

“A única constante é a mudança”

Heráclito de Efeso

...por isso é melhor programar com objectos! 😊

Divulgação

Android Training Program



Formação de programação para Android

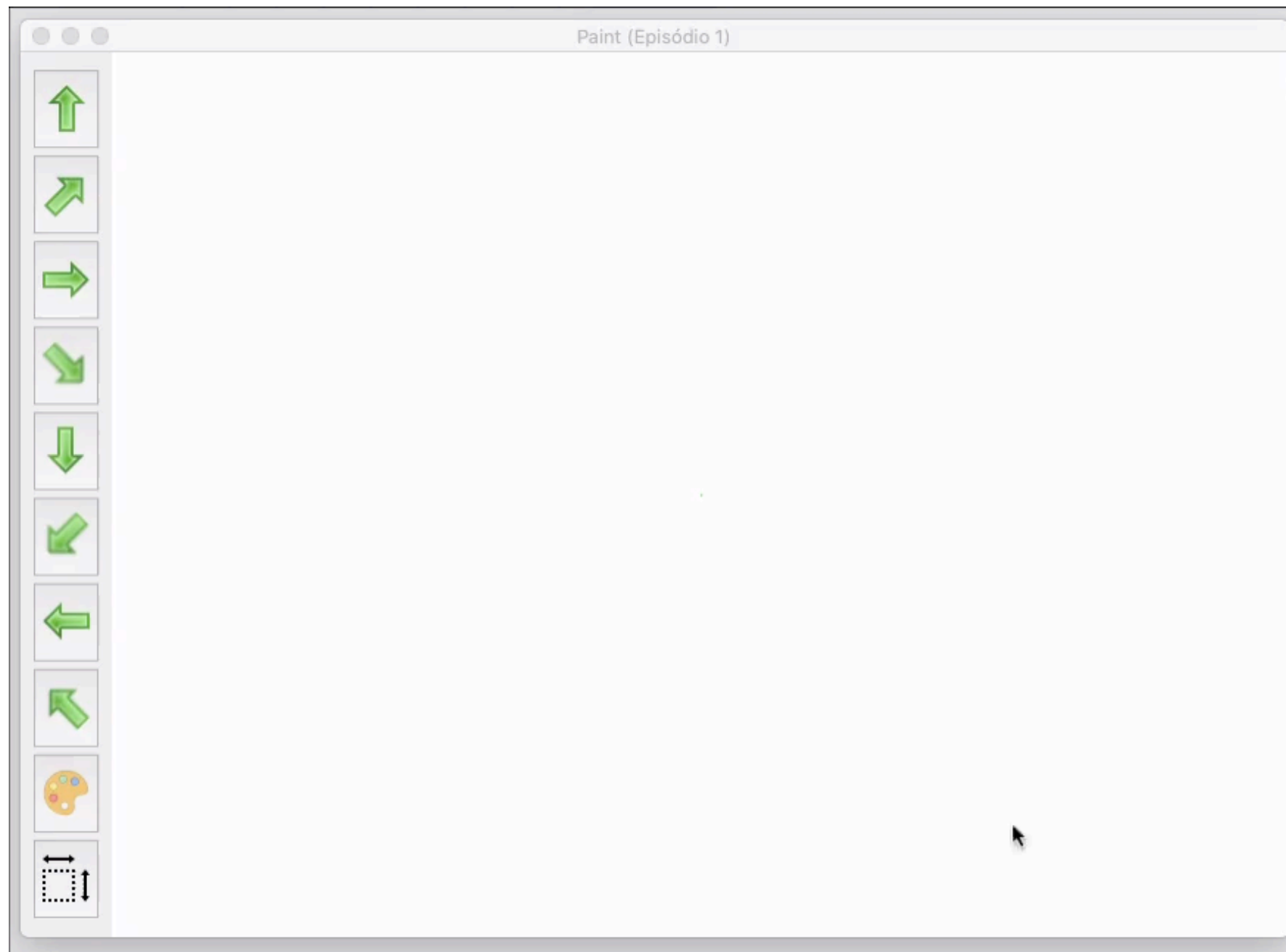
- Começa a 14 de Outubro
- Online e gratuita
- Em parceria com a Google
- Mais informações e inscrição:
<https://events.withgoogle.com/atp2020/>

Paint (episódio 1)

Que coelhos tiraram da cartola?



Hall of fame



Bernardo Amorim

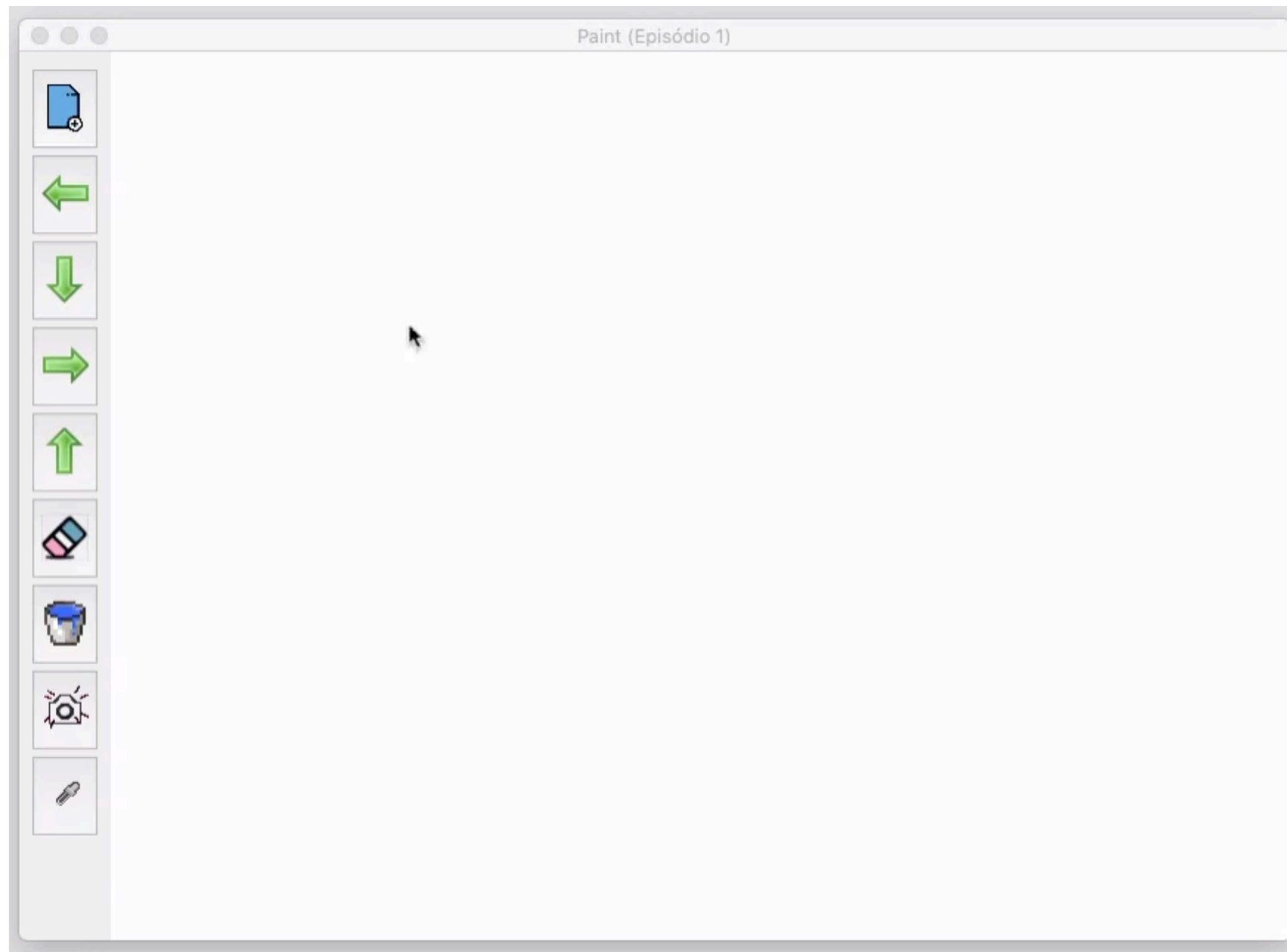
© Pedro Alves 2020

Hall of fame



Tiago Matos

Hall of fame



Paulo Pinto

Debate (breakout rooms)



Ciclo de vida do software

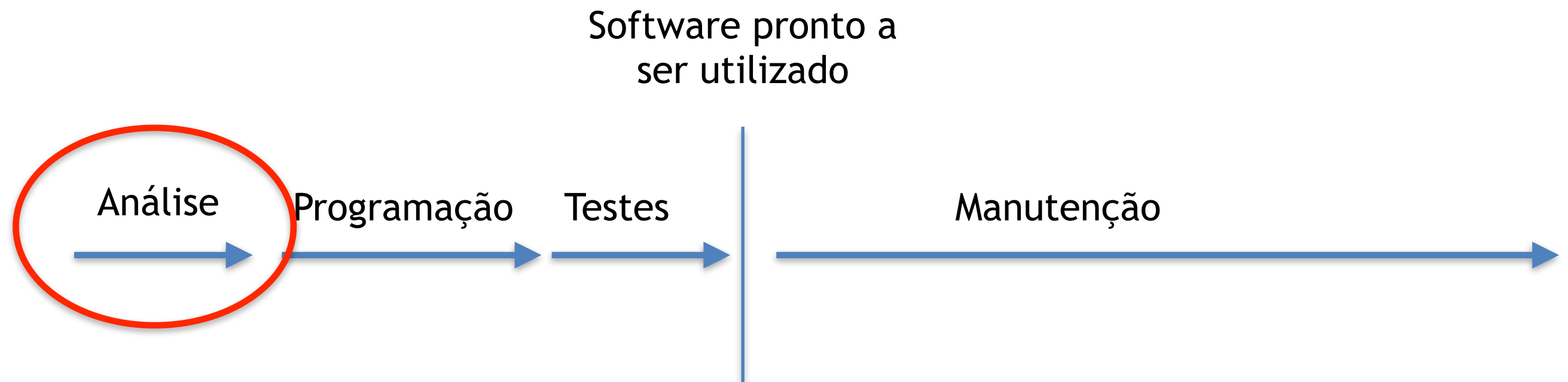
1. O que acontece em cada uma das 4 fases?
2. Como se podem reduzir os custos da fase de manutenção?

Duração: 10 min.

Devem escolher um porta-voz.

Quem já falou em sessões anteriores não pode ser porta-voz.

Análise



Entidades

Na fase de análise identificam-se as **entidades** de uma aplicação

Exemplos

Utilizador

Conta Bancária

Terminal de Pagamentos

Bola

Casa

...

Entidades são sempre nomes!
Nunca são verbos ou adjetivos

Entidades

Problema

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.

Quais as entidades desta aplicação?

Entidades

Problema

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.

Atributos

Os atributos caracterizam uma entidade

Entidade	Atributos
Carro	<ul style="list-style-type: none">• Matrícula• Côr• Número de portas• ???

Atributos

Os atributos caracterizam uma entidade

Entidade	Atributos
Carro	<ul style="list-style-type: none">• Matrícula• Côr• Número de portas• ???
Saco	<ul style="list-style-type: none">• Côr• Capacidade

Atributos

Os atributos caracterizam uma entidade

Entidade	Atributos
Carro	<ul style="list-style-type: none">• Matrícula• Côr• Número de portas• ???
Saco	<ul style="list-style-type: none">• Côr• Capacidade
Conta Bancária	<ul style="list-style-type: none">• Saldo• ????

Atributos

Os atributos caracterizam uma entidade

Entidade	Atributos
Carro	<ul style="list-style-type: none">• Matrícula• Côr• Número de portas• ???
Saco	<ul style="list-style-type: none">• Côr• Capacidade
Conta Bancária	<ul style="list-style-type: none">• Saldo• ????
Bola	<ul style="list-style-type: none">• ????

Atributos são nomes ou adjetivos mas têm que estar associados a uma entidade

Atributos

Problema

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.

Quais os atributos desta aplicação?

Atributos

Problema

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.

Quais os atributos desta aplicação?

Acções

A cada entidade estão associadas acções

Entidade	Acções
Carro	

Acções

A cada entidade estão associadas acções

Entidade	Acções
Carro	<ul style="list-style-type: none">• Ligar• Virar• Acelerar

Acções são verbos

Acções

A cada entidade estão associadas acções

Entidade	Acções
Carro	<ul style="list-style-type: none">• Ligar• Virar• Acelerar
Saco	

Acções são verbos

Acções

A cada entidade estão associadas acções

Entidade	Acções
Carro	<ul style="list-style-type: none">• Ligar• Virar• Acelerar
Saco	<ul style="list-style-type: none">• Introduzir coisas• Retirar coisas• Fechar

Acções são verbos

Acções

A cada entidade estão associadas acções

Entidade	Acções
Carro	<ul style="list-style-type: none">• Ligar• Virar• Acelerar
Saco	<ul style="list-style-type: none">• Introduzir coisas• Retirar coisas• Fechar
Conta Bancária	<ul style="list-style-type: none">• Ver saldo• ????

Acções são verbos

Acções

A cada entidade estão associadas acções

Entidade	Acções
Carro	<ul style="list-style-type: none">• Ligar• Virar• Acelerar
Saco	<ul style="list-style-type: none">• Introduzir coisas• Retirar coisas• Fechar
Conta Bancária	<ul style="list-style-type: none">• Ver saldo• ????
Bola	<ul style="list-style-type: none">• ????

Acções são verbos

Acções

Problema

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.

Quais as acções desta aplicação?

Acções

Problema

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.

Quais as acções desta aplicação?

Resumo

Entidades

Biblioteca

Livro

Utilizador

Atributos

título

ano de publicação

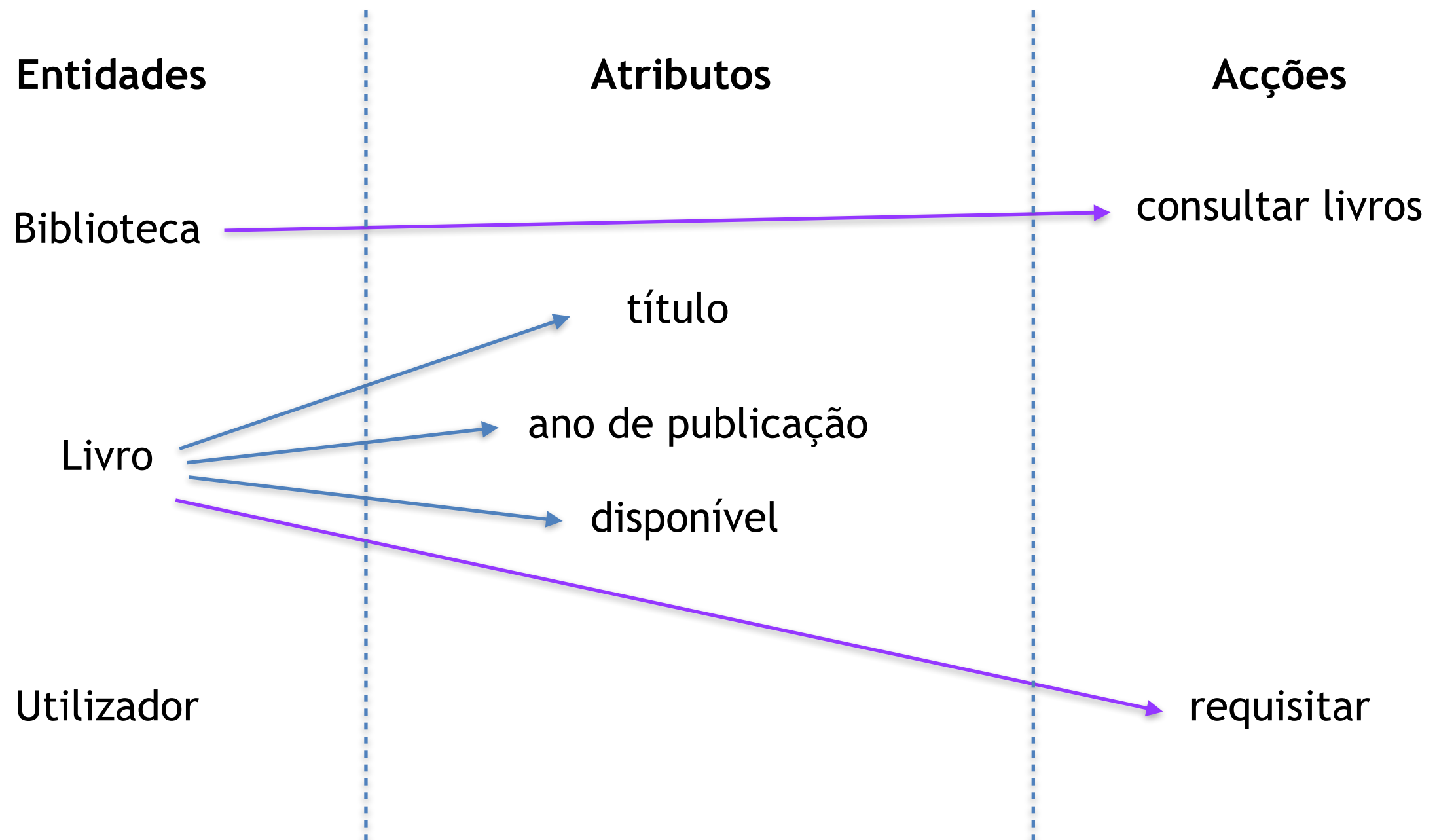
disponível

Acções

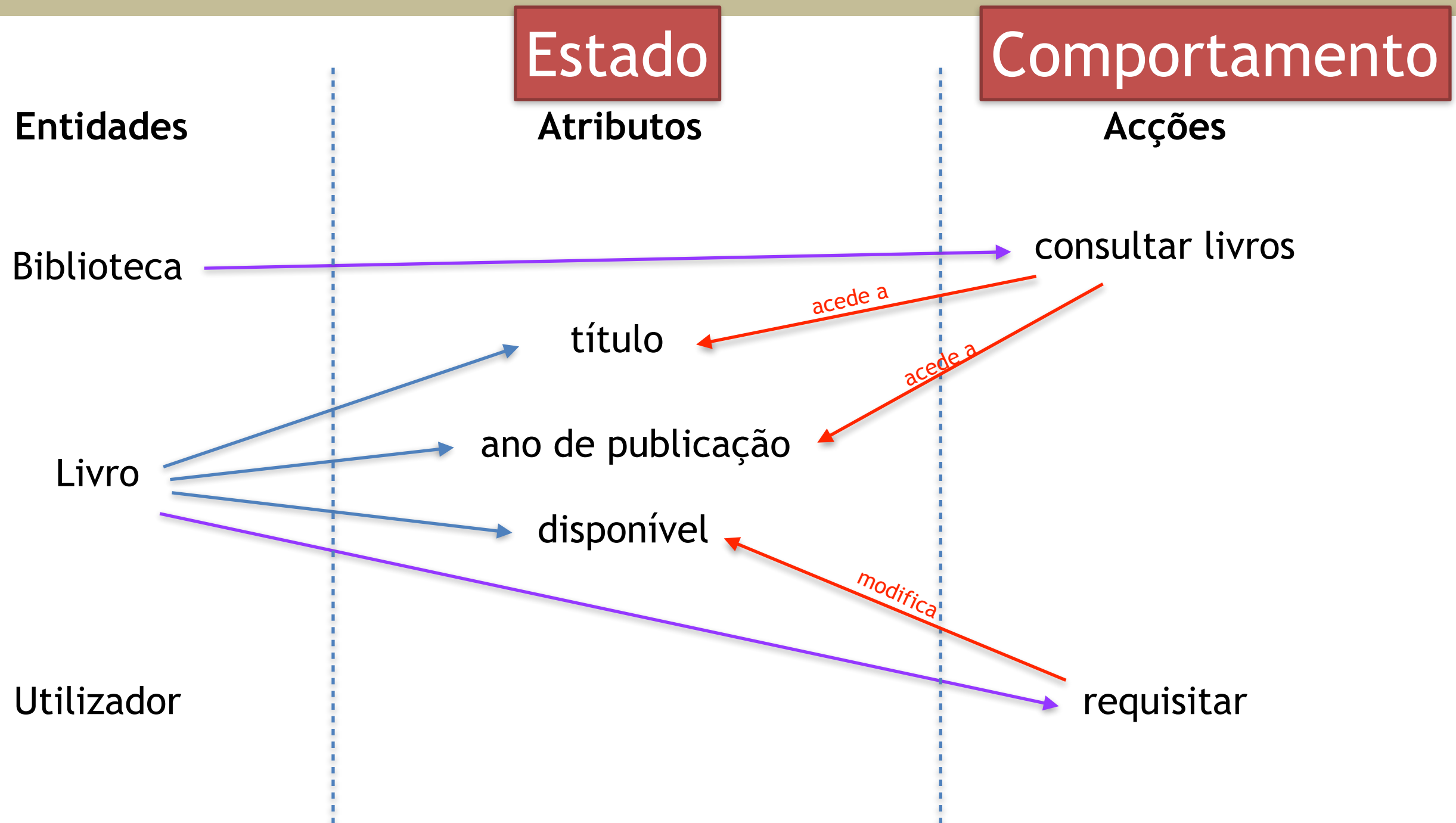
consultar livros

requisitar

Resumo



Resumo

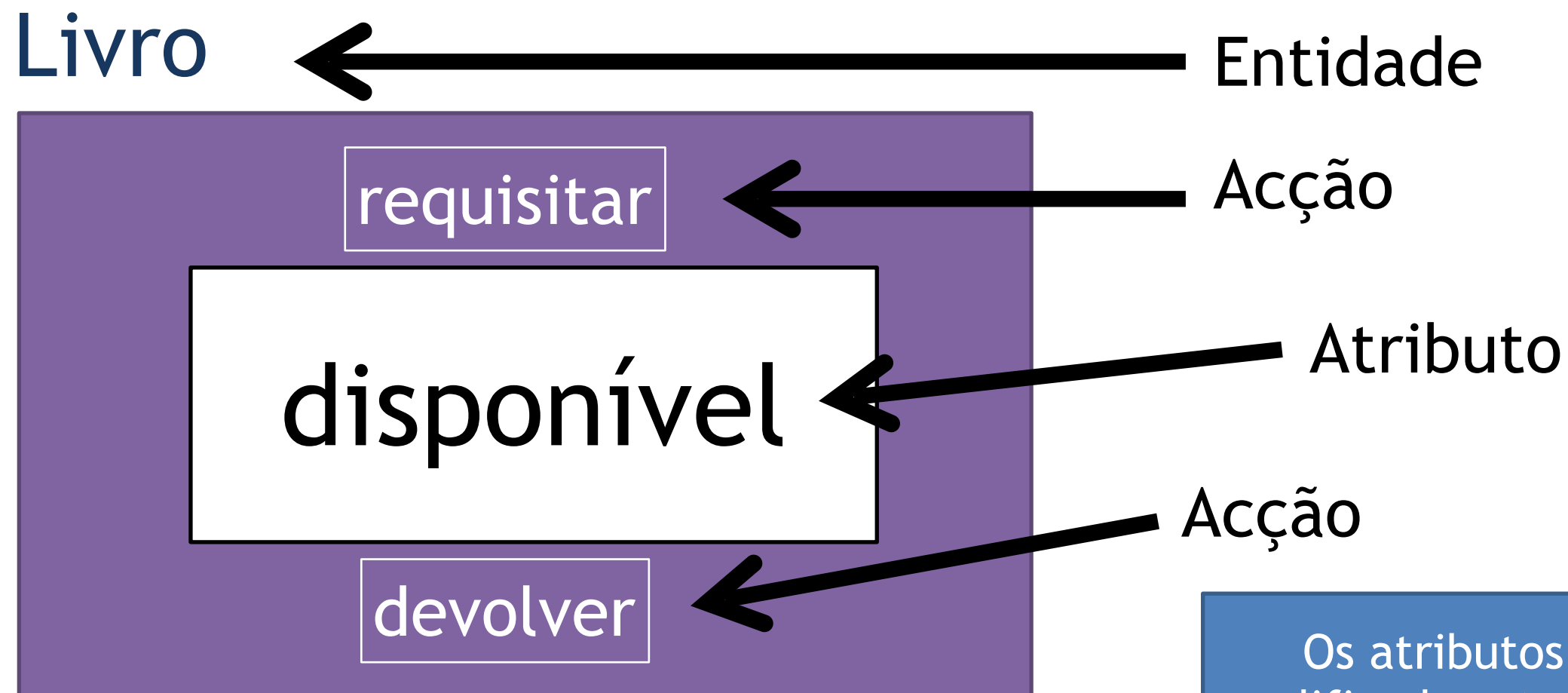


Entidades

Entidades têm dados/estado (atributos) e comportamento (acções)

Entidades

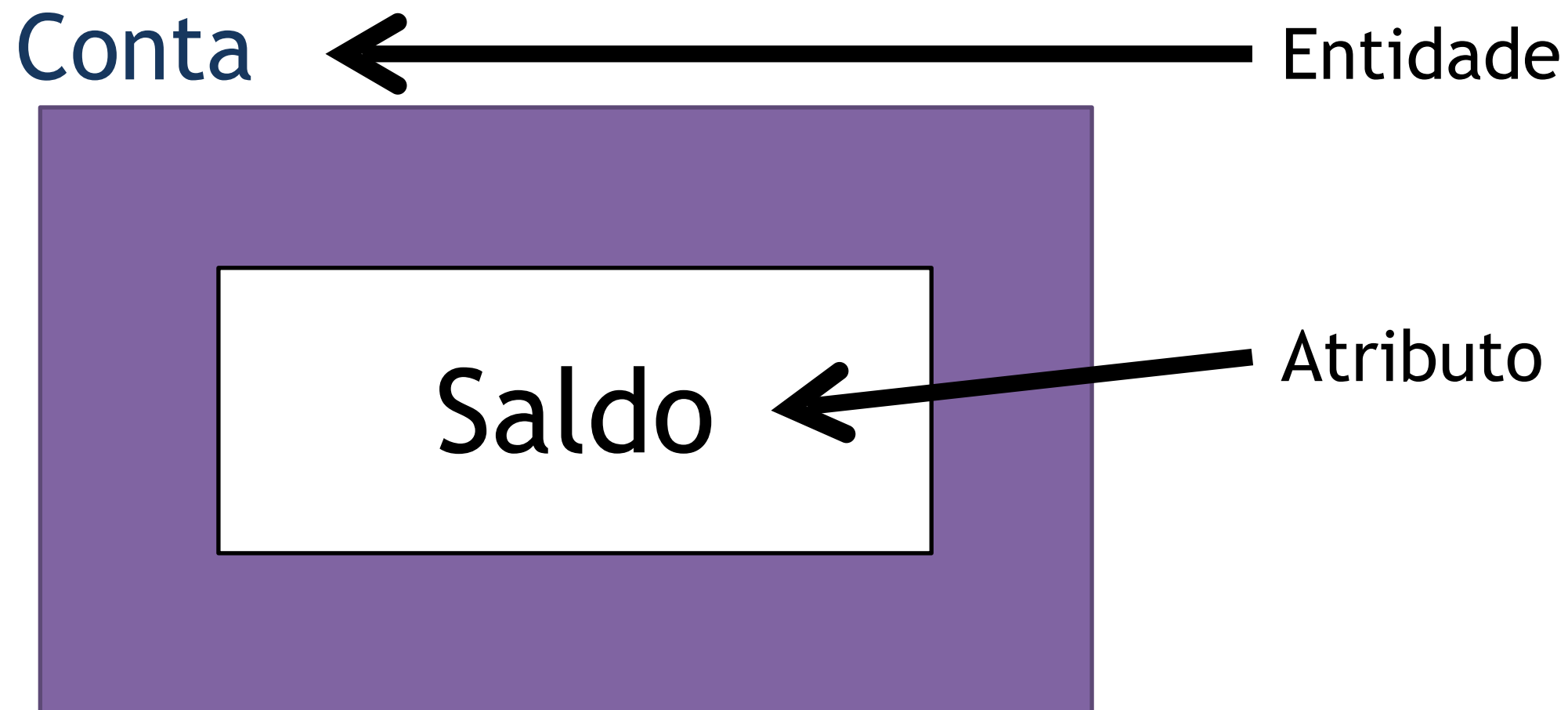
Entidades têm dados/estado (atributos) e comportamento (acções)



Os atributos são lidos/
modificados através de acções

Entidades

Entidades têm dados/estado (atributos) e comportamento (acções)



Entidades

Entidades têm dados/estado (atributos) e comportamento (acções)

Conta ← Entidade

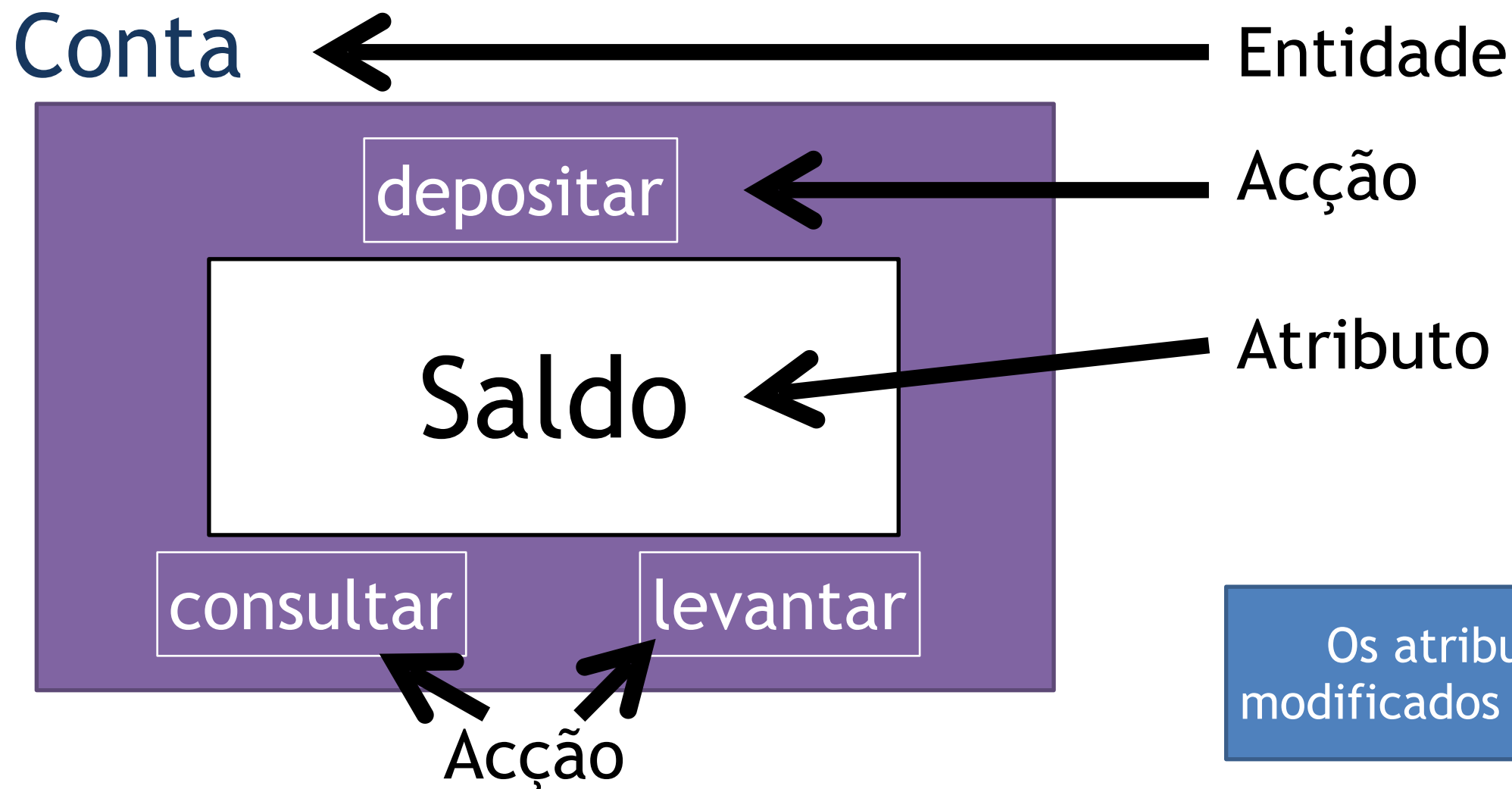


Atributo

O saldo faz parte do estado da conta

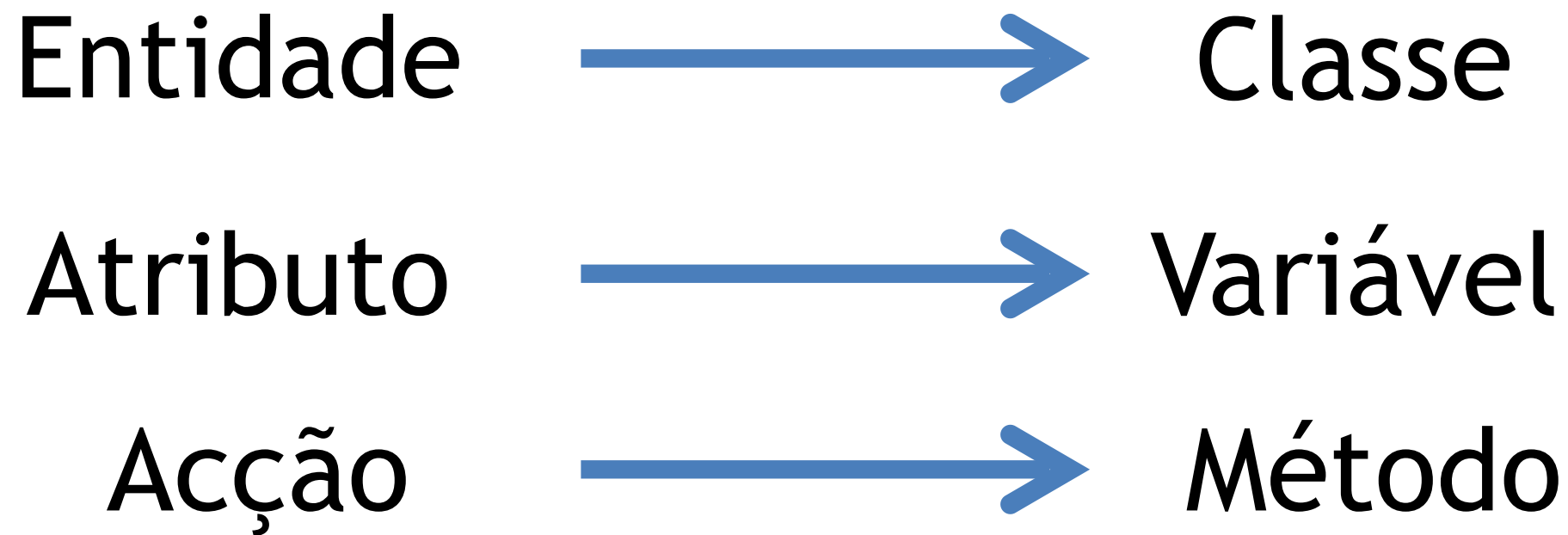
Entidades

Entidades têm dados/estado (atributos) e comportamento (acções)

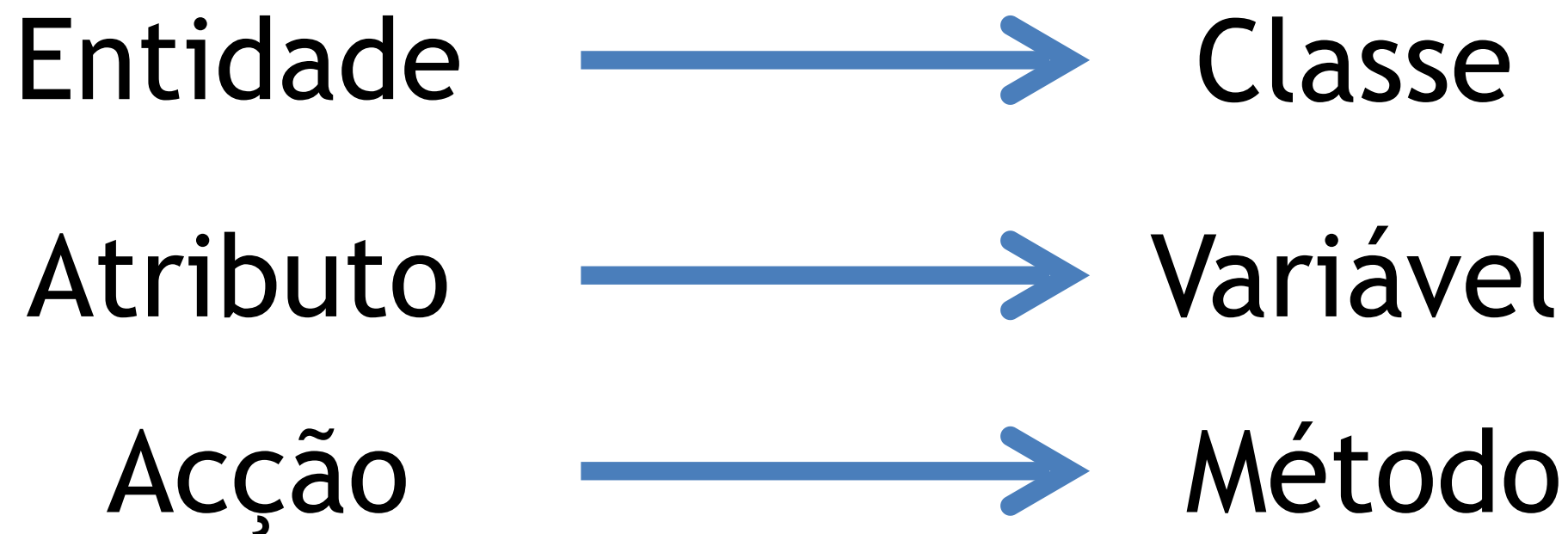


Os atributos são lidos/
modificados através de acções

Programação por objetos



Programação por objetos



A classe Carro tem as variáveis Estado e Velocidade e os métodos:

Ligar (altera Estado)
Desligar (altera Estado)
Acelerar (altera Velocidade)

Tipo vs Valor

Uma variável tem um tipo e (possivelmente) um valor

```
int idade = 30;  
String nome = "Pedro";
```

variável	tipo	valor
idade	int	30
nome	String	"Pedro"

Classe vs Objecto

Em Programação Orientada a Objectos, a variável continua a ter um tipo que é uma Classe e um valor que é um Objecto

```
Aluno aluno = new Aluno("Pedro", 21500000);  
Carro carro = new Carro("11-FF-33");
```

variável	Classe	Objecto
aluno	Aluno	aluno "Pedro" com o número 21500000
carro	Carro	carro com a matrícula 11-FF-33

Classe vs Objeto

Uma **Classe** representa uma entidade genérica.
Um **Objeto** é uma concretização específica de uma classe.

Classe	Objetos
Carro	<ul style="list-style-type: none">• Carro 45-GP-31• Carro 21-LM-10• Carro 67-GX-38
Saco	<ul style="list-style-type: none">• Saco do Manel• Saco da Maria

Classe vs Objeto

Um Objeto é sempre de uma Classe

O objecto

“Conta 435465567 CGD”

é da classe

Conta

Uma classe tem muitos objetos mas
um objecto só pertence a uma
classe

Classe vs Objeto

Um objecto é uma instância (ocorrência) de uma classe.

Java

Classes são declaradas com **class**
Objectos são criados/instanciados com **new**

```
class Carro {  
}  
  
class Mota {  
}  
  
public class App {  
    public static void main(String[] args) {  
        Carro carroDoPedro = new Carro();  
        Carro carroDaMaria = new Carro();  
        Mota motaComMatricula38GH54 = new Mota();  
    }  
}
```

Quantas classes estão declaradas?
Quantos objectos são criados?

Java

Classes são declaradas com **class**

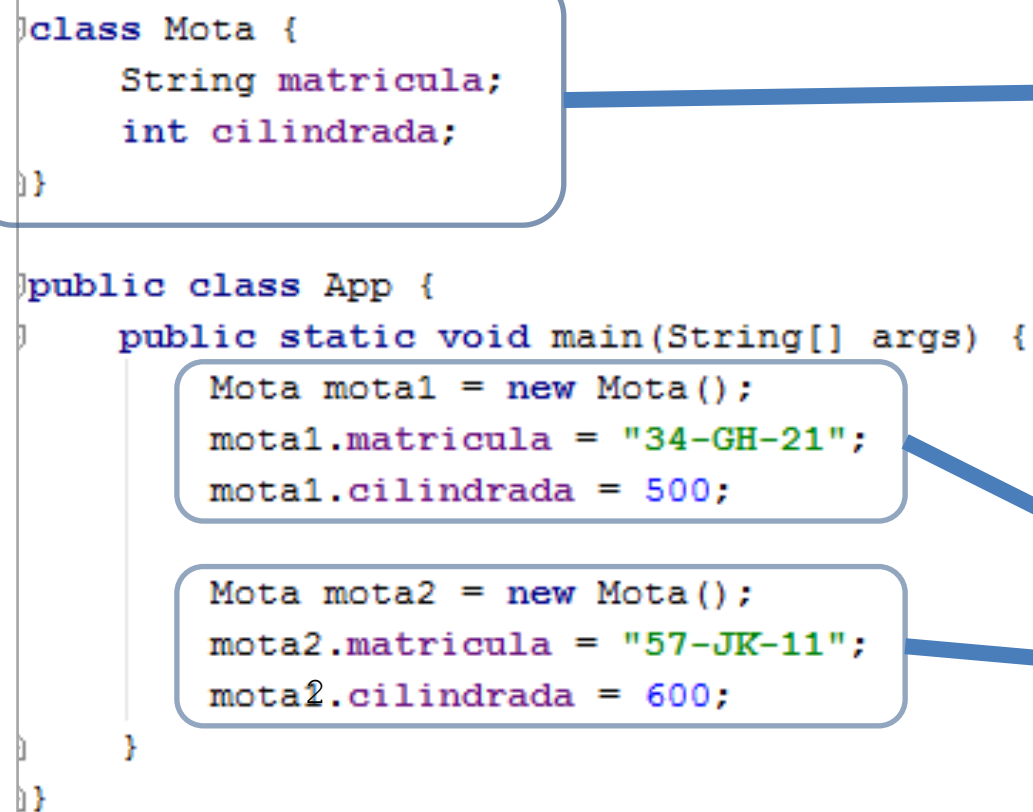
Objectos são criados com **new**

```
class Mota {  
}  
  
public class App {  
    public static void main(String[] args) {  
        for (int i=0; i < 10; i++) {  
            Mota mota = new Mota();  
        }  
    }  
}
```

Quantas classes estão declaradas?
Quantos objectos são criados?

Classe vs Objecto

```
class Mota {  
    String matricula;  
    int cilindrada;  
}  
  
public class App {  
    public static void main(String[] args) {  
        Mota mota1 = new Mota();  
        mota1.matricula = "34-GH-21";  
        mota1.cilindrada = 500;  
  
        Mota mota2 = new Mota();  
        mota2.matricula = "57-JK-11";  
        mota2.cilindrada = 600;  
    }  
}
```



A Classe define uma ficha com campos em branco (variáveis)

Cada Objeto preenche essa ficha de forma diferente

Classes vs Objetos

Classe Utilizador

The form is titled 'Classe Utilizador' and contains the following fields:

- Name ***: Two text boxes for 'First' and 'Last' names.
- Time ***: Two text boxes for 'HH' and 'MM', followed by a dropdown menu for 'AM/PM' (currently showing 'AM').
- Email ***: A single text box with the instruction 'Please use your office email address.' below it.
- Date ***: Three text boxes for 'MM', 'DD', and 'YYYY', separated by slashes, with a calendar icon to the right.
- Address ***: Three text boxes for 'Street Address', 'Street Address Line 2', and 'City'.
- Region**: A text box for the 'Region'.

Classes são “formulários” em branco.

- Só existe um por aplicação
- Tem que se instanciar antes de utilizar (fazer uma “fotocópia”)

```
utilizador = new Utilizador()
```

- Após instanciar ficamos com um objeto dessa classe - uma cópia do formulário que podemos preencher

Classes vs Objetos

Objetos da classe Utilizador

The image shows three overlapping user registration forms, each representing an instance (object) of the 'Utilizador' class. The forms are identical in structure but contain different data, demonstrating how objects are created from a single class template.

Name *

First: Last:

Time *

:

Email *

Please use your office email address.

Date *

/ /

Address *

Street Address

Street Address Line 2

City Region

Problema

E se eu quiser saber se uma moto tem alta cilindrada?



Hipótese 1

(programação imperativa “tudo na função main”)

```
class Mota {  
    String matricula;  
    int cilindrada;  
}  
  
public class App {  
  
    public static void main(String[] args) {  
        Mota mota = new Mota();  
        mota.matricula = "34-GH-21";  
        mota.cilindrada = 500;  
  
        if (mota.cilindrada > 1000) {  
            System.out.println("A mota " + mota.matricula + " tem alta cilindrada");  
        }  
    }  
}
```

Hipótese 1

(programação imperativa “tudo na função main”)

```
class Mota {  
    String matricula;  
    int cilindrada;  
}  
  
public class App {  
  
    public static void main(String[] args) {  
        Mota mota = new Mota();  
        mota.matricula = "34-GH-21";  
        mota.cilindrada = 500;  
  
        if (mota.cilindrada > 1000) {  
            System.out.println("A mota " + mota.matricula + " tem alta cilindrada");  
        }  
    }  
}
```

Problema: Para cada mota, tenho que repetir este código!

Hipótese 2

(programação imperativa com funções)

```
class Mota {  
    String matricula;  
    int cilindrada;  
}
```

```
public class App {
```

```
    static boolean temAltaCilindrada(Mota mota) {  
        if (mota.cilindrada > 1000) {  
            return true;  
        } else {  
            return false;  
        }  
    }  
}
```

```
    public static void main(String[] args) {  
        Mota mota = new Mota();  
        mota.matricula = "34-GH-21";  
        mota.cilindrada = 500;  
  
        if (temAltaCilindrada(mota)) {  
            System.out.println("A mota " + mota.matricula + " tem alta cilindrada");  
        }  
    }  
}
```

chama função



Hipótese 2

(programação imperativa com funções)

```
class Mota {  
    String matricula;  
    int cilindrada;  
}  
  
public class App {  
    static boolean temAltaCilindrada(Mota mota) {  
        if (mota.cilindrada > 1000) {  
            return true;  
        } else {  
            return false;  
        }  
    }  
}
```

Problema: Esta função não está na classe (módulo) certo! Está na “App” e devia estar na “Mota”

```
public static void main(String[] args) {  
    Mota mota = new Mota();  
    mota.matricula = "34-GH-21";  
    mota.cilindrada = 500;  
  
    if (temAltaCilindrada(mota)) {  
        System.out.println("A mota " + mota.matricula + " tem alta cilindrada");  
    }  
}
```

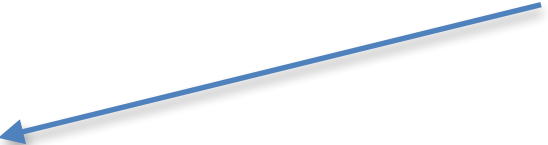
chama função

Hipótese 3

(programação orientada a objectos)

```
class Mota {  
    String matricula;  
    int cilindrada;  
  
    boolean temAltaCilindrada() {  
        if (cilindrada > 1000) {  
            return true;  
        } else {  
            return false;  
        }  
    }  
}
```

A função passou para dentro da Mota e deixou de ser “static”



```
public class App {  
  
    public static void main(String[] args) {  
        Mota mota = new Mota();  
        mota.matricula = "34-GH-21";  
        mota.cilindrada = 500;  
  
        if (mota.temAltaCilindrada()) {  
            System.out.println("A mota " + mota.matricula + " tem alta cilindrada");  
        }  
    }  
}
```

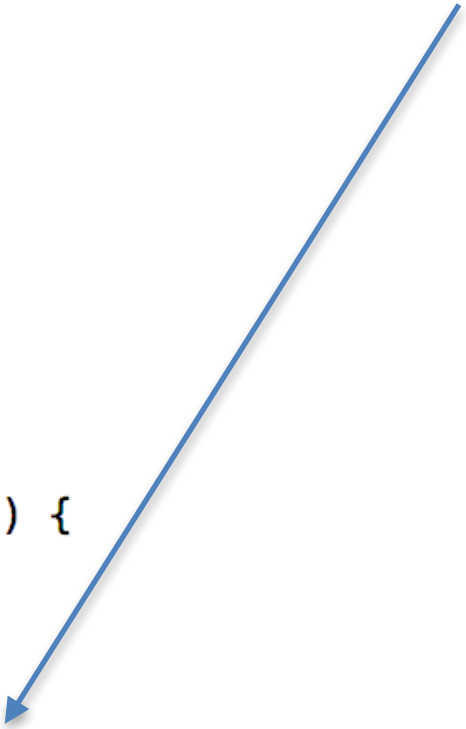
Hipótese 3

(programação orientada a objectos)

```
class Mota {  
    String matricula;  
    int cilindrada;  
  
    boolean temAltaCilindrada() {  
        if (cilindrada > 1000) {  
            return true;  
        } else {  
            return false;  
        }  
    }  
}
```

```
public class App {  
  
    public static void main(String[] args) {  
        Mota mota = new Mota();  
        mota.matricula = "34-GH-21";  
        mota.cilindrada = 500;  
  
        if (mota.temAltaCilindrada()) {  
            System.out.println("A mota " + mota.matricula + " tem alta cilindrada");  
        }  
    }  
}
```

Sempre que precisarmos
de saber se uma mota
tem alta cilindrada
basta perguntar-lhe



Comparação

```
class Mota {
    String matricula;
    int cilindrada;
}

public class App {

    static boolean temAltaCilindrada(Mota mota) {
        if (mota.cilindrada > 1000) {
            return true;
        } else {
            return false;
        }
    }

    public static void main(String[] args) {
        Mota mota = new Mota();
        mota.matricula = "34-GH-21";
        mota.cilindrada = 500;

        if (temAltaCilindrada(mota)) {
            System.out.println("A mota " + mota.m
        }
    }
}
```

Programação imperativa: A função recebe uma mota e diz-me se tem alta cilindrada

```
class Mota {
    String matricula;
    int cilindrada;

    boolean temAltaCilindrada() {
        if (cilindrada > 1000) {
            return true;
        } else {
            return false;
        }
    }
}

public class App {

    public static void main(String[] args) {
        Mota mota = new Mota();
        mota.matricula = "34-GH-21";
        mota.cilindrada = 500;

        if (mota.temAltaCilindrada()) {
            System.out.println("A mota " + mota.m
        }
    }
}
```

Programação orientada a objectos: Pergunto à mota se ela tem alta cilindrada

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 Aplicacao {  
    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

```
class ContaBancaria {  
    int saldo;  
}  
  
public class Aplicacao {  
    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

```
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;  
        }  
    }  
}
```

```
public class Aplicacao {  
    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

```
public class Aplicacao {  
    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 variável “saldo”

Princípio do encapsulamento

O estado de um objecto não deve ser manipulado diretamente

```
conta.saldo += 10;
```

Devem-se executar os métodos que o objecto nos dá

```
conta.deposita(10);
```

Paint (episódio 2)

Passos a seguir:

1. Se entregaram o episódio 1, já têm tudo preparado, basta clicarem neste link do episódio 2 e fazerem o mesmo que fizeram para o episódio 1:

<https://classroom.github.com/a/TLJIJtwu>

Vídeo explicativo no Moodle

2. Se não entregaram o episódio 1, devem ver o vídeo/slides desse episódio que explicam tudo o que devem fazer. A única coisa que muda é o link para o classroom (github) que passa a ser o que está em cima.