Programação Orientada a Objetos em

Flávio Miguel Varejão

Departamento de Informática

UFES

Exceções

- Condições Excepcionais
  - Erros
    - divisão por zero
    - acesso indevido a vetor
  - Múltiplos retornos
    - fim de arquivo
    - leitura de valor indevido

Exceções

Quais os problemas com as funções abaixo?

```
int fatorial(int n) {
   if (n<2) return 1;
   return n * fatorial (n - 1);
}
```

```
int fatorial(int n) {
   if (n<0) return -1;
   if (n<2) return 1;
   return n * fatorial (n - 1);
}
```

Linguagens sem Exceções

- Programador deve antecipar exceção
- Sinalização através de:
  - variável global
  - retorno de valor especial
  - parâmetro com código de erro
- Programador pode tratar ou não
  - geralmente esquece ou ignora
- Tratamento é realizado através de teste do sinalizador

Consequências

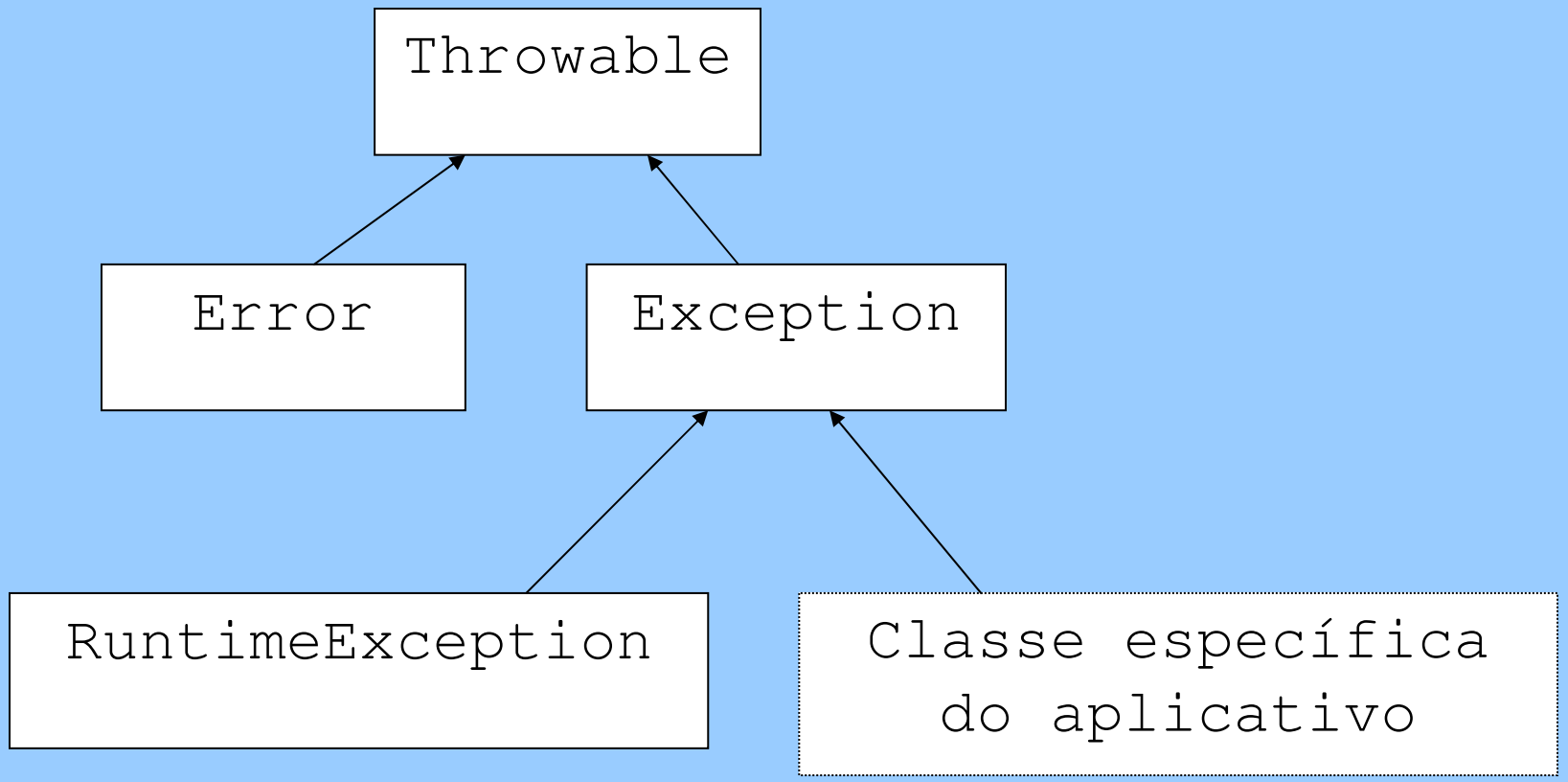
- Código mais obscuro
  - mistura de código que cumpre funcionalidade com código que trata os erros
- Código menos confiável
  - erros que não são tratados

Exceções

Como resolver problema ao chamar `new Impares(0)`?

```
public class Impares {
   private static int j = 1;
   private int i = j;
   private Impares prox;
   Impares(int n) {
       j = j + 2;
       if(--n > 0)
           prox = new Impares(n);
   }
   …
}
```

Tipos de Exceções



Tipos de Exceções

- Métodos de Throwable
  - void printStackTrace()
  - String getMessage()
  - String toString(0
  - outros
- Error - erros sérios (não devem ser tratados)
- Exception (podem ser tratados)
  - RunTimeException (especial)
  - outras (normal)

Blocos try-catch

```
try {
// código que pode lançar uma exceção específica
} catch (ExcecaoA exca) {
// código executado se ExcecaoA é diparada
} catch (ExcecaoB excb) {
// código executado se ExcecaoB é diparada
} ...
} catch (Exception e) {
// código executado se qualquer outra exceção é
// lançada
}
```

Bloco try-catch

```
public class Excecao {
  public static void main(String[] args) {
    String s;
    int a, b;
    try {
       s = Console.readString();
       a = Integer.valueOf(s).intValue();
       s = Console.readString();
       b = Integer.valueOf(s).intValue();
       resultado = a / b;
    } catch (ArithmeticException e){
       System.out.println("Divisao por zero");
    } catch (NumberFormatException e){
       System.out.println ("Erro na Formatacao ");
    } catch(Exception e) {
      System.out.println("Qualquer outra Excecao");
    }
  }
}
```

Lançamento de Exceções

```
// Excecao1.java
public class Excecao1 {
  public static void main(String[] args) {
    try {
       throw new Exception ("Uma primeira excecao");
    } catch(Exception e) {
      System.out.println("Excecao capturada");
    }
  }
}
```

Propagação de Exceções

```
// Excecao2.java
import java.io.*;
import java.util.*;
public class Excecao2 {
  static Random rand = new Random();
  static int pRand(int mod) {
     return Math.abs(rand.nextInt()) % mod + 1;
  }
```

Propagação de Exceções

```
public static void main(String[] args) {
  System.out.println("Primeiro try");
  try {
    System.out.println("Segundo try ");
    try {
      System.out.println("Terceiro try ");
      try {
        switch(pRand(4)) {
          default:
          case 1: throw new NumberFormatException();
          case 2: throw new EOFException();
          case 3: throw new NullPointerException();
          case 4: throw new IOException();
        }
```

Propagação de Exceções

```
            } catch (EOFException e) {
              System.out.println("Trata terceiro try");
            }
          } catch (IOException e) {
             System.out.println("Trata segundo try ");
          }
        } catch (NullPointerException e){
          System.out.println("Trata primeiro try");
        }
    }
}
```

A Cláusula throws

```
// Excecao2a.java
import java.io.*;
import java.util.*;
public class Excecao2a {
  static Random rand = new Random();
  static int pRand(int mod) {
     return Math.abs(rand.nextInt()) % mod + 1;
  }
  public static void main(String[] args)
                  throws IOException{
```

A Cláusula throws

```
System.out.println("Primeiro try");
try {
  System.out.println("Segundo try ");
  try {
    System.out.println("Terceiro try ");
    try {
      switch(pRand(4)) {
        default:
        case 1: throw new NumberFormatException();
        case 2: throw new EOFException();
        case 3: throw new NullPointerException();
        case 4: throw new IOException();
      }
```

A Cláusula throws

```
            } catch (EOFException e) {
              System.out.println("Trata terceiro try");
            }
          } catch (NumberFormatException e) {
             System.out.println("Trata segundo try ");
          }
        } catch (NullPointerException e){
          System.out.println("Trata primeiro try");
        }
    }
}
```

Propagação entre métodos

```
// Excecao2b.java
import java.io.*;
import java.util.*;
public class Excecao2b {
  static Random rand = new Random();
  static int pRand(int mod) {
     return Math.abs(rand.nextInt()) % mod + 1;
  }
```

Propagação entre métodos

```
public static void main(String[] args)
                throws IOException {
  System.out.println("Primeiro try");
  try {
    primeiro();
  } catch (NullPointerException e){
    System.out.println("Trata primeiro try");
  }
}
```

Propagação entre métodos

```
public static void primeiro() throws IOException,
                                     NullPointerException {
  System.out.println("Segundo try ");
  try {
    segundo();
  } catch (NumberFormatException e) {
      System.out.println("Trata segundo try ");
  }
}
```

Propagação entre métodos

```
public static void segundo() throws IOException,
                                    NullPointerException {
  System.out.println("Terceiro try ");
  try {
        switch(pRand(4)) {
          default:
          case 1: throw new NumberFormatException();
          case 2: throw new EOFException();
          case 3: throw new NullPointerException();
          case 4: throw new IOException();
        }
  } catch (EOFException e) {
        System.out.println("Trata terceiro try");
  }
 }
}
```

Relançamento de Exceções

```
// Excecao3.java
import java.io.*
public class Excecao3 {
  public static void main(String[] args) {
    try {
      try {
         throw new IOException();
      } catch (IOException e) {
          System.out.println("Trata primeiro aqui");
          throw e;
      }
    } catch (IOException e) {
         System.out.println("Continua aqui ");
    }
  }
}
```

Criação de Exceções

```
class MinhaExcecao extends Exception {
  public MinhaExcecao() {}
  public MinhaExcecao(String msg) { super(msg); }
  public MinhaExcecao(String msg, int x) {
     super(msg); i = x;
  }
  public int val() { return i; }
  private int i;
}
```

Criação de Exceções

```
public class TestaExcecao {
  public static void f() throws MinhaExcecao {
    System.out.println("Disparou MinhaExcecao");
    throw new MinhaExcecao();
  }
  public static void main(String[] args) {
    try {
      f();
    } catch(MinhaExcecao e) {
      System.out.println(e.val());
      e.printStackTrace();
    }
  }
}
```

A Cláusula finally

```
public class Sempre {
  public static void main(String[] args) {
    System.out.println("Primeiro try");
    try {
      System.out.println("Segundo try");
      try {
        throw new Exception();
      } finally {
        System.out.println("finally do segundo try");
      }
    } catch(Exception e) {
      System.out.println("excecao capturada");
    } finally {
      System.out.println("finally do primeiro try");
    }
  }
}
```

A Cláusula finally

```
public class CarroBomba {
  class SuperAquecimentoException extends Exception {}
  public void ligar() {}
  public void mover()
                  throws SuperAquecimentoException {
    String temperatura = Console.readString();
    if (temperatura.equals("anormal")) {
      throw new SuperAquecimentoException();
    }
  }
  public void desligar() {}
```

A Cláusula finally

```
  public static void main(String[] args) {
    CarroBomba c = new CarroBomba();
    try {
      c.ligar();
      c.mover();
    } catch(SuperAquecimentoException e) {
      System.out.println("vai explodir!!!");
    } finally {
      c.desligar();
    }
  }
}
```

Perda de Exceção

```
public class Perda {
  class InfartoException extends Exception {
    public String toString() { return "Urgente!"; }
  }
  void infarto() throws InfartoException {
    throw new InfartoException ();
  }
  class ResfriadoException extends Exception {
    public String toString() { return "Descanse!"; }
  }
  void resfriado() throws ResfriadoException {
    throw new ResfriadoException ();
  }
```

Perda de Exceção

```
  public static void main(String[] args)
                                throws Exception {
    Perda p = new Perda();
    try {
      p.infarto();
    } finally {
      p.resfriado();
    }
  }
}
```

Retomada

```
public class Retomada {
  static class NaoPositivoException
                                  extends Exception {}
  public static void main(String[] args) {
    boolean continua = true;
    while (continua) {
      continua = false;
      try {
        System.out.print (
          "Entre um inteiro positivo: ");
        int i = Console.readInteger();
        if (i <= 0) throw new NaoPositivoException();
      } catch(NaoPositivoException e) {
        System.out.println("Tente novamente!!!");
        continua = true;
      }
    }
  }
}
```

Herança com Exceções

```
class InfracaoTransito extends Exception {}
class ExcessoVelocidade extends InfracaoTransito {}
class AvancarSinal extends InfracaoTransito {}

abstract class Dirigir {
  Dirigir() throws InfracaoTransito {
   // Nao e' obrigado disparar a excecao
  }
  void irTrabalhar () throws InfracaoTransito {}
  abstract void viajar() throws ExcessoVelocidade,
                                AvancarSinal;
  void caminhar() {} // Nao dispara excecao
}
```

Herança com Exceções

```
class Acidente extends Exception {}
class Batida extends Acidente {}
class AltaVelocidade extends ExcessoVelocidade {}

interface Perigo {
  void irTrabalhar () throws Batida;
  void congestionamento() throws Batida;
}

public class DirecaoPerigosa extends Dirigir
                             implements Perigo {
```

Herança com Exceções

```
// novas excecoes podem ser adicionadas ao
// construtor, mas é obrigatório lidar com as
// excecoes do construtor da superclasse
DirecaoPerigosa() throws Batida, InfracaoTransito {}
DirecaoPerigosa (String s) throws ExcessoVelocidade,
                                  InfracaoTransito {}

// metodos devem ser declarados como na superclasse
//! void caminhar() throws AltaVelocidade {}
// erro de compilacao

// interfaces nao podem adicionar excecoes
// aos metodos da superclasse
//! public void irTrabalhar() throws Batida {}
```

Herança com Exceções

```
// adicao pode ser feita se metodo nao existe
// na superclasse
public void congestionamento() throws Batida {}

// Nao e obrigatorio disparar a excecao
// do metodo da superclasse
public void irTrabalhar() {}

// metodos sobrescritos podem disparar excecoes de
// subclasses da classe declarada no metodo
// da superclasse
void viajar() throws AltaVelocidade {}
```

Herança com Exceções

```
public static void main(String[] args) {
  try {
    DirecaoPerigosa dp = new DirecaoPerigosa ();
    dp.viajar ();
  } catch(AltaVelocidade e) {
  } catch(Batida e) {
  } catch(InfracaoTransito e) {}
  // AvancarSinal nao e' disparada em viajar()
  // da subclasse
```

Herança com Exceções

```
    try {
      Dirigir d = new DirecaoPerigosa();
      d.viajar ();
      // No caso de upcast deve-se tratar excecoes
      // da superclasse
    } catch(AvancarSinal e) {
    } catch(ExcessoVelocidade e) {
    } catch(Batida e) {
    } catch(InfracaoTransito e) {}
  }
}
```

Métodos de Console

```
public static int readInteger () {
      try {
         BufferedReader br = new BufferedReader (
                new InputStreamReader (System.in) );
         String s = br.readLine ();
         return Integer.parseInt (s);
      } catch (IOException e) {
         return 0;
      } catch (NumberFormatException e) {
         return 0;
      }
  }
```

Métodos de Console Problema

- Código de retorno é valor válido
  - readInteger(), readByte(), readShort(), readLong(), readFloat(), readDouble():

    retornam 0
  - readBoolean(): retorna false
  - readChar(): retorna '\0'
  - readString(): retorna ""
- Ao construir Console não se sabe como o usuário deseja tratar os problemas

Solução - Uso de Exceções

```
public static int readInteger () throws IOException {
      BufferedReader br = new BufferedReader (
                new InputStreamReader (System.in) );
      String s = br.readLine ();
      return Integer.parseInt (s);
}
```

Solução - Uso de Exceções

```
public class LeInteiro {
  public static void main(String[] args) {
    boolean continua = true;
    while (continua) {
      continua = false;
      try {
        System.out.print ("Entre um inteiro: ");
        int i = Console.readInteger();
      } catch(Exception e) {
        System.out.println("Tente novamente!!!");
        continua = true;
      }
    }
  }
}
```

Exercícios

- Refazer a classe Console propagando exceções
- Refazer os métodos leConsole das classes Data, Pessoa, MembroUniv, Assalariado, etc. usando os métodos da nova classe Console.