UNIRIO

Final / Classes de exceção

Aula 13

BSI — 2018.2

Jefferson Elbert Simões

CCET/DIA

2 de outubro de 2018

Previously on TP2...

- Fundamentos: classes, objetos, atributos, métodos
- Referências
- Construtores e destrutores
- Métodos e atributos estáticos
- Encapsulamento e visibilidade
- Tipos de enumeração
- Exceções (parte I)
- Agregação
- Herança
- Polimorfismo: sobrecarga, sobreposição
- Classes e métodos abstratos
- Interfaces

Princípio do privilégio mínimo

> "Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job."
>
> (Jerome Saltzer, 1974)

Fornecer o mínimo de recursos necessários para que uma tarefa seja completada é uma boa prática de projeto de sistemas

- Evita que as coisas saiam do controle, seja por erros de boa fé (bugs) ou má fé (ataques)

Princípio do privilégio mínimo

Já vimos um exemplo deste princípio em ação no Java:

- Visibilidade de métodos e atributos
  - Utilizado para limitar interfaces para interação de uma classe com outras
  - Evita que classes sejam manipuladas de maneira arbitrária e fiquem em estado inconsistente

Outro recurso de Java que segue esta filosofia: a palavra-chave `final`

- Significa que "algo" não pode ser modificado

Atributos finais

- Atributos finais não podem ser modificados
  - Variáveis e objetos só podem ser inicializados, na descrição da classe ou nos construtores

```
public class Pessoa {
    public String nome;
    public final int anoNasc;
    Pessoa( String nome, int anoNasc ) {
        this.nome = nome;
        this.anoNasc = anoNasc;
    }
}
Pessoa alguem = new Pessoa( "Zé da Silva", 1945 );
alguem.nome = "José da Silva";
alguem.anoNasc = 1950; // erro de compilação
```

Atributos finais

- Para atributos que são tipos primitivos (`int`, `float`, ...), o valor nunca pode ser modificado
- Para atributos objetos, a referência nunca pode ser modificada, mas o objeto em si pode

```
public class Pessoa {
    public String nome;
    public final Pessoa pai;
    Pessoa( String nome, Pessoa pai ) {
        this.nome = nome;
        this.pai = pai;
    }
}
Pessoa ze = new Pessoa ("Zé das Couves", null );
Pessoa ze_filho = new Pessoa ("Zé das Couves Filho", ze);
ze_filho.pai.nome = "José das Couves";
System.out.println( ze.nome );
Pessoa ze_fake = new Pessoa ("Zé das Couves Fake", null );
ze_filho.pai = ze_fake;
```

Atributos finais

- Para atributos que são tipos primitivos (`int`, `float`, ...), o valor nunca pode ser modificado
- Para atributos objetos, a referência nunca pode ser modificada, mas o objeto em si pode

```
public class Pessoa {
    public String nome;
    public final Pessoa pai;
    Pessoa( String nome, Pessoa pai ) {
        this.nome = nome;
        this.pai = pai;
    }
}
Pessoa ze = new Pessoa ("Zé das Couves", null );
Pessoa ze_filho = new Pessoa ("Zé das Couves Filho", ze);
ze.nome = "José das Couves";
System.out.println( ze_filho.pai.nome );
Pessoa ze_fake = new Pessoa ("Zé das Couves Fake", null );
ze_filho.pai = ze_fake;
```

Métodos finais

- Métodos finais não podem ser sobrepostos
  - Implementação de um método final em uma classe tem que ser herdada por todas as subclasses

```
public class Pessoa {
    String nome;
    public final void dizerNome() {
        System.out.println( "Meu nome é " + nome );
    }
}
public class Aluno extends Pessoa {
    public void dizerNome() {
        System.out.println( "Meu nome é " + nome
                            + " e eu sou aluno");
    }
}
```

Classes finais

- Classes finais não podem ser estendidas
  - Nenhuma classe herda de uma classe final
  - Utilizada por exemplo em classes importantes do Java, como `System` ou `String`
    - Segurança
    - Eficiência

Exceções

- Uma exceção é uma representação de um problema que ocorreu durante a execução de um comando
  -
- Exceções são uma forma mais sistemática e menos intrusiva de lidar com comportamentos imprevistos do código
  - Tratamento de exceção: fluxo de código paralelo, "descendo" na pilha de execução
  - Blocos `try/catch/finally`, comando `throw`

Exceções

- Uma exceção é uma representação de um problema que ocorreu durante a execução de um comando
  - Objeto da classe `Exception` ou alguma subclasse
- Exceções são uma forma mais sistemática e menos intrusiva de lidar com comportamentos imprevistos do código
  - Tratamento de exceção: fluxo de código paralelo, "descendo" na pilha de execução
  - Blocos `try/catch/finally`, comando `throw`

Exceções

Exemplo:

```
public class Pessoa {
    private String nome;
    private Pessoa pai;
    public String getNome() { return nome; }
    public String getNomeDoPai() {

        return pai.getNome(); // e se pai = null?

    }
}


Pessoa ze = new Pessoa ("Zé das Couves", null );
System.out.println( ze.getNomeDoPai() );
```

Exceções

Exemplo:

```
public class Pessoa {
    private String nome;
    private Pessoa pai;
    public String getNome() { return nome; }
    public String getNomeDoPai() {
        if( pai != null )
            return pai.getNome();
        else return ""; // bom retorno?
    }
}

Pessoa ze = new Pessoa ("Zé das Couves", null );
System.out.println( ze.getNomeDoPai() );
```

Exceções

Exemplo:

```
public class Pessoa {
    private String nome;
    private Pessoa pai;
    public String getNome() { return nome; }
    public String getNomeDoPai() {

        return pai.getNome();

    }
}

try {
    Pessoa ze = new Pessoa ("Zé das Couves", null );
    System.out.println( ze.getNomeDoPai() );
} catch (NullPointerException e) {
    System.out.println( "sem pai" );
}
```

Exceções

Exemplo:

```
public class Pessoa {
    private String nome;
    private Pessoa pai;
    public String getNome() { return nome; }
    public String getNomeDoPai() {

        return pai.getNome();

    }
}

try {
    Pessoa ze = new Pessoa ("Zé das Couves", null );
    System.out.println( ze.getNomeDoPai() );
} catch (NullPointerException e) { // todo NullPointer é pai nulo?
    System.out.println( "sem pai" );
}
```

Exceções

Em Java existem dois tipos de exceções:

não-verificada Utilizadas devido a erros de programação; podem acontecer em quase qualquer lugar

- Principal caso: `RuntimeException` e derivadas (como `NullPointerException`)
- Métodos não são forçados a tratá-las (podem causar interrupção imprevista)

verificada Ressaltam comportamentos pontuais imprevistos

- Métodos que geram a exceção devem possuir uma cláusula `throws`
- Métodos têm duas opções: ou tratam a exceção (em um bloco try-catch) ou deixar passar com uma cláusula `throws`

Exceções

Podemos criar nossas próprias classes de exceção!

- Diferenciação casos excepcionais, possíveis no nosso código, de casos mais gerais
- Tratamento particularizado destes erros

Como fazer isto?

- Basta criar a classe herdando da classe `Exception` ou suas subclasses
  - (para exceções não-verificadas, da classe `RuntimeException`)

Exceções

Exemplo:

```
public class SemPaiException extends Exception {
    SemPaiException( String message ) {
        super(message);
    }
}
public class Pessoa {
    private String nome;
    private Pessoa pai;
    public String getNome() { return nome; }
    public String getNomeDoPai() throws SemPaiException {
        try {
            return pai.getNome();
        } catch ( NullPointerException e ) {
            throw new SemPaiException( e.getMessage() ) ;
        }
    }
}
```

Exceções

Exemplo:

```
try {
    Pessoa ze = new Pessoa ("Zé das Couves", null );
    System.out.println( ze.getNomeDoPai() );
} catch (SemPaiException e) {
    System.out.println( "sem pai" );
}
```

Exercícios

1. Modifique seu exercício 6 da aula 6, criando uma classe para representar as exceções que ocorrem quando se tenta realizar uma divisão pelo número racional 0.