UNIRIO

Orientação a Objetos

Aula 4

BSI — 2018.2

Jefferson Elbert Simões

CCET/DIA

30 de setembro de 2018

Previously on TP2...

Novo paradigma: programação orientada a objetos

Objetos Conceitos do mundo real a serem representados computacionalmente

Classes Representam grupos de objetos semelhantes e delimitam sua descrição

- Servem de "molde" para objetos
- São os tipos aos quais os objetos pertencem

Atributos Representam características de objetos

Métodos Representam comportamento de objetos

- Ambos compõem a descrição presente em uma classe

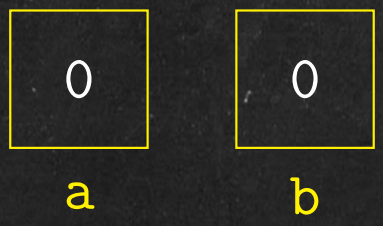
Aula de hoje

- Referências a objetos
- Construtores e destrutores
- UML

Objetos e referências (em Java)

- Ao contrário de tipos primitivos, a declaração de objetos não "cria" automaticamente um novo objeto no momento da declaração

```
class Pessoa {
    int idade;
    int altura;
}
int a, b;
Pessoa alice, beto;
```



Objetos e referências (em Java)

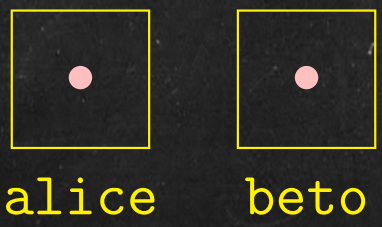
- Em vez disso, a declaração de um objeto de uma classe, na verdade, cria uma referência a um objeto dessa classe

```
class Pessoa {
    int idade;
    int altura;
}
int a, b;
Pessoa alice, beto;
```

Objetos e referências

- Um novo objeto só é criado (a classe é "instanciada") através do uso do operador `new`

```
class Pessoa {
    int idade;
    int altura;
}
Pessoa alice, beto;
```



Objetos e referências

- Um novo objeto só é criado (a classe é "instanciada") através do uso do operador `new`

```
class Pessoa {
    int idade;
    int altura;
}
Pessoa alice, beto;
alice = new Pessoa();
```

Objetos e referências

- Referências são, por default, nulas — `null`
  - Não referenciam nenhum objeto
- Atribuição: iguala referências
- Teste de igualdade: testa se referências são iguais

```
Pessoa alice, beto;
alice = new Pessoa ();
```

Objetos e referências

- Referências são, por default, nulas — `null`
  - Não referenciam nenhum objeto
- Atribuição: iguala referências
- Teste de igualdade: testa se referências são iguais

```
Pessoa alice, beto;
alice = new Pessoa ();
beto = alice;
```

Objetos e referências

- Referências são, por default, nulas — `null`
  - Não referenciam nenhum objeto
- Atribuição: iguala referências
- Teste de igualdade: testa se referências são iguais

```
Pessoa alice, beto;
alice = new Pessoa ();
beto = alice;
beto.idade = 35;
System.out.println(alice.idade)
```

Objetos e referências

- Referências são, por default, nulas — `null`
  - Não referenciam nenhum objeto
- Atribuição: iguala referências
- Teste de igualdade: testa se referências são iguais

```
Pessoa alice, beto;
alice = new Pessoa ();
beto = alice;
beto.idade = 35;
System.out.println(alice.idade)
```

Por que referências?

- Conceitualmente consistente
  - Todos os trechos de código que manipulam um objeto devem "enxergar" o mesmo objeto
  - Mudanças de estado são percebidas em todo o código, e não só em uma das "cópias"

  Nota: isto significa que verificações de estado são extremamente importantes (programação defensiva)
- Menor custo de processamento
  - Alocar objetos complexos em memória pode ser custoso, só deve ser feito se necessário
  - Evita "cópias" em passagem de parâmetros

Passagem de parâmetros

- Em Java, em toda chamada de função (método), os argumentos são recebidos por valor
  - A função possui sua própria "cópia" para trabalhar, com o mesmo valor do parâmetro passado
- Mas... isso tem implicações complicadas
- Tipos primitivos e classes funcionam diferente
  - Para tipos primitivos: o valor é a própria variável
  - Para classes: o valor é a referência à variável

Passagem de parâmetros

Exemplos:

```
// função de teste
public static void sobrescreve( int i ) {
    i = 10;
}

// na função main(...)
int meuNumero = 3;
sobrescreve( meuNumero );
System.out.println( meuNumero );
```

Passagem de parâmetros

Exemplos:

```
// função de teste
public static void sobrescreve( Pessoa vitima ) {
    vitima.idade = 10;
}

// na função main(...)
Pessoa minhaPessoa = new Pessoa();
minhaPessoa.idade = 3;
sobrescreve( minhaPessoa );
System.out.println( minhaPessoa.idade );
```

Passagem de parâmetros

Exemplos:

```
// função de teste
public static void sobrescreve( Pessoa vitima ) {
    vitima = new Pessoa();
    vitima.idade = 10;
}

// na função main(...)
Pessoa minhaPessoa = new Pessoa();
minhaPessoa.idade = 3;
sobrescreve( minhaPessoa );
System.out.println( minhaPessoa.idade );
```

Autoreferência

- Todo objeto possui uma referência especial: `this`
  - Palavra reservada em Java
  - Utilizada dentro de métodos, referência "aponta" ao próprio objeto que detem aquele método
- Uso principal I: passagem de parâmetros

```
public class Mecanico {
    public void checarOleo( Carro carro ) {...}
    public void checarPneus( Carro carro ) {...}
}
public class Carro {
    public Mecanico caraDeConfianca;
    public void fazerCheckUp() {
        caraDeConfianca.checarOleo( this );
        caraDeConfianca.checarPneus( this );
    }
}
```

Autoreferência

- Todo objeto possui uma referência especial: `this`
  - Palavra reservada em Java
  - Utilizada dentro de métodos, referência "aponta" ao próprio objeto que detem aquele método
- Uso principal 2: recuperar nome oculto

```
public class Pessoa {
    public String nome;
    public void setNome(String nome) {
        this.nome = nome;
    }
}
```

Como se instancia objetos?

- O que acontece por trás do operador `new`?
  - Chamada a um método especial: o construtor

```
class Pessoa {
    int idade;
    int altura;
    Pessoa () {

    }
}
Pessoa alice = new Pessoa ();
```

Como se instancia objetos?

- O que acontece por trás do operador `new`?
  - Chamada a um método especial: o construtor

```
class Pessoa {
    int idade;
    int altura;
    Pessoa () {

    }
}
Pessoa alice = new Pessoa ();
```

- Possui o mesmo nome da classe
- Não possui tipo de retorno
- Executada apenas na criação do objeto (não pode ser chamada explicitamente)

Como se instancia objetos?

- O que acontece por trás do operador `new`?
  - Chamada a um método especial: o construtor

```
class Pessoa {
    int idade;
    int altura;
    Pessoa () {

    }
}
Pessoa alice = new Pessoa ();
```

- Toda classe possui um construtor "padrão" oculto
  - Não recebe parâmetros, não executa comandos
- Construtores (com ou sem parâmetros) devem ser implementados para estruturar o código

Como se instancia objetos?

- O que acontece por trás do operador `new`?
  - Chamada a um método especial: o construtor

```
class Pessoa {
    int idade;
    int altura;
    Pessoa () {
        idade = 18;
    }
}
Pessoa alice = new Pessoa ();
```

- Toda classe possui um construtor "padrão" oculto
  - Não recebe parâmetros, não executa comandos
- Construtores (com ou sem parâmetros) devem ser implementados para estruturar o código

Como se instancia objetos?

- O que acontece por trás do operador `new`?
  - Chamada a um método especial: o construtor

```
class Pessoa {
    int idade;
    int altura;
    Pessoa (int id) {
        idade = id;
    }
}
int minhaIdade = 18;
Pessoa alice = new Pessoa (minhaIdade);
```

- Toda classe possui um construtor "padrão" oculto
  - Não recebe parâmetros, não executa comandos
- Construtores (com ou sem parâmetros) devem ser implementados para estruturar o código

Como se instancia objetos?

- Um construtor pode "pedir ajuda" a um outro construtor (encadeamento/chaining)
  - Palavra chave: `this`
  - Só é permitido fazer encadeamento com um outro construtor, e tem que ser o primeiro comando

```
class Pessoa {
    int idade;
    int altura;
    Pessoa (int id) {
        idade = id;
    }
    Pessoa (int id, int alt) {
        this(id);
        altura = alt;
    }
}
```

Como se instancia objetos?

- E para instanciar um array de objetos? Duas etapas:
  - Instanciar um array de referências
  - Instanciar um objeto em cada posição

Como se instancia objetos?

- E para instanciar um array de objetos? Duas etapas:
  - Instanciar um array de referências

```
Pessoa [] turma = new Pessoa[39];
```

  - Instanciar um objeto em cada posição

Como se instancia objetos?

- E para instanciar um array de objetos? Duas etapas:
  - Instanciar um array de referências

```
Pessoa [] turma = new Pessoa[39];
```

  - Instanciar um objeto em cada posição

```
for( int i = 0; i < 39; i++ )
    turma[i] = new Pessoa();
```

Como se instancia objetos?

- E para instanciar um array de objetos? Duas etapas:
  - Instanciar um array de referências

```
Pessoa [] turma = new Pessoa[39];
```

  - Instanciar um objeto em cada posição

```
for( Pessoa aluno : turma )
    aluno = new Pessoa();
```

E para destruir objetos?

- Responsabilidade do coletor de lixo (Garbage Collector) do Java
  - Verifica quando não existem mais referências a um objeto
  - Chama um método destrutor: `finalize()`
  - Desaloca memória
- Excelente prática de programação: não mexa com isso diretamente
  - Não há garantias de que o destrutor irá ser chamado
  - GC executa em uma thread própria: comportamentos inesperados
  - Se você precisa fazer algo quando o objeto for inutilizado, crie um método extra para ter controle sobre isso

UML

- Unified Modeling Language
  - Descrição e visualização de projetos de sistemas
- Diagrama de classes
  - Descreve a estrutura do sistema (código) em termos de classes, atributos, métodos e relacionamentos
  - Sem detalhes de implementação

Pessoa
- nome : String - idade : Integer
+ Pessoa (String, Integer) + fazerAniversário()

Exercícios

Modifique sua implementação dos exercícios 1–8 da aula 3, implementando um construtor padrão e/ou construtores adicionais

Créditos

- Baseado em slides de Gleison Santos