Introdução à Programação de Aplicações Científicas em Java

Escola de Verão do Laboratório Associado de Computação e Matemática Aplicada do INPE

Rafael Santos

Sobre o curso

- Dividido em quatro partes:
  1. Tecnologia, Linguagem e Orientação a Objetos.
  2. ***APIs* comuns.**
  3. Programação com Interfaces Gráficas.
  4. *APIs* para Processamento Científico (aplicações diversas).
- Cada parte tem aproximadamente 3 horas.
- O curso é somente teórico, sem laboratório.
  - Exemplos, *links*, etc. serão disponibilizados em http://www.lac.inpe.br/~rafael.santos
- Muitos exemplos formatados como *how-to*.

Exceções

Exceções

- Estritamente falando, não é uma API...
- Mecanismo de processamento de erros potenciais presente em várias linguagens modernas.
- Idéia básica: ao invés de testar erros de métodos/funções, testar blocos de código.
- Em Java: código em bloco try {}, exceções possíveis em blocos catch(Exception e} {}.
  - Adicionalmente um bloco finally {} para processamento independente de exceções.

Exceções

```
package excecoes;
public class Teste
  {
  public static void main(String[] args)
    {
    int[] a = new int[20];
    a[21] = 123;
    String x = "alfabeto";
    char y = x.charAt(10);
    }
  }
```

```
package excecoes;
public class TesteExcecoes
  {
  public static void main(String[] args)
    {
    try
      {
      int[] a = new int[20];
      a[21] = 123;
      String x = "alfabeto";
      char y = x.charAt(10);
      }
    catch(Exception e)
      {
      System.err.println("Exceção:");
      e.printStackTrace();
      }
    }
  }
```

```
Exceção:
java.lang.ArrayIndexOutOfBoundsException: 21
at excecoes.TesteExcecoes.main(TesteExcecoes.java:10)
```

Exceções

```
package excecoes;

public class TesteMaisExcecoes
  {
  public static void main(String[] args)
    {
    try
      {
      int[] a = new int[20];
      a[21] = 123;
      String x = "alfabeto";
      char y = x.charAt(10);
      }
    catch(ArrayIndexOutOfBoundsException e)
      {
      System.err.println("Exceção em um array:");
      e.printStackTrace();
      }
    catch(StringIndexOutOfBoundsException e)
      {
      System.err.println("Exceção em uma String:");
      e.printStackTrace();
      }
    }
  }
```

Exceções

- Algumas exceções forçam a inclusão de blocos catch, outras não.
- Múltiplos blocos catch possíveis: hierarquia de exceções.
- Delegação de exceções: métodos podem ser declarados como throws Exception, processamento com catch fica a cargo do método que o chamou.
- Tudo isso ficará mais claro com exemplos reais.

- Problema: ordem de try/catch/finally/return é confusa!

Arrays

Arrays

- Forma de representar várias instâncias ou variáveis com um nome e índice.
- Estritamente falando, também não é uma API...
  - ...mas existem mecanismos especiais de Java para processamento de arrays, particularmente...
    - Tamanho embutido no *array*
    - Verificação de validade/índice (exceção opcional)
- Não são mutáveis: tamanho deve ser conhecido na criação.
- Não são polimórficos: todos os elementos são da mesma classe/tipo...
  - ...exceto em casos previstos em OO.

Arrays Unidimensionais

- Declaração de arrays unidimensionais:
  tipo[] nome = new tipo[quantidade]
  tipo nome[] = new tipo[quantidade]
- Exemplos (tipos nativos e String):
  double[] dados = new double[1000];
  char[] alfabeto = new char[26];
  int[] vetor = new int[tamanho];
  String[] nomes = new String[35];

Arrays Unidimensionais

```
package arrays;

public class TermosDeSerie
  {
  public static void main(String[] args)
    {
    double[] termos = new double[10000];
    for(int x=0;x<termos.length;x++)
      {
      termos[x] = 1./((x+1)*(x+1));
      }
    }
  }
```

```
package arrays;

public class Alfabeto
  {
  public static void main(String[] args)
    {
    char[] alfabeto =
      new char[]{'a','b','c','d','e','f',
                 'g','h','i','j','k','l',
                 'm','n','o','p','q','r',
                 's','t','u','v','w','x',
                 'y','z'};
    System.out.println(alfabeto[18]); // s
    }
  }
```

Arrays Unidimensionais

```
package arrays;

public class ParametrosCL
  {
  public static void main(String[] args)
    {
    if (args.length == 0) System.exit(0);
    System.out.println("Os parâmetros passados foram:");
    for(int i=0;i<args.length;i++)
      System.out.println(i+":"+args[i]);
    }
  }
```

```
>java arrays.ParametrosCL dunga atchim soneca zangado dengoso feliz mestre
Os parâmetros passados foram:
0:dunga
1:atchim
2:soneca
3:zangado
4:dengoso
5:feliz
6:mestre
```

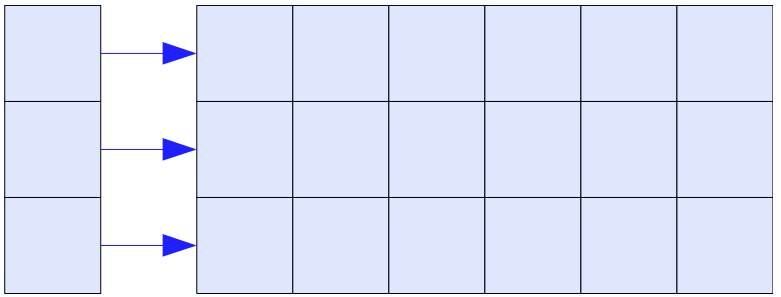
Arrays de Instâncias

- *Arrays* de instâncias são somente conjuntos de referências: instâncias **devem** ser inicializadas individualmente!
  - Exceto para Strings.

```
package arrays;

import java.net.*;

public class URLs
  {
  public static void main(String[] args)
    {
    URL[] endereços = new URL[args.length];
    try
      {
      for(int i=0;i<args.length;i++)
        {
        endereços[i] = new URL(args[i]);
        System.out.println(endereços[i].getProtocol());
        }
      }
    catch (MalformedURLException e) { e.printStackTrace(); }
    }
  }
```

Arrays Multidimensionais

- *Arrays* de *Arrays*



- Declaração de arrays multidimensionais:

```
tipo[][] nome =
  new tipo[quant1][quant2]
tipo[][][][][] nome =
  new tipo[qt1][qt2][qt3][qt4][qt5]
tipo nome[][] =
  new tipo[quant1][quant2]
tipo[] nome[] =
  new tipo[quant1][quant2]
```

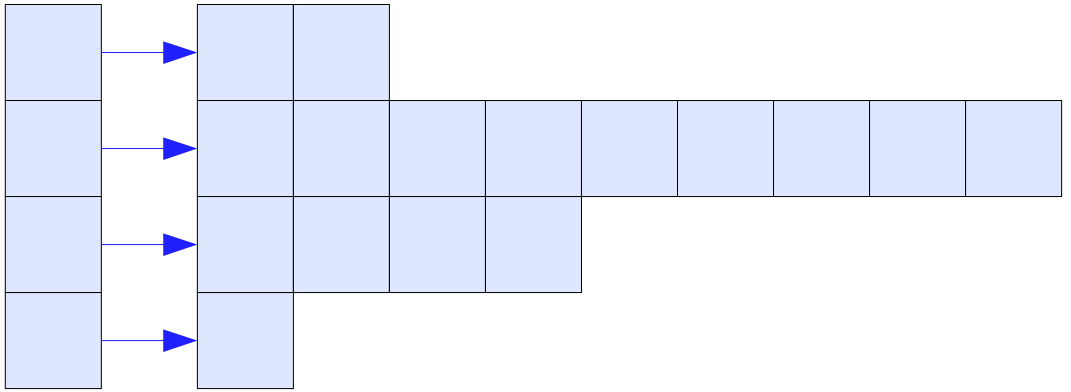
Arrays Multidimensionais

```
package arrays;

public class Multidimensionais
  {
  public static void main(String[] args)
    {
    float[][] matriz = new float[64][64];
    float[][][][][] cincodim;
    cincodim = new float[20][20][20][20][20];
    }
  }
```

```
package arrays;

public class Tabuleiro
  {
  private char[][] tabuleiro;

  public Tabuleiro(int linhas,int colunas)
    {
    tabuleiro = new char[linhas][colunas];
    }

  public int getLinhas() { return tabuleiro.length; }

  public int getColunas() { return tabuleiro[0].length; }
  }
```

Arrays Irregulares

- *Arrays* de *Arrays*



- Declaração igual à de arrays multidimensionais
  - Somente primeira dimensão é declarada.
  - Outras alocadas dinamicamente.

```
tipo[][] nome =
  new tipo[quant1][]
nome[0] = new tipo[comp1]
nome[1] = new tipo[comp2]
nome[2] = new tipo[comp3]
...
```

Arrays Irregulares

```
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
```

```
package arrays;
public class TrianguloDePascal
  {
  public static void main(String[] argumentos)
    {
    int númeroDeLinhas = 10;
    long[][] triânguloDePascal = new long[númeroDeLinhas][];
    // Cada linha é alocada independentemente.
    for(int linha=0;linha<númeroDeLinhas;linha++)
      triânguloDePascal[linha] = new long[2+linha];
    // Primeira linha constante.
    triânguloDePascal[0][0] = 1; triânguloDePascal[0][1] = 1;
    for(int linha=1;linha<númeroDeLinhas;linha++)
      {
      triânguloDePascal[linha][0] = 1;
      for(int coluna=1;coluna<triânguloDePascal[linha].length-1;coluna++)
        triânguloDePascal[linha][coluna] = triânguloDePascal[linha-1][coluna] +
                                           triânguloDePascal[linha-1][coluna-1];
      triânguloDePascal[linha][triânguloDePascal[linha].length-1] = 1;
      }
    for(int linha=0;linha<númeroDeLinhas;linha++)
      {
      for(int coluna=0;coluna<triânguloDePascal[linha].length;coluna++)
        System.out.print(triânguloDePascal[linha][coluna]+" ");
      System.out.println();
      }
    }
  }
```

Strings e *Tokens*

Strings

- Em Java, classe com status especial: não é necessário usar o new para criar novas instâncias.
- Métodos para extrair caracteres e substrings, localizar substrings, comparar de diversas formas, formatar valores, converter *case*, etc.
- Instâncias de String são *imutáveis*: métodos retornam novas instâncias.
- Permite concatenação com operador **+**.

Strings

```
package strings;

public class Strings
  {
  public static void main(String[] args)
    {
    String texto = "Democracia é eu mandar em você. "+
                   "Ditadura é você mandar em mim.";
    System.out.println(texto.length()); // 62
    System.out.println(texto.toLowerCase());
    System.out.println(texto.charAt(1)); // e
    System.out.println(texto.concat(" Millôr Fernandes."));
    System.out.println(texto.endsWith("mim.")); // true
    System.out.println(texto.startsWith("Ditadura")); // false
    System.out.println(texto.indexOf("você")); // 26
    System.out.println(texto.lastIndexOf("você")); // 43
    System.out.println(texto.substring(16,31)); // mandar em você.
    }
  }
```

Tokens

- Substrings, extraídas de uma String e separadas por uma expressão qualquer.
- Muito comuns em arquivos de texto, formato CSV, etc.
- Dois métodos para obtenção de *tokens* a partir de Strings:
  - String.split(), usa expressão regular e retorna *array* de Strings.
  - Classe StringTokenizer, funciona como um *Iterator*.

Tokens

```
package strings;
public class Tokens
  {
  public static void main(String[] args)
    {
    String linha = "operator:x:11:0:operator:/root:/sbin/nologin";
    String[] campos = linha.split(":");
    System.out.println("Usuário:"+campos[0]);
    System.out.println("Nome:"+campos[4]);
    System.out.println("Dir:"+campos[5]);
    System.out.println("Shell:"+campos[6]);
    }
  }
```

```
Usuário:operator
Nome:operator
Dir:/root
Shell:/sbin/nologin
```

```
package strings;
import java.util.StringTokenizer;
public class Tokenizer
  {
  public static void main(String[] args)
    {
    StringTokenizer st =
      new StringTokenizer("operator:x:11:0:operator:/root:/sbin/nologin",":");
    while(st.hasMoreTokens())
      System.out.println(st.nextToken());
    }
  }
```

```
operator
x
11
0
operator
/root
/sbin/nologin
```

Estruturas de Dados

Estruturas de Dados

- Java provê uma API para armazenamento de dados (instâncias de classes) na memória...
  - Simples e flexível.
  - Com diversas modalidades e variantes.
  - Correspondentes às principais estruturas clássicas de dados.
- Coleções: API de estruturas de dados.
- Java 5 provê duas outras características interessantes para manipulação de coleções:
  - Laço for para iteração.
  - *Autoboxing*.
- Classe Collections tem métodos estáticos úteis.

Estruturas de Dados: *Sets*

- Coleção de objetos que não admite objetos em duplicata.
- Interface Set: define contrato.
- Métodos:
  - add: adiciona um objeto ao set.
  - remove: remove um objeto do set.
  - contains: retorna true se o set contém o objeto.
- Classe HashSet: set com boa performance.
- Classe TreeSet: set que garante que elementos estarão em ordem implícita.

Estruturas de Dados: *Sets*

```
package ed;

import java.util.Date;
import java.util.HashSet;

public class ExemploSet1
{
public static void main(String[] args)
  {
  HashSet<Object> set = new HashSet<Object>();
  set.add(new Integer(123));
  set.add(123);
  set.add("ABC");
  set.add("ABC");
  set.add(new Date());
  set.remove("ABC");
  System.out.println(set);
  // [123, Tue Jan 23 08:13:59 BRST 2007]
  }
}
```

Estruturas de Dados: *Sets*

```
package ed;

import java.util.Date;
import java.util.TreeSet;

public class ExemploSet2
{
public static void main(String[] args)
  {
  TreeSet<Object> set = new TreeSet<Object>();
  set.add(new Integer(123));
  set.add(123);
  set.add("ABC"); // !!!!
  set.add("ABC");
  set.add(new Date());
  set.remove("ABC");
  System.out.println(set);
  // Exception in thread "main" java.lang.ClassCastException:
  //    java.lang.Integer
  }
}
```

Estruturas de Dados: *Sets*

```
package ed;

import java.util.Date;
import java.util.HashSet;
import java.util.Iterator;

public class ExemploSet3
  {
  public static void main(String[] args)
    {
    HashSet<Object> set = new HashSet<Object>();
    set.add(new Integer(123)); set.add(123);
    set.add("ABC");            set.add("ABC");
    set.add(new Date());
    Iterator<Object> i = set.iterator();
    while(i.hasNext())
      {
      Object o = i.next();
      if (o instanceof Integer)
        System.out.println("Achei um Integer:"+o);
      }
    // Achei um Integer:123
    }
  }
```

Estruturas de Dados: *Sets*

```
package ed;

import java.util.HashSet;

public class ExemploSet4
  {
  public static void main(String[] args)
    {
    HashSet<Integer> set = new HashSet<Integer>();
    set.add(new Integer(123));
    set.add(111);
    set.add(Integer.parseInt("877"));
    // set.add("877"); Erro de compilação: método não aplicável
    set.add(123);
    int sum = 0;
    System.out.print("Soma de ");
    for (Integer i : set)
      {
      System.out.print(i + " ");
      sum += i;
      }
    System.out.println("é " + sum);
    // Soma de 111 877 123 é 1111
    }
  }
```

Estruturas de Dados: *Sets*

- Mais operações úteis em *Sets*:
  - addAll: adiciona um *set* a outro.
  - retainAll: retém em um *set* tudo o que estiver em outro: interseção de sets.
  - removeAll: remove de um *set* tudo o que estiver em outro.
  - containsAll: retorna true se o *set* conter todos os elementos de outro.
- Estes métodos recebem como argumento uma outra instância de *Set* e retornam uma nova instância resultante da operação.

Estruturas de Dados: *Sets*

```
package ed;
import java.util.HashSet;

public class OperacoesSet
  {
  public static void main(String[] args)
    {
    HashSet<String> solteiros = new HashSet<String>();
    solteiros.add("Tom");
    solteiros.add("Larry");
    HashSet<String> casados = new HashSet<String>();
    casados.add("Nathan");
    casados.add("Jeffrey");
    casados.add("Randal");
    casados.add("Sriram");
    HashSet<String> tenistas = new HashSet<String>();
    tenistas.add("Tom");
    tenistas.add("Jeffrey");
    tenistas.add("Larry");
    HashSet<String> nadadores = new HashSet<String>();
    nadadores.add("Nathan");
    nadadores.add("Sriram");
    nadadores.add("Tom");
    // Todos os autores
    HashSet<String> todos = new HashSet<String>(casados);
    todos.addAll(solteiros);
    // [Nathan, Tom, Jeffrey, Larry, Randal, Sriram]
```

Estruturas de Dados: *Sets*

```
    // Nadadores e tenistas
    HashSet<String> nadadoresETenistas = new HashSet<String>(nadadores);
    nadadoresETenistas.retainAll(tenistas);    // [Tom]

    // Tenistas e casados
    HashSet<String> tenistasCasados = new HashSet<String>(tenistas);
    tenistasCasados.retainAll(casados);
    System.out.println(tenistasCasados);       // [Jeffrey]

    // Tenistas ou casados
    HashSet<String> tenistasOuCasados = new HashSet<String>(tenistas);
    tenistasOuCasados.addAll(casados);
    System.out.println(tenistasOuCasados);
    // [Nathan, Tom, Jeffrey, Larry, Randal, Sriram]

    // Casados mas não atletas
    HashSet<String> casadosMasNãoAtletas = new HashSet<String>(casados);
    casadosMasNãoAtletas.removeAll(tenistas);
    casadosMasNãoAtletas.removeAll(nadadores); // [Randal]

    // Todo nadador é tenista ?
    System.out.println(tenistas.containsAll(nadadores)); // false
    // Todo solteiro é tenista ?
    System.out.println(tenistas.containsAll(solteiros)); // true
    }
}
```

Estruturas de Dados: Listas

- Coleção de objetos em forma de lista, aceita duplicatas.
- Interface List: define contrato.
- Métodos:
  - add: adiciona um objeto à lista.
  - remove: remove um objeto da lista.
  - get: recupera um objeto da lista.
  - contains: retorna true se a lista contém o objeto.
  - indexOf: retorna o índice do objeto na lista ou −1.
  - size: retorna o número de elementos na lista.

Estruturas de Dados: Listas

- Classe LinkedList: performance razoável em todas as condições.
- Classe ArrayList: boa performance, mas pode cair quando tamanho é redimensionado.
- Classe Stack: métodos adicionais para push e pop.

Estruturas de Dados: Listas

```
package ed;

import java.util.ArrayList;
import java.util.Date;

public class ExemploLista1
  {
  public static void main(String[] args)
    {
    ArrayList<Object> lista = new ArrayList<Object>();
    lista.add(new Integer(123));
    lista.add(123);
    lista.add("ABC");
    lista.add("ABC");
    lista.add(new Date());
    lista.remove("ABC");
    System.out.println(lista);
    // [123, 123, ABC, Tue Jan 23 09:11:09 BRST 2007]
    }
  }
```

Estruturas de Dados: Listas

```
package ed;

import java.util.LinkedList;

public class ExemploLista2
  {
  public static void main(String[] args)
    {
    LinkedList<Float> lista = new LinkedList<Float>();
    lista.add(new Float(1.4));
    lista.add(1f);
    lista.add(new Float(2.61));
    float sum = 0;
    System.out.print("Soma de ");
    for(Float f:lista)
      {
      System.out.print(f+" ");
      sum += f;
      }
    System.out.println("é"+sum);
    // Soma de 1.4 1.0 2.61 é 5.01
    }
  }
```

Estruturas de Dados: Listas

```
package ed;

import java.util.ArrayList;
import java.util.Collections;

public class Sorteio
  {
  public static void main(String[] args)
    {
    ArrayList<Integer> números = new ArrayList<Integer>(60);
    for(int i=1;i<60;i++) números.add(i);
    Collections.shuffle(números);
    for(int i=0;i<6;i++) System.out.print(números.get(i)+" ");
    // 7 59 16 51 36 58
    }
  }
```

Estruturas de Dados: Mapas

- Também conhecidos como *arrays* associativos.
- Coleção de objetos como *arrays*, mas índices são objetos.
- Outra interpretação: conjunto de pares (chave,valor) de objetos. Chaves não podem ser duplicadas.
- Interface Map: define contrato.
- Métodos:
  - put: adiciona um objeto ao mapa.
  - remove: remove um objeto do mapa.
  - get: recupera um objeto do mapa.
  - keySet: retorna um set com todas as chaves.
  - values: retorna uma coleção com todos os valores.

Estruturas de Dados: Mapas

- Classe HashMap: boa performance.
- Classe TreeMap: elementos ordenados por chave.

```
package ed;

import java.util.HashMap;

public class ExemploMap1
  {
  public static void main(String[] args)
    {
    HashMap<Object, Object> mapa = new HashMap<Object, Object>();
    mapa.put(1, "um");
    mapa.put(2, "dois");
    mapa.put(3, "quatro");
    mapa.put(3, "três");
    mapa.remove("dois"); // ?
    mapa.remove(2); // ok
    mapa.put(0.0, "zero");
    mapa.put(0, "zero");
    System.out.println(mapa);    // {1=um, 3=três, 0=zero, 0.0=zero}
    }
  }
```

Estruturas de Dados: Mapas

```
package ed;

import java.util.TreeMap;

public class ExemploMap2
  {
  public static void main(String[] args)
    {
    TreeMap<String,Integer> mapa = new TreeMap<String,Integer>();
    mapa.put("um",1);
    mapa.put("dois",2);
    mapa.put("três",3);
    mapa.put("quatro",4);
    mapa.put("cinco",5);
    System.out.println(mapa);
    // {cinco=5, dois=2, quatro=4, três=3, um=1}
    System.out.println(mapa.get("quatro")+mapa.get("dois")); // 6
    }
  }
```

Matemática

Classe Math

- Contém campos estáticos para *e*,
- Contém métodos estáticos para:
  - Trigonometria (sin, cos, tan, asin, acos, atan, atan2)
  - Exponenciação e raízes (exp, log, log10, sqrt, cbrt)
  - Arredondamento (floor, ceil,round,rint)
  - Comparações (max, min)
  - Conversões (toDegrees, toRadians)
  - Geração de números aleatórios simples (random)
- Outros.

Classe Math

```
package math;

public class Trigonometria
  {
  public static void main(String[] args)
    {
    System.out.println("Ângulo\tSin\tCos\tTan");
    for(double ângulo=0;ângulo<Math.PI*2;ângulo += Math.PI/8.)
      {
      System.out.print(Math.toDegrees(ângulo)+"\t");
      System.out.print(Math.sin(ângulo)+"\t");
      System.out.print(Math.cos(ângulo)+"\t");
      System.out.print(Math.tan(ângulo)+"\t");
      System.out.println();
      }
    }
  }
```

```
Ângulo      Sin   Cos   Tan
0.0   0.0   1.0   0.0
22.5  0.3826834323650898    0.9238795325112867    0.41421356237309503
45.0  0.7071067811865475    0.7071067811865476    0.9999999999999999
67.5  0.9238795325112867    0.38268343236508984   2.414213562373095
90.0  1.0   6.123233995736766E-17 1.633123935319537E16
112.5 0.9238795325112867    -0.3826834323650897   -2.4142135623730954
135.0 0.7071067811865476    -0.7071067811865475   -1.0000000000000002
157.5 0.3826834323650899    -0.9238795325112867   -0.41421356237309515
180.0 1.2246467991473532E-16      -1.0  -1.2246467991473532E-16
202.49999999999997    -0.3826834323650897  -0.9238795325112868   0.41421356237309487
225.0 -0.7071067811865475   -0.7071067811865477   0.9999999999999997
247.49999999999997    -0.9238795325112865   -0.38268343236509034  2.414213562373091
270.0 -1.0  -1.8369701987210297E-16     5.443746451065123E15
292.50000000000006    -0.9238795325112866   0.38268343236509 -2.4142135623730936
315.00000000000006    -0.707106781186547    0.707106781186548     -0.9999999999999987
337.5000000000001     -0.3826834323650887   0.9238795325112872    -0.41421356237309376
```

Números de Precisão Arbitrária

- Duas classes do pacote java.math permitem representação de inteiros e decimais com precisão arbitrária: BigInteger e BigDecimal.
  - Classes, não tipos nativos!
  - É preciso criar instâncias e executar métodos.
  - Além de métodos para operações básicas, outros métodos específicos.
- Instâncias são imutáveis: operações retornam novas instâncias.

Matemática: BigInteger

```
package math;

import java.math.BigInteger;
import java.util.Random;

public class CriaBigInteger
  {
  public static void main(String[] args)
    {
    BigInteger b1 = new BigInteger("100000000000000000000000");
    System.out.println(b1); // 100000000000000000000000
    BigInteger b2 =
      new BigInteger(new byte[]{(byte)127,(byte)255,(byte)255,(byte)255,(byte)255});
    System.out.println(b2); // 549755813887 = 1111111 11111111 11111111 11111111
11111111
    BigInteger b3 = new BigInteger("BEBAMAISCAFE",32);
    System.out.println(b3); // 4124781563990614 86
    BigInteger b4 = new BigInteger(128,new Random());
    System.out.println(b4); // ex. 128162629103660182441501717705246294 28
    BigInteger b5 = BigInteger.probablePrime(128,new Random());
    System.out.println(b5); // ex. 291359484637392806951660141191384186061
    System.out.println(BigInteger.TEN); // 10 (!)
    }
  }
```

Matemática: BigInteger

```
package math;

import java.math.BigInteger;

public class UsaBigInteger
  {
  public static void main(String[] args)
    {
    BigInteger b1 = new BigInteger("100000000000000000000000");
    System.out.println(b1.toString());   // 100000000000000000000000
    System.out.println(b1.toString(16)); // d3c21bcecceda1000000
    System.out.println(b1.toString(36)); // 4iuaj6s1qvdv6nls
    BigInteger b2 = b1.add(new BigInteger("987654321"));
    System.out.println(b2); // 100000000000000987654321
    BigInteger b3 = b1.divide(new BigInteger("7"));
    System.out.println(b3); // 142857142857142857142857
    BigInteger b4 = b3.multiply(new BigInteger("7"));
    System.out.println(b4); // 999999999999999999999999
    BigInteger b5 = b1.max(b2);
    System.out.println(b5); // 100000000000000987654321
    BigInteger b6 = new BigInteger("2").pow(1000);
    System.out.println(b6); // 107...76 (302 dígitos)
    BigInteger b7 = b1.gcd(new BigInteger("222222222200"));
    System.out.println(b7); // 200
    }
  }
```

Matemática: BigInteger

```
package math;

import java.math.BigInteger;
import java.util.HashMap;

public class BigFatorial
  {
  private static HashMap<Integer,BigInteger> fatoriais =
    new HashMap<Integer, BigInteger>();

  public static BigInteger fatorial(int num)
    {
    if (fatoriais.get(num) == null)
      {
      BigInteger fat = BigInteger.ONE;
      for(int n=1;n<=num;n++)
        fat = fat.multiply(new BigInteger(""+n));
      fatoriais.put(num,fat);
      return fat;
      }
    else return fatoriais.get(num);
    }
```

Matemática: BigInteger

```
public static void main(String[] args)
  {
  BigInteger b10 = BigFatorial.fatorial(10);
  System.out.println(b10);   // 3628800
  calcula(1000); // 402...000 (2568 dígitos) em 31ms
  calcula(1000); // 402...000 (2568 dígitos) em 0ms
  calcula(20000); // 181...000 (77338 dígitos) em 2469ms
  calcula(20000); // 181...000 (77338 dígitos) em 0ms
  }

public static void calcula(int n)
  {
  long início = System.currentTimeMillis();
  BigInteger r = BigFatorial.fatorial(n);
  long fim = System.currentTimeMillis();
  System.out.println(r+" em "+(fim-início)+" ms");
  }

}
```

Matemática: BigDecimal

```
package math;

import java.math.BigDecimal;
import java.math.MathContext;

public class CriaBigDecimal
  {
  public static void main(String[] args)
    {
    BigDecimal b1 = new BigDecimal("100000000000000000000000.00000000000000001");
    System.out.println(b1); // 100000000000000000000000.00000000000000001
    System.out.println(b1.doubleValue()); // 1.0E24
    MathContext mc1 = new MathContext(35);
    BigDecimal b2 = new BigDecimal(b1.toString(),mc1);
    System.out.println(b2); // 100000000000000000000000.0000000000
    BigDecimal b3 = new BigDecimal(2.07);
    System.out.println(b3); // 2.069999999999998401278844539774581789970397949218 75
    BigDecimal b4 = BigDecimal.TEN;
    System.out.println(b4); // 10
    // BigDecimal b5 = b4.divide(new BigDecimal(7));
    // Exception in thread "main" java.lang.ArithmeticException:
    // Non-terminating decimal expansion; no exact representable decimal result.
    MathContext mc2 = new MathContext(48);
    BigDecimal b6 = b4.divide(new BigDecimal(7),mc2);
    System.out.println(b6); // 1.428571428571428571428571428571428571428571428571
    }
  }
```

Matemática: BigDecimal

```
package math;

import java.math.BigDecimal;
import java.math.MathContext;

public class UsaBigDecimal
  {
  public static void main(String[] args)
    {
    BigDecimal b1 = new BigDecimal("100000000000000000000000.00000000000000001");
    System.out.println(b1.precision()); // 42
    MathContext mc1 = new MathContext(35);
    BigDecimal b2 = new BigDecimal(b1.toString(),mc1);
    System.out.println(b2.precision()); // 35
    BigDecimal b3 = new BigDecimal("1.23456e789");
    System.out.println(b3.toString()); // 1.23456E+789
    System.out.println(b3.toPlainString()); // 12345600.....
    System.out.println(b3.toEngineeringString()); // 1.23456E+789
    System.out.println(b3.add(b3)); // 2.46912E+789
    System.out.println(b3.multiply(b3)); // 1.5241383936E+1578
    System.out.println(b3.divide(b3)); // 1
    }
  }
```

Matemática: BigDecimal

$$\frac{\pi}{4} = 4 * \arctan\left(\frac{1}{5}\right) - \arctan\left(\frac{1}{239}\right)$$

```
package math;

import java.math.BigDecimal;
import java.math.MathContext;

public class ReallyPrecisePi
  {
  // Calcula PI com fórmula de Machin: PI/4 = 4 * arctan(1/5) - arctan(1/239)
  private static MathContext mc = new MathContext(75);
  private static int calcSteps = 5;
  public static void main(String[] args)
    {
    BigDecimal frac1 = BigDecimal.ONE;
    frac1 = frac1.divide(new BigDecimal("5"),mc);
    BigDecimal t1 = arctan(frac1);
    BigDecimal frac2 = BigDecimal.ONE;
    frac2 = frac2.divide(new BigDecimal("239"),mc);
    BigDecimal t2 = arctan(frac2);
    BigDecimal pi = new BigDecimal("4");
    pi = pi.multiply(t1);
    pi = pi.subtract(t2);
    pi = pi.multiply(new BigDecimal("4"));
    System.out.println(pi);
    }
```

Matemática: BigDecimal

```
  // arctan(t) = t^1/1 -  t^3/3 + t^5/5 - t^7/7 + t^9/9..
  private static BigDecimal arctan(BigDecimal t)
    {
    BigDecimal res = BigDecimal.ZERO;
    for(int c=1;c<=calcSteps;c++)
      {
      BigDecimal sup = t.pow(c*2-1);
      BigDecimal inf = new BigDecimal(c*2-1);
      if ((c % 2) == 0) sup = sup.negate();
      res = res.add(sup.divide(inf,mc));
      }
    return res;
    }
  }
```

```
CalcSteps/MC
  5/75: 3.1415926824043995172402598360735758604897579967201355063017547035914903 4
 10/75: 3.1415926535897916969172796196201054481406519829326039638895900572007273 3
 25/75: 3.1415926535897932384626433832795028848774340169568717799892980661436541 5
 50/75: 3.1415926535897932384626433832795028841971693993751058209749445923078164 0
  Real: 3.1415926535897932384626433832795028841971693993751058209749445923078164 0
```

Linhas de Execução

Linhas de Execução (*Threads*)

- Código é executado na ordem explicitada dentro dos métodos.
- Métodos podem ser executados concorrentemente.
  - Ou com a aparência de concorrência em computadores com 1 CPU.
- Em Java, podemos indicar métodos podem ser executados concorrentemente em classes que herdam da classe Thread.
  - Estas classes devem implementar o método run (o método que será executado concorrentemente).

Linhas de Execução (*Threads*)

- Exemplo sem *Threads*

```
package threads;

public class CarroDeCorrida
  {
  private String nome;
  private int distância;
  private int velocidade;
  public CarroDeCorrida(String n, int vel)
    {
    nome = n;
    distância = 0;
    velocidade = vel;
    }
  public void executa()
    {
    while (distância <= 1200)
      {
      System.out.println(nome + " rodou " + distância + " km.");
      distância += velocidade;
      // Causa um delay artificial.
      for (int sleep = 0; sleep < 1000000; sleep++)
        double x = Math.sqrt(Math.sqrt(Math.sqrt(sleep)));
      }
    }
  }
```

Linhas de Execução (*Threads*)

```
package threads;

public class SimulacaoSemThreads
  {
  public static void main(String[] args)
    {
    // Criamos instâncias da classe CarroDeCorrida.
    CarroDeCorrida penélope =
      new CarroDeCorrida("Penélope Charmosa",60);
    CarroDeCorrida dick =
      new CarroDeCorrida("Dick Vigarista",100);
    CarroDeCorrida quadrilha =
      new CarroDeCorrida("Quadrilha da Morte",120);
    // Criados os carros, vamos executar as simulações.
    penélope.executa();
    dick.executa();
    quadrilha.executa();
    }
  }
```

```
Penélope Charmosa rodou 0 km.
Penélope Charmosa rodou 60 km.
Penélope Charmosa rodou 120 km.
...
Penélope Charmosa rodou 1140 km.
Penélope Charmosa rodou 1200 km.
Dick Vigarista rodou 0 km.
Dick Vigarista rodou 100 km.
Dick Vigarista rodou 200 km.
...
Dick Vigarista rodou 1100 km.
Dick Vigarista rodou 1200 km.
Quadrilha da Morte rodou 0 km.
Quadrilha da Morte rodou 120 km.
...
Quadrilha da Morte rodou 1080 km.
Quadrilha da Morte rodou 1200 km.
```

Linhas de Execução (*Threads*)

- Método mais simples de concorrência: classe herda de Thread e implementa o método run.
- Criamos instâncias desta classe e executamos o método start das mesmas.
  - Alternativa: implementar interface Runnable e ser executada como new Thread(instância).start.
- Podemos agrupar *threads* (instâncias de herdeiras de Thread) em um ThreadGroup.
  - Fácil manipular grupos e verificar quantas *threads* estão ativas.

Linhas de Execução (*Threads*)

- Exemplo com *Threads*

```
package threads;

public class CarroDeCorridaComThreads extends Thread
  {
  private String nome;
  private int distância;
  private int velocidade;
  public CarroDeCorridaComThreads(String n, int vel)
    {
    nome = n;
    distância = 0;
    velocidade = vel;
    }
  public void run()
    {
    while (distância <= 1200)
      {
      System.out.println(nome + " rodou " + distância + " km.");
      distância += velocidade;
      // Causa um delay artificial.
      for (int sleep = 0; sleep < 1000000; sleep++)
        double x = Math.sqrt(Math.sqrt(Math.sqrt(sleep)));
      }
    }
  }
```

Linhas de Execução (*Threads*)

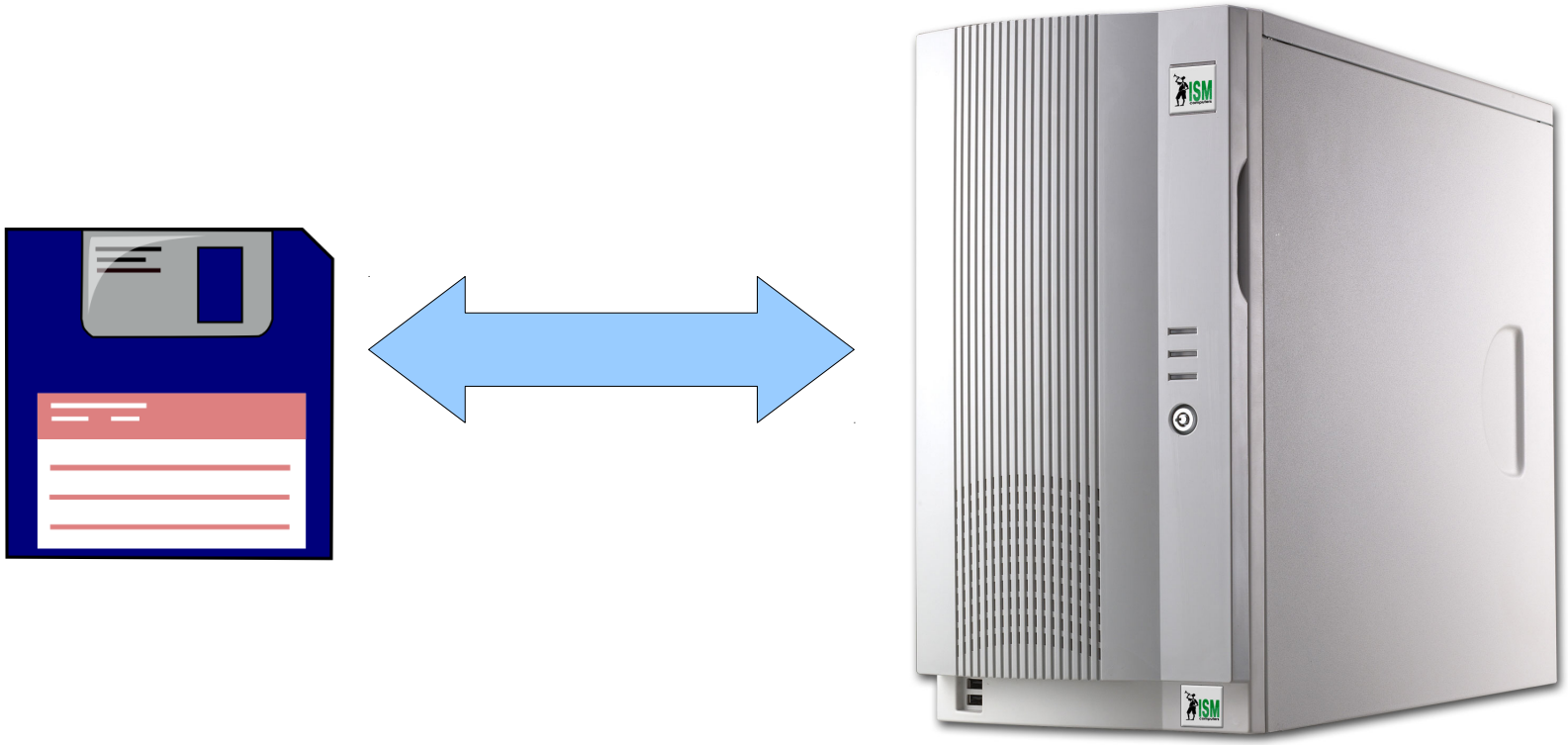
```
package threads;

public class SimulacaoComThreads
  {
  // Este método permite a execução da classe.
  public static void main(String[] args)
    {
    // Criamos instâncias da classe CarroDeCorrida.
    CarroDeCorridaComThreads penélope =
      new CarroDeCorridaComThreads("Penélope Charmosa",60);
    CarroDeCorridaComThreads dick =
      new CarroDeCorridaComThreads("Dick Vigarista",100);
    CarroDeCorridaComThreads quadrilha =
      new CarroDeCorridaComThreads("Quadrilha da Morte",120);
    // Criados os carros, vamos executar as simulações.
    penélope.run();
    dick.run();
    quadrilha.run();
    }
  }
```

```
Penélope Charmosa rodou 0 km.
Dick Vigarista rodou 0 km.
Quadrilha da Morte rodou 0 km.
Penélope Charmosa rodou 60 km.
Dick Vigarista rodou 100 km.
Quadrilha da Morte rodou 120 km.
...
Quadrilha da Morte rodou 1080 km.
Penélope Charmosa rodou 600 km.
Dick Vigarista rodou 1000 km.
...
Penélope Charmosa rodou 1140 km.
Penélope Charmosa rodou 1200 km.
```

Entrada e Saída

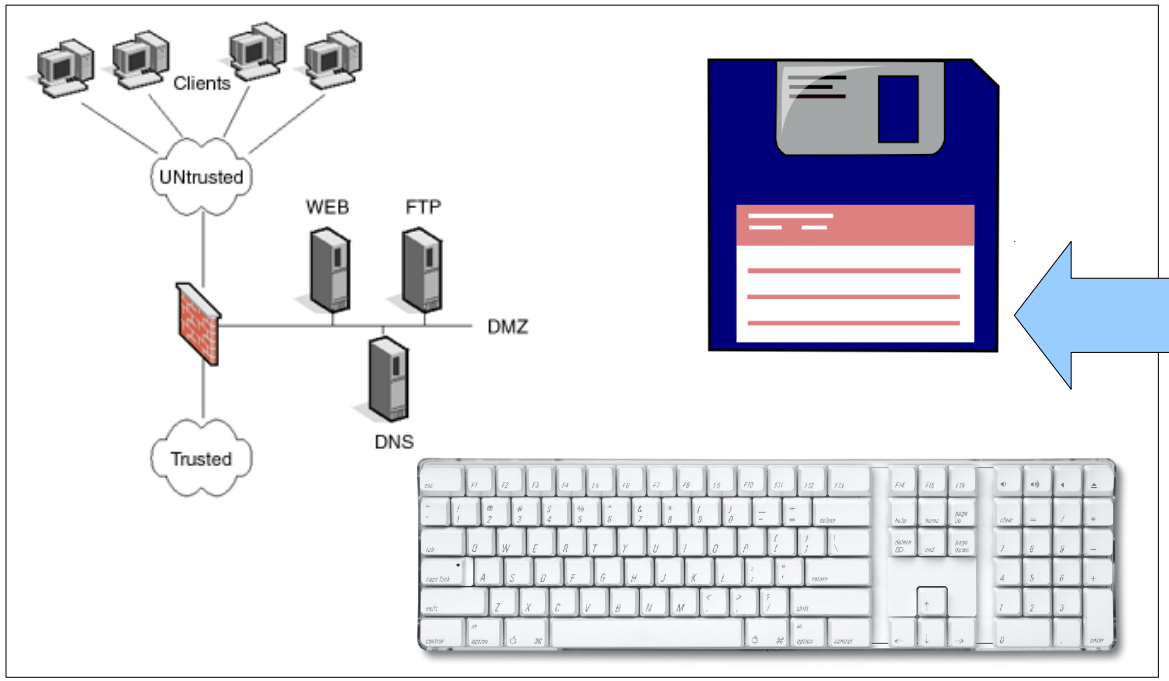
Entrada e Saída

- Uma das tarefas mais importantes de um programa de computador: possibilita reuso de algoritmos com dados diferentes a cada execução.
- Exemplo mais comum: **arquivos**.



Entrada e Saída

- Entrada e saída pode ser feita de diversas maneiras!
- Usaremos uma abstração: ***streams* (correntes) de dados**.
- Origem ou destino podem ser remotos.



Entrada e Saída: Acesso Seqüencial

Leitura:

0. Abrir a *Stream* (criando uma instância de uma classe que a representa)
1. Ler dados da *Stream*
   - Se os dados lidos forem **nulos**, chegamos ao final da *Stream*: terminar processamento
   - Senão processar dados e voltar ao passo 2
2. Fechar *Stream*

Escrita:

0. Abrir *Stream* (criando uma instância de uma classe que a representa)
1. Escrever dados na *Stream* enquanto houver dados a ser escritos
2. Fechar *Stream*

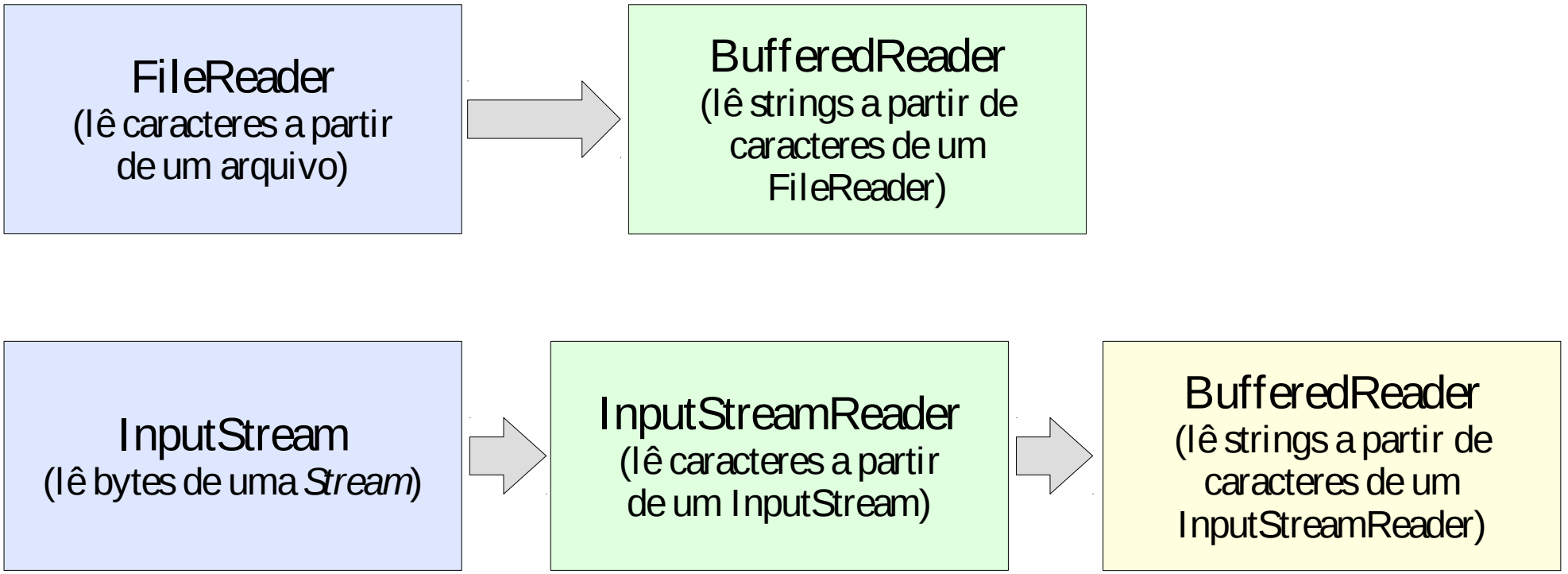
Entrada e Saída: Problemas Potenciais

- Algo pode dar errado!
- Se a *Stream* for de um arquivo de entrada:
  - O arquivo existe? Pode ser lido? Está no formato esperado?
- Se a *Stream* for de um arquivo de saída:
  - O arquivo pode ser criado? Temos espaço suficiente em disco?
- Se a *Stream* for de uma conexão de rede:
  - A conexão foi estabelecida corretamente? Os dados estão no formato esperado? O servidor/porta existe?

- Solução: processamento de exceções!

Entrada e Saída: Classes

- Classes de nível mais alto são criadas a partir de classes de nível mais baixo.



Entrada e Saída: Classes

	Entrada	Saída
Bytes	`BufferedInputStream` `DataInputStream`	`BufferedOutputStream` `DataOutputStream`
Caracteres	`InputStreamReader`	`OutputStreamWriter`
Tipos nativos	`DataInputStream`	`DataOutputStream`
Strings	`BufferedReader`	`BufferedWriter`
Objetos	`ObjectInputStream`	`ObjectOutputStream`

Escrevendo Strings

```
package io;

import java.io.*;

public class DemoBufferedWriter
  {
  public static void main(String[] args)
    {
    try
      {
      BufferedWriter bw = new BufferedWriter(new FileWriter("x.txt"));
      bw.write("Primeira linha de texto");
      bw.newLine();
      bw.write("Segunda linha de texto");
      bw.newLine();
      bw.write("Terceira linha de texto");
      bw.newLine();
      bw.close();
      }
    catch (IOException e)
      {
      System.out.println("Erro na gravação !");
      }
    }
  }
```

Lendo Strings

```
package io;
import java.io.*;

public class DemoBufferedReader
  {
  public static void main(String[] args)
    {
    try
      {
      BufferedReader br = new BufferedReader(new FileReader("Apl2.java"));
      String line;
      while (true)
        {
        line = br.readLine();
        if (line == null) break;
        else System.out.println(line);
        }
      br.close();
      }
    catch (FileNotFoundException e) { System.out.println("Arquivo não encontrado !"); }
    catch (IOException e) { System.out.println("Erro na leitura !"); }
    }
  }
```

Escrevendo Tipos Nativos

```
package io;

import java.io.*;

public class DemoDataOutputStream
  {
  public static void main(String[] args)
    {
    DataOutputStream dos;
    try
      {
      dos = new DataOutputStream(new FileOutputStream("tst.dat"));
      dos.writeBoolean(true);
      dos.writeInt(123456);
      dos.writeFloat(3.1416f);
      dos.writeUTF("String");
      dos.close();
      }
    catch (IOException e)
      {
      System.out.println("Erro na gravação !");
      }
    }
  }
```

Lendo Tipos Nativos

```
package io;

import java.io.*;

public class DemoDataInputStream
  {
  public static void main(String[] args)
    {
    DataInputStream dis;
    try
      {
      dis = new DataInputStream(new FileInputStream("tst.dat"));
      boolean b = dis.readBoolean();
      int i = dis.readInt();
      float f = dis.readFloat();
      String s = dis.readUTF();
      dis.close();
      }
    catch (FileNotFoundException e) { System.out.println("Arquivo não encontrado !"); }
    catch (IOException e) { System.out.println("Erro na leitura !"); }
    }
  }
```

Entrada e Saída de Instâncias de Classes

- Armazenamento de instâncias de classes: **persistência.**
- Mecanismo “manual”: métodos que armazenam e recuperam os valores de todos os campos.
- Mecanismo automático: **serialização.**
  - Basta declarar que a classe a ser armazenada implementa a interface *Serializable*.
  - Algumas classes de Java não são serializáveis.
- Problemas com versões de classes!

Uma Classe Serializável

```
package io;

import java.io.Serializable;

public class ContaBancaria implements Serializable
  {
  private String nomeCorrentista;
  private double saldo;

  public ContaBancaria(String n, double s)
    {
    nomeCorrentista = n;
    saldo = s;
    }

  public double getSaldo() { return saldo; }

  public String getNome() { return nomeCorrentista; }

  public void deposita(double quantia) { saldo = saldo + quantia; }

  public void retira(double quantia)
    {
    if (quantia < saldo) saldo = saldo - quantia;
    }
  }
```

Escrevendo Instâncias de uma Classe

```
package io;

import java.io.*;

public class DemoObjectOutputStream
  {
  public static void main(String[] args)
    {
    ObjectOutputStream oos;
    ContaBancaria c1 = new ContaBancaria("Sid Sackson",120000);
    ContaBancaria c2 = new ContaBancaria("R. C. Bell",900);
    try
      {
      oos = new ObjectOutputStream(new FileOutputStream("cnt.dat"));
      oos.writeObject(c1);
      oos.writeObject(c2);
      oos.close();
      }
    catch (IOException e)
      {
      System.out.println("Erro na gravação !");
      }
    }
  }
```

Lendo Instâncias de uma Classe

```
package io;

import java.io.*;

public class DemoObjectInputStream
  {
  public static void main(String[] args)
    {
    ObjectInputStream ois;
    try
      {
      ois = new ObjectInputStream(new FileInputStream("cnt.dat"));
      ContaBancaria ca = (ContaBancaria)ois.readObject();
      ContaBancaria cb = (ContaBancaria)ois.readObject();
      System.out.println(ca.getNome()+":"+ca.getSaldo());
      System.out.println(cb.getNome()+":"+cb.getSaldo());
      ois.close();
      }
    catch (FileNotFoundException e) { System.out.println("Arquivo não encontrado !"); }
    catch (IOException e) { System.out.println("Erro na leitura !"); }
    catch (ClassNotFoundException e)
      {
      System.out.println("Classe para leitura nÃ£o encontrada !");
      }
    }
  }
```

Entrada e Saída de Instâncias de Classes

- Lembrando sempre que:
  - Um arquivo serializado não pode ser lido de outra forma!
  - Em Linux, o comando file cnt.dat dá como resultado Java serialization data, version 5.
  - Para transporte entre máquinas virtuais, a classe serializável deve estar presente!
- É possível serializar objetos para arquivos XML...
  - Se os objetos forem instâncias de *Beans*.
  - Nem sempre nossas classes podem ou devem ser *Beans*...

Uma Classe Serializável (*Bean*)

```
package io;

public class ContaBancariaBean
  {
  private String nomeCorrentista;
  private double saldo;

  public ContaBancariaBean() { }

  public void setNomeCorrentista(String nc) { nomeCorrentista = nc; }

  public void setSaldo(double s) { saldo = s; }

  public String getNomeCorrentista() { return nomeCorrentista; }

  public double getSaldo() { return saldo; }

  public void deposita(double quantia) { saldo = saldo + quantia; }

  public void retira(double quantia)
    {
    if (quantia < saldo) saldo = saldo - quantia;
    }

  }
```

Escrevendo Instâncias de uma Classe (*Bean*)

```
package io;

import java.beans.XMLEncoder;
import java.io.*;

public class DemoXMLEncoder
  {
  public static void main(String[] args)
    {
    ContaBancariaBean c1 = new ContaBancariaBean();
    c1.setNomeCorrentista("Sid Sackson"); c1.setSaldo(120000);
    ContaBancariaBean c2 = new ContaBancariaBean();
    c2.setNomeCorrentista("R. C. Bell"); c2.setSaldo(900);
    try
      {
      XMLEncoder encoder = new XMLEncoder(new FileOutputStream("cnt.xml"));
      encoder.writeObject(c1);
      encoder.writeObject(c2);
      encoder.close();
      }
    catch (IOException e)
      {
      System.out.println("Erro na gravação !");
      }
    }
  }
```

Lendo Instâncias de uma Classe (*Bean*)

```
package io;

import java.beans.XMLDecoder;
import java.io.*;

public class DemoXMLDecoder
  {
  public static void main(String[] args)
    {
    try
      {
      XMLDecoder decoder = new XMLDecoder(new FileInputStream("cnt.xml"));
      ContaBancariaBean b1 = (ContaBancariaBean)decoder.readObject();
      ContaBancariaBean b2 = (ContaBancariaBean)decoder.readObject();
      System.out.println(b1.getNomeCorrentista()+" tem "+b1.getSaldo());
      System.out.println(b2.getNomeCorrentista()+" tem "+b2.getSaldo());
      decoder.close();
      }
    catch (IOException e)
      {
      System.out.println("Erro na leitura !");
      }
    }
  }
```

Classe Serializada em XML

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.5.0_09" class="java.beans.XMLDecoder">
 <object class="io.ContaBancariaBean">
  <void property="nomeCorrentista">
   <string>Sid Sackson</string>
  </void>
  <void property="saldo">
   <double>120000.0</double>
  </void>
 </object>
 <object class="io.ContaBancariaBean">
  <void property="nomeCorrentista">
   <string>R. C. Bell</string>
  </void>
  <void property="saldo">
   <double>900.0</double>
  </void>
 </object>
</java>
```

Manipulação de Arquivos

- Classe File permite encapsulamento e operações em arquivos e diretórios.
  - Verificação de atributos (leitura, escrita, etc.).
  - Criação de arquivos.
  - Lista de conteúdo (se diretório).

Manipulação de Arquivos

```
package io;

import java.io.File;
import java.util.Date;

public class Arquivos
  {
  public static void main(String[] args)
    {
    File f = new File("/etc/termcap");
    if (f.exists())
      {
      System.out.println("Arquivo "+f.getName());
      System.out.println("Path "+f.getParent());
      System.out.println("Tamanho: "+f.length()+" bytes");
      System.out.println("Pode ser lido? "+f.canRead());
      System.out.println("Pode ser escrito? "+f.canWrite());
      System.out.println("Modificado em "+new Date(f.lastModified()));
      }
    }
  }
```

```
Arquivo termcap
Path /etc
Tamanho: 807103 bytes
Pode ser lido? true
Pode ser escrito? false
Modificado em Wed Jul 12 22:18:39 BRT 2006
```

Manipulação de Arquivos

```
package io;
import java.io.File;
public class DiretorioImagens
  {
  public static void main(String[] args)
    {
    listaImagens(new File("/usr/java/default"));
    }

  private static void listaImagens(File base)
    {
    String[] conteúdo = base.list(new ImageFileFilter());
    for(String c:conteúdo)
      {
      File entrada = new File(base,c);
      if (entrada.isDirectory()) listaImagens(entrada);
      else System.out.println(entrada);
      }
    }
  }
```

Manipulação de Arquivos

```
package io;

import java.io.File;
import java.io.FilenameFilter;

public class ImageFileFilter implements FilenameFilter
  {
  public boolean accept(File dir, String name)
    {
    if (new File(dir,name).isDirectory()) return true;
    if (name.toLowerCase().endsWith(".jpeg") ||
        name.toLowerCase().endsWith(".jpg")) return true;
    if (name.toLowerCase().endsWith(".bmp")) return true;
    if (name.toLowerCase().endsWith(".gif")) return true;
    if (name.toLowerCase().endsWith(".png")) return true;
    return false;
    }
  }
```

```
/usr/java/default/demo/applets/TicTacToe/images/not.gif
/usr/java/default/demo/applets/TicTacToe/images/cross.gif
/usr/java/default/demo/applets/ImageMap/images/jim.graham.gif
/usr/java/default/demo/applets/Animator/images/SimpleAnimation/T1.gif
/usr/java/default/demo/applets/Animator/images/SimpleAnimation/T2.gif
/usr/java/default/demo/applets/Animator/images/Beans/T5.gif
/usr/java/default/demo/applets/Animator/images/Beans/T3.gif
/usr/java/default/demo/applets/Animator/images/Beans/T1.gif
/usr/java/default/demo/applets/Animator/images/Beans/T4.gif
/usr/java/default/demo/applets/Animator/images/Beans/T9.gif
/usr/java/default/demo/applets/Animator/images/Beans/T8.gif
/usr/java/default/demo/applets/Animator/images/Beans/T6.gif
...
```

Programação para Rede

Programação para Rede

- Programação baixo nível para Rede: Programação Cliente-Servidor.
- Assumimos que dois computadores em uma rede local ou na Internet precisam enviar dados de forma bem controlada.
- Existem alternativas: *web services*, HTTP.
- Motivação/justificativa: queremos ter controle (quase) total sobre comunicação entre aplicações.

- Veremos aplicações simples que usam TCP/IP.

Programação para Rede: Conceitos

- Termos simplificados:
  - **Servidor**: computador que contém o recurso que necessitamos.
  - **Serviço**: aplicação no servidor.
  - **Cliente**: aplicação que usará o recurso remoto.
  - **Endereço**: localização (lógica) de um computador em rede.
  - **Porta**: endereço numérico local em um servidor, associada a serviço.
    - **Porta Bem Conhecida**: porta com valor < 1024, geralmente corresponde a serviços bem conhecidos.
  - **Protocolo**: diálogo que será travado entre cliente e servidor para troca de informações.
  - **Socket**: possibilita a comunicação entre cliente e servidor.

Cliente:

0. Cria conexão (*socket*) e cria canais de IO.
1. Lê e envia dados via *socket* de acordo com o protocolo.
2. Fecha *socket* e canais de IO.

Servidor:

0. Cria serviço (*socket*).
1. Aguarda conexão (*socket* para o cliente) e cria canais de IO.
2. Lê e envia dados via *socket* de acordo com o protocolo.
3. Fecha *socket* para o cliente e volta ao passo 1.

Canais de IO

- Comunicação é feita com uma instância de Socket
  - Criada do lado do cliente com um nome de servidor e porta.
  - Criada do lado do servidor quando uma conexão é feita.
- Métodos de Socket:
  - getInputStream() retorna uma instância de InputStream que pode ser usada para construir BufferedReaders, ObjectInputStreams, DataInputStreams, etc.
  - getOutputStream() retorna uma instância de OutputStream que pode ser usada para construir BufferedWriters, ObjectOutputStreams, DataOutputStreams, etc.
  - close() fecha a conexão.

Exemplo: Cliente de Daytime

- Serviço em alguns computadores na rede que indica a hora local.
  - *Default*: porta 13.
  - Não requer envio de dados.
  - Resultados: em texto, pode ter várias linhas.
  - Alguns servidores em http://tf.nist.gov/service/time-servers.html.

Exemplo: Cliente de Daytime

```
package net;
import java.io.*;
import java.net.*;
public class ClienteDeDaytime
  {
  public static void main(String[] args)
    {
    String servidor = "time-nw.nist.gov";
    try
      {
      Socket conexão = new Socket(servidor,13);
      BufferedReader br =
        new BufferedReader(new InputStreamReader(conexão.getInputStream()));
      String linha;
      System.out.println("Resposta do servidor "+servidor+":");
      while(true)
        {
        linha = br.readLine(); if (linha == null) break;
        System.out.println(linha);
        }
      br.close(); conexão.close();
      }
    catch (UnknownHostException e) { System.out.println("O servidor não existe ou está"+
                                     " fora do ar."); }
    catch (IOException e) { System.out.println("Erro de entrada e saída."); }
    }
  }
```

Exemplo: Cliente de Echo

```
package net;

import java.io.*;
import java.net.*;
import javax.swing.JOptionPane;

public class ClienteDeEcho
  {
  public static void main(String[] args)
    {
    String servidor = "localhost";
    try
      {
      Socket conexão = new Socket(servidor,7);
      BufferedReader br =
        new BufferedReader(new InputStreamReader(conexão.getInputStream()));
      BufferedWriter bw =
        new BufferedWriter(new OutputStreamWriter(conexão.getOutputStream()));
```

Exemplo: Cliente de Echo

```
      while(true)
        {
        String linhaEnviada = JOptionPane.showInputDialog("Entre uma String");
        if (linhaEnviada == null) break;
        bw.write(linhaEnviada);
        bw.newLine();
        bw.flush();
        String linhaRecebida = br.readLine();
        System.out.println(linhaRecebida);
        }
      br.close();
      bw.close();
      conexão.close();
      }
    catch (UnknownHostException e)
      {
      System.out.println("O servidor não existe ou está fora do ar.");
      }
    catch (IOException e)
      {
      System.out.println("Erro de entrada e saída.");
      }
    System.exit(0);
    }

  }
```

Exemplo: Servidor de Strings Invertidas

```
package net;

import java.io.*;
import java.net.*;

public class ServidorDeStringsInvertidas
  {
  public static void main(String[] args)
    {
    ServerSocket servidor;
    try
      {
      servidor = new ServerSocket(10101);
      while(true)
        {
        Socket conexão = servidor.accept();
        processaConexão(conexão);
        }
      }
    catch (BindException e) { System.out.println("Porta já em uso."); }
    catch (IOException e) { System.out.println("Erro de entrada ou saída."); }
    }
  }
```

Exemplo: Servidor de Strings Invertidas

```
private static void processaConexão(Socket conexão)
  {
  try
    {
    BufferedReader entrada =
      new BufferedReader(new InputStreamReader(conexão.getInputStream()));
    BufferedWriter saída =
      new BufferedWriter(new OutputStreamWriter(conexão.getOutputStream()));
    String original = entrada.readLine();
    String invertida = "";
    for(int c=0;c<original.length();c++)
      {
      invertida = original.charAt(c)+invertida;
      }
    saída.write(invertida);
    saída.newLine();
    saída.flush();
    entrada.close();
    saída.close();
    conexão.close();
    }
  catch (IOException e)
    {
    System.out.println("Erro atendendo a uma conexão !");
    }
  }
}
```

Problemas Potenciais com Servidores

- Servidores simples: laço no servidor (aguarda conexão, atende conexão, volta a aguardar).
- Se atendimento à conexão demorar, outro pedido de serviço pode ser negado.
  - Mais freqüente quando atendimento à conexão envolve interação com usuário.
  - Limitar serviço? Forçar *timeout*?
- Solução: servidores *multithreaded*.
  - Uma classe que fornece o serviço herda de Thread.
  - Servidor cria instância e executa start para atender conexão.

Exemplo: Serviço de Números Aleatórios

```
package net;
import java.io.*;
import java.net.*;

public class ServicoDeNumerosAleatorios extends Thread
  {
  private Socket conexão = null;
  public ServicoDeNumerosAleatorios(Socket conn) { conexão = conn; }
  public void run()
    {
    try
      {
      DataInputStream entrada = new DataInputStream(conexão.getInputStream());
      DataOutputStream saída = new DataOutputStream(conexão.getOutputStream());
      int quantidade = entrada.readInt();
      for(int q=0;q<quantidade;q++)
        {
        double rand = Math.random();
        saída.writeDouble(rand);
        System.out.println("Acabo de enviar o número "+rand+" para o cliente "+
                           conexão.getRemoteSocketAddress());
        }
      entrada.close();       saída.close();         conexão.close();
      }
    catch (IOException e) { System.out.println("Erro atendendo a uma conexão !"); }
    }
  }
```

Exemplo: Servidor de Números Aleatórios

```
package net;

import java.io.*;
import java.net.*;

public class ServidorDeNumerosAleatorios
  {
  public static void main(String[] args)
    {
    ServerSocket servidor;
    try
      {
      servidor = new ServerSocket(9999);
      while(true)
        {
        // Quando uma conexão é feita,
        Socket conexão = servidor.accept();
        // ... criamos um serviço para atender a esta conexão...
        ServicoDeNumerosAleatorios s = new ServicoDeNumerosAleatorios(conexão);
        // ... e executamos o serviço (que será executado concorrentemente).
        s.start();
        }
      }
    catch (BindException e) { System.out.println("Porta já em uso."); }
    catch (IOException e) { System.out.println("Erro de entrada e saída."); }
    }
  }
```

Exemplo: Cliente de Números Aleatórios

```
package net;

import java.io.*;
import java.net.*;
import javax.swing.JOptionPane;

public class ClienteDeNumerosAleatorios
  {
  public static void main(String[] args)
    {
    String servidor = "localhost";
    try
      {
      Socket conexão = new Socket(servidor,9999);
      DataInputStream dis =
        new DataInputStream(conexão.getInputStream());
      DataOutputStream dos =
          new DataOutputStream(conexão.getOutputStream());
```

Exemplo: Cliente de Números Aleatórios

```
      int quantos = Integer.parseInt(JOptionPane.showInputDialog("Quantos números?"));
      dos.writeInt(quantos);
      for(int q=0;q<quantos;q++)
        {
        double valor = dis.readDouble();
        System.out.println("O "+(q+1)+"o valor é "+valor+".");
        }
      dis.close();
      conexão.close();
      }
    catch (UnknownHostException e)
      {
      System.out.println("O servidor não existe ou está fora do ar.");
      }
    catch (IOException e)
      {
      System.out.println("Erro de entrada e saída.");
      }
    System.exit(0);
    }
  }
```

Reflexão

Reflexão

- API que permite introspecção: identificação e manipulação de campos e métodos de classes presentes em tempo de execução.
- É possível, em tempo de execução:
  - Obter informações sobre classes, interfaces, métodos, campos, etc.
  - Criar instâncias de classes que não são conhecidas em tempo de execução e executar seus métodos.
- Exemplo simples: *plug-ins* dinâmicos.

Reflexão

```
package reflexao;
public interface ConstanteMatematica
  {
  public String getTítulo();   public String getDescrição();  public double getValor();
  }
```

```
package reflexao;
public class ConstantePI implements ConstanteMatematica
  {
  public String getDescrição() { return "O valor de PI com precisão dupla"; }
  public String getTítulo()    { return "pi"; }
  public double getValor()     { return Math.PI; }
  }
```

```
package reflexao;
public class ConstanteE implements ConstanteMatematica
  {
  public String getDescrição() { return "O valor de E com precisão dupla";  }
  public String getTítulo()    { return "e"; }
  public double getValor()     { return Math.E; }
  }
```

```
package reflexao;
public class ConstantePiSimples implements ConstanteMatematica
  {
  public String getDescrição() { return "O valor aproximado de PI"; }
  public String getTítulo()    { return "pi (aprox.)"; }
  public double getValor()     { return 355./113.; }
  }
```

Reflexão

```
package reflexao;

import java.io.File;
import java.util.ArrayList;

public class ListaConstantes
  {
  public static void main(String[] args)
    {
    File dirPlugins = new File("reflexao");
    // Recuperamos a lista de arquivos .class neste diretório.
    String[] arquivosClass = dirPlugins.list(new ClassFilenameFilter());
    // Mas quais herdam de ConstanteMatematica? Filtramos e criamos instâncias.
    ArrayList<ConstanteMatematica> constantes = filtraClasses(arquivosClass);
    for(ConstanteMatematica cm:constantes)
      {
      System.out.println(cm.getTítulo()+" ("+cm.getDescrição()+") :"+cm.getValor());
      }
    }
```

Reflexão

```
private static ArrayList<ConstanteMatematica> filtraClasses(String[] nomes)
  {
  ArrayList<ConstanteMatematica> lista = new ArrayList<ConstanteMatematica>();
  for(String nome:nomes)
    {
    boolean aceita = false;       Class c = null;
    String nomeClasse = "reflexao."+nome.substring(0,nome.indexOf(".class"));
    try
      {
      c = Class.forName(nomeClasse);
      Class[] interfacesImplementadas = c.getInterfaces();
      for (Class i:interfacesImplementadas)
        if (i == reflexao.ConstanteMatematica.class) aceita = true;
      }
    catch (ClassNotFoundException e) { aceita = false; }
    if (aceita)
      {
      ConstanteMatematica o = null;
      try
        {
        o = (ConstanteMatematica)c.newInstance();
        }
      catch (InstantiationException e) { e.printStackTrace(); }
      catch (IllegalAccessException e) { e.printStackTrace(); }
      if (o != null) lista.add(o);
      }
    }
  return lista;
  }
}
```

Reflexão

```
package reflexao;

import java.io.File;
import java.io.FilenameFilter;

public class ClassFilenameFilter implements FilenameFilter
  {
  public boolean accept(File dir, String name)
    {
    return (name.endsWith(".class"));
    }
  }
```

```
pi (aprox.) (O valor aproximado de PI) :3.14159292035398 25
pi (O valor de PI com precisão dupla) :3.141592653589793
e (O valor de E com precisão dupla) :2.718281828459045
```

Perguntas?