Programação Orientada a Objetos

Exceções

Ely – ely.miranda@ifpi.edu.br

Erros

- Os softwares devem estar preparados tratar e se recuperar de erros:
  - Erros devem ser tratados sejam eles quaisquer
- Tratamento de erros leva à robustez:
  - Capacidade de um software executar suas funções mesmo em condições anormais;
  - O software deve prever formas de execução não esperada e deve ser capaz de resistir e se recuperar de falhas;
  - Robustez leva à confiabilidade.

Alternativas a tratamento de erros

```
public class Conta {
   private String    numero;
   private double    saldo;

   public void sacar(double valor) {
       saldo = saldo - valor;
   }
   ...
}
```

Como evitar que o saldo fique negativo?

Alternativa 1: desconsiderar a operação

```
public class Conta {
   private String    numero;
   private double    saldo;

   public void sacar(double valor) {
     if (saldo > valor) {
       saldo = saldo - valor;
     }
   }
   ...
}
```

Alternativa 1: desconsiderar a operação

- Problema:
  - Ficamos na incerteza do sucesso ou não da operação.

```
// ...
Conta minhaConta = new Conta();
minhaConta.depositar(100);
minhaConta.sacar(1000);
// A chamada ao método "sacar" funcionou?
```


Alternativa 2: exibir uma mensagem de erro

```
public class Conta {
   private String    numero;
   private double   saldo;

   public void sacar(double valor) {
       if (saldo > valor)
         saldo = saldo - valor;
       else
         System.out.print("Saldo Insuficiente!");
   }...
}
```

Alternativa 2: exibir uma mensagem de erro

- Problema: ficamos atrelados à interface (gráfica ou texto;
- E se estivéssemos usando interface gráfica?
  - A mensagem não seria notada, pois não se iria olhar o prompt.

Alternativa 3: Retornar um código de erro

```
public class Conta { ...
  public boolean sacar(double valor) {
        if (saldo > valor) {
           saldo = saldo - valor;
           return true;
        }
        else
          return false;
    }...
}
```

Alternativa 3: Retornar um código de erro

```
public class Conta { ...
   public boolean transferir(Conta  conta, double v) {
      boolean sacou = conta.sacar(v);
      if (sacou) {
         this.depositar(v);
         return true;
      }else
         return false;
   } ...
}
```


Alternativa 3: Retornar um código de erro

```
   class Banco { ...
      public int sacar(String num, double v) {
         Conta c = this.consultar(num);
         if (c != null) {
            boolean sacou = c.sacar(v);
            if (sacou)
                  return 1;
            else
                  return 2;
         } else
               return 3;
      }
   }
```

Alternativa 3: Retornar um código de erro

- Problemas:
  - Temos que testar o valor de retorno para saber o que houve;
  - Quando o método já retorna valores, temos que reservar alguns para representar erros: “valores menores que 0 para erros”;
  - Isto é considerado uma má prática:
    - Uso de flags ou números mágicos;
    - Pode levar à mudança do tipo de retorno do método:
      - Um método com retorno boolean, pode ter que retornar inteiros: 1 (true) - 0 (false) e -1 (erro);
    - Se o método já retorna um número calculado, é difícil reservar códigos de erro que não se confundam um possível valor calculado;
  - O valor retornado não é intuitivo e carece de documentação para o entendimento.

Exceções

- Erros em Java são sinalizados por Exceções:
  - Classes especiais que contém informações sobre o tipo de erro gerado;
  - Causam a interrupção da execução do código se não tratados:
    - A partir da exceção, nada mais é executado, a menos que o erro seja tratado;
    - Com isso, um erro não causa implicações mais graves.
- Toda exceção deve ter uma mensagem associada que facilite a interpretação do erro;
- Exemplos de exceções:
  - ArrayIndexOutOfBoundsException;
  - NullPointerException;
  - Etc;

Exceções não tratadas

```
public static void main(String[] args) {
   int[] numeros = new int[5];

   for (int i = 0; i < 10; i++) {
      numeros[i] = i;
   }
   System.out.println("Nunca chegará aqui");
 }
```

ArrayIndexOutOfBoundsException

Exceções não tratadas

```
public static void main(String[] args) {
   Conta c = null;
   System.out.println("Erro:" +
                      c.getNumero());
 }
```

NullPointerException

Lançamento de exceções

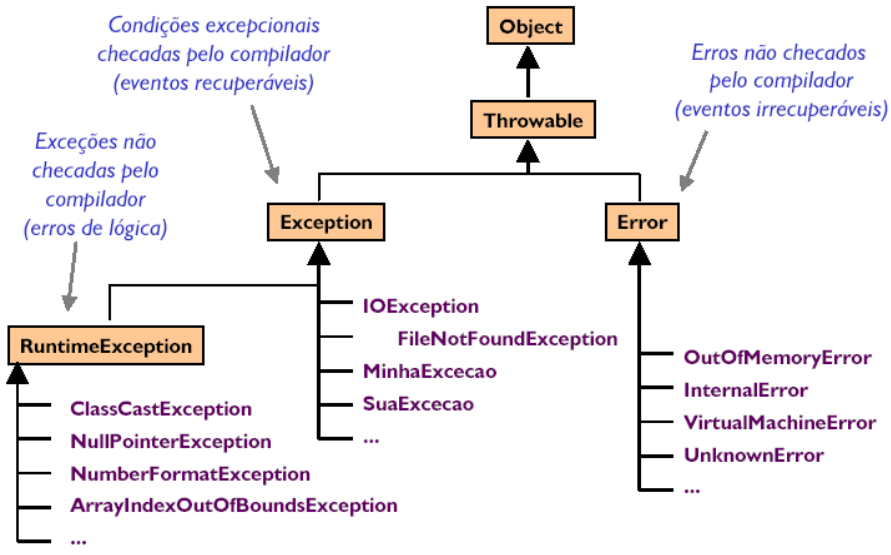
- Também podemos lançar exceções quando encontrarmos situações de erros;
- Lançar uma exceção é instanciar uma classe específica:
  - usa-se a palavra reservada **throw** ;
  - passa-se uma mensagem de erro;
- Podemos lançar exceções já existentes no java ou criar nossas próprias exceções.

Lançamento de exceção pré-existente

```
public class Conta {
    private double saldo;
    //...
    public void sacar(double valor){
        if(saldo < valor){
            throw new RuntimeException("Saldo insuficiente");
        }
        saldo-=valor;
    }
    //...
}
```

- O débito não é realizado caso caia na exceção;
- O código de **else** não é necessário, pois a exceção interrompe o código a partir da sua instanciação.

Hierarquia



Estendendo classes de exceções

```
//AplicacaoException.java
public class AplicacaoException extends RuntimeException {
    public AplicacaoException(String message) {
       super(message);
    }
}

//SaldoInsuficienteException.java
public class SaldoInsuficienteException extends AplicacaoException {
    public SaldoInsuficienteException(String message) {
       super(message);
    }
}

//ContaNaoCadastradaException .java
public class ContaNaoCadastradaException extends RuntimeException {
    public ContaNaoCadastradaException (String message) {
       super(message);
    }
}
```

Lançamento de exceção personalizada

```
public class Conta {
    private double saldo;
    //...
    public void sacar(double valor){
        if(saldo < valor){
            throw new AplicacaoException("Saldo insuficiente");
        }
        saldo-=valor;
    }
    //...
}
```

- O débito não é realizado caso caia na exceção;
- O código de **else** não é necessário, pois a exceção interrompe o código a partir da sua instanciação.

Capturando/tratando a exceção

```
// ...
Conta c = new Conta();
c.creditar(100);
try{
   c.sacar(100);
   c.sacar(1000);
   c.creditar(10); //não é executada
}catch(AplicacaoException e){
  System.out.println(e.getMessage());
}
// ...
```

Capturando/tratando a exceção

```
// ...
Conta c = new Conta();
c.creditar(100);
try{
   c.sacar(100);
   c.sacar(1000);
   c.creditar(10); //não é executada
}catch(SaldoInsuficienteException e){
  System.out.println(e.getMessage());
}
// ...
```


Exceções checadas e não checadas

- Não checadas:
  - Descendem de RuntimeException;
  - Exceções não checadas/verificadas (unchecked) em tempo de compilação;
  - Suas descendentes também são unchecked;
  - **Devem ser usadas preferencialmente.**
- Checadas:
  - Descendem de Exception;
  - Exceções checadas/verificadas em tempo de compilação;
  - Compilador exige que sejam ou tratadas ou declaradas (propagadas) pelo método que potencialmente as provoca;
  - Devem ser declaradas em cada método que não as tratar (próximo slide).

Exceções checadas

- Obrigam a quem chamar o método tratar a exceção;
- Palavra reservada **throws** indica que pode ser lançada no corpo do método;
- Deve-se colocar na assinatura do método, do contrário, ocorre um erro de compilação;

```
public class Caldeira {

    private int temperatura;

    public int getTemperatura() {
        return temperatura;
    }
    public void setTemperatura(int temperatura) throws Exception {
        if (temperatura > 100){
            throw new Exception("Não é permitido alterar a " +
                    "temperatura com valores acima de 100°C");
        }else{
           this.temperatura = temperatura;
        }
    }
}
```

23

Blocos try/catch/finally

- Trecho de código usado para tratar exceções;
- O bloco try "tenta" executar um bloco de código que pode levantar exceção;
- Deve ser seguido por:
  - Um ou mais blocos catch;
  - E/ou um bloco finally
- Um try não pode aparecer sozinho:
  - Deve ter pelo menos um catch ou por um finally.

24

Blocos Catch

- Blocos catch:
  - recebem um tipo de exceção como argumento:
  - Se ocorrer uma exceção no try, ela irá descer pelos **catch** até encontrar um que declare exceções de uma classe ou superclasse da exceção levantada;
  - Apenas um dos blocos **catch** é executado.
- Bloco finally:
  - Contém instruções que devem ser executadas independentemente da ocorrência ou não de exceções;
  - Exemplos disso são códigos de "limpeza" como fechamento de arquivos, liberação de recursos, etc.

try/catch/finally

```
...
try {
   // executa até linha onde ocorrer exceção
} catch (TipoExcecao1 ex) {
   // executa somente se ocorrer TipoExcecao1
} catch (TipoExcecao2 ex) {
   // executa somente se ocorrer TipoExcecao2
} finally {
   // executa sempre, mesmo sem exceção ...
}

// a partir daqui, executa se exceção for capturada ou
  se não ocorrer
...
```

Capturar e tratar as exceções

```
...
try {
   for(int i = 0; i < 10; i++) {
      numeros[i] = i;
      System.out.println(i);
   }
} catch (ArrayIndexOutOfBoundsException e) {
   System.out.println("Erro interno, contate
     o administrador do sistema");
}
...
```

Pode-se omitir detalhes "técnicos" no tratamento de exceções

Exemplo

```
...
public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        do {
           try {
                System.out.println("Digite o número menor que 5:");

                int num = sc.nextInt();
                int[] numeros = {1,2,3,4,5};
                for (int i = 0; i < num; i++) {
                    System.out.print(numeros[i] + " ");
                }
            } catch (InputMismatchException e) {
                System.out.println("Numero inválido");
            } catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("Limite do array estourado");
            }
            System.out.println("Digite 0 para sair.");

        } while (!sc.next().equals("0"));
    }
...
```

Hierarquia de exceções

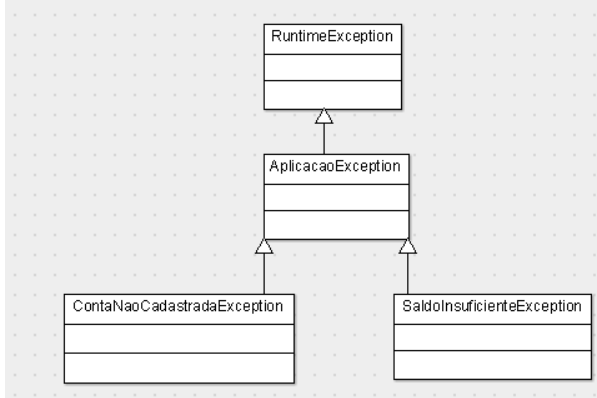
- Se uma classe de exceção mais genérica aparecer antes de uma mais específica, uma exceção do tipo da específica jamais será capturada.

```
...
try {
 ...
} catch (RuntimeException e) {
   System.out.println("Qualquer exceção vai cair aqui: " +
  e.getMessage());
} catch (InputMismatchException e) {
   System.out.println("Numero inválido");
} catch (ArrayIndexOutOfBoundsException e) {
   System.out.println("Limite do array estourado");
}
...
```


29

Exemplo de Hierarquia de Exceções

- Preferencialmente, as aplicações devem ter uma exceção principal e suas descendentes;
- A exceção principal deve descender de RuntimeException.



30

Tratamento na classe ExecutaBanco

```
public static void main(String[] args) {
    Banco b = new Banco(100);
    do {
        try {
            case "1":
            //opção cadastrar...
            break;
            //demais casos...
            }
        } catch (AplicacaoException e) {
            //exibo a mensagem, afinal foi uma exceção tratada
            System.out.println(e.getMessage());
        } catch (Exception e) {
            //qualquer outra exceção, msg padrão até que seja investigada
            System.out.println("Erro ao realizar operação. Contate o
            responsável pelo sistema.);
        }
    } while (!opcao.equals("9"));
}
```


Considerações sobre exceções

- Sempre que possível, use suas próprias exceções;
- Métodos sobrescritos não podem provocar mais exceções que os métodos originais;
- Nunca escreva um código de tratamento vazio:

```
try {
   // .. código que pode causar exceções
}
catch (MinhaException e) {}
```

- Sempre observe a pilha de erros de uma exceção.

Resumo

- Tratar:

```
try {  ...
} catch (MinhaExcecao ex) {
  // exibir a mensagem da exceção
} catch (Exception ex) {
  //emitir uma mensagem menos técnica
} finally {
  /*opcionalmente escrever um código
  que seja executado sempre (com ou sem
  exceção */
}
```


Resumo

- Lançar:

```
if (<condicao_nao_satisfeita>)
   throw new MinhaExcecao("Erro...");
```

- Criar / definir:

```
public class MinhaExcecao extends
  RuntimeException {...}
```

Programação Orientada a Objetos

Exceções

Ely – ely.miranda@ifpi.edu.br