UNIP
UNIVERSIDADE PAULISTA

Linguagem de Programação Orientada a Objeto

Exceções

Professora Sheila Cáceres

Exemplo de exceção não tratada

```
import java.util.Scanner;
public class DivideByZeroNoExceptionHandling {

   public static int quotient( int numerator, int denominator )   {
      return numerator / denominator; // possible division by zero
   }

   public static void main( String args[] ) {
      Scanner scanner = new Scanner( System.in );
      System.out.print( "Please enter an integer numerator: " );
      int numerator = scanner.nextInt();
      System.out.print( "Please enter an integer denominator: " );
      int denominator = scanner.nextInt();

      int result = quotient( numerator, denominator );
      System.out.printf("\nResult: %d / %d = %d\n", numerator,
        denominator, result );
   }
}
```

```
Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
```

```
Please enter an integer numerator: 100
Please enter an integer denominator: 0
Exception in thread "main" java.lang.ArithmeticException:
/ by zero
at DivideByZeroNoExceptionHandling.quotient(
DivideByZeroNoExceptionHandling.java:10)
at DivideByZeroNoExceptionHandling.main(
DivideByZeroNoExceptionHandling.java:22)
```

Exceção

Uma **exceção** é um evento que ocorre durante a execução de um programa que interrompe o fluxo normal de instruções.

Definição da Sun

Ex:

- falha ao ler um disco rígido;
- retirar um elemento de uma coleção vazia;
- falha na conexão de rede;
- etc

Tratamento de Exceções

Na programação tradicional, a detecção, documentação e tratamento de erros geralmente leva ao confuso código espaguete.
Por exemplo, considere a ação que lê um arquivo:

```
lerArquivo() {
        abrir o arquivo;
        determinar seu tamanho;
        alocar esta quantidade de memória;
        transferir o arquivo para a memória;
        fechar o arquivo;
}
```

Tratamento de Exceções

Em uma primeira olhada, esta função parece bastante simples, mas ela ignora vários erros em potencial:

- que acontece se o arquivo não puder ser aberto?
- que acontece se o tamanho do arquivo não puder ser determinado?
- que acontece se não puder ser alocada memória suficiente?
- que acontece se a leitura falhar?
- que acontece se o arquivo não puder ser fechado?

Para responder estas questões quanto ao procedimento para “lerArquivo”, faz-se necessário adicionar uma porção de código para fazer a detecção do erro, documentação e tratamento:

Tratamento de Exceções

```
lerArquivo() {
    inicializa codigoErro = 0;
    abrir o arquivo
    if (oArquivoEstaAberto) {
        determinar o tamanho do arquivo;
        if (conseguimosTamanhoArquivo) {
            alocar esta quantidade de memória;
            if (conseguimosAlocarEstaQuantidade) {
                transferir o arquivo para a a memória;
                if (leituraFalhou) {
                    codigoErro = -1; }
            } else {
                codigoErro = -2; }
        } else {
                codigoErro = -3; }
        fechar arquivo;
        if (ArquivoNaoFechou && codigoErro == 0) {
            codigoErro = -4;
        } else {
            codigoErro = codigoErro e -4; }
    } else {
        codigoErro = -5; }
    return codigoErro;
}
```

Tratamento de Exceções

Separando o código normal do código de tratamento de erros

As exceções permitem que você escreva o fluxo principal de seu código e trate dos casos excepcionais em outro lugar. Veja como fica a função lerArquivo usando exceções:

Tratamento de Exceções

```
lerArquivo() {
    try {
          abrir o arquivo;
          determinar seu tamanho;
          alocar aquela quantidade de memória;
          transferir o arquivo para a memória
          fechar o arquivo;
    } catch (aberturaArquivoFalhou) {
        fazer algo aqui;
    } catch (determinarTamanhoFalhou) {
        fazer algo aqui;
    } catch (alocacaoDeMemóriaFalhou) {
        fazer algo aqui;
    } catch (leituraFalhou) {
        fazer algo aqui;
    } catch (fechamentoFalhou) {
        fazer algo aqui;
    }
}
```

Captura de excepções
Try - Catch

Captura de excepções

- Para capturar uma exceção deve-se envolver o bloco que de código que deseja ser verificado com um bloco ***try.***
- *N*o bloco **catch**, se trata a exceção capturada. Para isso, deve-se declarar como parâmetro o tipo de excepção que se deseja capturar. Podem ser declarados vários blocos catch contendo diferentes tipos de exceções
- O bloco finally é executado independentemente de uma exceção ter acontecido ou não.

```
try {
        //Proteja uma ou mais instruções aqui.
} catch (exceptiontype name) {
        //  Informe da exceção e recuperação aqui.
} finally {
        statement(s)
}
```

O bloco try

1. Exceção lançada a partir daqui.

```
try{
     addressbook.saveToFile(filename);
     tryAgain = false;
}
catch(IOException e) {
     System.out.println("Unable to save to " + filename);
     tryAgain = true;
}
```

2. Controle transferido para cá.

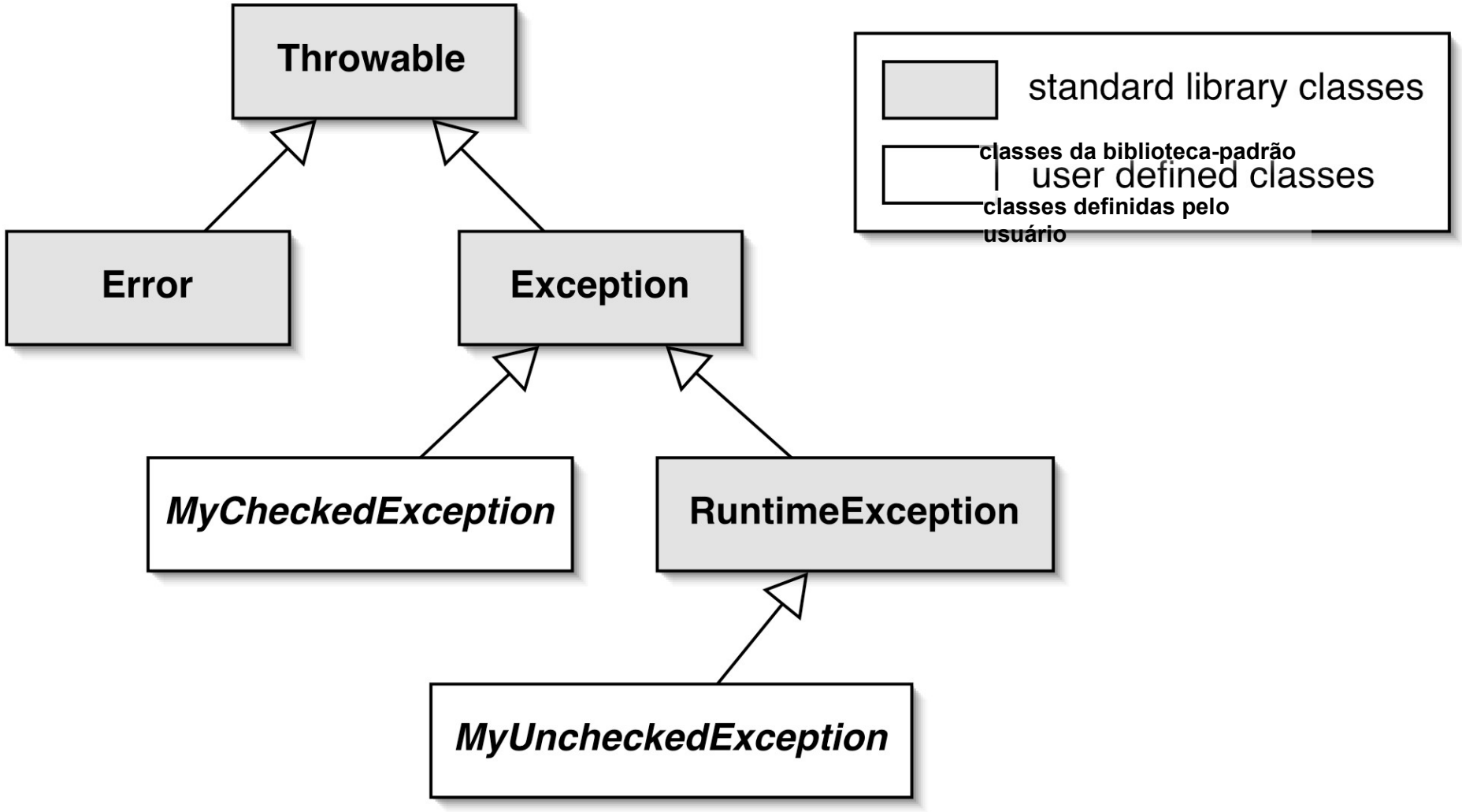
Exemplo

```
public class ExcepTest{
   public static void main(String args[]) {
      try {
         int a[] = new int[2];
         System.out.println("Access element three :" + a[3]);
      } catch (ArrayIndexOutOfBoundsException e) {
         System.out.println("Exception thrown  :" + e);
      }
      System.out.println("Out of the block");
   }
}
```

```
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
Out of the block
```

Hierarquicas de classes
e
Checked vs. Unckecked

A hierarquia de classes de exceção



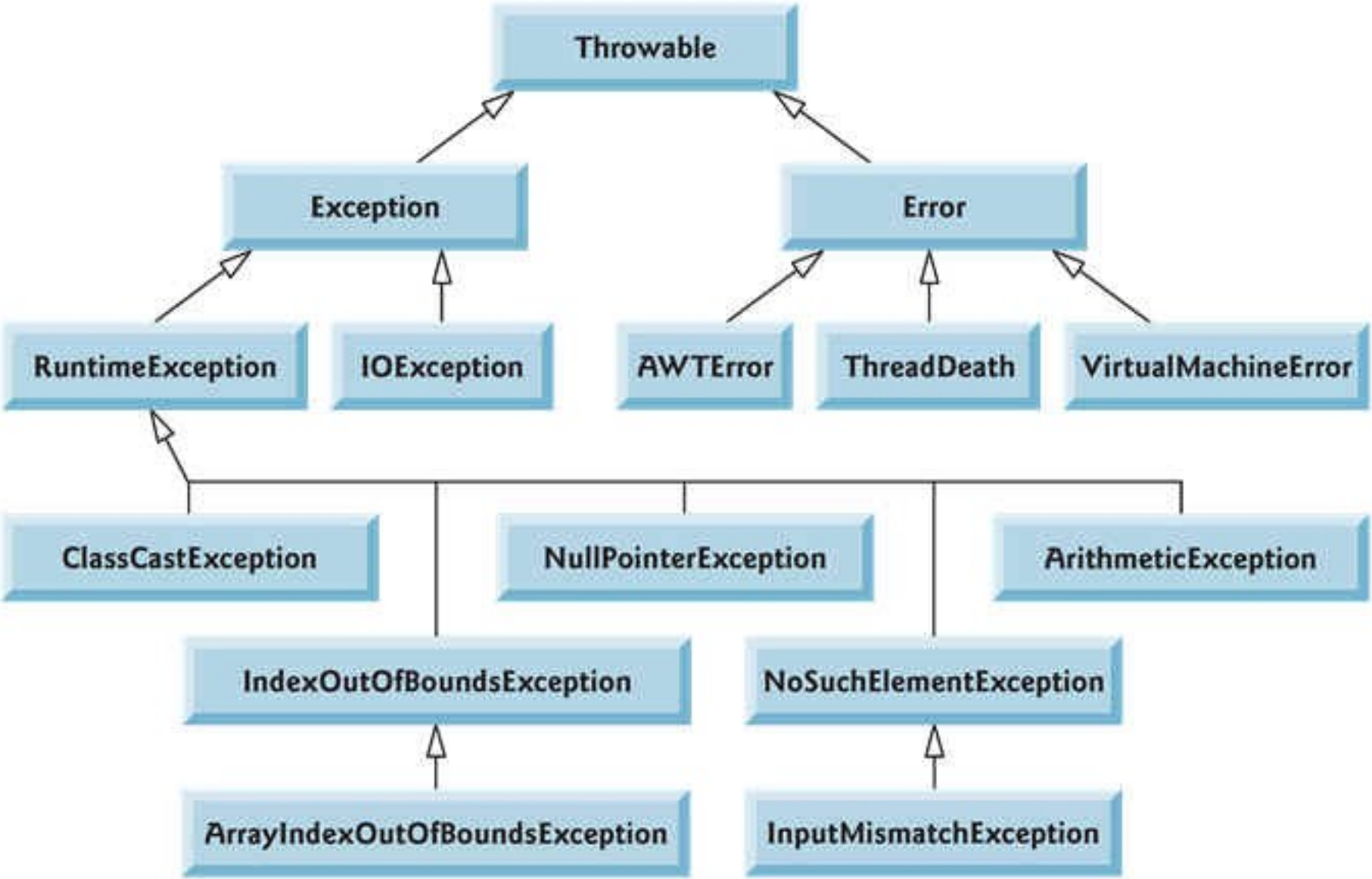


Fig. 11.3 | Portion of class `Throwable`'s inheritance hierarchy.

Source: Deitel & Deitel

Categorias de exceção

- Exceções verificadas (**checked**):
  - subclasse de **`Exception`**;
  - Representa errores que não estão no controle do programa, não podem ser previstos pelo programador, ex: um arquivo é aberto mas não existe.
  - É sempre verificada pelo compilador para garantir que seja tratada quando recebida e declarada pelos métodos que a lançam.
  - **Atenção:** Um código com um método que lance uma ***checked exception*** sem capturá-la em algum local **não compila!!**

- Exceções não-verificadas (**unchecked**):
  - subclasse de **`RuntimeException`**;
  - Podem ser lançadas sem necessidade de ser especificadas, não dará erro se não forem tratadas.
  - Utilizadas para falhas não-antecipadas, erros de programação.
  - O compilador java não da erro se não forem declaradas ou tratadas, porém o programa pode terminar durante a execução se acharmos uma exceção desse tipo.

Exemplo de unchecked exception

```
public class UncheckedException {
   public static void main(String args[]) {
      int a[] = new int[2];
      System.out.println("Access element three :" + a[3]);
   }
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
   at exceptions.UncheckedException.main(UncheckedException.java:7)
```

Exemplo de Checked exception

```
1  package exceptions;
2  import java.io.*;
3
4  public class Copy {
5     public static void main(String[] args)  {
6        File inputFile = new File("Copy.txt");
7        File outputFile = new File("Output.txt");
8        FileReader in = new FileReader(inputFile);
9        FileWriter out = new FileWriter(outputFile);
10       int c;
11       while ((c = in.read()) != -1)
12           out.write(c);
13
14       in.close();
15       out.close();
16    }
17 }
```

Compilador acusa erro.

Exemplo de Checked exception

```
public class Copiador {
    public static void main(String[] args)  {
        File inputFile = new File("Copy.txt");
        File outputFile = new File("Output.txt");
        try{
        FileReader in = new FileReader(inputFile);
        FileWriter out = new FileWriter(outputFile);
        int c;

        while ((c = in.read()) != -1)
            out.write(c);

        in.close();
        out.close();
        System.out.println("Cópia realizada com sucesso.");
        }catch(IOException ioo){System.out.println(
        "Erro encontrado");}
        System.out.println("Continuando fluxo do programa");
    }
}
```

Compilador pode continuar o fluxo do programa mesmo se não achar o arquivo

Erro encontrado
Continuando fluxo do programa

Lançando exceções

Lançando excepções

- Um método que queira lançar uma exceção deverá ter duas coisas a mais.
  1. Declarar no seu cabeçalho que pode lançar uma exceção (`throws`).
  2. Ao detectar um erro, lançar a exceção (`throw`).

Observação:

Um método que chama um outro método que pode lançar uma exceção PRECISA declarar no cabeçalho a possibilidade do lançamento (`throws`), apesar de não ter o `throw` no seu corpo.:

Lançando excepções

1. A cláusula throws

- Métodos que lançam uma exceção verificada devem incluir uma cláusula throws :

```
public void sacar(double valor) throws Exception { . . . }
```

2. A cláusula throw

- Para lançar uma exceção, usa-se a clausula throw.
- Exemplo:

```
throw new Exception("Exceção lançada");
```

Lançando excepções: Exemplo 1

```
public void sacar(double valor) throws Exception {
      if (valor <= saldo) {
         saldo -= valor;
      } else {
         double quantidade = valor - saldo;
         throw new Exception(quantidade);
      }
   }
```

Lançando exceções: Exemplo 2

```
public class UsingExceptions {
    public static void main(String args[]) {
        try {     method1();
        }
        catch (Exception exception) {
            System.err.println("Exceção tratada " +
                exception.getMessage() );
            exception.printStackTrace();
        }
    }
    public static void method1() throws Exception {
        method2();
    }
    public static void method2() throws Exception {
        method3();
    }
    public static void method3() throws Exception {
        throw new Exception("Exception thrown in method3");
    }
}
```

```
Exceção tratada Exception thrown in method3
java.lang.Exception: Exception thrown in method3
    at exceptions.UsingExceptions.method3(UsingExceptions.java:26)
    at exceptions.UsingExceptions.method2(UsingExceptions.java:22)
    at exceptions.UsingExceptions.method1(UsingExceptions.java:18)
    at exceptions.UsingExceptions.main(UsingExceptions.java:9)
```

Excepções proprias

Criando Exceções Proprias

- Pode-se criar exceções próprias.
- Se desejamos criar uma exceção que seja manipulada (checked) obrigatoriamente devemos herdar da classe Exception.

```
class MyException extends Exception{
}
```

- Se desejamos criar uma exceção unchecked, podemos herdar de RuntimeException.

Exemplo

- Considerando uma conta bancaria, criaremos uma exceção que é lançada quando tentamos sacar mais dinheiro do que possuimos.
- Para isso criaremos as classes:
  - Conta
  - SaldoInsuficienteException: nossa exceção
  - BankDemo: onde testaremos a exceção

Criando a exceção propria

```
public class SaldoInsuficienteException extends Exception {
    private double quantidade;

    public SaldoInsuficienteException(double quantidade) {
        this.quantidade = quantidade;
    }

    public double getAmount() {
        return quantidade;
    }
}
```

```java
public class Conta {
    private double saldo;
    private int numeroConta;

    public Conta(int numeroConta) {
        this.numeroConta = numeroConta;
    }

    public void depositar(double valor) {
        saldo += valor;
    }

    public void sacar(double valor) throws SaldoInsuficienteException {
        if (valor <= saldo) {
            saldo -= valor;
        } else {
            double quantidade = valor - saldo;
            throw new SaldoInsuficienteException(quantidade);
        }
    }

    public double getSaldo() {  return saldo;   }

    public int getNumeroConta() {  return numeroConta;    }
}
```

```
public class BankDemo {
    public static void main(String[] args) {
        Conta c = new Conta(101);
        System.out.println("Depositando $500...");
        c.depositar(500.00);
        try {
            System.out.println("Saque de $100...");
            c.sacar(100.00);
            System.out.println("Saque de $600...");
            c.sacar(600.00);
        } catch (SaldoInsuficienteException e) {
            System.out.println("Desculpe, vc esta com deficit de $"
                + e.getAmount());
            StackTraceElement[] traceElements = e.getStackTrace();
            System.out.println("FileName\t\t\tMethod\tLine number");
            for(StackTraceElement element : traceElements){
                System.out.printf("%s\t",element.getFileName());
                System.out.printf("%s\t",element.getMethodName());
                System.out.printf("%s\n",element.getLineNumber());
            }
        }
        System.out.println("Transações possiveis realizadas com
            sucesso");
    }
}
```

Saída do programa

```
Depositando $500...
Saque de $100...
Saque de $600...
Desculpe, porem voce esta com deficit de $200.0
FileName        Method      Line number
Conta.java      sacar       23
BankDemo.java   main        16
Transações possiveis realizadas com sucesso
```

Bibliografia

- Programação orientada a objetos com Java.
  Autores David J. Barnes e Michael Kolling.
- Java: Como programar.
  Autores: H. M. Deitel e P. J. Deitel
  Editora: Pearson – 9a Edição
- Slides do POO, Depto. de Informática e Estatística,UFSC.
- Java exception Handling:
  http://www.tutorialspoint.com/java/java_exceptions.htm
  Acessado o 4 de maio do 2014.
- Nota: Vários dos Exemplos, Figuras e Definições desta apresentação foram extraídos das fontes aqui apresentadas.