


Métodos

 Paragem
nome : String
temNome(nome:String): boolean obterNome(): String registarNome(nome: String): void

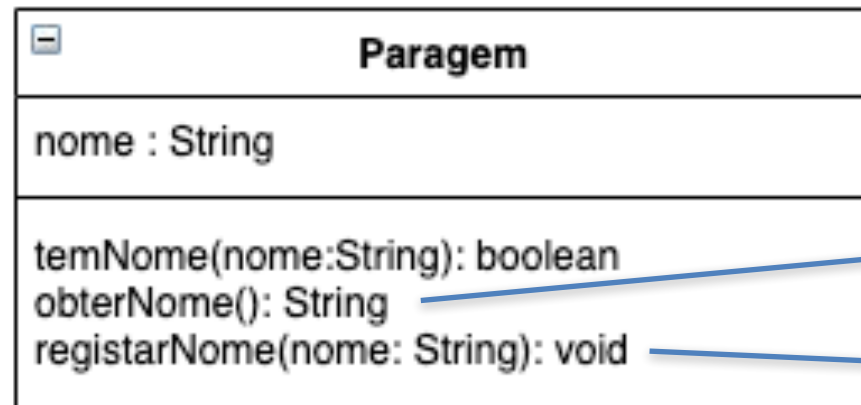
Métodos

Paragem
nome : String
temNome(nome:String): boolean obterNome(): String registarNome(nome: String): void

Lê a variável nome (e retorna-a)

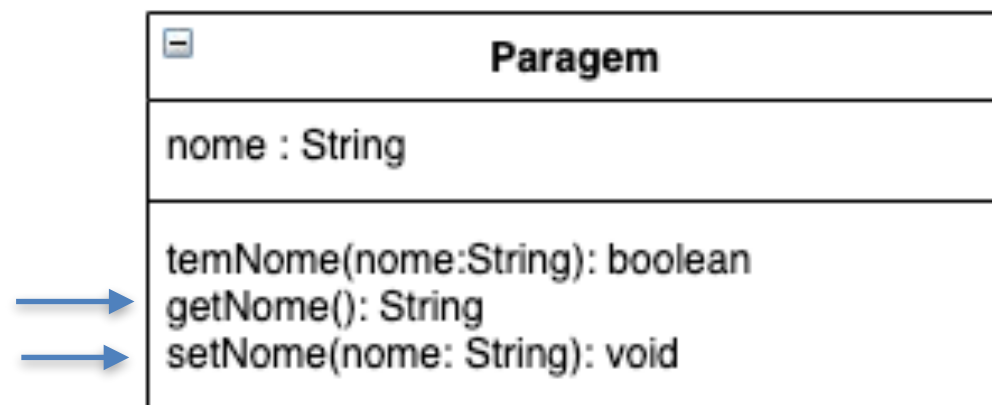
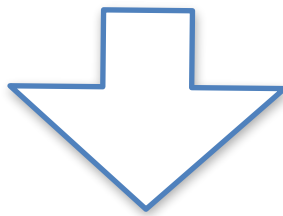
Escreve a variável nome

Métodos



Lê a variável nome (e retorna-a)

Escreve a variável nome



Métodos

Getters e Setters

Métodos que permitem aceder às variáveis mantendo o princípio do encapsulamento

Em Java, usa-se a notação:

```
get<Variavel>()  
set<Variavel>()
```

Em C#:

```
private string type;  
public string Type  
{  
    get { return this.type; }  
    set { this.type = value; }  
}
```

Exercício

Completa a classe abaixo

```
class ContaBancaria {  
  
    int saldo;  
  
    int getSaldo() {  
        return saldo;  
    }  
  
    void setSaldo(int saldoACarregar) {  
        saldo = saldoACarregar;  
    }  
  
    void deposita(int valor) {  
        saldo += valor;  
    }  
}
```

Enviar via teams para p4997

Getters e setters

```
class ContaBancaria {
```

```
    int saldo;
```

```
    int getSaldo() {  
        return saldo;  
    }
```

Consulta o estado (*getter*)

```
    void setSaldo(int saldoACarregar) {  
        saldo = saldoACarregar;  
    }
```

Injeta estado (*setter*)

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

Altera estado

```
}
```

Getters e setters

Paragem
nome : String
temNome(nome:String): boolean getNome(): String setNome(nome: String): void

```
class Paragem {  
    String nome;  
  
    boolean temNome(String nome) {  
        return nome.equals(nome);  
    }  
  
    String getNome() {  
        return nome;  
    }  
  
    void setNome(String nome) {  
        nome = nome;  
    }  
}
```

Getters e setters

Paragem
nome : String
temNome(nome:String): boolean getNome(): String setNome(nome: String): void

```
class Paragem {  
    String nome;  
  
    boolean temNome(String nome) {  
        return nome.equals(nome);  
    }  
  
    String getNome() {  
        return nome;  
    }  
  
    void setNome(String nome) {  
        nome = nome;  
    }  
}
```

Isto está correcto?

Getters e setters

Estamos a comparar uma variável com ela própria???

```
class Paragem {  
    String nome;  
  
    boolean temNome(String nome) {  
        return nome.equals(nome);  
    }  
  
    String getNome() {  
        return nome;  
    }  
  
    void setNome(String nome) {  
        nome = nome;  
    }  
}
```

Qual a variável “nome” que estamos a alterar???

this

“**this**” é uma variável implícita que todos os objetos têm e que representa o próprio objeto


this

```
class Paragem {  
    String nome;  
  
    boolean temNome(String nome) {  
        return this.nome.equals(nome);  
    }  
  
    String getNome() {  
        return this.nome;  
    }  
  
    void setNome(String nome) {  
        this.nome = nome;  
    }  
}
```

Assim, já é claro que estamos a comparar variáveis diferentes!



Na realidade, aqui nem se costuma usar o “this”, pois não há necessidade de desambiguar a variável



Assim, já é claro qual a variável “nome” que estamos a alterar!



Boas práticas

Os getters e setters devem ser usados com cuidado pois expõem detalhes “íntimos” do objecto.

```
class ContaBancaria {  
    BigDecimal saldo;  
  
    void deposita(int valor) {  
        this.saldo.add(new BigDecimal(valor));  
    }  
  
    BigDecimal getSaldo() {  
        return saldo;  
    }  
}
```

Não expõe o tipo do saldo

Expõe o tipo do saldo

Boas práticas

(retirado da cábula UML)

Evitar setters, reflectir as acções que modificam o objecto

Classe	Má prática	Boa prática
Conta	setSaldo()	levantar() depositar()
Veiculo	<u>setVelocidade()</u>	acelerar() travar()
Ponto	setX() setY()	moveCima() moveEsquerda()
Aparelho	setLigado()	ligar()

Exercício 1

```
class Pessoa {
    String nome;
    Pessoa outraPessoa, outraPessoa2;

    void fazCoisas() {

        outraPessoa = new Pessoa();
        outraPessoa2 = new Pessoa();
        outraPessoa.nome = "joana";
        outraPessoa2.nome = "antonio";

        System.out.println("->" + outraPessoa2.nome);
        System.out.println("->" + this.nome);
        System.out.println("->" + outraPessoa.nome);
    }
}

public class Aplicacao {
    public static void main(String args[]) {
        Pessoa pessoaA = new Pessoa();
        pessoaA.nome = "pedro";
        pessoaA.fazCoisas();
    }
}
```

Qual o output deste programa?

Enviar via teams para p4997

Exercício 1 (resolução)

```
class Pessoa {
    String nome;
    Pessoa outraPessoa, outraPessoa2;

    void fazCoisas() {

        outraPessoa = new Pessoa();
        outraPessoa2 = new Pessoa();
        outraPessoa.nome = "joana";
        outraPessoa2.nome = "antonio";

        System.out.println("->" + outraPessoa2.nome);
        System.out.println("->" + this.nome);
        System.out.println("->" + outraPessoa.nome);
    }
}

public class Aplicacao {
    public static void main(String args[]) {
        Pessoa pessoaA = new Pessoa();
        pessoaA.nome = "pedro";
        pessoaA.fazCoisas();
    }
}
```

Qual o output deste programa?

->antonio
->pedro
->joana

Exercício 2

```
class Pessoa {
    String nome = "filipa";
    Pessoa outraPessoa, outraPessoa2;

    void fazCoisas() {

        outraPessoa = new Pessoa();
        outraPessoa2 = new Pessoa();
        outraPessoa2.nome = "antonio";

        System.out.println("->" + outraPessoa2.nome);
        System.out.println("->" + this.nome);
        System.out.println("->" + outraPessoa.nome);
    }
}

class Aplicacao {
    public static void main(String args[]) {
        Pessoa pessoaA = new Pessoa();
        pessoaA.nome = "pedro";
        pessoaA.fazCoisas();
    }
}
```

Qual o output deste programa?

Enviar via teams para p4997

Exercício 2

(resolução)

```
class Pessoa {
    String nome = "filipa";
    Pessoa outraPessoa, outraPessoa2;

    void fazCoisas() {

        outraPessoa = new Pessoa();
        outraPessoa2 = new Pessoa();
        outraPessoa2.nome = "antonio";

        System.out.println("->" + outraPessoa2.nome);
        System.out.println("->" + this.nome);
        System.out.println("->" + outraPessoa.nome);
    }
}

class Aplicacao {
    public static void main(String args[]) {
        Pessoa pessoaA = new Pessoa();
        pessoaA.nome = "pedro";
        pessoaA.fazCoisas();
    }
}
```

Qual o output deste programa?

->antonio
->pedro
->filipa

Exercício 3

```
class Pessoa {
    String nome = "filipa";
    Pessoa outraPessoa, outraPessoa2;

    void fazCoisas() {

        outraPessoa = new Pessoa();
        outraPessoa2 = new Pessoa();
        outraPessoa2.nome = "antonio";

        System.out.println("->" + outraPessoa2.nome);
        System.out.println("->" + this.nome);
        System.out.println("->" + outraPessoa.nome);

        if (this.nome.equals("pedro")) {
            outraPessoa.fazCoisas();
        }
    }
}

class Aplicacao {
    public static void main(String args[]) {
        Pessoa pessoaA = new Pessoa();
        pessoaA.nome = "pedro";
        pessoaA.fazCoisas();
    }
}
```

Qual o output deste programa?

Enviar via teams para p4997

Exercício 3

```
class Pessoa {
    String nome = "filipa";
    Pessoa outraPessoa, outraPessoa2;

    void fazCoisas() {

        outraPessoa = new Pessoa();
        outraPessoa2 = new Pessoa();
        outraPessoa2.nome = "antonio";

        System.out.println("->" + outraPessoa2.nome);
        System.out.println("->" + this.nome);
        System.out.println("->" + outraPessoa.nome);

        if (this.nome.equals("pedro")) {
            outraPessoa.fazCoisas();
        }
    }
}

class Aplicacao {
    public static void main(String args[]) {
        Pessoa pessoaA = new Pessoa();
        pessoaA.nome = "pedro";
        pessoaA.fazCoisas();
    }
}
```

Qual o output deste programa?

->antonio
->pedro
->filipa
->antonio
->filipa
->filipa

Construtores

Inicialização de objetos



Construtores

```
class Carro {  
    String cor;  
  
    boolean eAmarelo() {  
        if (this.cor.equals("amarelo")) {  
            return true;  
        } else {  
            return false;  
        }  
    }  
}  
  
public class App {  
  
    public static void main(String[] args) {  
        Carro carro = new Carro();  
  
        if (carro.eAmarelo()) {  
            System.out.println("É amarelo!!!");  
        }  
    }  
}
```

Este programa vai crashar!
Porquê?

Construtores

```
class Carro {  
    String cor;  
  
    boolean eAmarelo() {  
        if (this.cor.equals("amarelo")) {  
            return true;  
        } else {  
            return false;  
        }  
    }  
}  
  
public class App {  
  
    public static void main(String[] args) {  
        Carro carro = new Carro();  
  
        if (carro.eAmarelo()) {  
            System.out.println("É amarelo!!!");  
        }  
    }  
}
```

Rebenta!!! A cor do carro não foi inicializada!



Construtores

```
public static void main(String[] args) {  
    Carro carro = new Carro();  
    carro.setCor("azul");  
  
    if (carro.eAmarelo()) {  
        System.out.println("É amarelo!!!");  
    }  
}
```

← Aparentemente resolve!

```
void fazCoisas(Carro carro) {  
    if (carro.eAmarelo()) {  
        System.out.println("É amarelo!!!");  
    }  
}
```

← Será que foi inicializado? Ou vai rebentar??

Construtores

Construtor é o método que é chamado quando se cria o objecto (por exemplo, quando se faz “new” em Java)

Pode receber zero, um ou mais argumentos

```
Carro carro = new Carro();  
Carro carro2 = new Carro("azul");  
Carro carro3 = new Carro("azul", "45-HG-21");
```

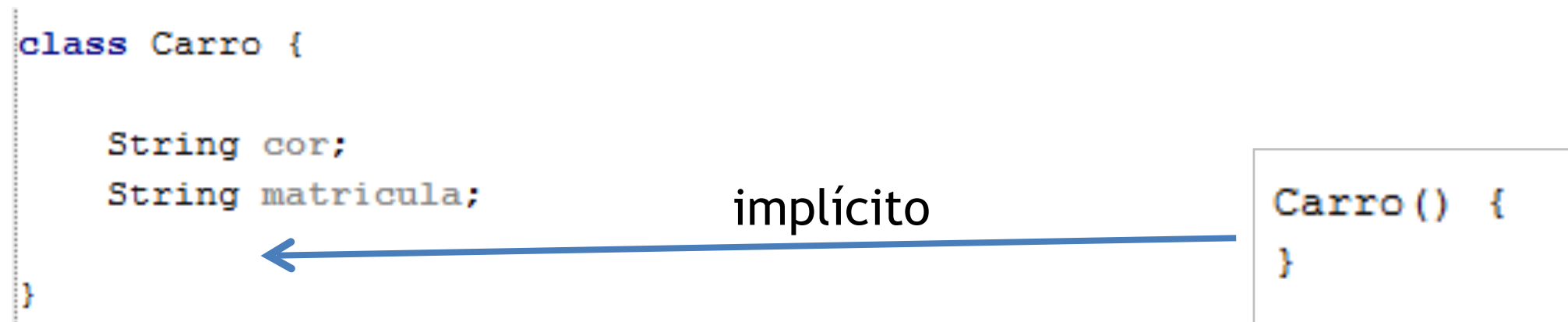

Construtores

Construtores têm um nome igual à classe e não retornam nada (nem void)

```
class Carro {  
  
    String cor;  
    String matricula;  
  
    Carro() {  
    }  
  
    Carro(String cor) {  
        this.cor = cor;  
    }  
  
    Carro(String cor, String matricula) {  
        this.cor = cor;  
        this.matricula = matricula;  
    }  
}
```

Construtores

Se a classe não declarar construtores, é criado um construtor implícito sem argumentos



Construtores

Se a classe declarar construtor(es), já não há construtor implícito

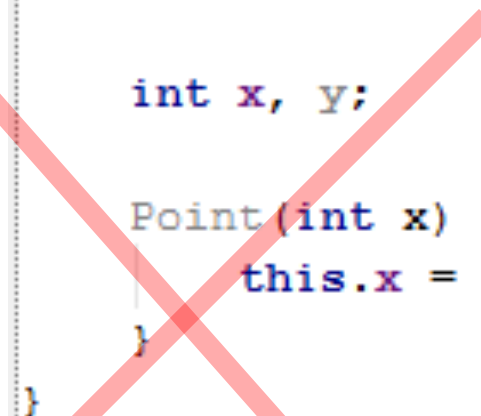
```
class Carro {  
    String cor;  
    String matricula;  
  
    Carro(String cor) {  
        this.cor = cor;  
    }  
}
```

implícito

```
Carro() {  
}
```

Construtores

O Construtor deve deixar o objeto num estado completo



```
class Point {  
  
    int x, y;  
  
    Point(int x) {  
        this.x = x;  
    }  
}
```

```
class Point {  
  
    int x, y;  
  
    Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

Construtores

Construtor sem argumentos

```
class Carro {  
    int cilindrada = 1000;  
}
```

Construtor Implícito

```
class Carro {  
    int cilindrada;  
  
    Carro() {  
        this.cilindrada = 1000;  
    }  
}
```

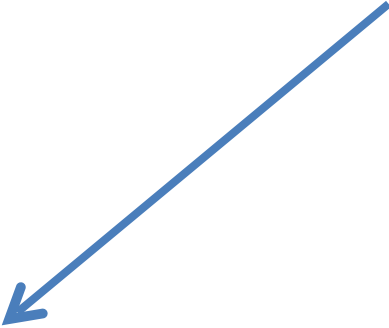
Construtor Explícito

Ambos são válidos!!

Conta Bancária revisitada

```
class ContaBancaria {  
  
    int saldo;  
  
    ContaBancaria(int saldoInicial) {  
        this.saldo = saldoInicial;  
    }  
}  
  
public class App {  
  
    public static void main(String[] args) {  
  
        ContaBancaria conta1 = new ContaBancaria();  
  
        ContaBancaria conta2 = new ContaBancaria(3455000);  
  
    }  
}
```

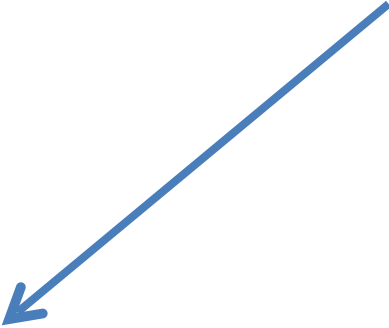
Qual o erro?



Conta Bancária revisitada

```
class ContaBancaria {  
  
    int saldo;  
  
    ContaBancaria(int saldoInicial) {  
        this.saldo = saldoInicial;  
    }  
}  
  
public class App {  
  
    public static void main(String[] args) {  
  
        ContaBancaria conta1 = new ContaBancaria();  
  
        ContaBancaria conta2 = new ContaBancaria(3455000);  
  
    }  
}
```

Erro! Não há construtor sem argumentos! Ou seja, a conta tem sempre que ser criada com um saldo inicial!



Correcto



Conta Bancária revisitada

```
class ContaBancaria {  
  
    int saldo;  
  
    ContaBancaria(int saldoInicial) {  
        if (saldoInicial <= 0) {  
            throw new IllegalArgumentException();  
        } else {  
            this.saldo = saldoInicial;  
        }  
    }  
}
```

Este construtor
garante que a conta
tem sempre saldo
inicial positivo!

Exercício

```
class Carro {  
  
    String matricula;  
    int cilindrada;  
    int velocidadeAtual;  
  
}
```

Construtor?
Setters?

Enviar via teams para p4997

Resolução

```
class Carro {  
  
    String matricula;  
    int cilindrada;  
    int velocidadeAtual;  
  
    Carro(String matricula, int cilindrada) {  
        this.matricula = matricula;  
        this.cilindrada = cilindrada;  
        this.velocidadeAtual = 0;    // o carro começa sempre parado  
    }  
  
    void setVelocidadeAtual(int velocidadeAtual) {  
        this.velocidadeAtual = velocidadeAtual;  
    }  
}
```

Resolução

Não há setters para a matrícula e para a cilindrada porque uma vez criado o objecto, já não vão mudar

Também não é possível criar um carro que já esteja em movimento

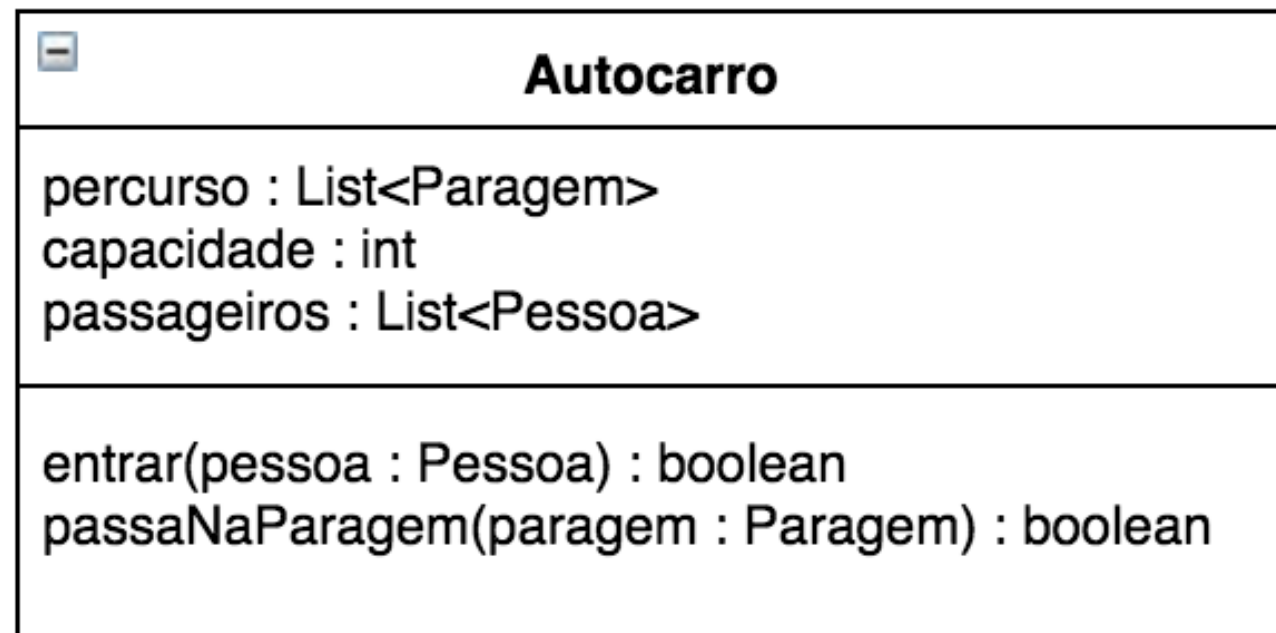
```
class Carro {  
  
    String matricula;  
    int cilindrada;  
    int velocidadeAtual;  
  
    Carro(String matricula, int cilindrada) {  
        this.matricula = matricula;  
        this.cilindrada = cilindrada;  
        this.velocidadeAtual = 0;    // o carro começa sempre parado  
    }  
  
    void setVelocidadeAtual(int velocidadeAtual) {  
        this.velocidadeAtual = velocidadeAtual;  
    }  
}
```

Resolução

```
class Carro {  
  
    String matricula;  
    int cilindrada;  
    int velocidadeAtual;  
  
    Carro(String matricula, int cilindrada) {  
        this.matricula = matricula;  
        this.cilindrada = cilindrada;  
        this.velocidadeAtual = 0;    // o carro começa sempre parado  
    }  
  
    void setVelocidadeAtual(int velocidadeAtual) {  
        this.velocidadeAtual = velocidadeAtual;  
    }  
}
```

Mas melhor ainda que usar o setVelocidadeAtual() seriam os métodos acelera() e trava()

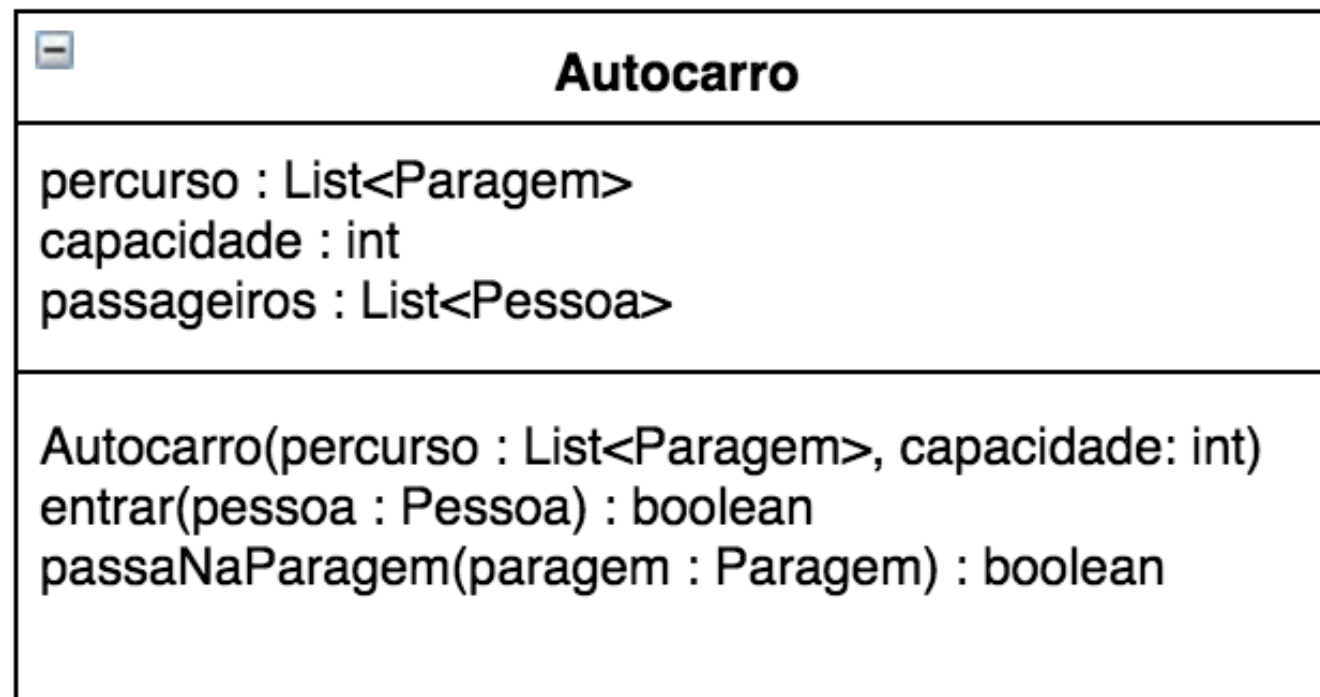
Exercício



Enviar via teams para p4997

Que construtor deve ter esta classe?

Resolução



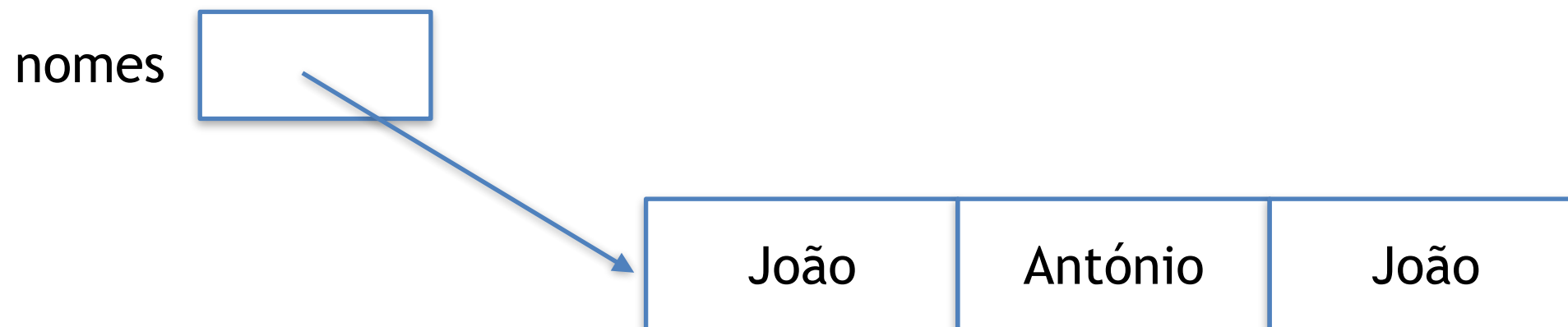
Resolução

```
class Autocarro {  
    List<Paragem> percurso;  
    int capacidade;  
    List<Pessoa> passageiros;  
  
    public Autocarro(List<Paragem> percurso, int capacidade) {  
        this.percurso = percurso;  
        this.capacidade = capacidade;  
        this.passageiros = new ArrayList<>();  
    }  
}
```

Representação em memória

```
List<String> nomes = new ArrayList<>();  
  
// adiciona 3 elementos à lista  
nomes.add("João");  
nomes.add("António");  
nomes.add("João");
```

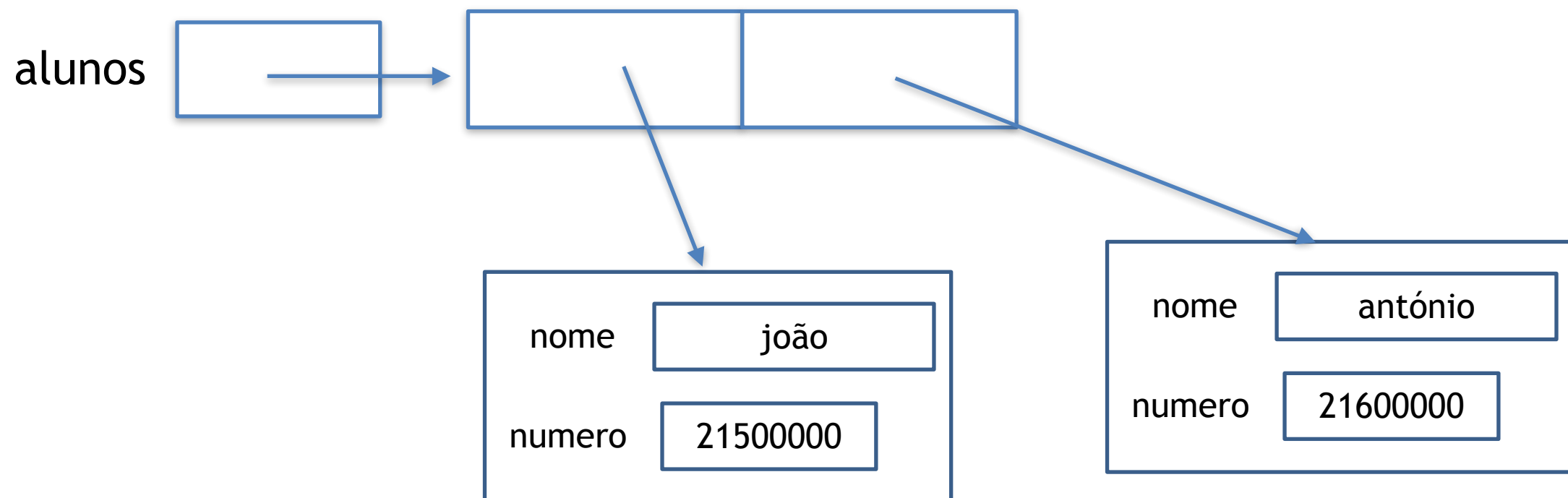
Estrutura de memória



Representação em memória

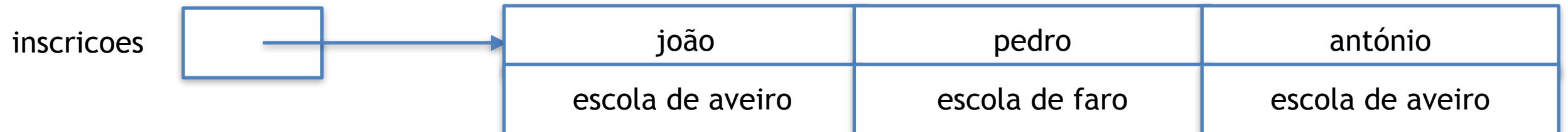
```
List<Aluno> alunos = new ArrayList<>();  
  
// adiciona 2 elementos à lista  
alunos.add(new Aluno("João", 21500000));  
alunos.add(new Aluno("António", 21600000));
```

Estrutura de memória



Map

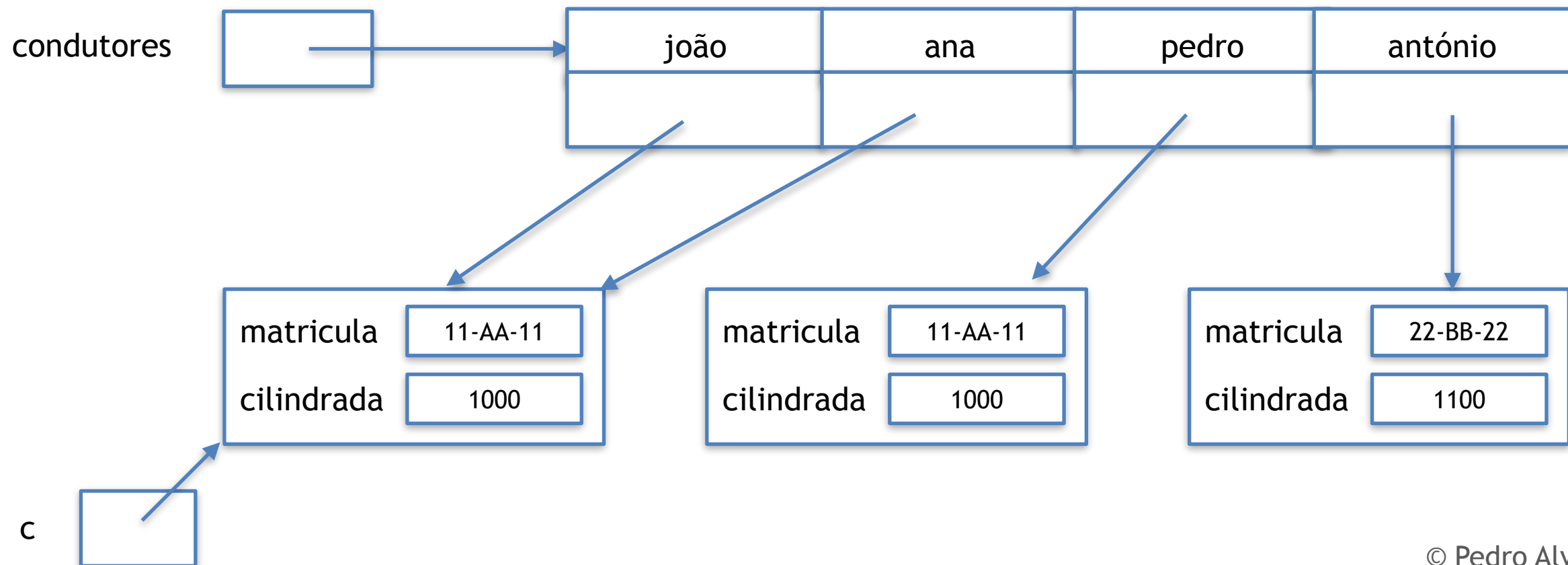
```
Map<String,String> inscricoes = new HashMap<>();  
  
// adiciona 3 pares (chave, valor) ao map  
inscricoes.put("joão", "Escola de Aveiro");  
inscricoes.put("antónio", "Escola de Faro");  
inscricoes.put("pedro", "Escola de Aveiro");
```



Map

```
Map<String,Carro> condutores = new HashMap<>();
```

```
Carro c = new Carro("11-AA-11", 1000);  
condutores.put("joão", c);  
condutores.put("antónio", new Carro("22-BB-22", 1100));  
condutores.put("pedro", new Carro("11-AA-11", 1000));  
condutores.put("ana", c);
```



Exercício

Desenhe a estrutura de memória no final da execução deste programa
construtores e classe/função principais omitidos por simplificação

```
class Garagem {
    List<Carro> carros = new ArrayList<>();

    void estaciona(Carro carro) {
        carros.add(carro);
    }
}

class Carro {
    String matricula;
    int velocidadeMaxima;

    void aplicaTuning() {
        velocidadeMaxima += 20;
    }
}

(...)
Map<String, Garagem> garagens = new HashMap<>();

Carro c = new Carro("11-AA-11", 200);

Garagem g1 = new Garagem();
g1.estaciona(new Carro("22-BB-22", 210));
g1.estaciona(new Carro("33-CC-33", 170));

Garagem g2 = new Garagem();
g2.estaciona(c);

garagens.put("Garagem Auto", g1);
garagens.put("Garagem da esquina", g2);

g2.carros.get(0).aplicaTuning();
```

Enviar via teams para p4997