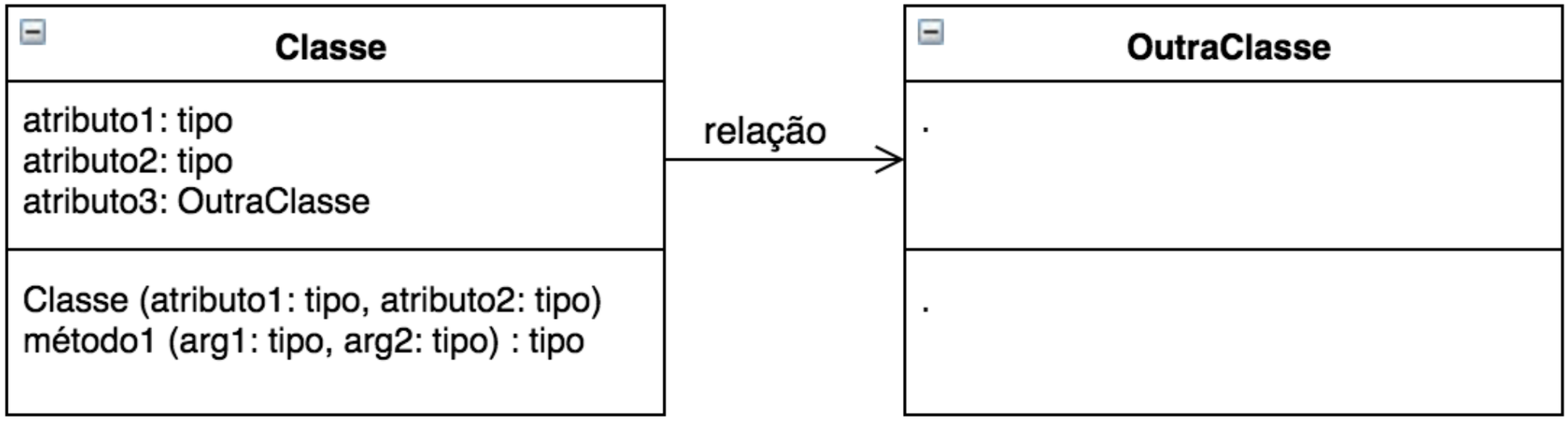
Variáveis

Construtores e funções

UML

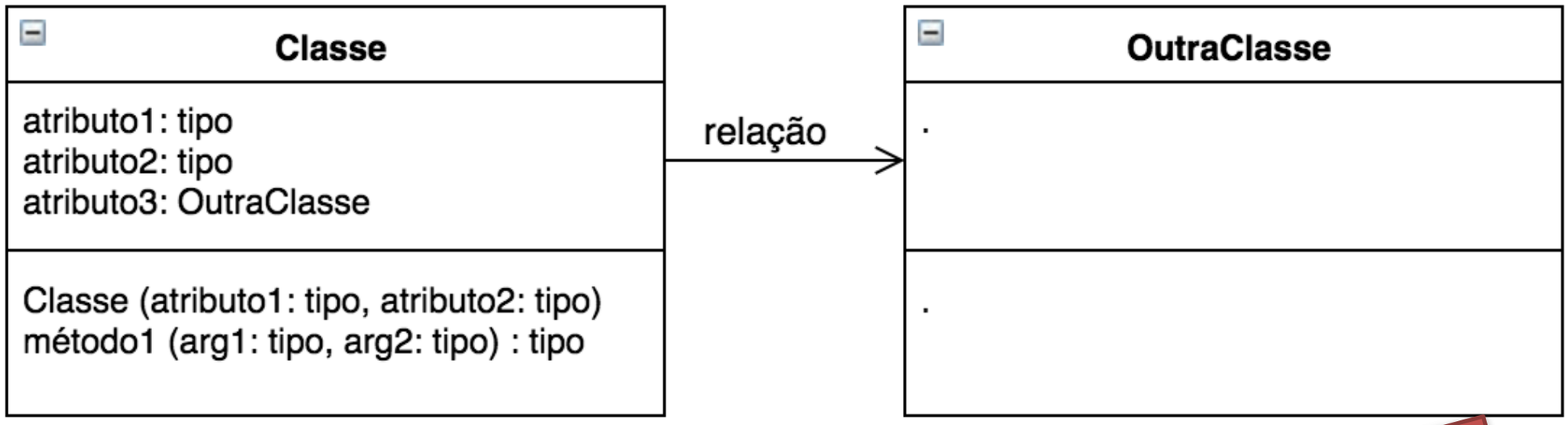


UML



- Cada classe ocupa uma caixa, que descreve os seus atributos e métodos
- Para cada atributo é preciso indicar qual o seu tipo (ex: int, String)
- Para cada método é preciso indicar a sua assinatura - tipo de retorno e argumentos (ex: somar(op1: int, op2: int) : int)
- A seta indica uma relação com outra classe - tem que ser dado um nome à seta indicando qual a relação (ex: “tem”, “está contida em”, “é administrada por”, etc.)
- Por convenção, o nome da classe deve começar sempre por maiúscula
- Por convenção, o nome dos atributos e métodos devem começar por minúscula
- Para nomes compostos deve-se escrever “quantidadeDeBananas” e não “quantidade_de_bananas” (CamelCase)

UML



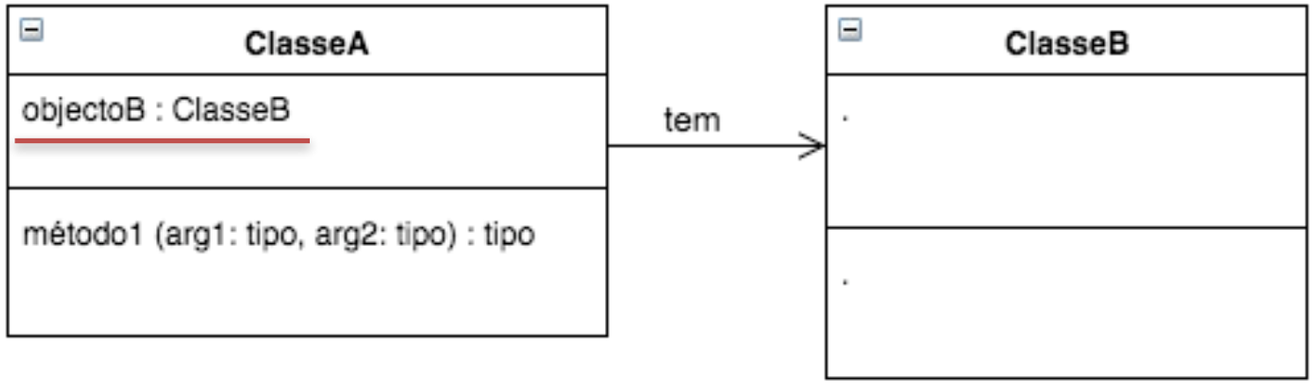
- Cada classe ocupa uma caixa, que descreve os seus atributos
- Para cada atributo é preciso indicar qual o seu tip
- Para cada método é preciso indicar a s
argumentos (ex: somar(op1: int
- A seta indica uma r dado um nome à seta indicando q em", "é administrada por", etc.)
- Por
sse deve começar sempre por maiúscula
- Por c nome dos atributos e métodos devem começar por minúscula
- Para n mes compostos deve-se escrever "quantidadeDeBananas" e não "quantidade_de_bananas" (CamelCase)

Vai ser disponibilizada uma cábula UML no Moodle com estas e outras regras

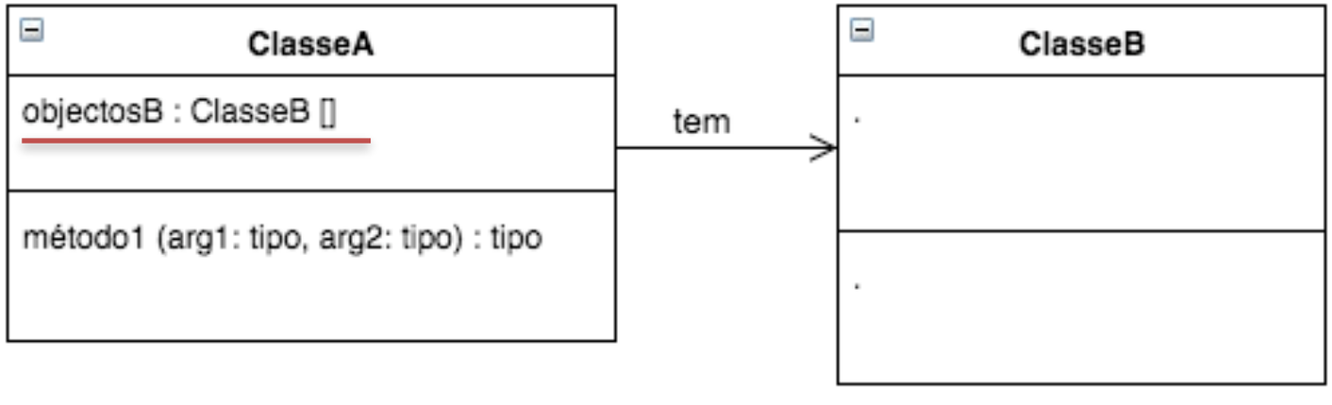
UML - Relações

A ClasseA tem uma relação com a ClasseB porque:

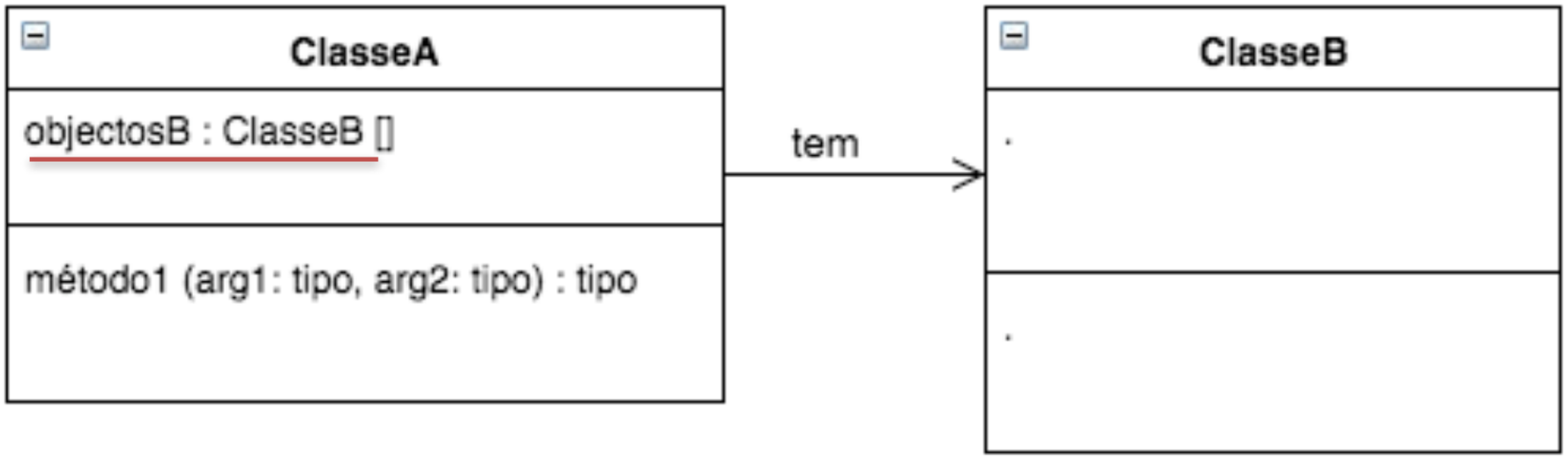
Cada objecto da ClasseA referencia um objecto da ClasseB



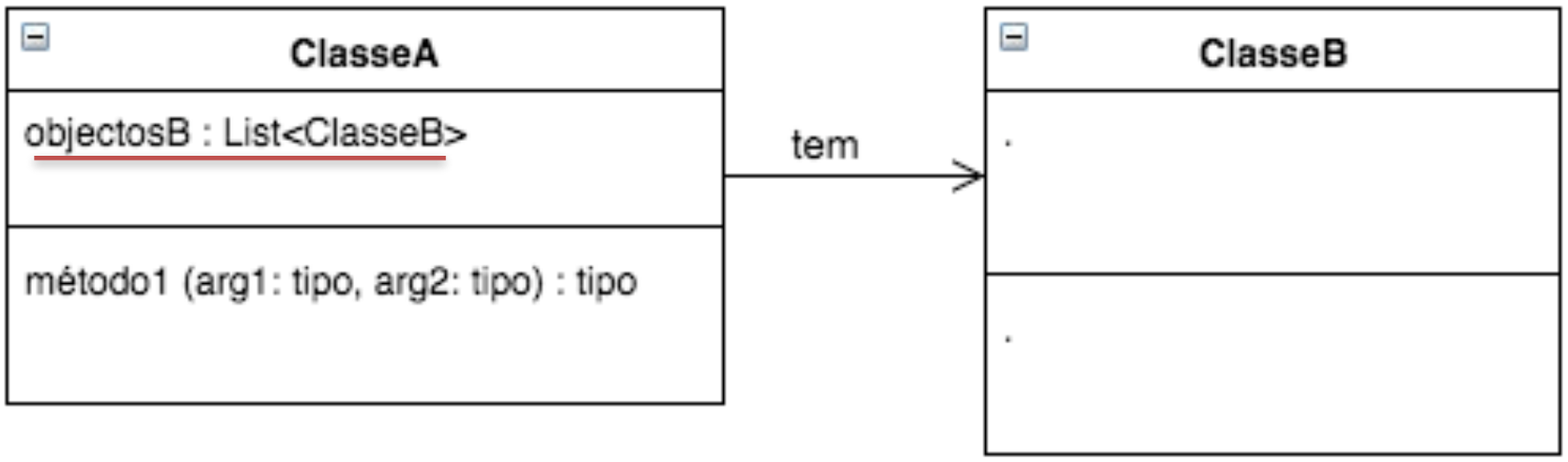
Cada objecto da ClasseA referencia múltiplos objectos da ClasseB



UML - Relações



Em vez de array, pode-se usar o List, no seguinte formato:



Exercício (Breakout Rooms)

Pretende-se desenvolver uma aplicação para gerir os autocarros da cidade de Lisboa. Em particular, a aplicação tem que suportar a entrada e saída de pessoas nas diversas paragens. Tem também que ajudar as pessoas a decidir se devem apanhar um certo autocarro (enquanto esperam na paragem) e validar se podem apanhar um certo autocarro.

Identificar as classes, variáveis e métodos desta aplicação.
Identificar também as relações entre as classes.

Desenhar o diagrama UML respetivo usando draw.io
e enviar-me via teams para p4997

Resolução

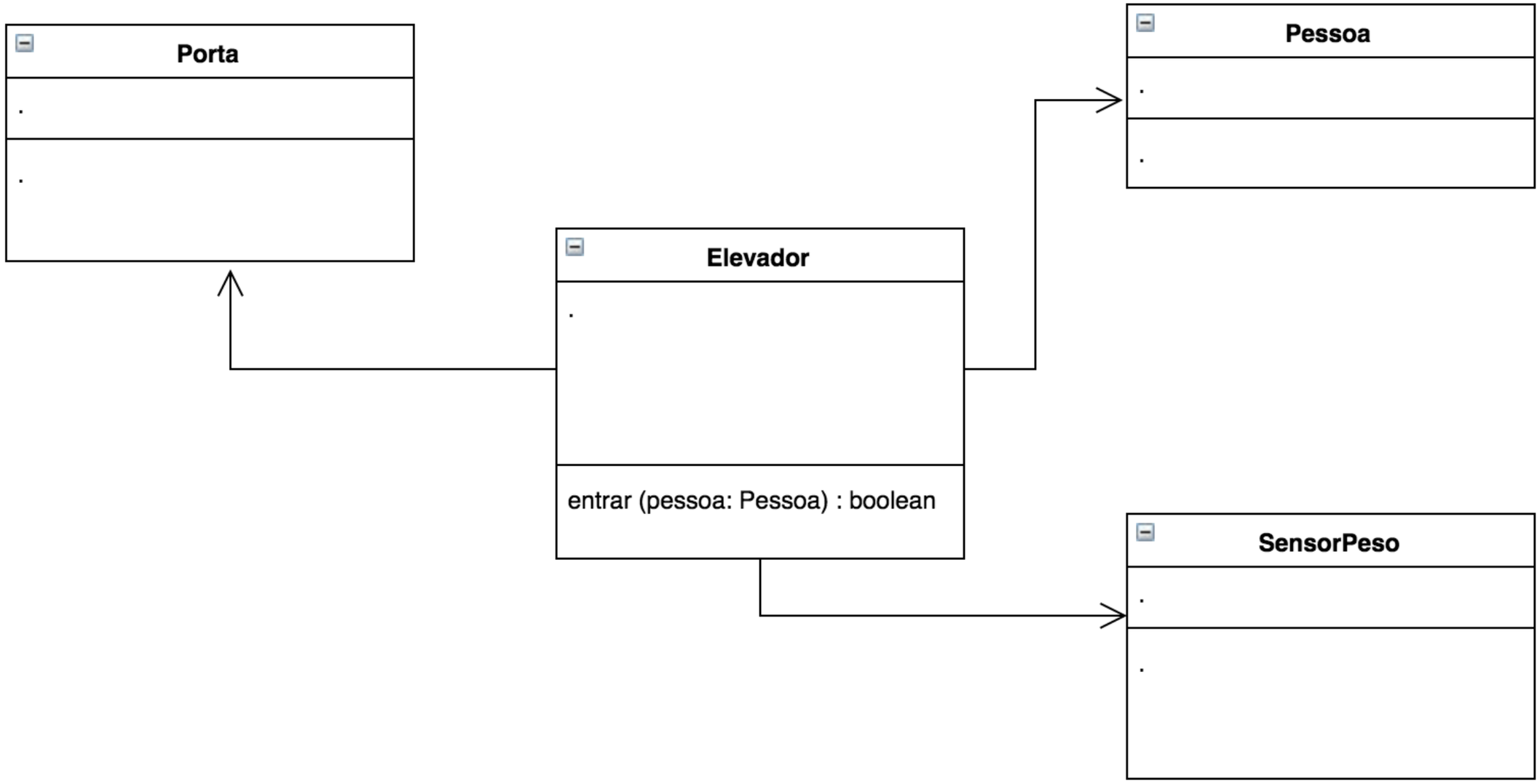
A explicação está em dois vídeos no Moodle

Nota: a classe Bilhete está omitida para simplificar

Guião de implementação do Autocarro

Será publicado no Moodle um guião de exercícios relacionados com o problema do Autocarro que deverão tentar resolver antes da próxima aula prática

TPC teórico 1



Completa o seguinte diagrama de classes (variáveis, métodos e nome das relações) de forma a reflectir o bom funcionamento de um elevador, nomeadamente para permitir validar se uma pessoa pode entrar no elevador.

TPC teórico 1

Regras de submissão

- A submissão deve ser feita no Moodle, no trabalho *TPC Teórico 1 - Diagrama do Elevador*
- A submissão não pode conter elementos identificadores do aluno tais como o número ou nome
- O diagrama tem que ser desenhado à mão
- O diagrama tem que ser legível
- O diagrama tem que respeitar o diagrama incompleto que serve de partida a este problema - não pode incluir novas classes nem alterar as existentes
- O diagrama deve respeitar todas as regras descritas na cábula UML

TPC teórico 1

Regras de avaliação

- Cada trabalho vai ser avaliado por 3 colegas, de forma anónima. Os avaliadores têm um conjunto de critérios objetivos de avaliação, que irão aplicar a cada um desses trabalhos.
- Os professores irão monitorizar as avaliações para evitar injustiças

TPC teórico 1

Nota final

A nota final do TPC 1 é a combinação de duas notas:

- A média das 3 avaliações de colegas (80%)
- A qualidade das avaliações que fizeste ao trabalho dos teus colegas (20%). Isto tem a ver com o grau de concordância entre as várias avaliações

TPC teórico 1

Calendário