Aula 21: Exceções e controle de erros

Professor(a): Virgínia Fernandes Mota
http://www.dcc.ufmg.br/∼virginiaferm

ALGORITMOS E ESTRUTURAS DE DADOS - SETOR DE INFORMÁTICA

Exceções e controle de erros

- Ao término desta aula, você será capaz de:
  - controlar erros e tomar decisões baseadas nos mesmos;
  - criar novos tipos de erros para melhorar o tratamento deles em sua aplicação ou biblioteca;
  - assegurar que um método funcionou como diz em seu "contrato".

Exceções e controle de erros

- Voltando às Contas que criamos, o que aconteceria ao tentar chamar o método saca com um valor fora do limite?
- O sistema mostraria uma mensagem de erro, mas quem chamou o método saca não saberá que isso aconteceu.
- Como avisar aquele que chamou o método de que ele não conseguiu fazer aquilo que deveria?

Exceções e controle de erros

- Em Java, os métodos dizem qual o **contrato** que eles devem seguir.
- Se, ao tentar sacar, ele não consegue fazer o que deveria, ele precisa, ao menos, avisar ao usuário que o saque não foi feito.

- Uma solução possível:

```
1  boolean saca(double quantidade) {
2    // posso sacar até saldo+limite
3    if (quantidade > this.saldo + this.limite) {
4        System.out.println("Não posso sacar fora do
             limite!");
5        return false;
6      } else {
7        this.saldo = this.saldo - quantidade;
8        return true;
9      }
10   }
```

```
1  Conta minhaConta = new Conta();
2  minhaConta.deposita(100);
3  minhaConta.setLimite(100);
4  if (!minhaConta.saca(1000)) {
5    System.out.println("Não saquei");
6  }
```

- Essa não é a solução mais elegante!
- Mesmo invocando o método e tratando o retorno de maneira correta, o que faríamos se fosse necessário sinalizar quando o usuário passou um valor negativo como quantidade?
- SOLUÇÃO: utilizamos um código diferente em Java para tratar aquilo que chamamos de **exceções**!

Exceções e controle de erros

- Exceção:
  - Uma exceção representa uma situação que normalmente não ocorre e representa algo de estranho ou inesperado no sistema.

Um exemplo

- Antes de resolvermos o nosso problema, vamos ver como a Java Virtual Machine age ao se deparar com situações inesperadas, como divisão por zero ou acesso a um índice de array que não existe.

Um exemplo

```
1  class TesteErro {
2    public static void main(String[] args) {
3      System.out.println("inicio do main");
4      metodo1();
5      System.out.println("fim do main");
6    }
7
8    static void metodo1() {
9      System.out.println("inicio do metodo1");
10     metodo2();
11     System.out.println("fim do metodo1");
12   }
13
14   static void metodo2() {
15     System.out.println("inicio do metodo2");
16     int[] array = new int[10];
17     for (int i = 0; i <= 15; i++) {
18       array[i] = i;
19       System.out.println(i);
20     }
21     System.out.println("fim do metodo2");
22   }
23 }
```

Um exemplo

Rode o código. Qual é a saída? O que isso representa? O que ela indica?

Um exemplo

- Rastro da pilha (stacktrace).
- O sistema de exceções do Java funciona da seguinte maneira: quando uma exceção é lançada (throw), a JVM entra em estado de alerta e vai ver se o método atual toma alguma precaução ao tentar executar esse trecho de código.
- Como o metodo2 não está tratando o problema, a JVM pára a execução dele anormalmente, sem esperar ele terminar, e volta um stackframe pra baixo, onde será feita nova verificação.

Um exemplo

- Adicione um try/catch em volta do for, pegando ArrayIndexOutOfBoundsException. O que o código imprime?

```
1  try {
2    for (int i = 0; i <= 15; i++) {
3      array[i] = i;
4      System.out.println(i);
5    }
6  } catch (ArrayIndexOutOfBoundsException e) {
7    System.out.println("erro: " + e);
8  }
```

Um exemplo

- Em vez de fazer o try em torno do for inteiro, tente apenas com o bloco de dentro do for. Qual é a diferença?

```
1  for (int i = 0; i <= 15; i++) {
2    try {
3      array[i] = i;
4      System.out.println(i);
5    } catch (ArrayIndexOutOfBoundsException e) {
6      System.out.println("erro: " + e);
7    }
8  }
```

Um exemplo

- Retire o try/catch e coloque ele em volta da chamada do metodo2.

```
1  System.out.println("inicio do metodo1");
2  try {
3    metodo2();
4  } catch (ArrayIndexOutOfBoundsException e) {
5    System.out.println("erro: " + e);
6  }
7  System.out.println("fim do metodo1");
```

Um exemplo

- Faça o mesmo, retirando o try/catch novamente e colocando em volta da chamada do metodo1. Rode os códigos, o que acontece?

```
1  System.out.println("inicio do main");
2  try {
3    metodo1();
4  } catch (ArrayIndexOutOfBoundsException e) {
5    System.out.println("Erro : "+e);
6  }
7  System.out.println("fim do main");
```

Um exemplo

- Repare que, a partir do momento que uma exception foi catched (pega, tratada, handled), a execução volta ao normal a partir daquele ponto.

Exceções de Runtime mais comuns

- Que tal tentar dividir um número por zero?
- Ou acessar uma referência nula?
- Um ArrayIndexOutOfBoundsException ou um NullPointerException poderia ser facilmente evitado!

Outro tipo de exceção: Checked Exceptions

- Os exemplos, com ou sem o try/catch, compilam e rodam. Em um, o erro termina o programa e, no outro, é possível tratá-lo.
- Mas não é só esse tipo de exceção que existe em Java $\rightarrow$ Um outro tipo, obriga a quem chama o método ou construtor a tratar essa exceção.
- Chamamos esse tipo de exceção de *checked*, pois o compilador checará se ela está sendo devidamente tratada, diferente das anteriores, conhecidas como *unchecked*.

Outro tipo de exceção: Checked Exceptions

- Um exemplo interessante é o de abrir um arquivo para leitura, onde pode ocorrer o erro do arquivo não existir:

```
1  class Teste {
2    public static void metodo() {
3      new java.io.FileInputStream("arquivo.txt");
4    }
5  }
6  //Não compila e avisa para tratar FileNotFoundException
```

Outro tipo de exceção: Checked Exceptions

- Podemos tratar o problema de duas maneiras.
- Tratá-lo com o try e catch ou delegar ele para quem chamou o nosso método.

```
1  public static void metodo() {
2    try {
3      new java.io.
           FileInputStream("
           arquivo.txt");
4    } catch (java.io.
         FileNotFoundException e)
          {
5      System.out.println("Nao
           foi possível abrir o
           arquivo para leitura")
           ;
6    }
7  }
```

```
1  public static void metodo()
       throws java.io.
       FileNotFoundException {
2    new java.io.FileInputStream(
         "arquivo.txt");
3  }
```

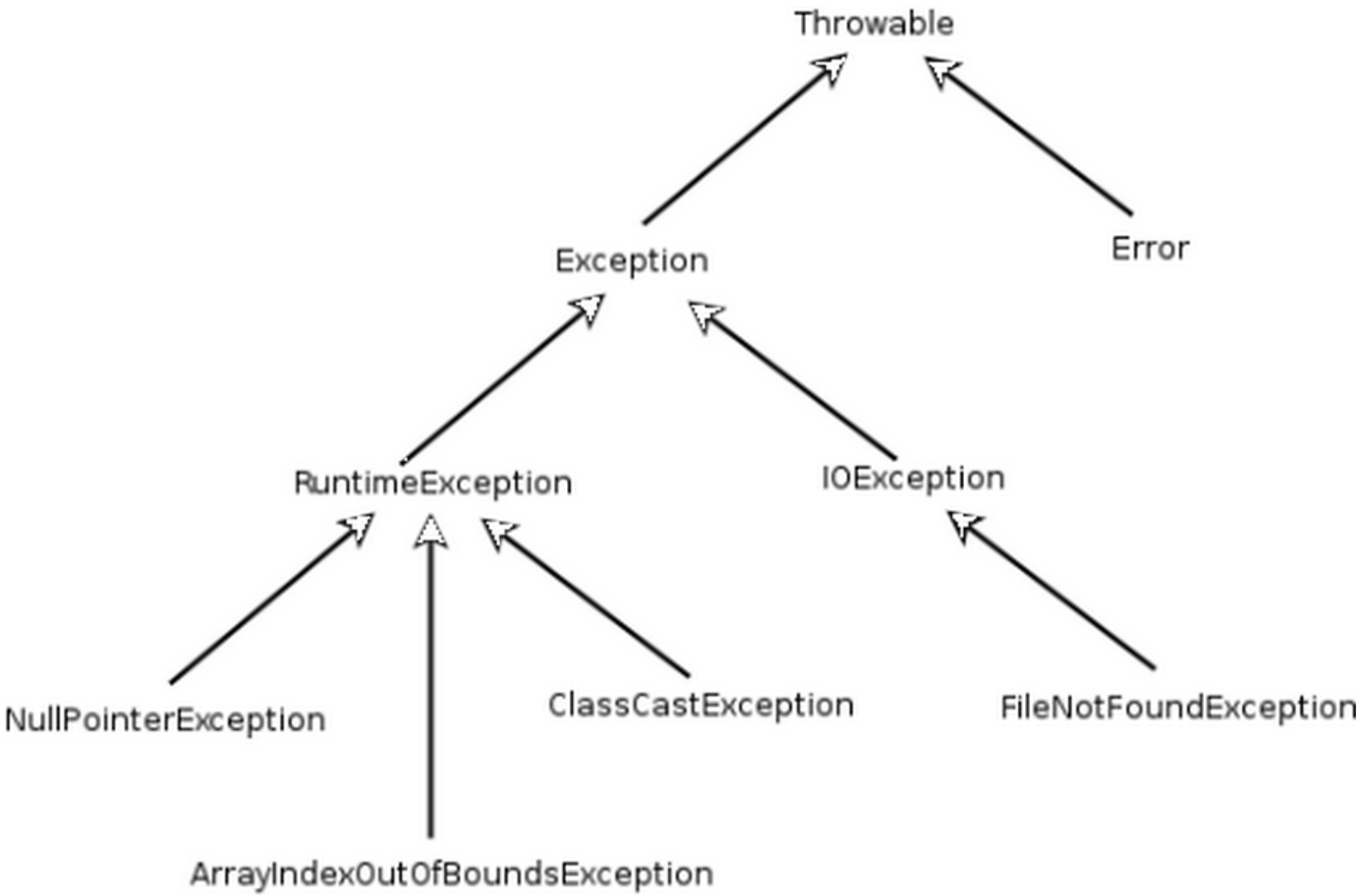
Outro tipo de exceção: Checked Exceptions

- No início, existe uma grande tentação de sempre passar o problema pra frente para outros o tratarem.
- Pode ser que faça sentido, dependendo do caso, mas não até o main, por exemplo.
- Quem tenta abrir um arquivo sabe como lidar com um problema na leitura. Quem chamou um método no começo do programa pode não saber.

Outro tipo de exceção: Checked Exceptions

- Não há uma regra para decidir em que momento do seu programa você vai tratar determinada exceção.
- Isso vai depender de em que ponto você tem condições de tomar uma decisão em relação àquele erro.

Família Throwable



Mais de um erro

- É possível tratar mais de um erro quase que ao mesmo tempo.
- Com o try/catch:

```
1  try {
2    objeto.metodoQuePodeLancarIOeSQLException();
3  } catch (IOException e) {
4    // ..
5  } catch (SQLException e) {
6    // ..
7  }
```

- Com o throws:

```
1  public void abre(String arquivo) throws IOException,
       SQLException {
2    // ..
3  }
```

Mais de um erro

- Você pode, também, escolher tratar algumas exceções e declarar as outras no throws:

```
1  public void abre(String arquivo) throws IOException {
2    try {
3      objeto.metodoQuePodeLancarIOeSQLException();
4    } catch (SQLException e) {
5      // ..
6    }
7  }
```

- É desnecessário declarar no throws as exceptions que são unchecked, porém é permitido e às vezes, facilita a leitura e a documentação do seu código.

Lançando exceções

- Lembre-se do método saca da nossa classe Conta. Ele devolve um boolean caso consiga ou não sacar:

```
1  boolean saca(double valor) {
2    if (this.saldo < valor) {
3      return false;
4    } else {
5      this.saldo-=valor;
6      return true;
7    }
8  }
```

- Podemos, também, lançar uma Exception, o que é extremamente útil. Dessa maneira, resolvemos o problema de alguém poder esquecer de fazer um if no retorno de um método.

Lançando exceções

- A palavra chave **throw** lança uma Exception (bem diferente de throws!! Throws apenas avisa da possibilidade daquele método lançá-la, obrigando o outro método que vá utilizar deste de se preocupar com essa exceção em questão.)

```
1  void saca(double valor) {
2    if (this.saldo < valor) {
3      throw new RuntimeException();
4    } else {
5      this.saldo-=valor;
6    }
7  }
```

Lançando exceções

- No nosso caso, lança uma do tipo unchecked. RuntimeException é a exception mãe de todas as exceptions unchecked.
- Desvantagem: Muito genérica!

Lançando exceções

- Podemos então usar uma Exception mais específica:

```
1  void saca(double valor) {
2    if (this.saldo < valor) {
3      throw new IllegalArgumentException();
4    } else {
5      this.saldo-=valor;
6    }
7  }
```

- IllegalArgumentException diz um pouco mais: algo foi passado como argumento e seu método não gostou.
- IllegalArgumentException é a melhor escolha quando um argumento sempre é inválido como, por exemplo, números negativos, referências nulas, etc

Lançando exceções

- Para pegar esse erro, não usaremos um if/else e sim um try/catch, porque faz mais sentido já que a falta de saldo é uma exceção:

```
1  Conta cc = new ContaCorrente();
2  cc.deposita(100);
3
4  try {
5    cc.saca(100);
6  } catch (IllegalArgumentException e) {
7    System.out.println("Saldo Insuficiente");
8  }
```

- Podemos melhorar??

Lançando exceções

- Passar para o construtor da IllegalArgumentException o motivo da exceção:

```
1  void saca(double valor) {
2    if (this.saldo < valor) {
3      throw new IllegalArgumentException("Saldo
           insuficiente");
4    } else {
5      this.saldo-=valor;
6    }
7  }
```

Lançando exceções

- O método getMessage() definido na classe Throwable (mãe de todos os tipos de erros e exceptions) vai retornar a mensagem que passamos ao construtor da IllegalArgumentException.

```
1  try {
2    cc.saca(100);
3  } catch (IllegalArgumentException e) {
4    System.out.println(e.getMessage());
5  }
```

O que colocar dentro do try?

- Imagine que vamos sacar dinheiro de diversas contas:

```
1  Conta cc = new ContaCorrente();
2  cc.deposita(100);
3
4  Conta cp = new ContaPoupanca();
5  cp.deposita(100);
6
7  // sacando das contas:
8
9  cc.saca(50);
10 System.out.println("consegui sacar da corrente!");
11
12 cp.saca(50);
13 System.out.println("consegui sacar da poupança!");
```

- Podemos escolher vários lugares para colocar try/catch!

O que colocar dentro do try?

- O que você vai colocar dentro do try influencia muito a execução do programa!
- Pense direito nas linhas que dependem uma da outra para a execução correta da sua lógica de negócios.

Criando seu próprio tipo de exceção

- É bem comum criar uma própria classe de exceção para controlar melhor o uso de suas exceções.
- Dessa maneira, podemos passar valores específicos para ela carregar, que sejam úteis de alguma forma.
- Vamos criar a nossa!

Criando seu próprio tipo de exceção

- Voltamos para o exemplo das Contas, vamos criar a nossa Exceção de SaldoInsuficienteException:

```
1  public class SaldoInsuficienteException extends
       RuntimeException {
2    SaldoInsuficienteException(String message) {
3      super(message);
4    }
5  }
```

Criando seu próprio tipo de exceção

- Em vez de lançar um IllegalArgumentException, vamos lançar nossa própria exception, com uma mensagem que dirá "Saldo Insuficiente":

```
1  void saca(double valor) {
2    if (this.saldo < valor) {
3      throw new SaldoInsuficienteException("Saldo
           Insuficiente," +
4                            "tente um valor menor");
5    } else {
6      this.saldo-=valor;
7    }
8  }
```

Criando seu próprio tipo de exceção

- E, para testar, crie uma classe que deposite um valor e tente sacar um valor maior:

```
1  public static void main(String[] args) {
2    Conta cc = new ContaCorrente();
3    cc.deposita(10);
4
5    try {
6      cc.saca(100);
7    } catch (SaldoInsuficienteException e) {
8      System.out.println(e.getMessage());
9    }
10 }
```

Criando seu próprio tipo de exceção

- Podemos transformar essa Exception de unchecked para checked, obrigando a quem chama esse método a dar try-catch, ou throws:

```
1  public class SaldoInsuficienteException extends
       Exception {
2
3    SaldoInsuficienteException(String message) {
4      super(message);
5    }
6  }
```

Finally

- Os blocos try e catch podem conter uma terceira cláusula chamada finally que indica o que deve ser feito após o término do bloco try ou de um catch qualquer.
- É interessante colocar algo que seja imprescindível de ser executado, caso o que você queria fazer tenha dado certo, ou não.
- Caso comum: Liberar um recurso (arquivo, conexão...).

Finally

```
1  try {
2    // bloco try
3  } catch (IOException ex) {
4    // bloco catch 1
5  } catch (SQLException sqlex) {
6    // bloco catch 2
7  } finally {
8    // bloco que será sempre executado, independente
9    // se houve ou não exception e se ela foi tratada ou não
10 }
```

Exercícios

1. Na classe Conta, modifique o método deposita(double x): Ele deve lançar uma exception chamada IllegalArgumentException.
2. Crie uma classe TestaDeposita com o método main. Crie uma ContaPoupanca e tente depositar valores inválidos.
3. Adicione o try/catch para tratar o erro.
4. Ao lançar a IllegalArgumentException, passe via construtor uma mensagem a ser exibida. Lembre que a String recebida como parâmetro é acessível depois via o método getMessage() herdado por todas as Exceptions.

Exercícios

5. Crie sua própria Exception, ValorInvalidoException. Para isso, você precisa criar uma classe com esse nome que seja filha de RuntimeException.
6. Coloque um construtor na classe ValorInvalidoException que receba valor inválido que ele tentou passar.
7. Declare a classe ValorInvalidoException como filha direta de Exception em vez de RuntimeException. Ela passa a ser checked. O que isso resulta?

Mais Exercícios

- Vamos alterar nosso sistema! Crie as seguintes classes:
  - Cliente: nome, conta corrente, conta poupança, segura de vida.
  - Banco: nome, clientes.
  - TestaBanco: Para fazer os testes.
- Cuidado: Não apague os códigos antigos!