UNIRIO

Modularização e Encapsulamento

Aula 5

BSI — 2018.2

Jefferson Elbert Simões

CCET/DIA

4 de setembro de 2018

Previously on TP2...

Programação orientada a objetos: objetos, classes, atributos, métodos, ...

- Objetos: Conceitos do mundo real
- Classes: Descrição ("molde") de objetos
- Atributos: Características de objetos
- Métodos: Comportamento de objetos

Modularização

- Característica de um bom software
- Suas funcionalidades (internas ou não) são separadas em módulos autocontidos e parcialmente independentes com propósitos específicos e evidentes
- Classes fornecem uma estrutura para modularizar código
  - Todas as atividades pertinentes a uma determinada classe se tornam métodos dessa classe

Modularização

- Em muitos casos, determinadas informações ou atividades não parecem "pertencer aos objetos"
- Para estes casos, podemos declarar atributos ou métodos como sendo estáticos
  - Palavra-chave: `static`
- Quase equivalentes a variáveis e funções globais
  - Escopo limitado à classe na qual foram declarados

Atributos e métodos estáticos

- O que significa um atributo ser estático?
  - Somente existe "uma cópia" do atributo, comum a todos os objetos da classe

```
class Turma {
    ArrayList<String> alunosInscritos;
        // ex: tp2_2018_1.alunosInscritos
    static int capacidade = 40;
        // sempre: Turma.capacidade
};
```

Atributos e métodos estáticos

- O que significa um método ser estático?
  - Sua chamada e execução não dependem da existência de um objeto

```
class Carro {
    int kmRodados;
    void zeraKm() {
        kmRodados = 0;
    }
    // ex: celtaPretoDoMeuAvo.zeraKm()
    static float converteKmParaMilha (float km) {
        return km/1.6;
    };
    // sempre: Carro.converteKmParaMilha(...)
};
```

  - Implementação não pode utilizar nenhum atributo ou método não-estático

Atributos e métodos estáticos

Regra geral (subjetiva): faz sentido acessar o atributo ou chamar o método, mesmo sem um objeto daquela classe?

- Atributos estáticos — candidatos:
  - Poucos casos, geralmente constantes universais
    - `Math.PI`
    - `Integer.MAX_VALUE`
    - `Fisica.VEL_LUZ`

Atributos e métodos estáticos

Regra geral (subjetiva): faz sentido acessar o atributo ou chamar o método, mesmo sem um objeto daquela classe?

- Métodos estáticos — candidatos:
  - Funções em classes utilitárias (cálculos, conversão de medidas)

```
class Carro {
    static float converteKmParaMilha( float km ) {
        ...
    };
}
Math.toRadians( 180 );
```

Atributos e métodos estáticos

Regra geral (subjetiva): faz sentido acessar o atributo ou chamar o método, mesmo sem um objeto daquela classe?

- Métodos estáticos — candidatos:
  - Operações que utilizem múltiplos objetos "equivalentes"

```
class Racional {
    Racional soma( Racional num1,
                   Racional num2 ) {...};
}
```

Atributos e métodos estáticos

Regra geral (subjetiva): faz sentido acessar o atributo ou chamar o método, mesmo sem um objeto daquela classe?

- Métodos estáticos — candidatos:
  - Métodos que não acessam atributos (ou outros métodos) não-estáticos
    - É provável que o objetivo desses métodos não dependa de nenhum objeto em particular

Fundamentos de OO

Quatro conceitos fundamentais:

- Abstração de dados
  - Construção de uma visão de mundo representativa que ignore detalhes desnecessários
- Encapsulamento (aula de hoje)
- Herança
- Polimorfismo

Encapsulamento

Termo ambíguo: refere-se a dois conceitos de linguagens de programação (às vezes ao mesmo tempo)

Agrupamento entre dados e funções que operem nestes dados

Ocultamento de informações irrelevantes

Encapsulamento

Termo ambíguo: refere-se a dois conceitos de linguagens de programação (às vezes ao mesmo tempo)

Agrupamento entre dados e funções que operem nestes dados (atributos e métodos)

Ocultamento de informações irrelevantes (?)

Encapsulamento

Termo ambíguo: refere-se a dois conceitos de linguagens de programação (às vezes ao mesmo tempo)

Agrupamento entre dados e funções que operem nestes dados

Ocultamento de informações relevantes

Encapsulamento é a prática de ocultar a estrutura interna de componentes do mundo com o qual eles interagem

- Eu não preciso saber como você funciona, se você me disser como interagir com você

Encapsulamento

Benefícios de encapsulamento:

- Modularização
  - Código que mexe com estruturas internas dos componentes fica isolado do código que utiliza estes componentes
- Flexibilidade
  - Implementação dos componentes pode ser modificada de forma arbitrária, desde que a interface com os demais componentes seja mantida

Encapsulamento

Duas analogias:

1. Envio de mensagens
   - Componentes são independentes (um não mexe no outro)
   - Interação é feita apenas através de envio de mensagens
2. Oferta de serviço
   - Componente oferece determinados serviços
   - Interação é feita através de pedidos destes serviços

Encapsulamento em Java

- Em Java, objetos interagem chamando métodos uns dos outros...
  - "envio de mensagem"
  - "pedido de serviço"
- ...e operando em seus atributos
  - Hmm... Isto pode trazer problemas...

Encapsulamento em Java

- Problema I: objetos em estado inconsistente

```
class Racional {
    int numerador;
    int denominador;
    Racional (int num, int denom);
    Racional incremento (Racional inc) {...};
}
Racional x = new Racional(3, 5);
Racional y = new Racional(2, 5);
x.denominador = 0;
x.incremento(y);
```

Encapsulamento em Java

- Problema 2: mudança de implementação

```
class Racional {
    int numer;
    int denomin;
    Racional (int num, int denom);
    Racional incremento (Racional inc);
}
Racional x = new Racional(3, 5) {...};
Racional y = new Racional(2, 5) {...};
x.denominador = 0;
x.incremento(y);
```

Encapsulamento em Java

Lições:

- Implemente bem suas classes
  - Métodos devem sempre manter objetos em um estado consistente
- Projete bem suas classes
  - Deve ser simples descobrir como interagir com seus objetos
- Não deixe margem para erro
  - Se existir uma forma errada de interagir com seus objetos, alguém vai descobrir isso

Visibilidade

Em Java, podemos escolher a visibilidade de atributos e métodos de classes. Opções válidas:

`public` Qualquer um pode acessar

`protected` (veremos na aula sobre herança)

- Somente classes do mesmo módulo (package) podem acessar

`private` Somente a própria classe pode acessar

- Para atributos: acessar = ler e modificar valor
- Para métodos: acessar = realizar chamadas

Visibilidade

```
class Racional {
    public int numer;
    public int denomin;
    Racional (int num, int denom);
    public Racional incremento (Racional inc) ...;
}
// em outro ponto do código...
Racional x = new Racional(3, 5);
Racional y = new Racional(2, 5);
x.denomin = 0;
x.incremento(y);
```

O que acontece?

Visibilidade

```
class Racional {
    private int numer;
    private int denomin;
    Racional (int num, int denom);
    public Racional incremento (Racional inc) ...;
}
// em outro ponto do código...
Racional x = new Racional(3, 5);
Racional y = new Racional(2, 5);
x.denomin = 0;
x.incremento(y);
```

O que acontece?

Visibilidade

UML: descrição mais geral de visibilidade

- Indicada por caracteres antes do nome do atributo/método
  - + Público
  - # Protegido (herança)
  - - Privado

Aluno
- nome : String - idade : Integer
+ fazerAniversário()

Visibilidade

Em Java, podemos também escolher a visibilidade das próprias classes. Opções válidas:

`public` Qualquer um pode acessar e instanciar

- Somente outras classes do mesmo módulo (package) podem acessar

- Classes públicas devem ser declaradas em um arquivo `.java` próprio, com o mesmo nome da classe

Visibilidade

Recomendação geral da filosofia OO: deixe todos os atributos privados e todos os métodos públicos

- Maximiza controle, minimiza bugs
- Recomendação pessoal: torne privados tantos métodos quanto for possível
  - Sempre pense se realmente vale a pena tornar um método público
  - Documente seu código quando tomar decisões difíceis e quando a escolha entre público e privado não for óbvia
- "setters"/"getters": métodos especiais para manipular atributos privados
  - Cuidado: frequentemente não vale a pena criar sets padrão

Exercícios

- Modifique os exercícios da aula 3 para incorporar restrições de visibilidade. Cabe a você decidir quais atributos e métodos serão públicos e quais serão privados, e por quê.
- Implemente os seguintes métodos nas classes que você criou nos exercícios da aula 3:

Racional	soma e produto de dois números racionais
Vetor em $\mathbb{R}^3$	soma, subtração, produto interno e produto vetorial

Sobre vetores em $\mathbb{R}^3$:

Produto interno de dois vetores (x, y, z) e (x', y', z') é um número inteiro:

$$(x, y, z) \cdot (x', y', z') = \sqrt{xx' + yy' + zz'}$$

Produto vetorial de dois vetores (x, y, z) e (x', y', z') é um novo vetor em $\mathbb{R}^3$:

$$(x, y, z) \times (x', y', z') = (yz' - zy', zx' - xz', xy' - yx')$$

Créditos

- Baseado em slides de Gleison Santos