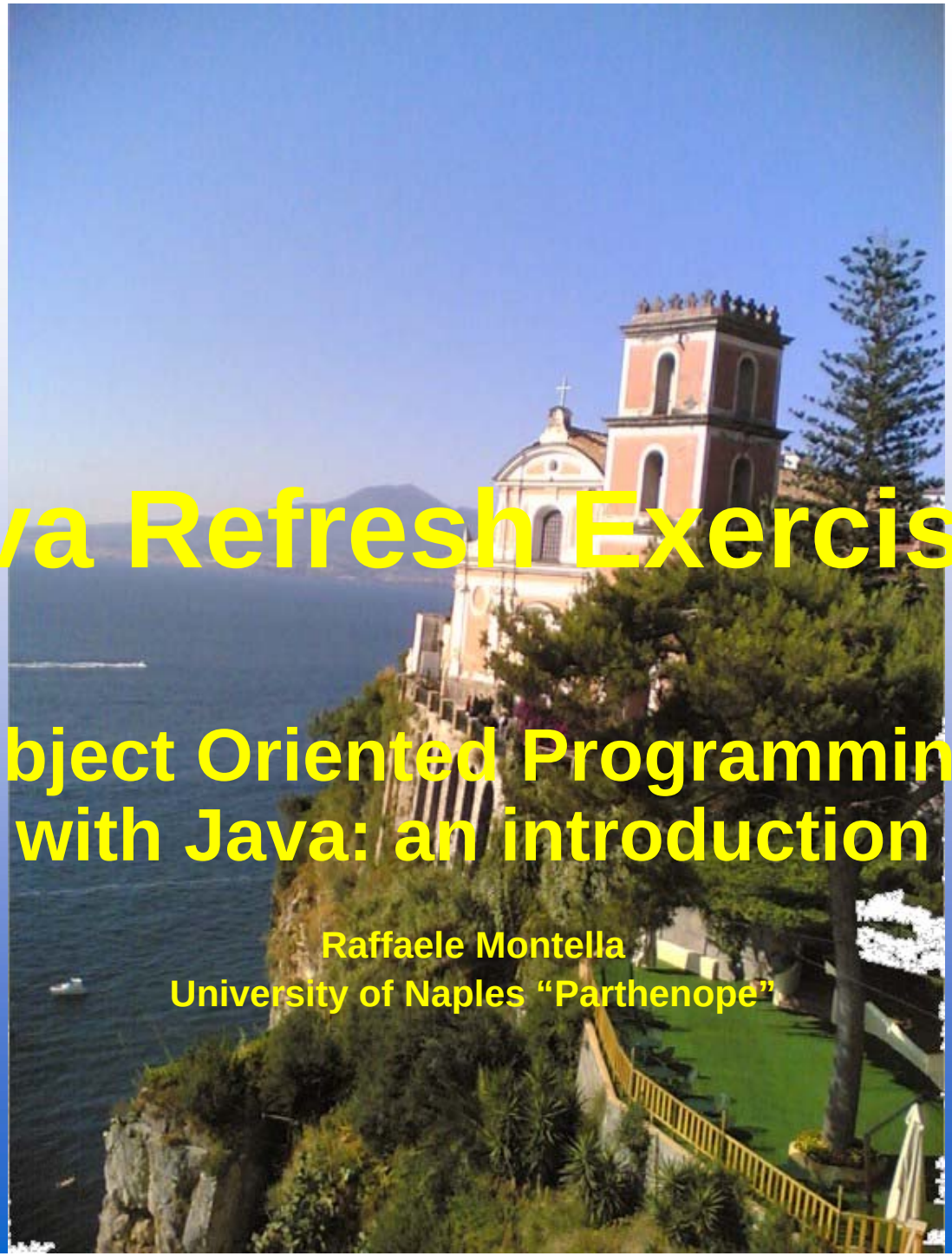

Java Refresh Exercises

Object Oriented Programming with Java: an introduction

Raffaele Montella
University of Naples “Parthenope”

Outline

- Introduction to Java

- The HelloWorld program
- Code features
- Comments
- Packages and Namespaces
- OOP Basics: Classes, Objects, Methods and Attributes
- Primitive and Class Types
- Loops and Conditions

- Exceptions
- Input / Output

- Documentation

Welcome to the Java world (1/2)

- Java was developed at Sun Microsystems in the first part of ’90s with the following goals:
  - **One language**
  - **One binary**
  - **More architectures**
  - **The same running application!**
- The Java compiler produces a “bytecode” binary
- The bytecode binary is executed by a virtual machine (JVM, Java Virtual Machine)

Welcome to the Java world (2/2)

- The JVM abstracts the real machine
- The same compiled code can be executed on different hardware and software architectures each provided by a JVM
- Java is freely downloadable from Sun website
  - Java Development Kit (JDK)
  - Java Runtime Environment (JRE)
- JDK & JRE are not Open Source, but an Open Source implementation is avalaiable (Kaffe)

Java Features (1/2)

- Simple and powerful:
  - The use of best practices and design patterns is suggested
- Secure:
  - The executable code is very far from the real machine
- Object Oriented:
  - In Java everything is an object: from the ‘main class’ to ‘Integer’
- Robust:
  - No pointers allowed, only references
  - Strongly typed

Java Features (2/2)

- Interactive:
  - Graphical User Interface
  - Multithread
  - Networking
- Architecture independent:
  - Everything runs inside the virtual machine
  - The same binary code runs on Linux, Mac-OS, Windows, etc…
- Interpreted, but “high performance”:
  - The bytecode is easy to translate into machine code
  - Just in time compilers can improve this feature
  - NB: high performance, but not so high for high performance computing!
- Easy to learn: I will demonstrate it…

Java@Work

- What you need:
  - Java Development Kit
  - A text editor
    - vi or notepad are enough
    - jEdit is a dedicated editor (developed in Java)
    - Netbeans and Eclipse are powerful, free and very cool (IDE, Integrated Development Environment)
    - Commercial tools: JBuilder, IBM Visual Age for Java
  - A book
    - "**Thinking in Java**" by *Bruce Eckel* (freely downloadable)
- All needed files are on the ISSGC05 repository.

Hello World!

- We break the process of programming in Java into three steps:
  - create the program by typing it into a text editor and saving it to a file named **`HelloWorld.java`**

    `vi HelloWorld.java`
  - compile it by typing in the terminal window

    `javac HelloWorld.java`
  - run (or execute) it by typing in the terminal window

    `java HelloWorld`

HelloWorld.java

- Java public classes are implemented in a file named as the class
- Any Java public class can contain one main method
- Here you are the HelloWorld.java file:

```
// A very simple HelloWorld Java code
public class HelloWorld {

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

Look at the Code

- Each statement line ends with a semicolon ; as in C/C++
- Each code block is inside curly brackets {,}
- Variables defined inside a code block are local to it
- Java is **not positional**:
  carrige return and space sequences outside quotes are ignored:

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

  is a correct code, but very hard to be managed by humans!

```
// A very simple HelloWorld Java code
public class HelloWorld {

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

Comments

- C++ style comment

```
// this is a single line slash
// slash C++ style comment
```

- C style comment

```
/* this are few comment lines in
a C style fashion */
```

- Javadoc auto documentation comments

```
/**
* @params args Command Line Arguments
**/
```

more information about Javadoc later in this lesson

```
// A very simple HelloWorld Java code
public class HelloWorld {

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

Packages and Namespaces

- Java classes definitions are grouped into namespaced packages.
- Packages stored in compressed zip or jar (Java ARchive) files must be specified in the CLASSPATH environment variable.
- Namespaces are used:
  - to avoid omonimous class ambiguity
  - to produce better code
- The **`import`** statement imports a package namespace:

```
// Imports all namespaces in java.io
import java.io.*;

// Imports the specified namespace
import java.io.InputStreamReader;
```

Primitive Types

- In Java primitive types are similar to their C counterparts:

```
int num1=0;
float num2;
num2=5.3;
double num3=10.0;
boolean condition=true;
```

- A variable can be
  - declared and assigned a value in two different steps…
  - …or in one statement.

- Any variable specified as **`final`** is considered as a constant:

```
final double pi=3.1415;
```

Exercise 1

- Type, compile and run the HelloWorld.java program

Solution 1

```
// File: HelloWorld.java
public HelloWorld {
  public static void main(String[] args) {
    System.out.println(“Hello World!”);
  }
}
```

Defining a Class

- The skeleton of a Java program is the class definition
- One file can contain more than one class definition
- Only **one** public class can be defined for each .java file
- Nested classes are allowed
- A class can extends another one by inheritance

```
// File HelloWorld.java
// A demo HelloWorld Java code
class A {
  ...
}
class B extends A {
  ...
}
public class HelloWorld {
  class C {
    ...
  }
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

OOP Basics: Classes

- A class represents an abstract data type
- A class is an object generalization
- An object is an instance of a specific class

- Example:
  myComputingMachine object is an instance of the ComputingMachine class of objects

- The class is the ‘idea’, the general model, of an object

OOP Basics: Objects

- An object stores some data (its state);
- Provides methods to access and manipulate that data (its behavior);
- Represents an individual unit (its identity).

- **An example:**
  the computing machine object…

Example:
The ComputingMachine Object

- The state of a computing machine might be characterized by
  - The type of CPU
  - The number of CPUs
  - The speed of each CPU
- We can communicate with the computing machine and do something.
- We can identify a computing machine by its fully qualified domain name or IP.

Example:
The Computing Machine Class

- Coding the ComputingMachine class in Java:

```
// File: ComputingMachine.java
public class ComputingMachine {
    // Attributes
    public String fqdn;
    public String cpuType;
    public int cpuCount;
    public float cpuSpeed;

    // Methods
    public double doSomething(double a, double b) {
      double result=a*b;
      return result;
    }
}
```

The ComputingMachine class is public:
…mented in the
…**hine.java** file

Attributes definition

Methods definition

- Using the ComputingMachine class in Java:

```
ComputingMachine myCM=new ComputingMachine();
myCM.fqdn="193.205.230.114";
double myResult = myCM.doSomething(14.5f, 40.85f);
```

OOP Basics: Access Specifiers

- Attributes and methods can be declared with the following access specifiers:
  - **`public`:** Accessible both inside and outside the class
  - **`private`:** Accessible only inside the class
  - **`protected`:** Accessible inside the class and inside derived classes, but not outside

```
// A very simple HelloWorld Java code
public class HelloWorld {

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

OOP Basics: Attributes

- stores the object state
- can be both primitive and abstract data types
- can be defined as static
  (shared between all object instances)

```
class ComputingMachine {
   // Attributes
   public String fqdn;
   public String cpuType;
   public int cpuCount;
   public float cpuSpeed;
   private float cpuLoad;

   public static int count;
  // Methods
  ...
}
```

OOP Basics: Methods

- are functions acting on the object behaviour
- can be qualified as public, private and protected like attributes
- can be defined as static (allowing the invocation by statically used objects)

```
class ComputingMachine {
   // Attributes
   ...
   // Methods
   public double doSomething(double a, double b) {
     double result=a+b;
     return result;
   }
   private void updateCPULoad() {
     ...
   }
}
```

The Constructor

- Is a special method invoked when the object is instanced using the **new** operator
- The name of the constructor must be the same of the class
- In java there is no destructor counterpart as in C++ (except the finalize method)

```
class ComputingMachine {
  // Attributes
  ...
  // Methods
  ...

  ComputingMachine() {
   fqdn="0.0.0.0";
   cpyType="Unknown";
   cpuSpeed=0;
   cpuCount=0;
   count++;
  }
}
```

The Main Method

- Is the entry point for a Java program
- Interfaces the program with the command line shell throw parameters and returned value (using System.exit(..))
- Any public class can have one main method
- The main method MUST be defined as public AND static so the JVM can execute it
- For class testing purposes I suggest to write a main method for each class.

```
// A very simple HelloWorld Java code
public class HelloWorld {

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

Class Types

- Are considered as reference to object instance

```
ComputingMachine myCM;
```

  - myCM is a reference to an object belonging to ComputingMachine class: myCM is not the object
  - We can use only static attributes and methods of ComputingMachine class
  - The constructor is not invoked

- Have to be first declared and then instanced

```
myCM=new ComputingMachine();
```

  - myCM references to an instance of a ComputingMachine object
  - The ComputingMachine constructor is invoked
  - We can use any kind of attributes and methods (both static and not static)
  - There is no destructor: the Java garbage collector frees automatically allocated, but unreferenced objects

Exercise 2

- Implement the ComputingMachine class in the file named ComputingMachine.java
- Modify the HelloWorld main method to show the result of the ComputerMachine doSomething method invocation.

Solution 2

```
// File: ComputingMachine.java
public class ComputingMachine {
   // Attributes
   public String fqdn;
   public String cpuType;
   public int cpuCount;
   public float cpuSpeed;

   // Methods
   public double doSomething(double a, double b) {
     double result=a+b;
     return result;
   }
}

// File: Exercise2.java
public class Exercise2 {
  public static void main(String[] args) {
    ComputingMachine cm = new ComputingMachine();
    double r = cm.doSomething(3.5,2.7);
    System.out.println("doSomething=" + r);
  }
}
```

Classes for Primitive Types

- For each primitive type there is a predefined wrapper class:
  - Double
  - Float
  - Integer
  - Boolean
- Static members for primitive types operations
  - Parsing
  - To string conversions

```
String myAge = age;
System.out.println("I'm " + myAge
  + " years old");
```

The String class

- Is a class with a special behaviour:
  - Strings are used as primitive types
  - The new operator is not needed

```
String s1;
s1 = “ISS on Grid Computing 2005”;
String s2 = “”;
String s3 = “Hello”;
```

  - String concatenation is done with the + operator

```
s1 = s2 + “ there!”;
String s3 = “number “+1;
```

String class methods

- The String class is provided by power manipulation metho

```
int len = s1.length();
boolean eq = s1.e      s(s3);
int
Str                    (",");
```

Returns the number of chars belonging to the string s1

Compares the string s1 with s3.
Returns true if s1==s3, false otherwise.

Splits the string s1 in an array of Strings using the specified separator ","

Compares the string s1 with s2.
The returned value is the same of the strcmp C function.

Returns the index of the first occurrence of the "<form" string in s1, -1 if the string isn't in s1.

Arrays

- can be declared and instantiated using the new operator;
- the array elements values can be assigned in a separate step;

```
int[] a1;
a1 = new int[5];
double[] a2 = new double[10];
a2[0] = 1.0;
```

- elements can be initialized when the array is declared;

```
String[] colors = {“red”,”blue”,”green”};
int len = colors.length;
```

- The length of an array is stored in the **`length`** class attribute

Command Line Arguments

- Are passed to the main method using an array of strings

```
public class ShowArguments {

  public static int main(String[] args) {
    System.out.println(args.length);
    System.out.println(args[0]);
    System.out.println(args[1]);
    System.out.println(args[2]);
  }
}
```

Loops

- Loops are implemented in the same way as in C/C++

```
for(
  set_index_variable,
  loop_condition,
  iteration_alteration_to_index) {…}

while (condition) {…}

do {…} while (condition);
```

- Example:

```
for (a=0;a<args.length:a++) System.out.println(args[a])
```

Exercise 3

- Modify the HelloWorld program to show each argument passed in separate rows on the console window.

Solution 3

```
// File Exercise3.java
public class Exercise3 {

  public static void main(String[] args) {
    for(int i=0;i<args.length;i++) {
      System.out.println(i+”:”+args[i]);
    }
  }
}
```

Conditions

- The if –else statement is the same as in C/C++ and can take three forms:

```
if(condition){…}
if(condition){…} else {…}
if(condition1){…}
elseif (condition2){…}
else{…}
```

- where a condition can be formed using the comparative and logical operators (==, >, <, >=, <=, !=, &&, ||).

Exercise 4

- Modify the previous exercise to accept just and only 3 parameters on the command line.

Solution 4

```
// File Exercise4.java
public class Exercise4 {

  public static void main(String[] args) {
    if (args.length==3)
      for(int i=0;i<args.length;i++)
        System.out.println(i+”:”+args[i]);

  }
}
```

Converting strings to numbers

- All text input is treated as a string.
- Any number must be converted to their primitive type.

- This can be done using static methods contained in the classes corresponding to each primitive type.

```
String number_string = “1.5”;
double number = Double.parseDouble(number_string);
String number_string2 = “10”;
int number2 = Integer.parseInt(number_string2);
```

- At run-time there is the possibility that the String variable may not contain the string representation of a number.

- Rather than leaving the program to crash, this possibility is managed by enclosing the conversion in a try-catch statement to handle an exceptional condition.

OOP Basics: Exceptions

- The term exception is shorthand for the phrase "exceptional event.“
- **Definition:**
  An exception is an event that occurs during the execution of a program that disrupts the normal instructions flow.
- Java has an efficient way to catch and manage exceptions derived as an evolution of C++ fashion.

Catch them all!

- The try/catch construct permits to catch exceptions thrown inside a code block:

```
try {
  block of code
} catch (Exception ex) {
  exception manager code block
}
```

- If inside the try block of code an exception rises, is executed the code inside the catch block.

Example: string to number

- When the conversion fails an exception of type NumberFormatException is thrown by the parse method and the program jumps to the statements in the catch block.

```
try {
  number = Double.parseDouble(number_string);
}
catch(NumberFormatException nfe) {
  System.err.println("Error in converting to double");
  System.exit(1);
}
```

- If the catch block does not include a statement for the program to exit, the program will continue with any statements that come after it
- error messages are printed using the standard error stream System.err rather than System.out.
- This is useful when the program output is redirected to a file.

Exercise 5

- Write a Java adder program accepting operands on command line and displaying the result on the console window.

Solution 5

```
// File Exercise5.java
public class Exercise5 {
  public static void main(String[] args) {
    double r=0;
    for(int i=0;i<args.length;i++) {
      try {
        r=r+Double.parseDouble(args[i]);
      } catch (NumberFormatException nfe) {
        System.out.println(args[i] +
          " is not a valid operand!");
      }
    }
    System.out.println("Result="+r);
  }
}
```

The Stack Trace

- The stack trace output is a very useful feature used to discover the position of a runtime error.
- Example: string to number conversion exception

```
public class Test {
  public static void main(String[] args) {
    double d = Double.parseDouble(args[0]);
    System.out.println(d);
  }
}
```

- Passing the value ‘foo’ as argument (foo is not convertible to double)

```
Exception in thread "main" java.lang.NumberFormatException: For input
   string: “foo"
at sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:1224)
at java.lang.Double.parseDouble(Double.java:482)
at Test.main(Test.java:4)
```

Input and output streams

- In previous examples inputs have been performed via the command line arguments.
- Now we will read the input from the console using the stream System.in.
- Input and output streams allow the use of file redirecting with Java programs.

The input stream

- System.in can only read bytes.
- System.in is used to initialize an InputStreamReader object
- The InputStreamReader object is passed as argument to the BufferedReader object constructor.
- A line of text can be read using the BufferedReader method readLine()

```
InputStreamReader isr = new
  InputStreamReader(System.in);
BufferedReader br = new BufferedReader(isr);
String name = br.readLine();
```

Input stream exceptions

- I/O problems can occur when trying to perform the reading.
  - The readLine() method throws a IOException
  - The read operation must be enclosed in an appropriate try-catch block.
- After all input is read from the stream it should be closed using br.close().

Example:
Read input stream

```
import java.io.*;

public class ReadInputStream {
  public static void main() {

  try{
    InputStreamReader isr = new
      InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(isr);
    String name = br.readLine();
    br.close();
    System.out.println(name);
}
catch(IOException ioe){
   System.err.println("Error in reading name");
   System.exit(1);
}
```

Manage multiple exceptions

- The following not trivial example points out the multiple exception management tecnique.

```
try {
   block of code
} catch (IOException exIO) {
  catch input output exception block
} catch (ArithmeticException exAr) {
  catch arithmetic exception block
} finally {
  in any case do something
}
```

- Different exception types are handled in separate catch blocks
- The finally clause specifies a block of code executed in any case

Exercise 6

- Write a Java adder program accepting 2 operands from the standard input and showing the output on the console window.
- Remember to manage both NumberFormatException and IOException!

Solution 6 (1/2)

```
// Exercise6.java
import java.io.*;
public class Exercise6 {

  private double getOperand(String prompt) {
    double operand=0;
    System.out.print(prompt);

    try {

      BufferedReader br = new BufferedReader(
        new InputStreamReader(System.in));
      operand = Double.parseDouble(br.readLine());

    } catch (IOException exIO) {
      System.out.println(exIO.getMessage());
    } catch (NumberFormatException exNF) {
      System.out.println(exIO.getMessage());
    }

    return operand;
  }
  ...
```

Solution 6 (2/2)

```
   ...

  public static void main(String[] args) {
    Exercse6 e6 = new Exercise6();
    double a = e6.getOperand("A:");
    double b = e6.getOperand("B:");
    double sum = a + b;
    System.out.println("A+B="+sum);
  }
}
```

The output stream

- Output to the console can be handled in a similar way using the stream System.out.
- The stream can be converted to a OutputStreamReader object and then to a PrintWriter object.
- The PrintWriter class has the methods print() and println() for printing.

- However, this is unecessary because, the stream System.out already has methods println() and print().

```
// Solution 1
OutputStreamReader osw = new
   OutputStreamReader(System.out);
PrintWriter pw = new PrintWriter(osw);
pw.println("Hello World!");

// Solution 2
System.out.println("Hello World!");
```

Reading and Writing files

- Rather than read the inputs from a file using a stream, a file can be read directly programmatically.
- Similar to streams, a FileReader object, which reads characters, can be created using the name of the input file and this is used to initialize a BufferedReader object for reading a line of text:

```
try {
  FileReader fr = new FileReader(“filename”);
  BufferedReader br = new BufferedReader(fr);
  String name = br.readLine();
  br.close();
}
catch(IOException ioe) {
  System.out.println(“Error reading name”);
}
```

File Troubles

- The previous code is enough to read from a file.
- It does not provide any useful information on the source of an I/O problem:
  - the input file does not exist,
  - the input filename does not refer to a file,
  - the permissions on the file are such the file cannot be read,
  - the file is empty,
  - the file contains not enough data.
- Almost all of these events can be tested using a File object,

The File Object

```
try {
  File inFile = new File("filename");

  if(!inFile.exists()) throw new IOException("Input file does not exist");
  if(!inFile.isFile()) throw new IOException("Input file is not a file");
  if(!inFile.canRead()) throw new IOException("Cannot read input file");
  if(inFile.length() == 0) throw new IOException("Input file is empty");

  FileReader fr = new FileReader(inFile);
  BufferedReader br = new BufferedReader(fr);
  String name = br.readLine();
  fr.close();
}
catch(IOException ioe) {
  System.err.println("Error reading name");
  System.err.println(ioe.getMessage());
  System.exit(1);
}
```

- The File methods exists(), isFile() etc do not automatically throw an IOExeception
- If a problem is detected an exception can be thrown.
- A FileReader object can also be created using a File object..

Writing Files

- Writing to a file follows a similar pattern.

```
File outFile = new File("filename");
if(!outFile.createNewFile())
   throw new IOException("Output file already exists.");
FileWriter fw = new FileWriter(outFile);
PrintWriter pw = new PrintWriter(fw);
pw.println(name);
pw.close();
```

- This code is enclosed in a suitable try-catch.
- A new file is created for the output.
- If an existing file is to be used, the File method canWrite() can be used to check that the file is writeable.
- The statement

```
FileWriter fw = new FileWriter(outFile,true);
```

will cause output to be appended to the file.

Exercise 7

- Write a Java program reading a text file, showing it line by line on the console window and then write it on another file.
- Get input and output filenames from the comand line.

Solution 7 (1/2)

```
// Exercise7.java
import java.io.*;

public class Exercise7 {
  public static void main(String[] args) {
   try {
     BufferedReader br=new BufferedReader(new
   FileReader(args[0]));
     try {
       PrintWriter pw=new PrintWriter(
         new FileWriter( args[1] );
       String in;
       while (in = br.readLine() != null) {
         System.out.println(in);
         pw.println(in);
       }
       ...
```

Solution 7 (2/2)

```
        ...
        pw.close();
      } catch(IOException ex) {
         System.out.println(ex.getMessage());
       }
       br.close();
     } catch(IOException ex) {
      System.out.println(ex.getMessage());
    }
  }
}
```

Documentation

- Html pages of API documentation can be automatically generated from the source code using documentation comments that are included in the program and the javadoc tool.
- The comments are of the form /** comment */ and appear (immediately) before:
  - the class definition,
  - the attribute declarations
  - definitions of the constructors
  - definitions of methods.
- The comments should describe the class, attributes and methods without detailing the implementation.
- Implementation details can be added using // or /* comments */.

Class Comments

```
/** The ComputingMachine class is a grid computing node
 * @author R. Montella
 * @version 0.1
 */
public class ComputingMachine {
```

- As with all document comments it must be placed immediately before the declaration/definition it is describing.
- If an **`import`** statement is placed between '*/' and 'public ..' the comment will not be found by the javadoc tool.
- The comment begins with a brief description of the class and is followed by two tagged comments.:
  - 'author',
  - 'version'

Attribute Comments

```
/** Fully Qualified Domain Name */
public String fqdn;

/** The type of CPU as vendor and/or model */
public String cpuType;
```

- Both public and private attributes and methods can be commented
- With default javadoc settings only public methods and attributes are published.

Constructor and Methods Comments

```
/** Do a very computing intensive task: the sum
  * @param a the first operand of sum operation
  * @param b the second operand of sum operation
  * @return the sum of a and b
  */
public double doSomething(double a, double b) {
  double result=a+b;
  return result;
}
```

- The format for constructors and methods is to start with a brief description followed by the tagged parameter comments
- For methods which return a value, a tagged return comment
- The return tagged comment is avoided in the case of void methods

Producing Documentation

- Generating the html documentation is straightforward.
- The command

  `javadoc ComputingMachine.java`

  will generate the file ComputingMachine.html containing the class documentation and additional html files giving information on class hierarchy and including a help guide and an index of methods and classes.
- This command uses the default settings and will only document public attributes, constructors and methods.
- Private class members can be included using

  `javadoc –private ComputingMachine.java`
- If the documentation comments are omitted, the html file will still be generated but without any description of the class or of its members.

Exercise 8

- Write a documented version of ComputingMachine class implementation and generate both public and private HTML documentation.

Solution 8

```
/** The ComputingMachine class is a grid computing node
 * @author R. Montella
 * @version 0.1
 */
public class ComputingMachine {
   /** Fully Qualified Domain Name */
   public String fqdn;
   /** The type of CPU as vendor and/or model */
   public String cpuType;
   /** Number of  CPUs in SMP nodes */
   public int cpuCount;
   /** CPU speed */
   public float cpuSpeed;

   /** Do a very computing intensive task: the sum
    * @param a the first operand of sum operation
    * @param b the second operand of sum operation
    * @return the sum of a and b
    */
   public double doSomething(double a, double b) {
     double result=a+b;
     return result;
   }
}
```

Links

- Java Development Kit
  - http://java.sun.com
- jEdit
  - http://www.jedit.org
- Eclipse
  - http://www.eclipse.org
- Thinking in Java
  - http://www.mindview.net/Books/TIJ
- An online course
  - http://www.cs.princeton.edu/introcs