Orientação a Objetos em Java

Universidade Federal do Amazonas

Instituto de Computação

Técnicas Avançadas de Programação

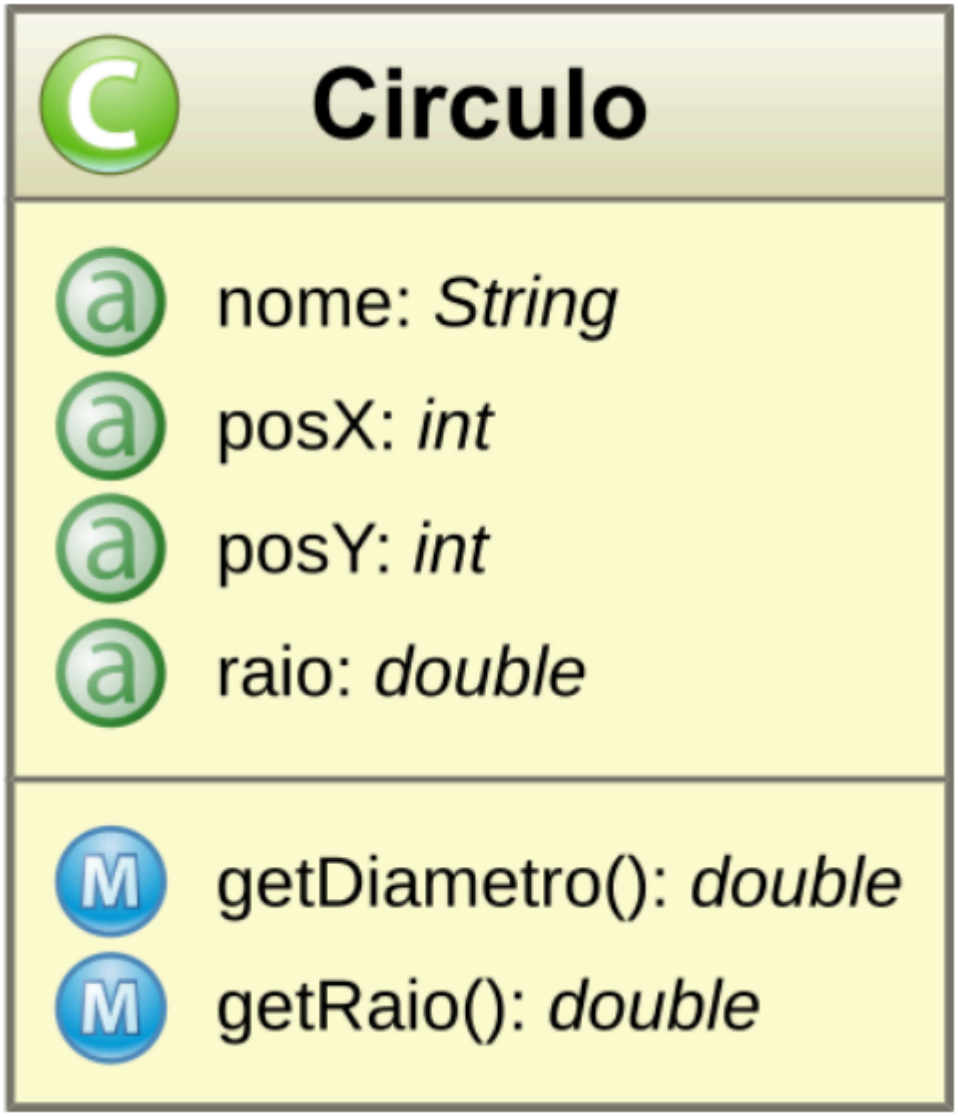
Data: 13 de setembro de 2024
Professor: Horacio Fernandes
Aula: OO em Java

1.1. Classe

Definem um novo tipo de dado e como este dado é implementado - e também definem as características que o objeto daquela classe terá.

Uma classe é separada em:
a. Atributos: funções e operações que os objetos podem executar;
b. Métodos: são os dados e variáveis relacionados aquela classe.

Uma classe acaba sendo estruturada como um diagrama de classes, representando a estrutura e as relações entre elas.



A classe é apenas um pequeno módulo do sistema. Sendo assim, é fundamental que o mesmo tenha apenas uma responsabilidade. Este conceito é conhecido como **coesão**.

Além disso, uma classe deve depender o menos possível das outras - este conceito é conhecido como **acoplamento**.

Sintaxe de uma Classe em Java:

```
class Circulo { // classe circulo
  String nome;
  int posX, posY;
  double raio; // atributos

  double getDiametro() { // método getDiametro
    return 2 * raio;
  }

  double getArea() { // método getArea
    return 3.14159 * raio * raio;
  }
}
```

1.2. Objeto

Um objeto é uma instância de uma classe, criada no tempo de execução. As classes:

- Ocupam espaço na memória;
- Alocam espaço à medida que são criados
- São alocados e desalocados dinamicamente.

Para criar um objeto, deve-se usar o operador `new` para representar um novo objeto.

```
class Principal {
  public static void main(String args[]) {
    Circulo circ = new Circulo(); // Instanciando a classe Circulo e criando o objeto circ

    circ.nome = "FN-2187";
    circ.posX = 7;
```

```
    circ.posY = 3;
    circ.raio = 2.5; // Modificando atributos do objeto circ

    System.out.println("O circulo circ está em ("
                       + circ.posX + ", " + circ.posY // Acessando o atributo posY
                       + "). Possui raio " + circ.raio
                       + " e area " + circ.getArea()) // Executando o método getArea()
  }
}
```

Ex: operações realizadas na linha abaixo:

```
Circulo circ = new Circulo();
```

- Declaração: estou dizendo que `circ` será um objeto da classe `circulo`. Obs: a declaração **não** cria um novo objeto.
- Instanciação: a sintaxe `new` aloca dinamicamente um novo objeto na memória.
- Inicialização: Chamada ao construtor da classe `Circulo`.
- Atribuição: A referência do objeto retornada pelo operador new é atribuída à variável `circ`.

---

2. Atributo

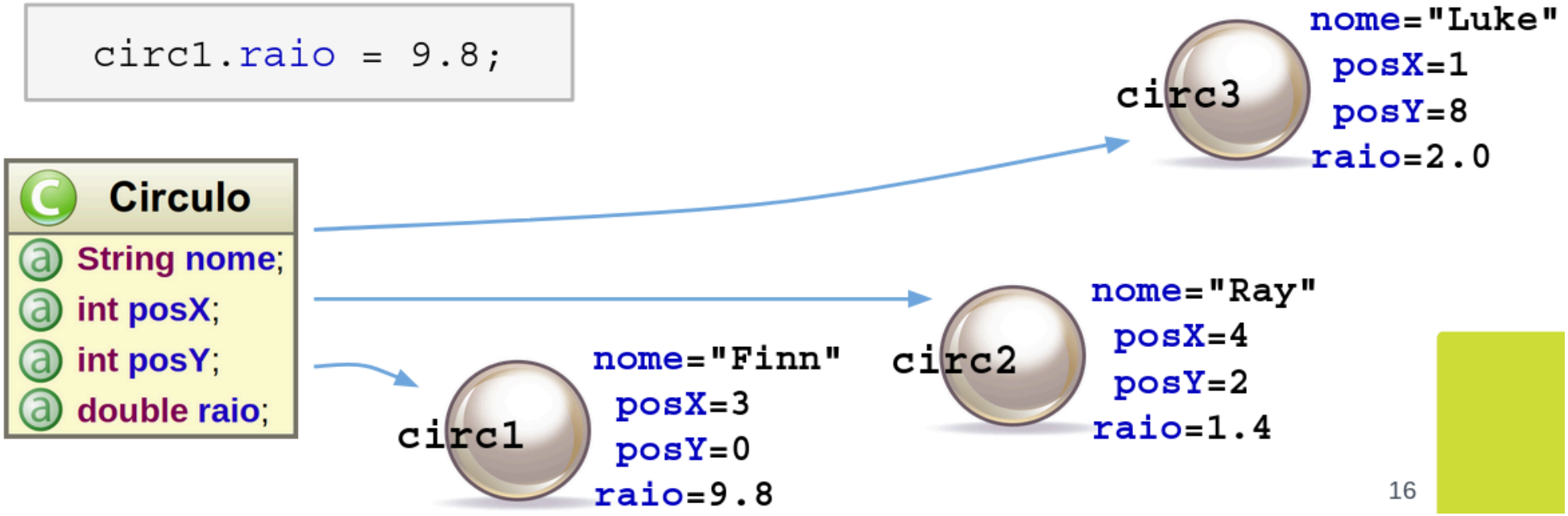
Atributos são dados relacionados a classe.
-> São variáveis dentro do escopo de uma classe e armazenam as informações relacionadas a ele.
-> Cada atributo possui um tipo e um nome. E, as vezes, um modificador de acessos (cenas dos próximos markdowns).

```
int anoPublicacao;
 double raio;
```

```
float notaFinal = 9.8f;
String nomeMestre = "Yoda";
```

Os atributos seguem alguns padrões (boas práticas, né). Nas especificações do Java, existem algumas convenções:
-> Use nomes completos (as vezes até frases!)
-> Separe palavras compostas com a Teoria do `camelCase` (primeira palavra minúscula, as subsequentes começam com maiúscula).

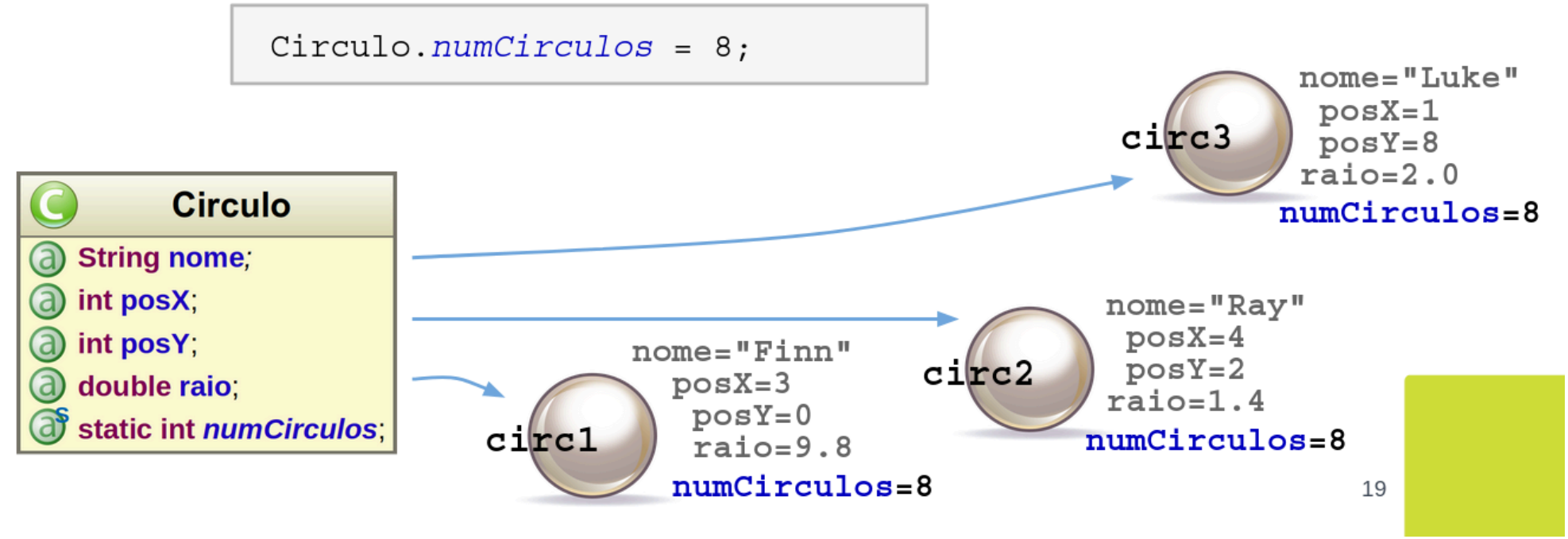
Os atributos podem ser:
-> **De instância**: cada objeto terá uma região reservada na memória para o seu atributo, e pode ter um valor distinto para cada. Estes são acessados através do seu nome `Ex: circ1.raio` (acessei o atributo raio da classe `circ1`). Possíveis outras classes seguem com o valor inalterado.



Outro tipo de atributo:
-> **De classe**: só ocupa uma posição na memória (mesmo se você tiver um zilhão de objetos). Todos os objetos vêem e acessam esta posição (logo, o mesmo valor). É um paizão. Se ele, porventura, ser alterado, todos os objetos da classe sofrem a alteração (por isso é comumente chamado de um atributo da classe). É declarado usando o modificador `static`.

```java
static int numCirculos; // declarei o atributo numCirculos
Circulo.numCirculos = 3; // atribuí o valor 3 a numCirculos
```

Quando eu alterar a variável `numCirculos`, eu mudo o valor dele em todos os objetos.



Mais um tipo de atributo:

-> **Constante**: é um atributo cujo valor não muda depois da inicialização (alguns chamariam de teimoso). São declarados usando o modificador de acesso `final`, e não precisam necessariamente serem inicializados.

-> Em geral, constantes também são estáticas e públicas, buscando facilitar o acesso - normalmente utilizamos para armazenar utilitários (lembra de `Math.PI`?)

```java
public static final double E = 2.7182818284590452354;
public static final double PI = 3.14159265358979323846;
public static final int MIN_VALUE = 0x80000000;
public static final int MAX_VALUE = 0x7fffffff;
```

```
public static final int SIZE = 32;
public static final int BYTES = SIZE / Byte.SIZE; // acessei a constante SIZE da classe BYTE
```

- **Estado de um Objeto**: O estado de um objeto são os valores atuais de seus atributos. O estado de um objeto pode mudar quando alteramos o valor de um atributo diretamente; ou quando executamos um método do objeto (que altera o valor de algum atributo).

---

3. Métodos

Os métodos são as funções que podemos executar relacionadas a uma classe. São os métodos que executam algoritmos que podem acessar e modificar atributos, retornando valores ou voids (lembra de C?). E sim, variáveis locais podem coexistir.

Cada método possui modificadores de acesso (mas depende); um retorno, um nome, parâmetros e um algoritmo de implementação (isso aqui não depende não).

```
// exemplo da classe Circulo
double getArea() {
  return 3.14159 * raio * raio;
}
```

```
// exemplo da classe String (nativo do Java)
public char charAt(int index) {
  if ( (index < 0) || (index >= value.length) ) {
    throw new StringIndexOutOfBoundsException(index);
  }

  return value[index];
}
```

Tal qual os atributos, também existem **métodos de instância e métodos de classe**.

-> Métodos Estáticos: São declarados com o modificador `static`, não precisam de uma instância para ser executados; mas eles não podem acessar atributos de instância (apenas atributos estáticos).

3.2. Sobrecarga de métodos

Em Java, dois ou mais métodos podem ter o mesmo nome! (calma, não é magia. Eles precisam ter parâmetros de tipo e/ou quantidades distintos). Isto é conhecido como **sobrecarga de métodos**.

```
public void println()           { /* ... */ }
public void println(boolean x) { /* ... */ }
public void println(char x)    { /* ... */ }
public void println(float x)   { /* ... */ }
public void println(int x)     { /* ... */ }
public void println(long x)    { /* ... */ }
public void println(Object x)  { /* ... */ }
public void println(String x)  { /* ... */ }
```

3.3. Métodos Especiais

Em Java, temos dois métodos especiais: **destrutor e construtor**.
-> Método construtor: é executado quando um novo objeto de uma classe X é criado. Ele inicializa os atributos do novo objeto - e mais coisas também. Os métodos construtores possuem o mesmo nome da classe, permitem sobrecarga (diversos construtores desde que tenhamos distintos parâmetros) e não tem nenhum retorno. Nem mesmo `void`.

```
// Este construtor não possui parâmetros. Para criar um objeto usando este construtor: Circulo circ = new Circulo();
class Circulo {
  int posX, posY;
  double raio;
```

```
// Este construtor possui um parâmetro do tipo double. Para criar um objeto: Circulo circ = new Circulo(1.2);
  Circulo() {
        posX = 0;
        posY = 0;
        raio = 0.0;
  }

// Este this serve para diferenciar o atributo raio do parâmetro (variável local) raio.
  Circulo(double raio) {
        posX = 0;
        posY = 0;
        this.raio = raio;
  }

// Este construtor possui três parâmetros do tipo double. Para criar um objeto: Circulo circ =  new Circulo(1, 3, 9.8);
  Circulo(int posX, int posY, double raio) {
        this.posX = posX;
        this.posY = posY;
        this.raio = raio;
  }

  // Métodos getDiametro, getArea ...
}
```

Para simplificar os códigos (sonho de consumo dos devs), podemos fazer um construtor chamar outro! (meio mágico. mas é tudo ciência!). Chamamos isto de **encadeamento de construtores**. Inclusive, os Senhores Java recomendam que você faça isso.

```
class Circulo {
  int posX, posY;
  double raio;

// Este construtor usa a palavra reservada this para chamar o construtor que possui três parâmetros do tipo double (último)
  Circulo() {
    this(0, 0, 0.0);
  }
```

```
// Mesma coisa, mas passando o raio
  Circulo(double raio) {
    this(0, 0, raio);
  }

// Este construtor é o principal. Inicializa todos os atributos e será usado pelos outros construtores
  Circulo(int posX, int posY, double raio) {
    this.posX = posX;
    this.posY = posY;
    this.raio = raio;
  }

  // Métodos getDiametro, getArea ...

}
```

Faltou mais um!
-> Método Destrutor: adoramos destruir, afinal, somos seres humanos. O método destrutor é executado quando o objeto é removido da memória. É implementado através do método `finalize`. Porém... assim como na vida real não é muito garantido destruir o meio ambiente; em Java é meio parecido - não temos controle sobre o coletor de lixo do Java, logo não podemos garantir se o método vai ser chamado ou quando ele vai ser chamado.

```
protected void finalize() { /*...*/ }
```

Da série "Não é Magia, é Arquitetura": o **coletor de lixo** do Java é uma das tecnologias chave da linguagem. Como já foi explicado anteriormente, o free(System32) não fica por nossa conta, e sim tudo é feito automaticamente pelo coletor de lixo do Java.

Na real, o funcionamento é bem simples: se um objeto O não tem mais referências em sua direção, então ele pode ser liberado, coletado e removido. Como o Java não permite acesso direto a memória, isso funciona. (os coletores de lixo público podiam ser assim também!).