

POO I

Vinícius Machado

Revisão de Conceitos

Além da revisão de sintaxe em Java, já passamos por alguns conceitos. Dentre eles:

- Classe
 - Atributos
 - Métodos
 - Construtores
- Modificadores de visibilidade
 - `public`
 - `private`
 - `protected`

Classe, Atributo e Método

- Classe
 - Modelo abstrato de algo
- Atributo
 - Características que fazem parte do modelo
- Método
 - Ações / eventos que o modelo tem.

Visibilidade

- `public`
 - Todos têm acesso
- `private`
 - Apenas a classe tem acesso
- `protected`
 - A classe e as classes que herdaram têm acesso
- **NÃO DECLARADO: `public package`**
 - Apenas classes declaradas no mesmo pacote tem acesso
 - Não utilizado na prática

Métodos construtores

- São métodos utilizados no momento da criação do meu objeto, isto é, no momento do “new”
- O método construtor é responsável por retornar uma nova instância do objeto.
- Formato:

```
public NomeClasse( params** ) { ... }
```

- Não tem return.

** os parâmetros são opcionais

Métodos construtores

- Por padrão, o compilador do Java cria um método construtor vazio caso a gente não defina um na classe.
- Caso o meu construtor seja diferente do padrão, isto é, com parâmetros, o construtor vazio fica indisponível.
- Existe um conceito muito utilizado na criação de classes que é o polimorfismo dos métodos, a ideia é que o “mesmo” método pode ser chamado com diferentes tipos de parâmetros.

Get / Set

Get / Set

Visibilidade está associada a segurança e a tentativa de diminuir a quantidade de possíveis futuros bugs, quando o código fonte tem suas partes bem encapsuladas, evita que o desenvolvedor utilize as classes de maneira inadequada.

Na aula passada falamos sobre public e private, eventualmente, utilizaremos classes que são chamadas “Data Classes”. Que fazem uma abstração de um conjunto de dados.

Get / Set

Desta forma, nem todo método GET e SET terá lógica incluída, a função destes métodos será simplesmente dar acesso a determinada informação contida na classe.

Por exemplo a classe Pessoa com os atributos nome e cpf. Quando uma pessoa nasce, deve ser registrada no cartório e a partir daí, existe um nome associado a esse bebê.

Quando essa pessoa cresce, ela precisa fazer um documento, então, ignorando a parte burocrática, alguém dá / coloca um CPF nessa pessoa.

Get / Set

Em POO ...

Existe uma classe de objetos de compartilham das mesmas informações (Pessoa). Cada instância dessa classe de objetos tem seus valores para seus atributos. No método construtor, é passada a informação do atributo nome do objeto que está sendo instanciado. Existe um método para buscar o nome, e outros dois métodos relacionados ao CPF, um para colocar o CPF e outro para buscar a informação.

Get / Set

```
public class Pessoa {  
    private String nome;  
    private long cpf;  
    public Pessoa (String novoNome) { nome = novoNome; }  
    public void setCpf(long novoCpf) { cpf = novoCpf; }  
    public long getCpf() { return cpf; }  
}
```

Get / Set versus dados public

Parece que fornecer as capacidades de set e get é essencialmente o mesmo que tornar public as variáveis de instância da classe. Essa é uma das sutilezas que torna o Java tão desejável para a engenharia de software. Uma variável de instância public pode ser lida ou gravada por qualquer método que tem uma referência que contém a variável. Se uma variável de instância for declarada private, um método get public certamente permitirá que outros métodos a acessem, mas o método get pode controlar como o cliente pode acessá-la.

Get / Set versus dados public

Por exemplo, um método `get` poderia controlar o formato dos dados que ele retorna, e assim proteger o código do cliente na representação dos dados real. Um método `set public` pode, e deve, examinar cuidadosamente tentativas de modificar o valor da variável e lançar uma exceção, se necessário. Por exemplo, tentativas para definir o dia do mês como 37 ou peso de uma pessoa como um valor negativo devem ser rejeitadas. Portanto, embora os métodos `set` e `get` possam fornecer acesso a dados `private`, o acesso é restrito pela implementação dos métodos. Isso ajuda a promover uma boa engenharia de software

Get / Set

```
public class Pessoa {  
    private String nome;  
    private long cpf;  
    public Pessoa (String novoNome) { nome = novoNome; }  
    public void setCpf(long novoCpf) { cpf = novoCpf; }  
    public long getCpf() { return cpf; }  
}
```

this

this

Notaram no exemplo anterior o poder da criatividade? Toda vez que temos que passar um parâmetro para alterar o valor de um atributo de uma classe, precisamos deixar claro na assinatura do método o que é aquele valor. No entanto, se a pessoa desenvolvedora está utilizando a nossa API, basta para ela ler a palavra `cpf` que, associada ao método `set`, isto servirá para alterar a informação. Existe em JAVA (e em outras linguagens também), uma palavra chave para fazer referência a sua própria instância e, com isso, aos seus atributos e métodos.

this

Cada objeto pode acessar uma referência a si próprio com a palavra-chave `this` (às vezes chamada de referência `this`). Quando um método de instância é chamado para um objeto particular, o corpo do método utiliza implicitamente a palavra-chave `this` para referenciar as variáveis de instância do objeto e outros métodos. Isso permite que o código da classe saiba qual objeto deve ser manipulado.

this

Com essa informação, não precisamos ser tão criativos e podemos utilizar apenas `this` para representar quando estamos acessando um atributo da classe ou quando não estamos.

```
public void setCpf(long cpf) {  
    this.cpf = cpf;  
}
```

this

Existe também uma outra funcionalidade para a palavra chave this.
Chamar outro método construtor dentro da própria classe

...

this

Basicamente, a ideia é que muitas vezes podemos fornecer em nossa API, várias formas de se iniciar uma classe, podemos considerar parâmetros default por exemplo.

Para evitar o CTRL+C / CTRL+V de códigos, lembrem-se que um bloco nunca deve ser repetido, se assim for, transforme em um método... Para evitar, podemos criar vários construtores com valores padrão e, estes chamam o construtor que contém a lógica.

this

Exemplo:

```
public class A {  
    private int a, b;  
  
    A(int a) { this(a, 0); }  
    A(int a, int b) { this.a = a; this.b = b; }  
  
}
```

Membros da classe - static

- Uma variável static representa informações por toda a classe, que são compartilhadas entre os objetos da classe.
- Variáveis static têm escopo de classe. Os membros public static de uma classe podem ser acessados por meio de uma referência a qualquer objeto da classe ou qualificando o nome de membro com o nome de classe e um ponto (.). O código cliente só pode acessar os membros da classe static de uma classe private por meio dos métodos da classe.
- Membros da classe static existem assim que a classe é carregada na memória.

Membros da classe - static

- Um método declarado static não pode acessar as variáveis de instância e os métodos de instância da classe, porque um método static pode ser chamado mesmo quando nenhum objeto da classe foi instanciado.
- A referência this não pode ser utilizada em um método static.

Composição

Composição

Uma classe pode ter referências a objetos de outras classes como membros. Isso é chamado composição e, às vezes, é referido como um relacionamento tem um. Por exemplo, um objeto `AlarmClock` precisa saber a data/hora atual e a data/hora em que ele supostamente deve soar o alarme, por isso é razoável incluir duas referências a objetos da classe `Time` em um objeto `AlarmClock`.

Enumerações

- Tipo especial em Java para armazenar grupos de constantes.
- Podem ser associados valores via construtor
- For melhorado itera sobre os valores declarados
- Não são utilizadas para criar classes!

```
enum Level {  
    LOW,  
    MEDIUM,  
    HIGH  
}
```

Enumerações

Todos os tipos enum são tipos por referência. Um tipo enum é declarado com uma declaração enum, que é uma lista separada por vírgulas de constantes enum. A declaração pode incluir opcionalmente outros componentes das classes tradicionais, como construtores, campos e métodos.

- Constantes enum são implicitamente final, porque declaram constantes que não devem ser modificadas.
- Constantes enum são implicitamente static.

Enumerações

- Qualquer tentativa de criar um objeto de um tipo enum com um operador new resulta em um erro de compilação.
- Constantes enum podem ser utilizadas em qualquer lugar em que constantes podem ser usadas, como nos rótulos case das instruções switch e para controlar instruções for aprimoradas.
- Cada constante enum em uma declaração enum é opcionalmente seguida por argumentos que são passados para o construtor enum.
- Para cada enum, o compilador gera um método static chamado values que retorna um array das constantes do enum na ordem em que elas foram declaradas.

Leitura Recomendada

Java Como Programar - Deitel

- Capítulos até o 8º

Online

- https://en.wikipedia.org/wiki/Object-oriented_programming
- <http://www.dca.fee.unicamp.br/cursos/PooJava/classes/conceito.html>
- <https://www.devmedia.com.br/abstracao-encapsulamento-e-heranca-pilares-da-poo-em-java/26366>

Referências

Livro

Java: Como programar.

Use a Cabeça! Java

Contato

vinicius.machado@osorio.ifrs.edu.br

