Programação Orientada a Objetos

Prof. Paulo André Castro
pauloac@ita.br
www.comp.ita.br/~pauloac
ITA – Stefanini

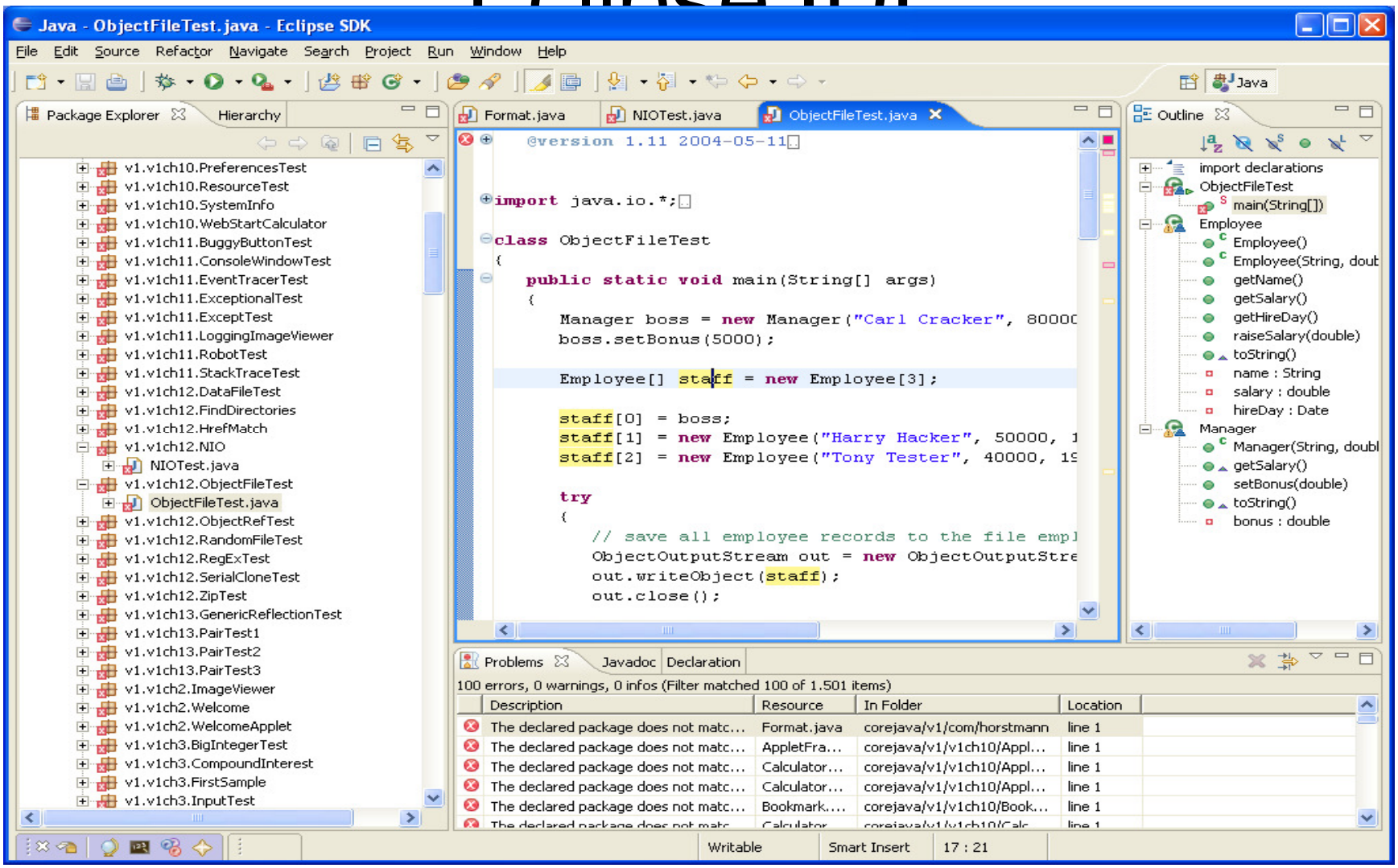
Rules of the lab

- Não deixar lixar e não comer nas bancadas do lab.
- Não desplugar cabos (mouse, teclado, monitor, rede…)
- Desligar o computador e monitor ao deixar o lab.
- Durante a aula, deve-se utilizar as máquinas apenas para questões relativas a aula
- Navegar, redes sociais, email podem ser usados apenas no intervalo

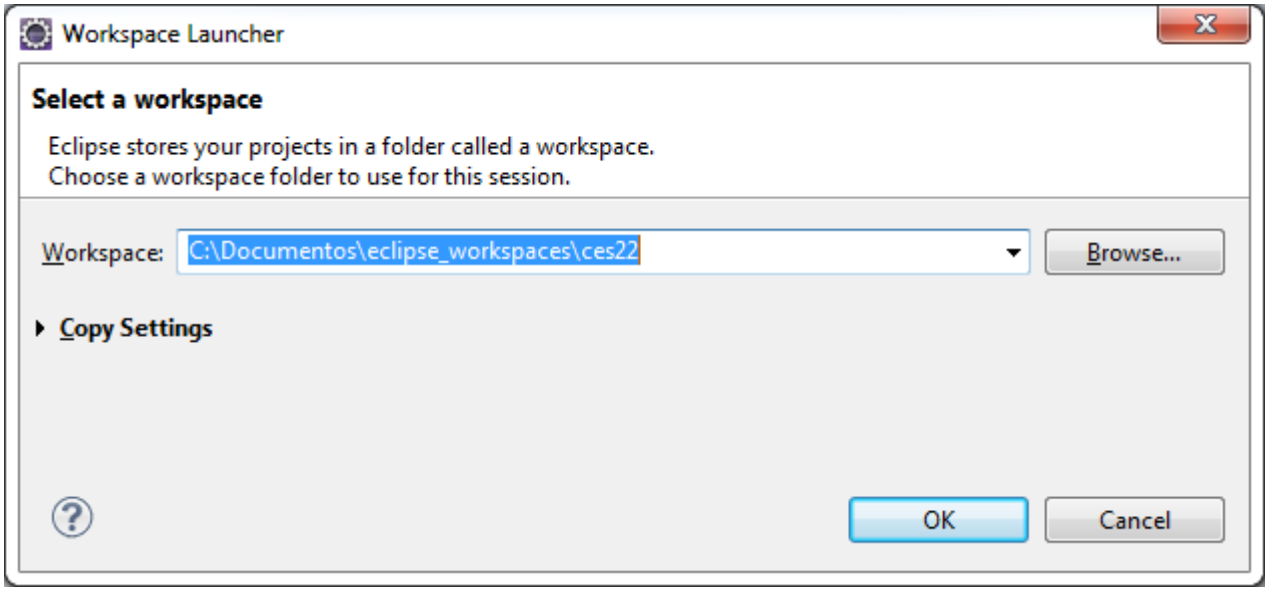
Sumário de Hoje

- Introdução ao Ambiente Eclipse
  - Criando workspaces e projetos
  - Compilando e executando programas
- Desenvolvimento de Programas básicos (modo texto)
  - Os conceitos de CLASSPATH, package e import
  - Formatando texto
- O sistema de I/O Orientado a Objetos do Java
  - Acessando arquivos Texto
  - Acessando arquivos Binários
  - Serialização e armazenamento de Objetos

Eclipse IDE

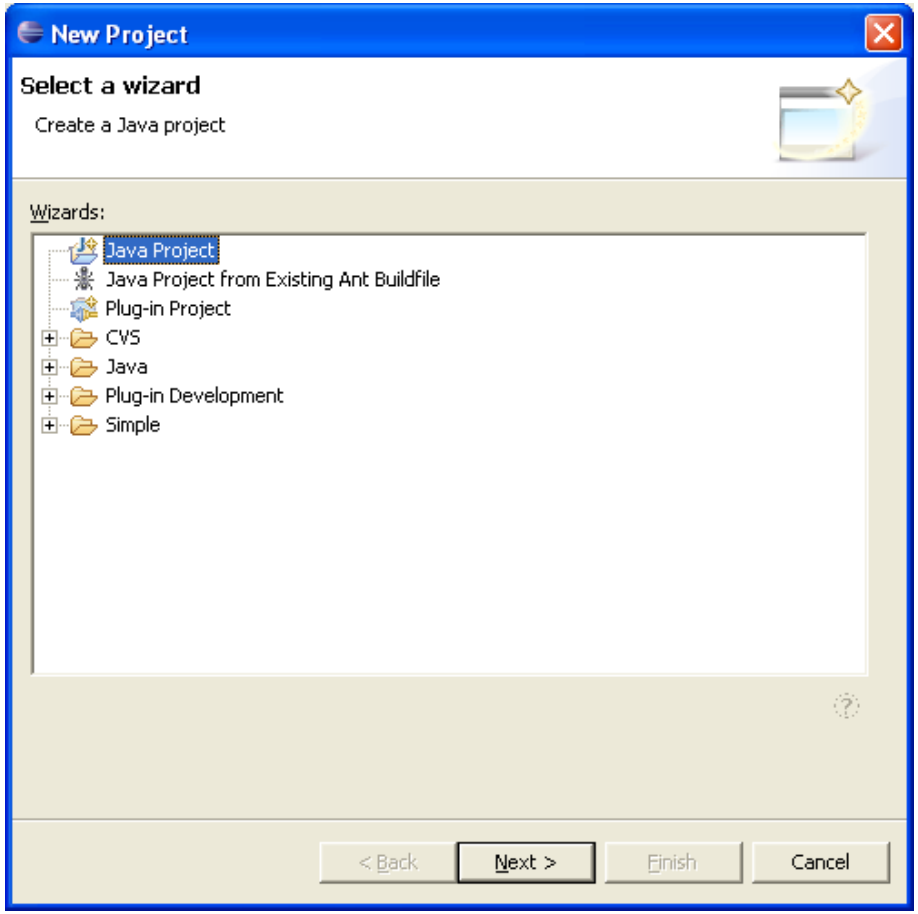

Java - ObjectFileTest.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Hierarchy
v1.v1ch10.PreferencesTest
v1.v1ch10.ResourceTest
v1.v1ch10.SystemInfo
v1.v1ch10.WebStartCalculator
v1.v1ch11.BuggyButtonTest
v1.v1ch11.ConsoleWindowTest
v1.v1ch11.EventTracerTest
v1.v1ch11.ExceptionalTest
v1.v1ch11.ExceptTest
v1.v1ch11.LoggingImageViewer
v1.v1ch11.RobotTest
v1.v1ch11.StackTraceTest
v1.v1ch12.DataFileTest
v1.v1ch12.FindDirectories
v1.v1ch12.HrefMatch
v1.v1ch12.NIO
NIOTest.java
v1.v1ch12.ObjectFileTest
ObjectFileTest.java
v1.v1ch12.ObjectRefTest
v1.v1ch12.RandomFileTest
v1.v1ch12.RegExTest
v1.v1ch12.SerialCloneTest
v1.v1ch12.ZipTest
v1.v1ch13.GenericReflectionTest
v1.v1ch13.PairTest1
v1.v1ch13.PairTest2
v1.v1ch13.PairTest3
v1.v1ch2.ImageViewer
v1.v1ch2.Welcome
v1.v1ch2.WelcomeApplet
v1.v1ch3.BigIntegerTest
v1.v1ch3.CompoundInterest
v1.v1ch3.FirstSample
v1.v1ch3.InputTest
Format.java
NIOTest.java
ObjectFileTest.java
@version 1.11 2004-05-11
import java.io.*;
class ObjectFileTest
{
public static void main(String[] args)
{
Manager boss = new Manager("Carl Cracker", 80000
boss.setBonus(5000);
Employee[] staff = new Employee[3];
staff[0] = boss;
staff[1] = new Employee("Harry Hacker", 50000, 1
staff[2] = new Employee("Tony Tester", 40000, 19
try
{
// save all employee records to the file empl
ObjectOutputStream out = new ObjectOutputStre
out.writeObject(staff);
out.close();
Outline
import declarations
ObjectFileTest
main(String[])
Employee
Employee()
Employee(String, dout
getName()
getSalary()
getHireDay()
raiseSalary(double)
toString()
name : String
salary : double
hireDay : Date
Manager
Manager(String, doubl
getSalary()
setBonus(double)
toString()
bonus : double
Problems
Javadoc
Declaration
100 errors, 0 warnings, 0 infos (Filter matched 100 of 1.501 items)
Description
Resource
In Folder
Location
The declared package does not matc... Format.java corejava/v1/com/horstmann line 1
The declared package does not matc... AppletFra... corejava/v1/v1ch10/Appl... line 1
The declared package does not matc... Calculator... corejava/v1/v1ch10/Appl... line 1
The declared package does not matc... Calculator... corejava/v1/v1ch10/Appl... line 1
The declared package does not matc... Bookmark.... corejava/v1/v1ch10/Book... line 1
Writable
Smart Insert
17 : 21

Criando Workspace



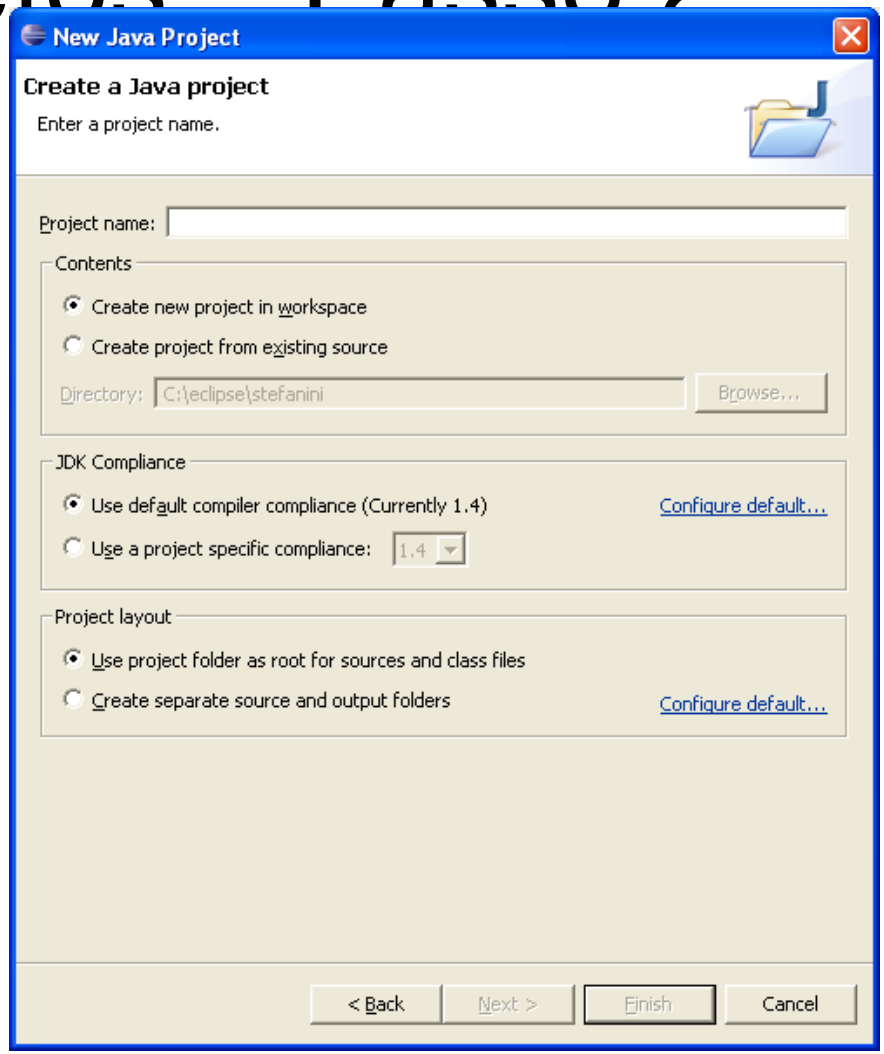
Criando projetos

- Menu File | New | Project
- Java Project
- A partir de código pré-existente arquivos ant
- CVS
- Java
  - Java Project
  - Java Project from ..Ant
- Plug-In Development



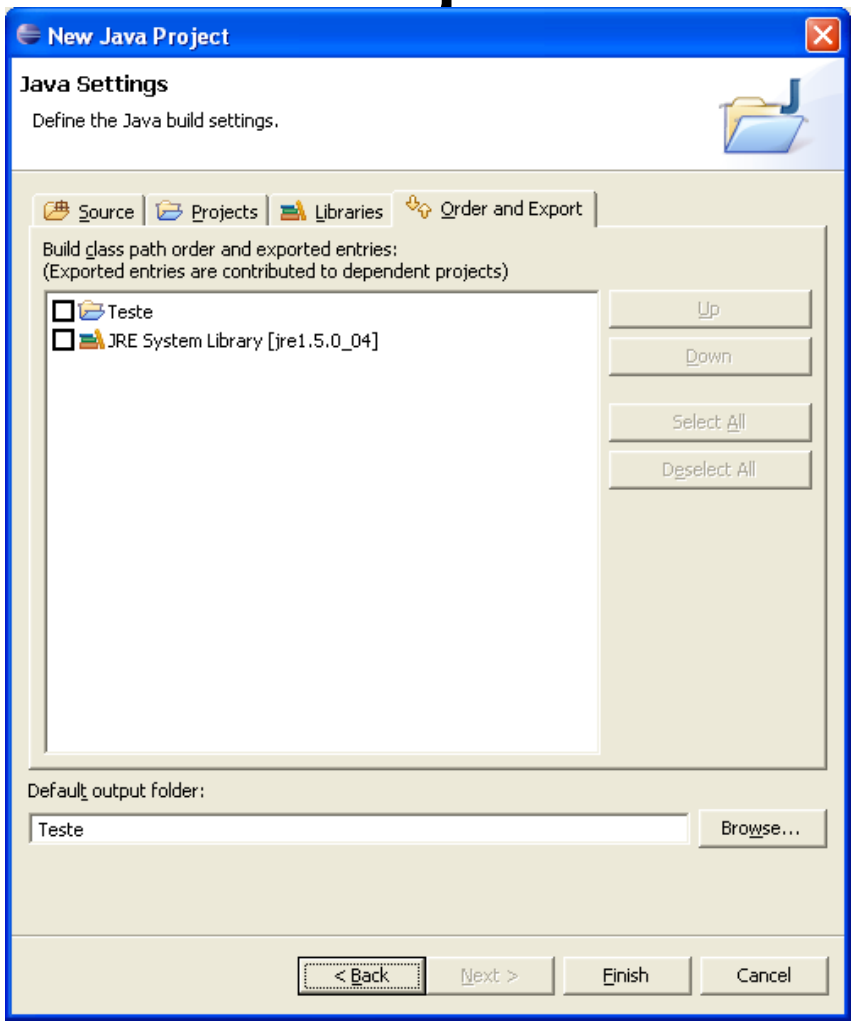
Criando Projetos – Passo 2

- Escolha
  - Nome do Projeto
  - Projeto vazio ou criado a partir de código pré-existente
  - JDK alvo
  - Project Layout



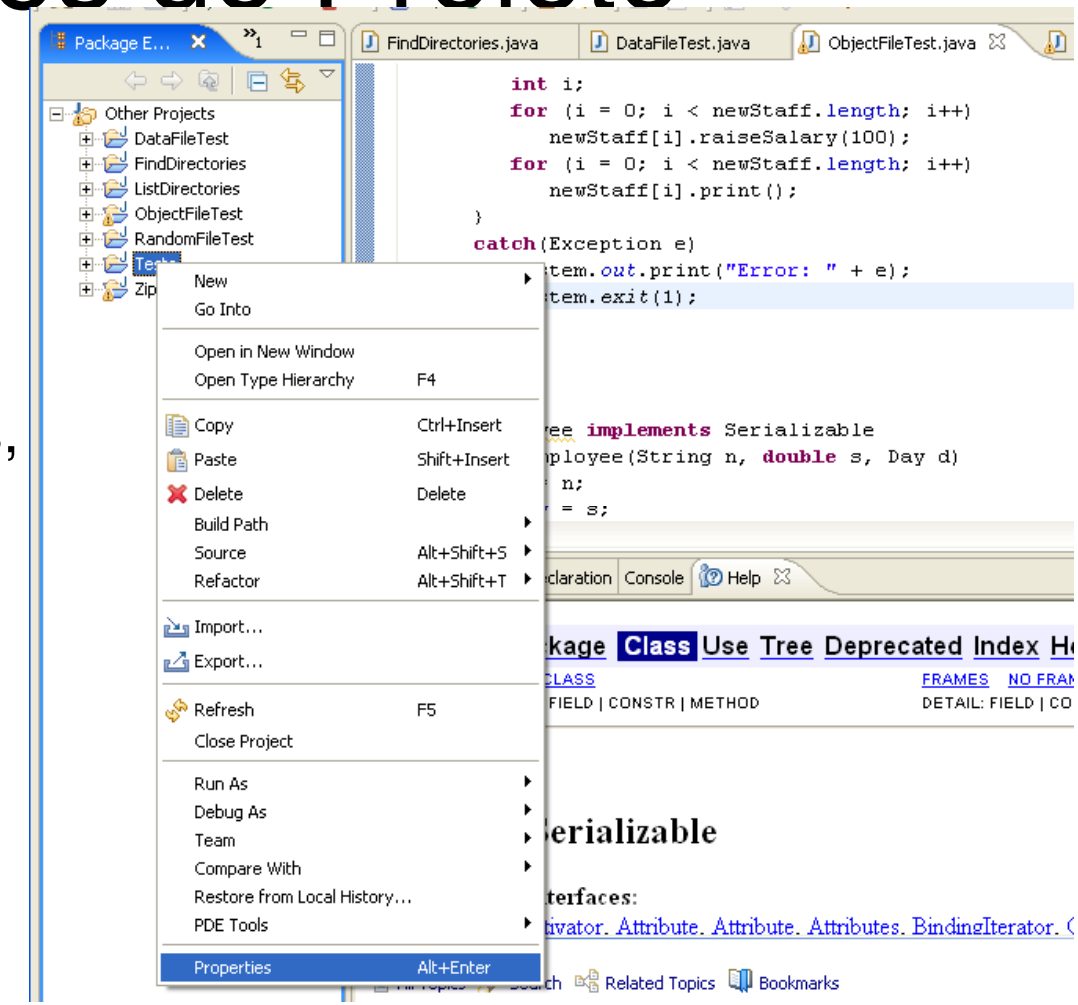
Configuração do Projeto

- Escolha dos diretórios com código-fonte
- Bibliotecas utilizadas
- Projetos requeridos
- Ordem de importação/exportação



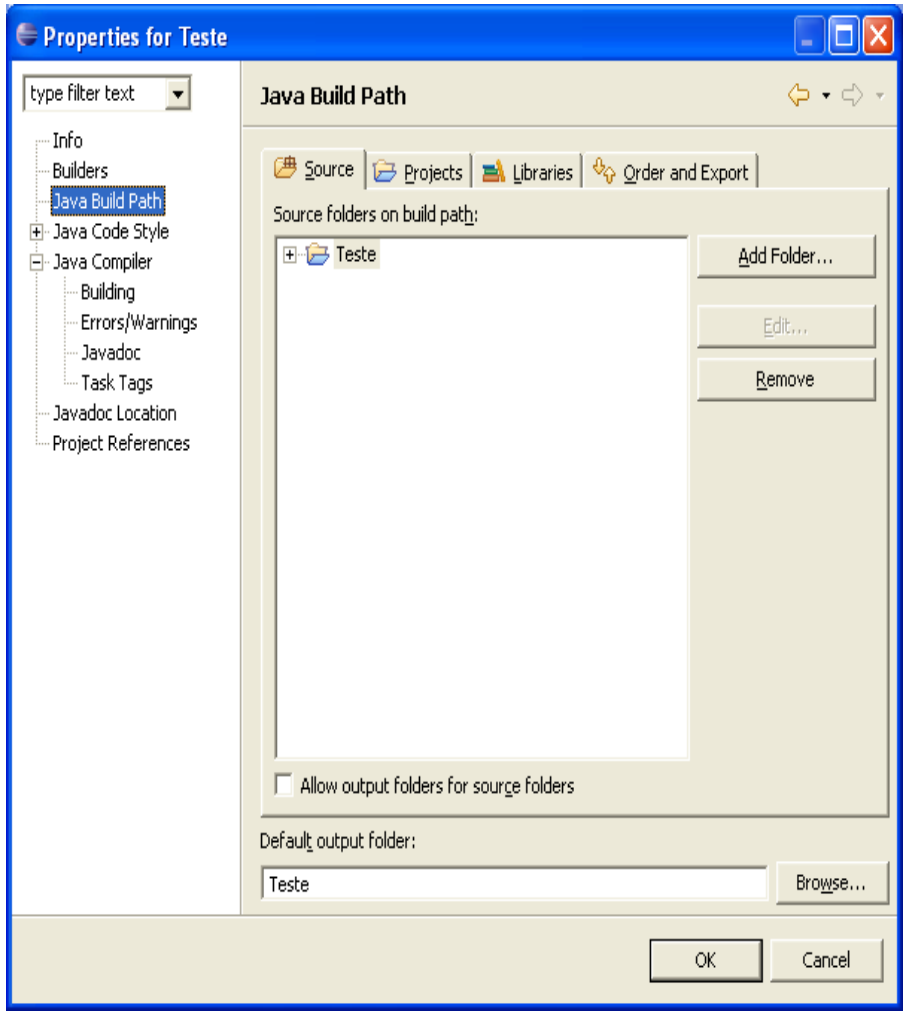
Opções de Projeto

- Criação de elementos do Projeto
  - Classes, interfaces, etc.
- Refactoring
- Propriedades,etc.

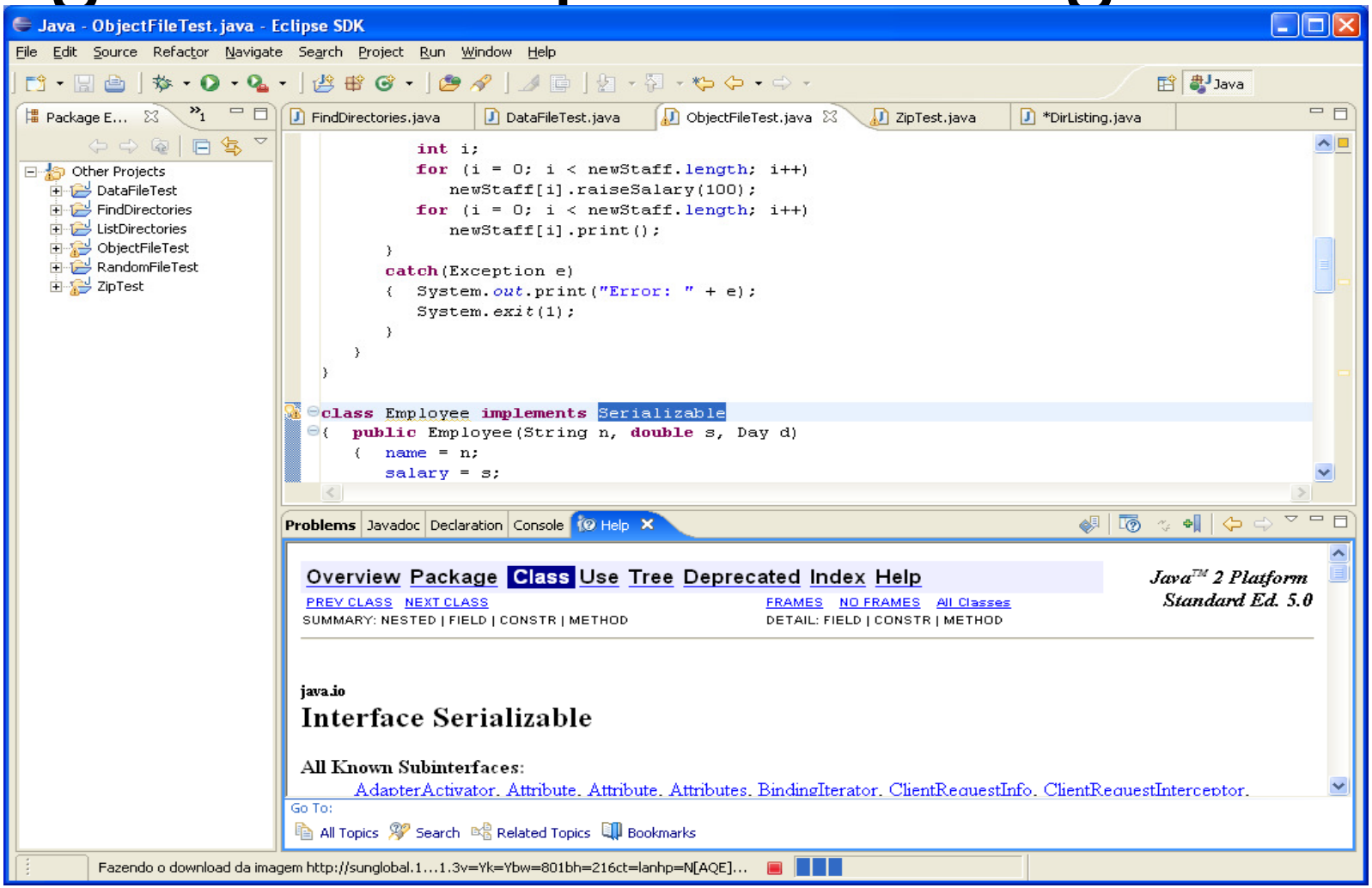


Propriedades do Projeto

- Configuração do Classpath e acesso a bibliotecas
- Configuração de diretórios de destino e fonte
- Configuração de destino do Javadoc
- Referências a outros projetos, etc.



Alguns Exemplos de Programas



Alguns Exemplos de Programa

- Exemplo 1:

```
public class Hello {
   public static void main(String args[])
   {  System.out.println(“Hello World!”);
   }
}
```

- Compile e Execute o programa acima através do Eclipse

Exemplo 2

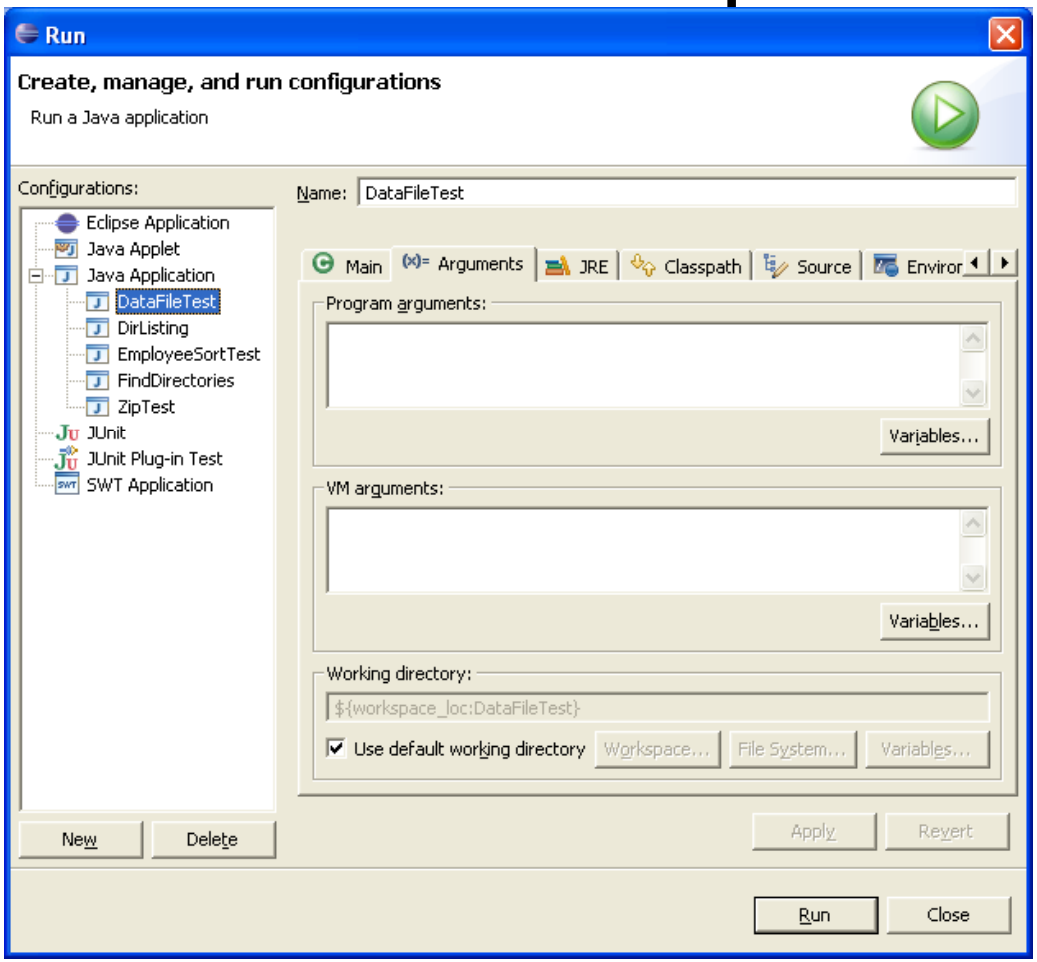
- **File: ShowTwoArgs.java**

```
public class ShowTwoArgs {
  public static void main(String[] args) {
    System.out.println("First arg: " +
                       args[0]);
    System.out.println("Second arg: " +
                       args[1]);
  }
}
```

Compile e Execute o programa acima

Inserindo argumentos para os programas através do Eclipse

- Menu Run | Run ...
  - Tab Arguments



Exemplo 3 – Usando Loops

```
public class ShowArgs {
  public static void main(String[] args) {
    for(int i=0; i<args.length; i++) {
      System.out.println("Arg " + i +
                         " is " +
                         args[i]);
    }
  }
}
```

Exemplo 4 – Loops Aninhados

```
public class TriangleArray {
  public static void main(String[] args) {

    int[][] triangle = new int[10][];

    for(int i=0; i<triangle.length; i++) {
      triangle[i] = new int[i+1];
    }

    for (int i=0; i<triangle.length; i++) {
      for(int j=0; j<triangle[i].length; j++) {
        System.out.print(triangle[i][j]);
      }
      System.out.println();
    }
  }
}
```

Exemplo 5

```
class Ship3 {
  public double x, y, speed, direction;
  public String name;

  public Ship3(double x, double y,
               double speed, double direction,
               String name) {
    this.x = x; // "this" differentiates instance vars
    this.y = y; //  from local vars.
    this.speed = speed;
    this.direction = direction;
    this.name = name;
  }

  private double degreesToRadians(double degrees) {
    return(degrees * Math.PI / 180.0);
  }
```

Exemplo 5

```
  public void move() {
    double angle = degreesToRadians(direction);
    x = x + speed * Math.cos(angle);
    y = y + speed * Math.sin(angle);
  }
  public void printLocation() {
    System.out.println(name + " is at ("
                       + x + "," + y + ").");
  }
}

public class Test3 {
  public static void main(String[] args) {
    Ship3 s1 = new Ship3(0.0, 0.0, 1.0,   0.0, "Ship1");
    Ship3 s2 = new Ship3(0.0, 0.0, 2.0, 135.0, "Ship2");
    s1.move();
    s2.move();
    s1.printLocation();
    s2.printLocation();
  }
}
```

Saída do Exemplo 5

- **Compiling and Running:**

```
javac Test3.java
java Test3
```

- **Output:**

```
Ship1 is at (1,0).
Ship2 is at (-1.41421,1.41421).
```

Onde estão as classes ? CLASSPATH

- **The `CLASSPATH` environment variable defines a list of directories in which to look for classes**
  - Default = current directory and system libraries
  - Best practice is to not set this when first learning Java!
- **Setting the CLASSPATH**

```
set CLASSPATH = .;C:\java;D:\cwp\echoserver.jar
setenv CLASSPATH .:~/java:/home/cwp/classes/
```

  - **The period indicates the current working directory**
- **Supplying a CLASSPATH**

```
javac -classpath .;D:\cwp  WebClient.java
java -classpath .;D:\cwp WebClient
```

Pacotes...

- **A package lets you group classes in subdirectories to avoid accidental name conflicts**
  - To create a package:
    1. Create a subdirectory with the same name as the desired package and place the source files in that directory
    2. Add a package statement to each file

```
package packagename;
```

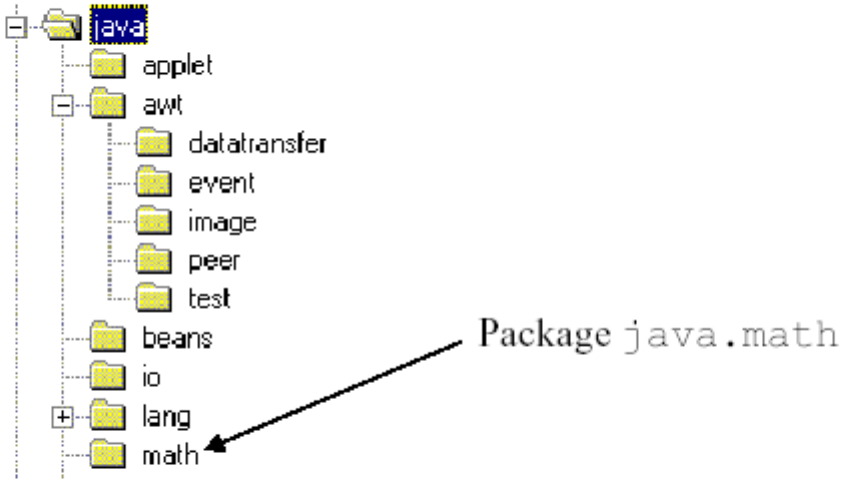
    3. Files in the main directory that want to use the package should include

```
import packagename.*;
```

- **The package statement must be the first statement in the file**
- **If a package statement is omitted from a file, then the code is part of the default package that has no name**

Pacotes...

- **The package hierarchy reflects the file system directory structure**



  - The root of any package must be accessible through a Java system default directory or through the `CLASSPATH` environment variable

Exercício

- Criar packages hello, triangle, showArgs e ship
- Mover programas correspondentes
- Definir variáveis de instância de Ship como private e criar métodos de acesso

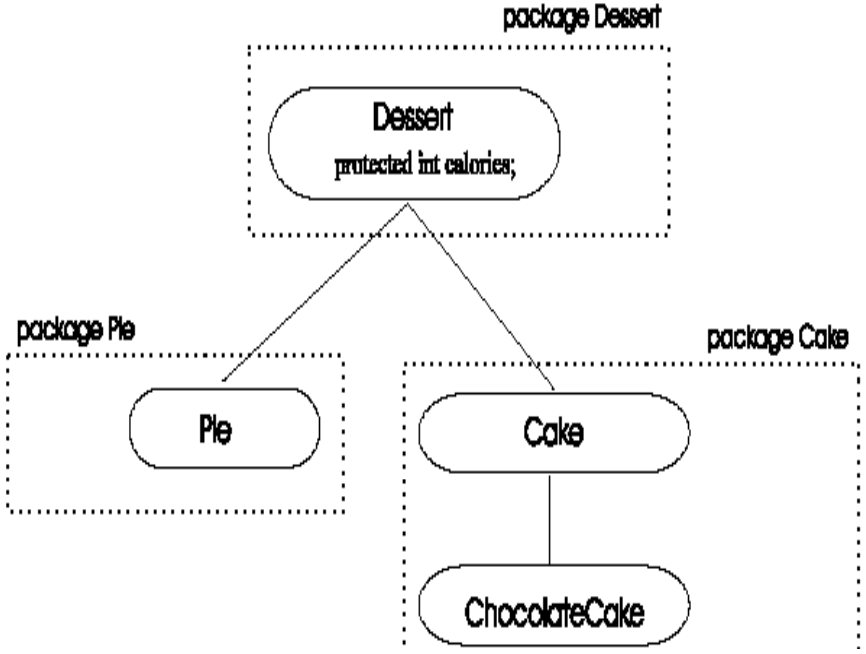
Mais sobre modificadores (métodos e variáveis)

- **public**: o método ou variável ao qual se refere é acessível de “qualquer lugar” no código
  - Uma classe deve ser declarada public para ser acessível por outras classes
  - Uma classe pública deve estar declarada num arquivo com o mesmo nome da classe. Ex. “ public class Ship ...” deve estar no arquivo Ship.java
- **private:** O método ou variável ao qual se refere é acessível exclusivamente por métodos da mesma classe
  - Declarar uma variável private a faz acessível pelo resto do código apenas através de métodos públicos

Mais sobre modificadores (métodos e variáveis)

- protected: Acessível apenas a variáveis e métodos da classe, das classes filhas (herdadas) e das classes que pertencem ao mesmo pacote
  - Variáveis e métodos protected são herdados mesmo por classes que não pertencem ao mesmo pacote
- [default]: Similar ao protected, exceto por:
  - Variáveis e métodos [default] NÃO são herdados por classes que não pertencem ao mesmo pacote
  - Em outras palavras: Variáveis e métodos [default] são herdadas APENAS por classes que pertencem ao mesmo pacote

Exemplo - protected

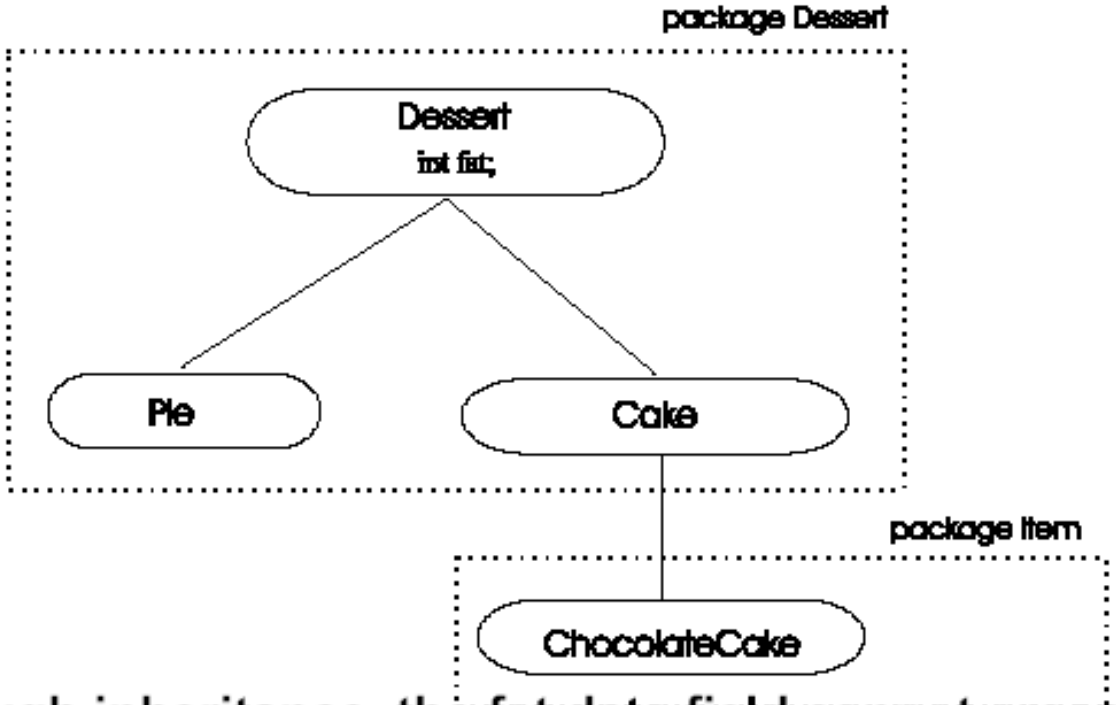


• Cake, ChocolateCake e Pie herdam o campo calories

•Entretanto, se o código na classe Cake tem uma referência ao objeto Pie, o campo calories de Pie não pode ser acessado em Cake.

- •Campos protected de uma classe não podem ser acessados fora de um mesmo pacote, exceto se na mesma árvore de hierarquia

Exemplo – [default]



- **Even through inheritance, the fat data field cannot cross the package boundary**
  - Thus, the `fat` data field is accessible through any `Dessert`, `Pie`, and `Cake` object within any code in the `Dessert` package
  - However, the `ChocolateCake` class does not have a fat data field, nor can the fat data field of a `Dessert`, `Cake`, or `Pie` object be accessed from code in the `ChocolateCake` class

Sumário de modificadores de acesso

Data Fields and Methods	Modifiers			
	public	protected	default	private
Accessible from same class?	yes	yes	yes	yes
Accessible to classes (**nonsubclass**) from the **same package**?	yes	yes	yes	no
Accessible to **subclass** from the **same package**?	yes	yes	yes	no
Accessible to classes (**nonsubclass**) from **different package**?	yes	no	no	no
Accessible to **subclasses** from **different package**?	yes	no	no	no
Inherited by subclass in the same package?	yes	yes	yes	no
Inherited by subclass in different package?	yes	yes	no	no

Outros modificadores

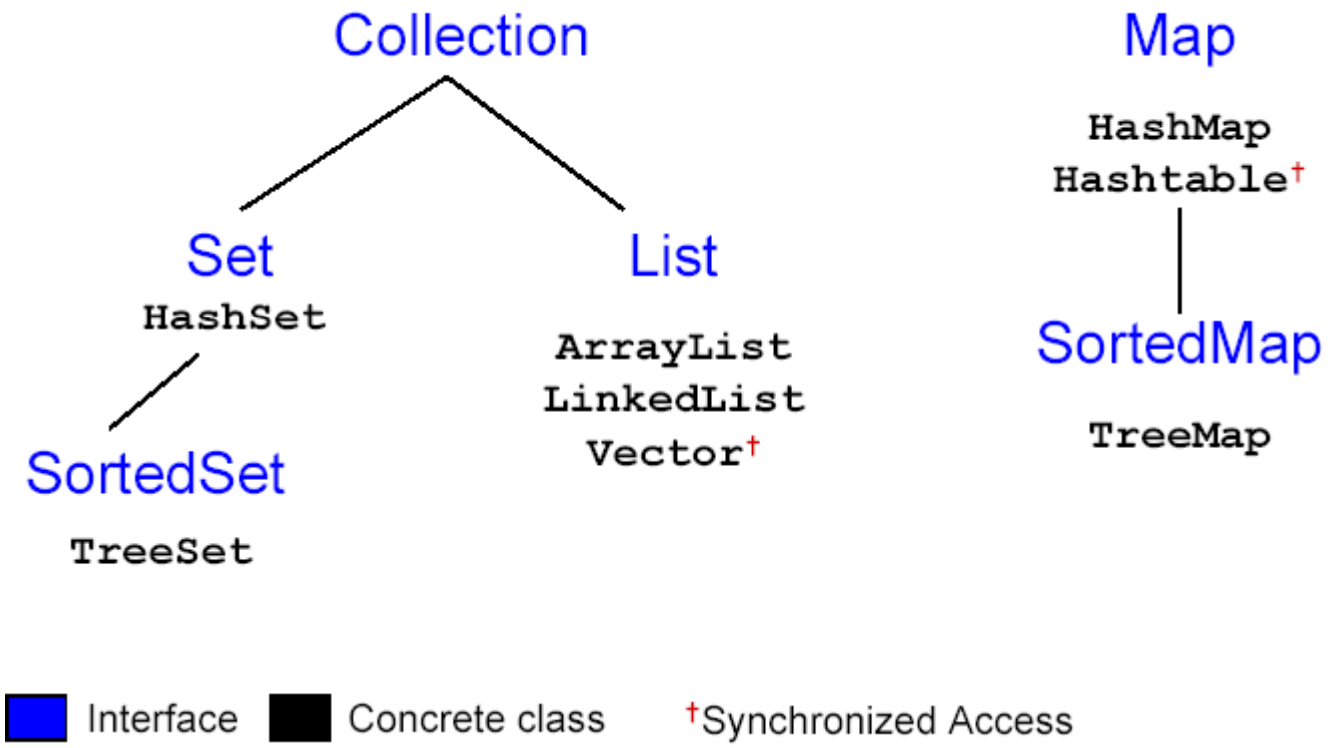
- **final**
  - For a class, indicates that it cannot be subclassed
  - For a method or variable, cannot be changed at runtime or overridden in subclasses
- **synchronized**
  - Sets a lock on a section of code or method
  - Only one thread can access the same synchronized code at any given time
- **transient**
  - Variables are not stored in serialized objects sent over the network or stored to disk
- **native**
  - Indicates that the method is implement using C or C++

Algumas Diretrizes para gerar bom código

- Uma classe deve o menor número possível de métodos públicos (mas deve ter pelo menos um!)
  - Isto diminui o acoplamento entre as classes do projeto, o que facilita a manutenção
- Deve-se evitar variáveis públicas. Crie métodos de acesso get/set. Exemplo:

```
Class Ship {
 private double speed;
public double getSpeed() { return speed; }
public void setSpeed(double speed) {
  this.speed=speed;}
}
```

Estruturas de Dados no Java 2



Collection Interfaces

- **Collection**
  - Abstract class for holding groups of objects
- **Set**
  - Group of objects containing no duplicates
- **SortedSet**
  - Set of objects (no duplicates) stored in ascending order
  - Order is determined by a `Comparator`
- **List**
  - *Physically* (versus logically) ordered sequence of objects
- **Map**
  - Stores objects (unordered) identified by unique keys
- **SortedMap**
  - Objects stored in ascending order based on their key value
  - Neither duplicate or `null` keys are permitted

Duas Estruturas de Dados Muito Úteis

- Vector
  - Um array de ***Object*** de tamanho variável
  - Tempo para acessar um objeto é independente da sua posição na lista
  - No jdk 1.2 ou superior, pode-se utilizar ArrayList
  - ArrayList não é sincronizado (thread-safe), por isso tende a ser mais rápido
- Hashtable
  - Armazena pares: nome-valor como Object
  - Valores não podem ser nulos
  - No jdk 1.2 ou superior, pode-se utilizar HashMap
  - HashMap não é sincronizado (thread-safe), por isso tende a ser mais rápido

Métodos úteis em Vector

- **addElement/insertElementAt/setElementAt**
  - Add elements to the vector
- **removeElement/removeElementAt**
  - Removes an element from the vector
- **firstElement/lastElement**
  - Returns a reference to the first and last element, respectively (without removing)
- **elementAt**
  - Returns the element at the specified index
- **indexOf**
  - Returns the index of an element that equals the object specified
- **contains**
  - Determines if the vector contains an object

Utilizando Vector

- **elements**
  - Returns an `Enumeration` of objects in the vector

```
Enumeration elements = vector.elements();
while(elements.hasMoreElements()) {
  System.out.println(elements.nextElement());
}
```

- **size**
  - The number of elements in the vector
- **capacity**
  - The number of elements the vector can hold before becoming resized

Métodos úteis em Hashtable

- **put/get**
  - Stores or retrieves a value in the hashtable
- **remove/clear**
  - Removes a particular entry or all entries from the hashtable
- **containsKey/contains**
  - Determines if the hashtable contains a particular key or element
- **keys/elements**
  - Returns an enumeration of all keys or elements, respectively
- **size**
  - Returns the number of elements in the hashtable
- **rehash**
  - Increases the capacity of the hashtable and reorganizes it

Exemplo de Uso de um Hashtable

```
import java.util.Hashtable;

public class ExemploHashtable {
    public static void main(String[] args) {
            Hashtable numbers = new Hashtable();
        numbers.put("one", new Integer(1));
        numbers.put("two", new Integer(2));
        numbers.put("three", new Integer(3));
        String key="three";
        Integer n = (Integer)numbers.get(key);
        if (n != null) {
            System.out.println(key+" = " + n);
        }
    }
}
```

Resultado

- >three = 3

Exemplo de Uso de uma Coleção Vector

```
import java.util.Iterator;
import java.util.Vector;

public class Colecoes {

      public static void main(String[] args) {
            Vector vetStrings=new Vector();
            for(int i=1;i<=5;i++)
              vetStrings.add("Linha "+i);
            //Laços de Iteração
            for (Iterator iter = vetStrings.iterator(); iter.hasNext();) {
                        String element = (String) iter.next();
                         System.out.println(element);
            }
      }
}
```

Resultado

>Linha 1

Linha 2

Linha 3

Linha 4

Linha 5

Classes Genéricas no JDK 5.0

- Classe genéricas: classes que podem ser parametrizadas para trabalharem sobre classes específicas
  - Tipos parametrizáveis: (Design Patterns, GoF)
  - Templates: C++
  - Classes genéricas: Java,C#

Avanços em Collections no JDK 5.0

- Coleções genéricas:

```
Vector<String> vetStrings=new
   Vector<String>();
```

- Laços de Iteração Aprimorados

```
for(String element: vetStrings) {
  System.out.println(element);
}
```

Exemplo de Uso de uma Coleção Genérica

```
import java.util.Vector;

public class ColecoesGenericas {

    public static void main(String[] args) {

        Vector<String > vetStrings=new Vector<String>();
        for(int i=1;i<=5;i++)
          vetStrings.add("Linha "+i);
        //Laços de Iteração Aprimorados
        for(String element: vetStrings) {
          System.out.println(element);
        }
    }
}
```

Resultado

>Linha 1

Linha 2

Linha 3

Linha 4

Linha 5

HashTable Genérico

```
import java.util.Hashtable;

public class ExemploHashtableGenerico {
    public static void main(String[] args) {
            Hashtable<String,Integer> numbers = new Hashtable<String,Integer>();
        numbers.put("one", new Integer(1));
        numbers.put("two", new Integer(2));
        numbers.put("three", new Integer(3));
        String key="three";
        Integer n = numbers.get(key);
        if (n != null) {
            System.out.println(key+" = " + n);
        }
    }
}
```

Formatando Texto

- Números
- Use DecimalFormat para formatação
  - Nas versões mais novas, Java implementa um método printf, semelhante ao C/C++
- Abordagem DecimalFormat
  1. Crie um objeto DecimalFormat descrevendo a formatação
     - DecimalFormat formatter= new DecimalFormat("#,###.##");
  2. Utilize o método de instância format para convertar valores em string formatadas
     - formatter.format(24.99);

Formatando Números

Symbol	Meaning
0	Placeholder for a digit.
#	Placeholder for a digit. If the digit is leading or trailing zero, then don't display.
.	Location of decimal point.
,	Display comma at this location.
-	Minus sign.
E	Scientific notation. Indicates the location to separate the mattissa from the exponent.
%	Multipy the value by 100 and display as a percent.

Exemplo – Formatação de Números

```
import java.text.*;

public class NumFormat {
  public static void main (String[] args) {
    DecimalFormat science = new DecimalFormat("0.000E0");
    DecimalFormat plain = new DecimalFormat("0.0000");

    for(double d=100.0; d<140.0; d*=1.10) {
      System.out.println("Scientific: " + science.format(d) +
                         " and Plain: " + plain.format(d));
    }
  }
}
```

Resultado – Formatando Números

```
> java NumFormat

Scientific: 1.000E2 and Plain: 100.0000
Scientific: 1.100E2 and Plain: 110.0000
Scientific: 1.210E2 and Plain: 121.0000
Scientific: 1.331E2 and Plain: 133.1000
```

Trabalhando com Datas

- Classe Date
  - Vários métodos deprecated!

- Classe Calendar
  - Uso: Calendar cal=Calendar.getInstance();
    - cal.setTime(new Date());
    - Date hoje=cal.getTime();
    - cal.setTime(int year,int month,int day, int hour,int minute,int second);
    - cal.set(Calendar.MONTH,Calendar.SEPTEMBER);

Formatando datas

```
/** @param Data a ser formatada
    @return Data formatada em texto DD/MM/YY
*/
public static String FormatDate_UK(Date d) {
      return
  DateFormat.getDateInstance(DateFormat.SHORT,
  Locale.UK).format(d);
}
/** @param Data a ser formatada
    @return Data formatada em texto MM/DD/YY
*/
public static String FormatDate_USA(Date d) {
      return
  DateFormat.getDateInstance(DateFormat.SHORT,
  Locale.US).format(d);
}
```

Formatando datas

```
/** @param Data a ser formatada
    @return Data formatada em texto corrido na
  língua do computador
*/
public static String FormatDate_Default(Date
  d) {
     return
  DateFormat.getDateInstance(DateFormat.LONG,
  Locale.getDefault()).format(d);
}
```

- Em computador configurado para português (br):
  - Retorna: 7de março de 2013

Formatando datas – Abordagem Direta

```
public static String FormatDate(Date d) {
      if(d==null)
        return new String("");
      Calendar cal=Calendar.getInstance();
      cal.setTime(d);
      String year=Integer.toString(cal.get(Calendar.YEAR));
      String month=Integer.toString(cal.get(Calendar.MONTH)+1);
      String day=Integer.toString(cal.get(Calendar.DAY_OF_MONTH));
      String hour=Integer.toString(cal.get(Calendar.HOUR_OF_DAY));
      String minute=Integer.toString(cal.get(Calendar.MINUTE));
      String second=Integer.toString(cal.get(Calendar.SECOND));
      if(month.length()<2)
       month="0"+month;
      if(day.length()<2)
       day="0"+day;
      if(hour.length()<2)
       hour="0"+hour;
      if(minute.length()<2)
       minute="0"+minute;
      if(second.length()<2)
       second="0"+second;

      return day+"/"+month+"/"+year+" "+hour+":"+minute+":"+second;

    }
```

Criando datas

```
public static Date getDate
    (int year,int month,int day) {
       Calendar cal=Calendar.getInstance();
        cal.set(year,month,day,0,0,0);
      return cal.getTime();
     }
```

- Programa de Testes de Data

```
public class DataTest {
/* funções de manipulação de datas....
......... */

public static void main(String[] args) {
        Date hoje=new Date();

            System.out.println("FormatDate_UK:
   "+DatasTest.FormatDate_UK(hoje));
            System.out.println("FormatDate_USA:
   "+DatasTest.FormatDate_USA(hoje));
            System.out.println("FormatDate_Default:
   "+DatasTest.FormatDate_Default(hoje));
            Date outroDia=getDate(2007,4,16); // mes inicia em zero
            System.out.println("FormatDate_Default:
   "+DatasTest.FormatDate_Default(outroDia));

   }
}
```

Resultado

- FormatDate_UK: 16/04/07
- FormatDate_USA: 4/16/07
- FormatDate_Default: 16 de Abril de 2007
- FormatDate_Default: 16 de Maio de 2007

Calendar

- Pode-se utilizar GregorianCalendar (subclasse de Calendar) para adiantar ou movimentar datas
  - GregorianCalendar gcal=new GregorianCalendar();
- Adiantando ou atrasando datas
  - cal.add(10,Calendar.MONTH)
  - cal.add(-15,Calendar.YEAR)

- Acessando campos de uma data
  - cal.get(Calendar.MONTH) //retorna inteiro
  - cal.get(Calendar.SECONDS)

Sumário de Hoje

- Introdução ao Ambiente Eclipse
  - Criando workspaces e projetos
  - Compilando e executando programas
- Desenvolvimento de Programas básicos (modo texto)
  - Primeiros Programas
  - Javadoc, Os conceitos de CLASSPATH, package e import
  - Formatando texto

- **O sistema de I/O Orientado a Objetos do Java**
  - Acessando arquivos Texto
  - Acessando arquivos Binários
  - Serialização e armazenamento de Objetos

Tratamento de Erros: Tradicional

- O tratamento de erros em linguagens sem Exceções, gera um código “sujo” com código tratamento de erro:
  - ret=funcao1();
  - if(ret==ERRO)
    - //Trata erro

    ret=funcao2();
  - if(ret==ERRO)

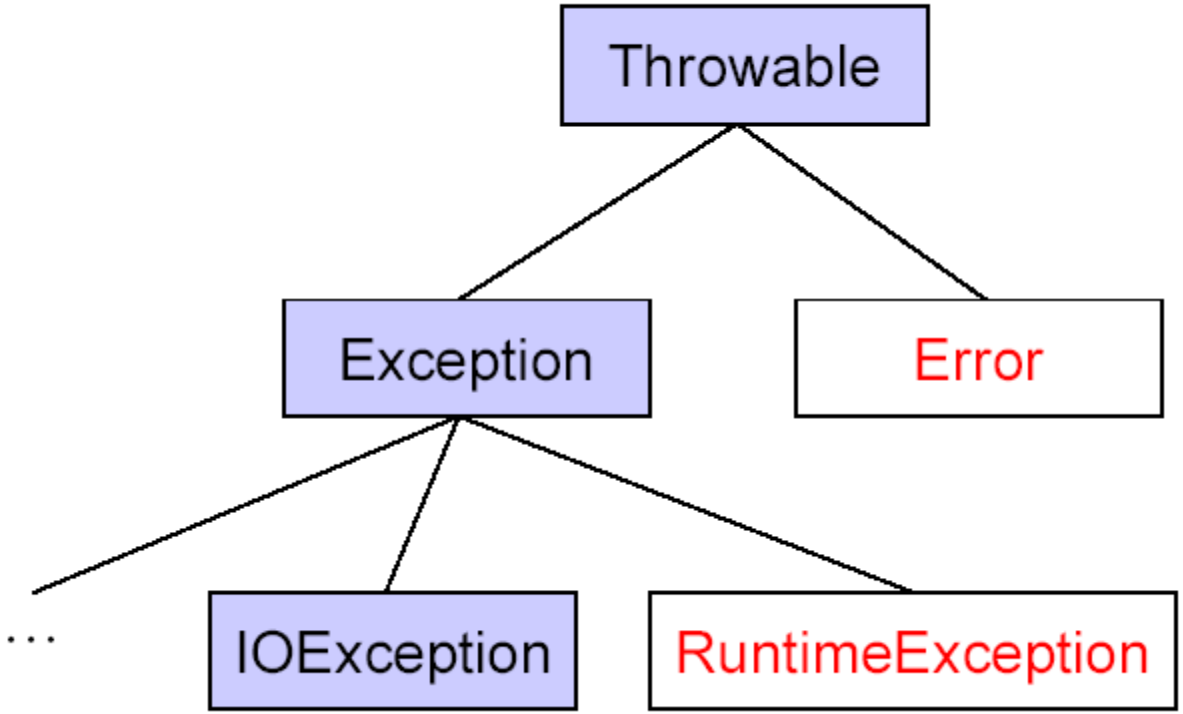
    //Trata Erro 2

Tratamento de Erros: Exceções

- Em Java, o sistema de tratamento de erros é baseado exceções
  - Exceções devem ser tratados em blocos **try/catch**
  - Quando ocorre uma exceção esta é direcionada para o correspondente **catch**
- Formato:

```
try {
  statement1;
  statement2;
  ...
} catch(SomeException someVar) {
  handleTheException(someVar);
}
```

Diagrama Simplicado de Exceções



Try-catch

- Um bloco try pode ter associados vários blocos catch

```
try {
  ...
} catch (ExceptionType1 var1) {
  // Do something
} catch (ExceptionType2 var2) {
  // Do something else
}
```

- A exceção será tratado pelo bloco catch mais específico
- Caso não seja encontrado algum apropriado, a exceção será direcionada para blocos try mais externos
  - Caso não seja encontrado nenhum try apropriado dentro do método, este irá jogar a exceção

Um exemplo de Try-catch

```
...
BufferedReader in = null;
String lineIn;
try {
  in = new BufferedReader(new FileReader("book.txt"));
  while((lineIn = in.readLine()) != null) {
    System.out.println(lineIn);
  }
  in.close();
} catch (FileNotFoundException fnfe ) {
  System.out.println("File not found.");
} catch (EOFException eofe) {
  System.out.println("Unexpected End of File.");
} catch (IOException ioe) {
  System.out.println("IOError reading input: " + ioe);
  ioe.printStackTrace(); // Show stack dump
}
```

A cláusula finally

- Ao final de um conjunto de blocos catch pode-se, opcionalmente, incluir uma cláusula **finally**. Caso nenhum bloco catch, seja executado o **finally** será sempre executado

```
try {
   ...
} catch (SomeException someVar) {
   // Do something
} finally {
   // Always executed
}
```

O sistema de IO em Java

- A biblioteca java.io tem mais de 60 classes(stream) de input/output
- Dois grandes grupos
  - Classes baseadas em tráfego de bytes
    - DataStreams:
  - Classes baseados em tráfego de caracteres
    - Reader e Writer
- Em qualquer operação de IO pode ocorrer uma exceção do tipo IOException

A classe File

- **A `File` object can refer to either a file or a directory**

```
File file1 = new File("data.txt");
File file1 = new File("C:\java");
```

  - To obtain the path to the current working directory use

```
System.getProperty("user.dir");
```

  - **To obtain the file or path separator use**

```
System.getProperty("file.separator");
System.getProperty("path.separator");
```

or

```
File.separator()
File.pathSeparator()
```

Métodos úteis em File

- **isFile/isDirectory**
- **canRead/canWrite**
- **length**
  - Length of the file in bytes (`long`) or `0` if nonexistant
- **list**
  - If the `File` object is a directory, returns a `String` array of all the files and directories contained in the directory; otherwise, `null`
- **mkdir**
  - Creates a new subdirectory
- **delete**
  - Deletes the directory and returns `true` if successful
- **toURL**
  - Converts the file path to a URL object

Exemplo de File: Programa que lista o diretório do usuário

```
import java.io.*;

public class DirListing {
  public static void main(String[] args) {

    File dir = new File(System.getProperty("user.dir"));

    if(dir.isDirectory()){
      System.out.println("Directory of " + dir);
      String[] listing = dir.list();
      for(int i=0; i<listing.length; i++) {
        System.out.println("\t" + listing[i]);
      }
    }
  }
}
```

Resultado

```
> java DirListing

Directory of C:\java\
        DirListing.class
        DirListing.java
        test
        TryCatchExample.class
        TryCatchExample.java
        XslTransformer.class
        XslTransformer.java
```

Exercício: Listar o conteúdo de um diretório

- Faça um programa em Java que liste o conteúdo de um diretório passado na linha de comando ou o diretório corrente, caso não seja solicitado nenhum
- Exemplos de uso:
  - C:>eclipse\ java ListDir “c:\Arquivos de Programa”
    - Lista o conteúdo do diretório Arquivos de Programa
  - C:>eclipse\java ListDir
    - Lista o conteúdo do diretório eclipse:

Solução

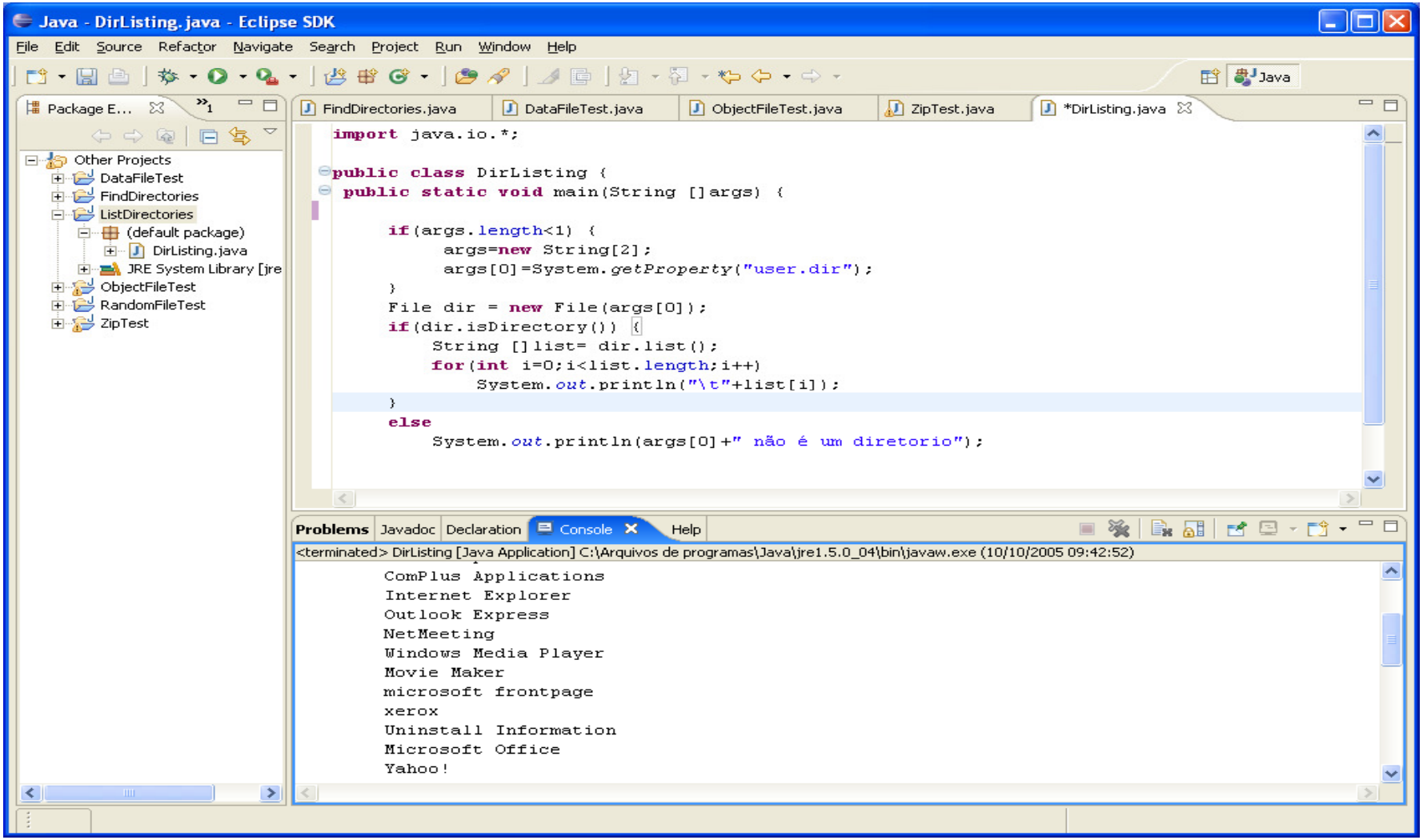
```
import java.io.*;

public class DirListing {
 public static void main(String []args) {

    if(args.length<1) {
          args=new String[2];
          args[0]=System.getProperty("user.dir");
    }
    File dir = new File(args[0]);
    if(dir.isDirectory()) {
         String []list= dir.list();
         for(int i=0;i<list.length;i++)
                 System.out.println("\t"+list[i]);
    }
    else
         System.out.println(args[0]+" não é um diretorio");


 }
}
```

Resultado


Java - DirListing.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Package E...
Other Projects
DataFileTest
FindDirectories
ListDirectories
(default package)
DirListing.java
JRE System Library [jre
ObjectFileTest
RandomFileTest
ZipTest
FindDirectories.java
DataFileTest.java
ObjectFileTest.java
ZipTest.java
*DirListing.java
import java.io.*;
public class DirListing {
public static void main(String []args) {
if(args.length<1) {
args=new String[2];
args[0]=System.getProperty("user.dir");
}
File dir = new File(args[0]);
if(dir.isDirectory()) {
String []list= dir.list();
for(int i=0;i<list.length;i++)
System.out.println("\t"+list[i]);
}
else
System.out.println(args[0]+" não é um diretorio");
Problems Javadoc Declaration Console Help
<terminated> DirListing [Java Application] C:\Arquivos de programas\Java\jre1.5.0_04\bin\javaw.exe (10/10/2005 09:42:52)
ComPlus Applications
Internet Explorer
Outlook Express
NetMeeting
Windows Media Player
Movie Maker
microsoft frontpage
xerox
Uninstall Information
Microsoft Office
Yahoo!

Classes para escrever Texto

Desired ...	Methods	Construction
Character File Ouput	**FileWriter** write(int char) write(byte[] buffer) write(String str)	File file = new File("filename"); FileWriter fout = new FileWriter(file); *or* FileWriter fout = new FileWriter("filename");
Buffered Character File Output	**BufferedWriter** write(int char) write(char[] buffer) write(String str) newLine()	File file = new File("filename"); FileWriter fout = new FileWriter(file); BufferedWriter bout = new BufferedWriter(fout); *or* BufferedWriter bout = new BufferedWriter(new FileWriter(new File("filename")));

Classes para escrever Texto

Desired ...	Methods	Construction

Character Output	**PrintWriter**
	write(int char)
	write(char[] buffer)
	writer(String str)
	print(…)
	println(…)
		FileWriter fout = new FileWriter("filename");
		PrintWriter pout = new PrintWriter(fout);
		or
		PrintWriter pout = new PrintWriter(
		new FileWriter("filename"));
		or
		
		PrintWriter pout = new PrintWriter(
		new BufferedWriter(
		new FileWriter("filename")));

Exemplo de Escrita de Arquivo de Texto

```
import java.io.*;

public class CharacterFileOutput {
  public static void main(String[] args) {
    FileWriter out = null;

    try {
      out = new FileWriter("book.txt");
      System.out.println("Encoding: " + out.getEncoding());
      out.write("Core Web Programming");
      out.close();
      out = null;
    } catch(IOException ioe) {
      System.out.println("IO problem: " + ioe);
      ioe.printStackTrace();
      try {
        if (out != null) {
          out.close();
        }
      } catch(IOException ioe2) { }
    }
  }
}
```

Codificação de caracteres

```
> java CharacterFileOutput
Encoding: Cp1252

> type book.txt
Core Web Programming
```

- **Note: Cp1252 is Windows Western Europe / Latin-1**
  - To change the system default encoding use
    `System.setProperty("file.encoding", "encoding");`
  - To specify the encoding when creating the output steam, use an `OutputStreamWriter`

```
OutputStreamWriter out =
   new OutputStreamWriter(
     new FileOutputStream("book.txt", "8859_1"));
```

Classes de Leitura de Streams de Texto

Desired ...	Methods	Construction
Character File Input	**FileReader** read() read(char[] buffer)	File file = new File("filename"); FileReader fin = new FileReader(file); *or* FileReader fin = new FileReader("filename");
Buffered Character File Input	**BufferedReader** read() read(char[] buffer) readLine()	File file = new File("filename"); FileReader fin = new FileReader(file); BufferedReader bin = new BufferedReader(fin); *or* BufferedReader bin = new BufferedReader(new FileReader(new File("filename")));

UCS Transformation Format – UTF 8

- **UTF encoding represents a 2-byte Unicode character in 1-3 bytes**
  - Benefit of backward compatibility with existing ASCII data (one-byte over two-byte Unicode)
  - Disadvantage of different byte sizes for character representation

UTF Encoding	
Bit Pattern	**Representation**
0xxxxxxx	ASCII (0x0000 - 0x007F)
10xxxxxx	Second or third byte
110xxxxx	First byte in a 2-byte sequence (0x0080 - 0x07FF)
1110xxxx	First byte in a 3-byte sequence (0x0800 - 0xFFFF)

Exemplo - FileReader

```
import java.io.*;

public class CharacterFileInput {
  public static void main(String[] args) {

    File file = new File("book.txt");
    FileReader in = null;

    if(file.exists()) {
      try {
        in = new FileReader(file);
        System.out.println("Encoding: " + in.getEncoding());
        char[] buffer = new char[(int)file.length()];
        in.read(buffer);
        System.out.println(buffer);
        in.close();
      } catch(IOException ioe) {
        System.out.println("IO problem: " + ioe);
        ioe.printStackTrace();
        ...
      }
    }
  }
}
```

Resultado - FileReader

```
> java CharacterFileInput

Encoding: Cp1252
Core Web Programming
```

- **Alternatively, could read file one line at a time:**

```
BufferedReader in =
  new BufferedReader(new FileReader(file));
String lineIn;
while ((lineIn = in.readLine()) != null) {
  System.out.println(lineIn);
}
```

I/O em Arquivos (Streams) Binários

- **Handle byte-based I/O using a DataInputStream or DataOutputStream**

DataType	DataInputStream	DataOutputStream
byte	readByte	writeByte
short	readShort	writeShort
int	readInt	writeInt
long	readLong	writeLong
float	readFloat	writeFloat
double	readDouble	writeDouble
boolean	readBoolean	writeBoolean
char	readChar	writeChar
String	readUTF	writeUTF
byte[]	readFully	

  - The `readFully` method blocks until all bytes are read or an EOF occurs
  - Values are written in big-endian fashion regardless of computer platform

Classes para Escrita em Streams Binárias

Desired ...	Methods	Construction
Binary File Output bytes	**FileOutputStream** write(byte) write(byte[] buffer)	File file = new File("filename"); FileOutputStream fout = new FileOutputStream(file); *or* FileOutputStream fout = new FileOutputStream("filename");
Binary File Output byte short int long float double char boolean	**DataOutputStream** writeByte(byte) writeShort(short) writeInt(int) writeLong(long) writeFloat(float) writeDouble(double) writechar(char) writeBoolean(boolean) writeUTF(string) writeBytes(string) writeChars(string)	File file = new File("filename"); FileOutputStream fout = new FileOutputStream(file); DataOutputStream dout = new DataOutputStream(fout); *or* DataOutputStream dout = new DataOutputStream(new FileOutputStream(new File("filename")));

Classes para Escrita em Streams Binárias

Desired ...	Methods	Construction
Buffered Binary File Output	**BufferedOutputStream** flush() write(byte) write(byte[] buffer, int off, int len)	File file = new File("filename"); FileOutputStream fout = new FileOutputStream(file); BufferedOutputStream bout = new BufferedOutputStream(fout); DataOutputStream dout = new DataOutputStream(bout); *or* DataOutputStream dout = new DataOutputStream(new BufferedOutputStream(new FileOutputStream(new File("filename"))));

Exemplo – Escrita em Arquivos Binários

```
import java.io.*;

public class BinaryFileOutput {

  public static void main(String[] args) {
    int[] primes = { 1, 2, 3, 5, 11, 17, 19, 23 };
    DataOutputStream out = null;

    try {
      out = new DataOutputStream(
              new FileOutputStream("primes.bin"));

      for(int i=0; i<primes.length; i++) {
        out.writeInt(primes[i]);
      }
      out.close();
    } catch(IOException ioe) {
       System.out.println("IO problem: " + ioe);
       ioe.printStackTrace();
    }
  }
}
```

Classes para Leitura em Streams Binárias

Desired ...	Methods	Construction
Binary File Input bytes	**FileInputStream** read() read(byte[] buffer)	File file = new File("filename"); FileInputStream fin = new FileInputStream(file); *or* FileInputStream fin = new FileInputStream("filename");
Binary File Input byte short int long float double char boolean	**DataInputStream** readByte() readShort() readInt() readLong() readFloat() readDouble() readchar() readBoolean() readUTF() readFully(byte[] buffer)	File file = new File("filename"); FileInputStream fin = new FileInputStream(file); DataInputStream din = new DataInputStream(fin); *or* DataInputStream din = new DataInputStream(new FileInputStream(new File("filename")));

Classes para Leitura em Streams Binárias

Desired ...	Methods	Construction
Bufferred Binary File Input	**BufferedInputStream** read() read(byte[] buffer, int off, int len) skip(long)	File file = new File("filename"); FileInputStream fin = new FileInputStream(file); BufferedInputStream bin = new BufferedInputStream(fin); DataInputStream din = new DataInputStream(bin); *or* DataInputStream din = new DataInputStream(new BufferedInputStream(new FileInputStream(new File("filename"))));

Exemplo – Leitura de Arquivos Binários

```
import java.io.*;

public class BinaryFileInput {
  public static void main(String[] args) {

    DataInputStream in = null;
    File file = new File("primes.bin");
    try {
      in = new DataInputStream(
              new FileInputStream(file));
      int prime;
      long size = file.length()/4;  // 4 bytes per int
      for(long i=0; i<size; i++) {
        prime = in.readInt();
        System.out.println(prime);
      }
      in.close();
    } catch(IOException ioe) {
       System.out.println("IO problem: " + ioe);
       ioe.printStackTrace();
    }
  }
}
```

Resumo

- Um objeto File pode referir-se tanto a um arquivo quanto a um diretório
- Use classes Reader/Writer para lidar com streams baseadas em caracteres
  - Para ler linhas use BufferedReader
  - Use classes de formatação para a formatação de texto (Ex. DecimalFormat, package java.text)
- Use classes DataStream para lidar com streams baseadas em bytes.
- Associe um FileOutputStream a um DataOutputSteam para escrever em arquivos binários tipos básicos do Java
- Associe um FileInputStream a um DataInputStream para ler de arquivos binários usando tipos básicos do Java
- Para ler ou escrever objetos em streams deve-se fazer uso das classes ObjectOutputStream e ObjectInputStream.

Exercício 1

- Crie um programa que armazene uma lista de Funcionários (name,salary,hireDay) em um arquivo texto
  - Crie três instâncias de funcionário, escreva no arquivo texto
  - Leia do arquivo e liste na tela os dados dos funcionários lidos
  - Use no mínimo duas classes: DataFileTest e Employee
  - Formato do registro:
    - [name] | [salary]
  - Utilizar StringTokenizer para separar campos
    - StringTokenizer t=new StringTokenizer(str,"|");
    - String s=t.nextToken();

Solução – Exercício 1 – Classe DataFileTest

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class DataFileTest
{  static void writeData(Employee[] e,
       PrintWriter os)
     throws IOException
   {  os.println(e.length);

     int i;
     for (i = 0; i < e.length; i++)
       e[i].writeData(os);
   }

static Employee[] readData(BufferedReader
     is)
   throws IOException
 {  int n = Integer.parseInt(is.readLine());
   Employee[] e = new Employee[n];
   int i;
   for (i = 0; i < n; i++)
   {  e[i] = new Employee();
     e[i].readData(is);
   }
   return e;
 }
```

```
public static void main(String[] args)
  {  Employee[] staff = new Employee[3];
     staff[0] = new Employee("Harry Hacker", 35500     );
     staff[1] = new Employee("Carl Cracker", 75000);
     staff[2] = new Employee("Tony Tester", 38000);
   try
     {  PrintWriter os = new PrintWriter(new
           FileWriter("employee.dat"));
        writeData(staff, os);
        os.close();
     }      catch(IOException e)
     {  System.out.print("Error: " + e);
        System.exit(1);      }
      try
     {  BufferedReader is = new BufferedReader(new
           FileReader("employee.dat"));
        Employee[] in = readData(is);
        for (int i = 0; i < in.length; i++) in[i].print();
        is.close();
     }   catch(IOException e)
     {  System.out.print("Error: " + e);     System.exit(1);     }   } }
```

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.StringTokenizer;

import utils.Format;


public class Employee
{  public Employee(String n, double s)
   {  name = n;
      salary = s;

   }
   public Employee() {}
   public void print()
   {  System.out.println(name + " " + salary);
   }
   public void raiseSalary(double
      byPercent)
   {  salary *= 1 + byPercent / 100;
   }
   public void writeData(PrintWriter os) throws IOException
   {  os.print(name+ "|");
      os.print( salary+ "|");

   }

   public void readData(BufferedReader is) throws
IOException
   {  String s = is.readLine();
      StringTokenizer t = new StringTokenizer(s, "|");
      name = t.nextToken();
      salary = Double.parseDouble(t.nextToken());

   }

   private String name;
   private double salary;

}
```

Exercício 1.1

- Crie um método em Employee que incrementa o salário de um funcionario pelo percentual passado como parâmetro. Após a leitura dos dados do arquivo aumente o salário de todos em 10%. Apresente os dados

Solução 1.1

- Na classe DataFileTest

```
public void raiseSalary(Employee[] e) {
    int i;
    for (i = 0; i < staff.length; i++)
      staff[i].raiseSalary(10.0);
}
```

- Na classe Employee,

```
public void raiseSalary(double byPercent)
{  salary *= 1 + byPercent / 100;
}
```