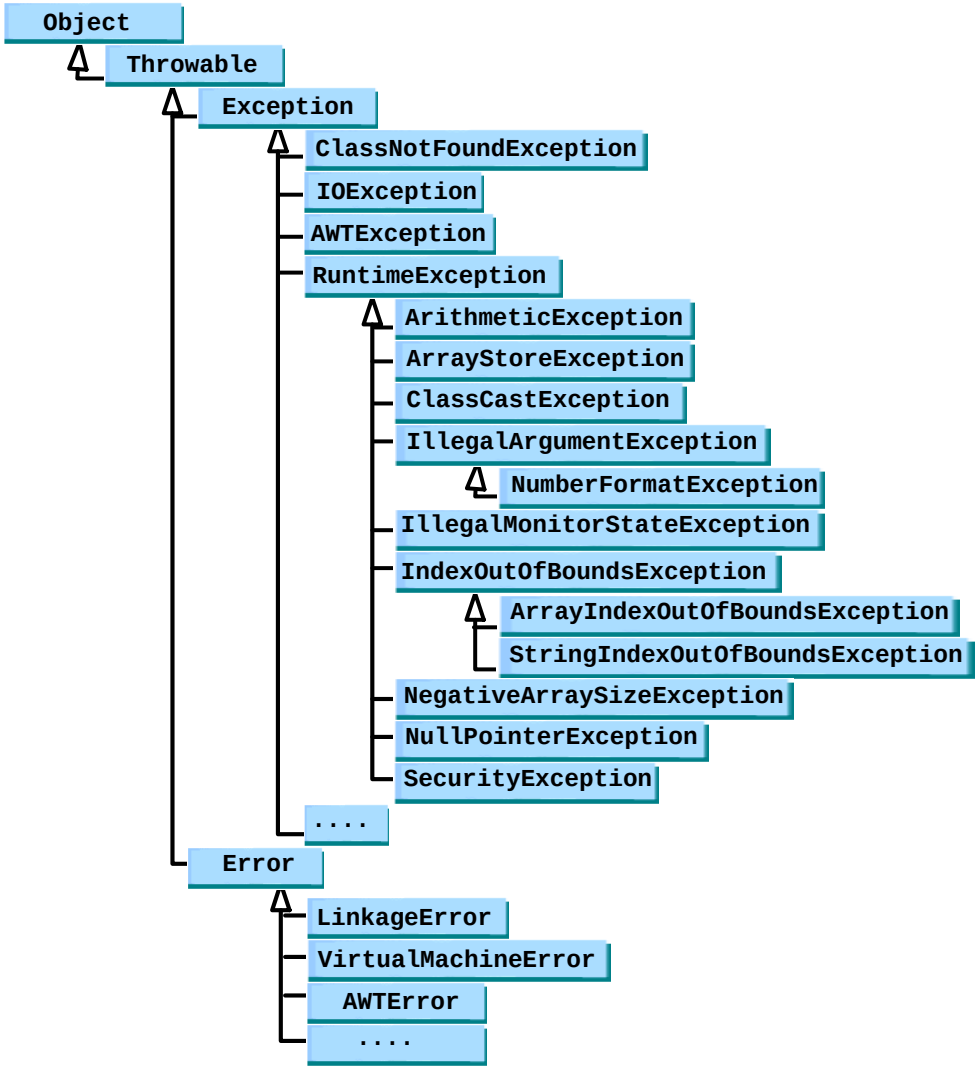
Chapter 14: File Input/Output

- Streams and Files
- File Stream Processing
- Text File I/O
- The File Class
- Binary File I/O

Review Ch 13: Exception Handling

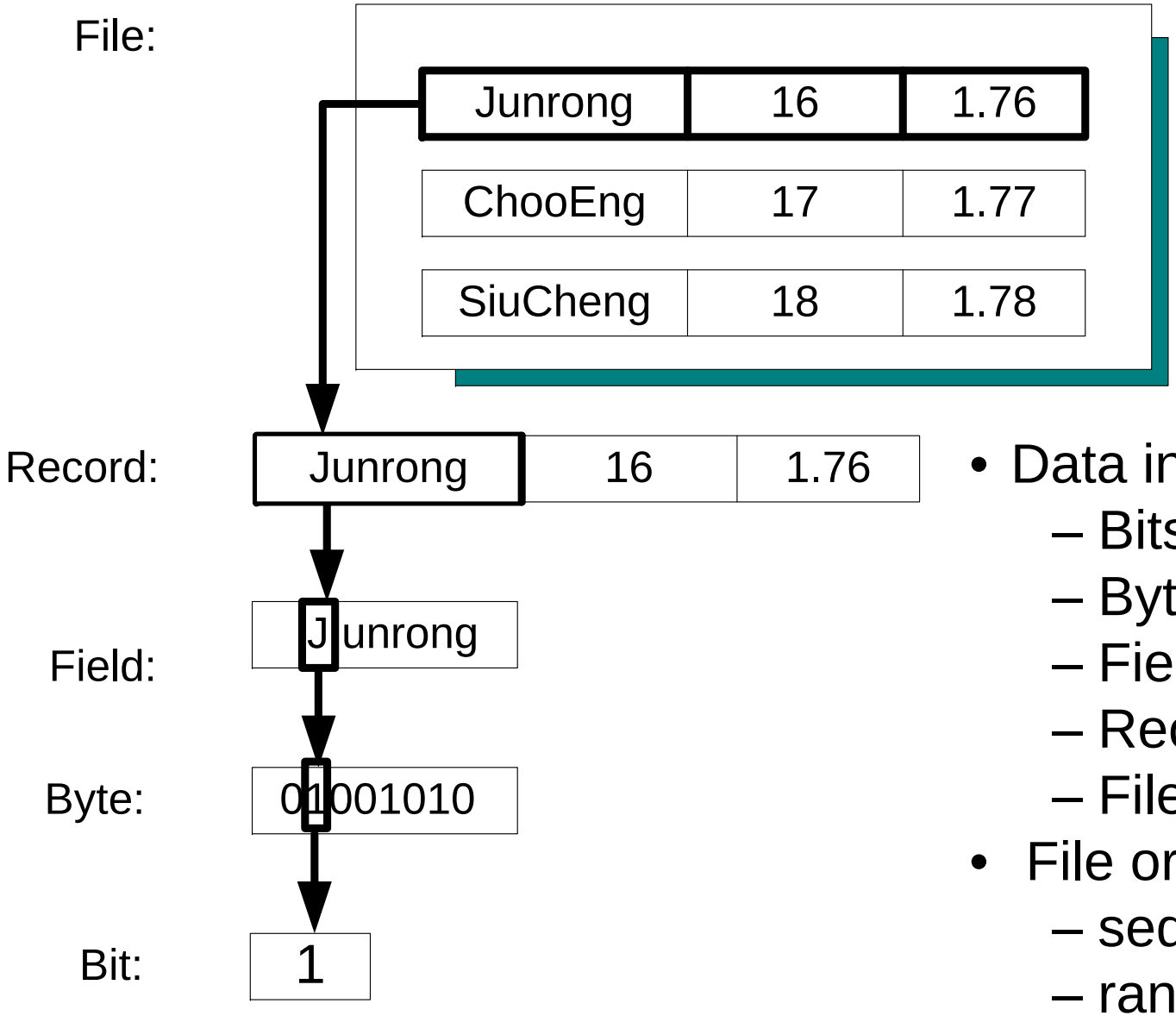
- Error Handling
- Java's Exception Handling
- Java's Exception Hierarchy
- Using Exception Classes
- Creating Your Own Exception Classes



Chapter 14: File Input/Output

- **Streams and Files**
- File Stream Processing
- Text File I/O
- The File Class
- Binary File I/O

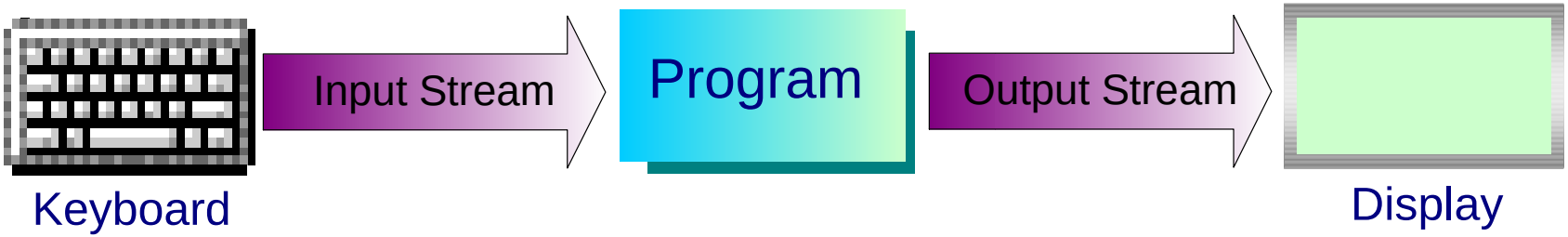
Files



- Data in Files
  - Bits
  - Byte
  - Fields
  - Records
  - Files
- File organization
  - sequential files
  - random files

Stream I/O

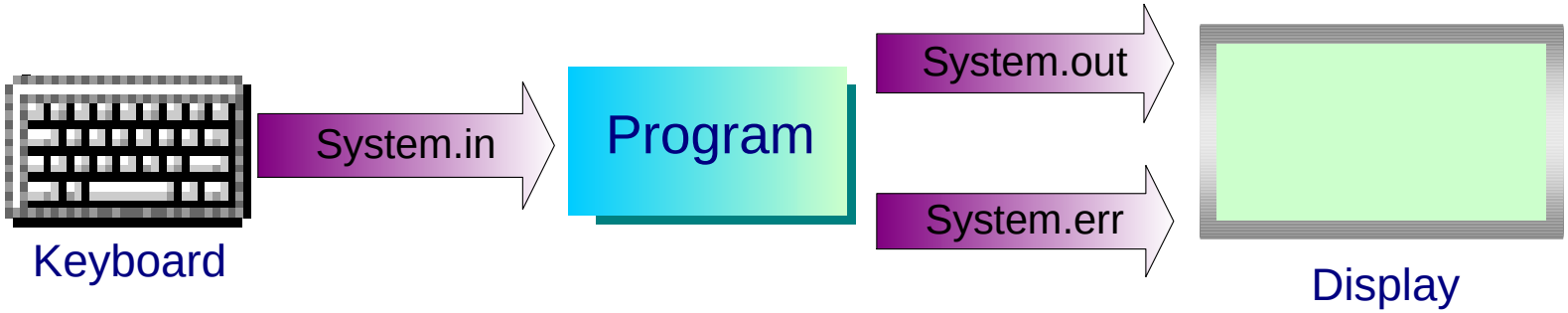
- All file I/O operations are carried out by means of streams.
- A stream is a sequence of characters. It connects a program to a device (or file) and transfers the data in (input stream) and out (output stream) from the program.



- Advantage: The programmer does not need to write specific input/output functions for different devices such as keyboard, display, printer, etc.
- Example Input files: keyboard, camera, disk drive, ..
  Output files: disk drive, tape drive, printer, monitor, ...

The Standard Java Stream Objects

Stream	Object	Device
standard input	**System.in**	input - normally the keyboard
standard output	**System.out**	output - normally the display
standard error	**System.err**	error message - the display



Binary VS Text Files

- Disk file streams work in a similar way as the standard input/output streams.
- A file stream can be viewed as a continuous sequence of bytes.
- Two modes of disk access: text mode and binary mode.
- Text mode:
  - Used for text files, data are stored and accessed as a sequence of characters
  - Character-based files, e.g., `.txt`, `.java`, etc.
- Binary mode:
  - Used for binary files, data are stored and accessed as a sequence of bytes
  - Byte-based files: data are stored in binary digits of 0 and 1, e.g., `.exe`, `.ppt`, `.class`, etc.

End-of-File Marker

- A special character in the file to indicate the end of a file.
- **Text file**
  - The character '\n' means end-of-line; it is placed at the end of a line.
  - end-of-file (EOF) character is placed at the end of a file.
  - Reading of files can be controlled using the EOF marker.
- **Binary file**
  - No end-of-file character
  - Need a special technique to read data from a binary file – to be discussed later

Chapter 14: File Input/Output

- Streams and Files
- **File Stream Processing**
- Text File I/O
- The File Class
- Binary File I/O

File Stream Processing

Basic file stream classes for text and binary files:

Mode	For Text Files	For Binary Files
Input	Use the **FileReader** class	Use the **FileInputStream** class
Output	Use the **FileWriter** class	Use the **FileOutputStream** class

See http://java.sun.com/j2se/1.5.0/docs/api/java/io/FileReader.html
In java IO package

Processing Text Files

```
import java.io.* ;
public class ProcessingFileStreams {
  public static void main( String[] args ) {
    try {
      // Step 1: Create and open file streams
      FileReader iStream = new FileReader( "input.txt"  );
      FileWriter oStream = new FileWriter( "output.txt" );
      // Step 2: Perform Read/Write Operation
      int data;
      data = iStream.read();
      oStream.write( data );
      // Step 3: Close file streams
      iStream.close();
      oStream.close();
    }
    catch ( IOException e ) {
      System.out.println( "IO Error!" + e.getMessage() );
      e.printStackTrace();
      System.exit( 0 );
    }
  }
}
```

Processing Text Files

input.txt

```
This is a test.
This is only a test.
```

output.txt

```
T
```

Program opens file: “`input.txt`”

Reads first character from the file.

*If `input.txt` doesn’t exist, an exception will be thrown (and should be caught).

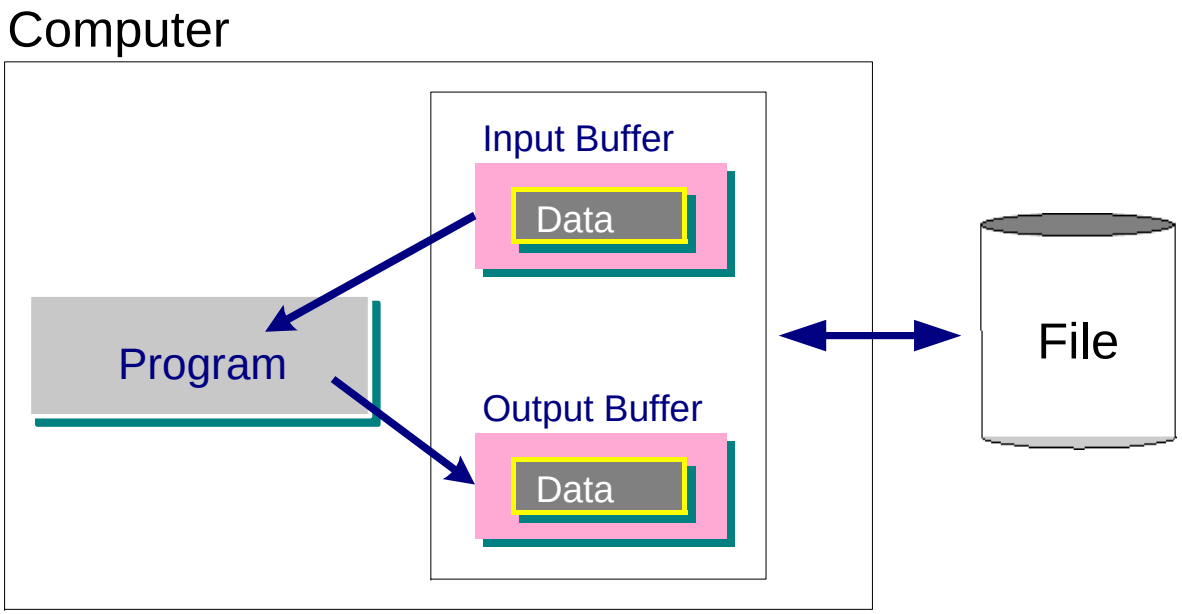
Program open file: “`output.txt`”

Writes a character to the file.

Both files are “closed” and Program ends.

*If `output.txt` already exist, its content is delete.

Buffered Stream I/O



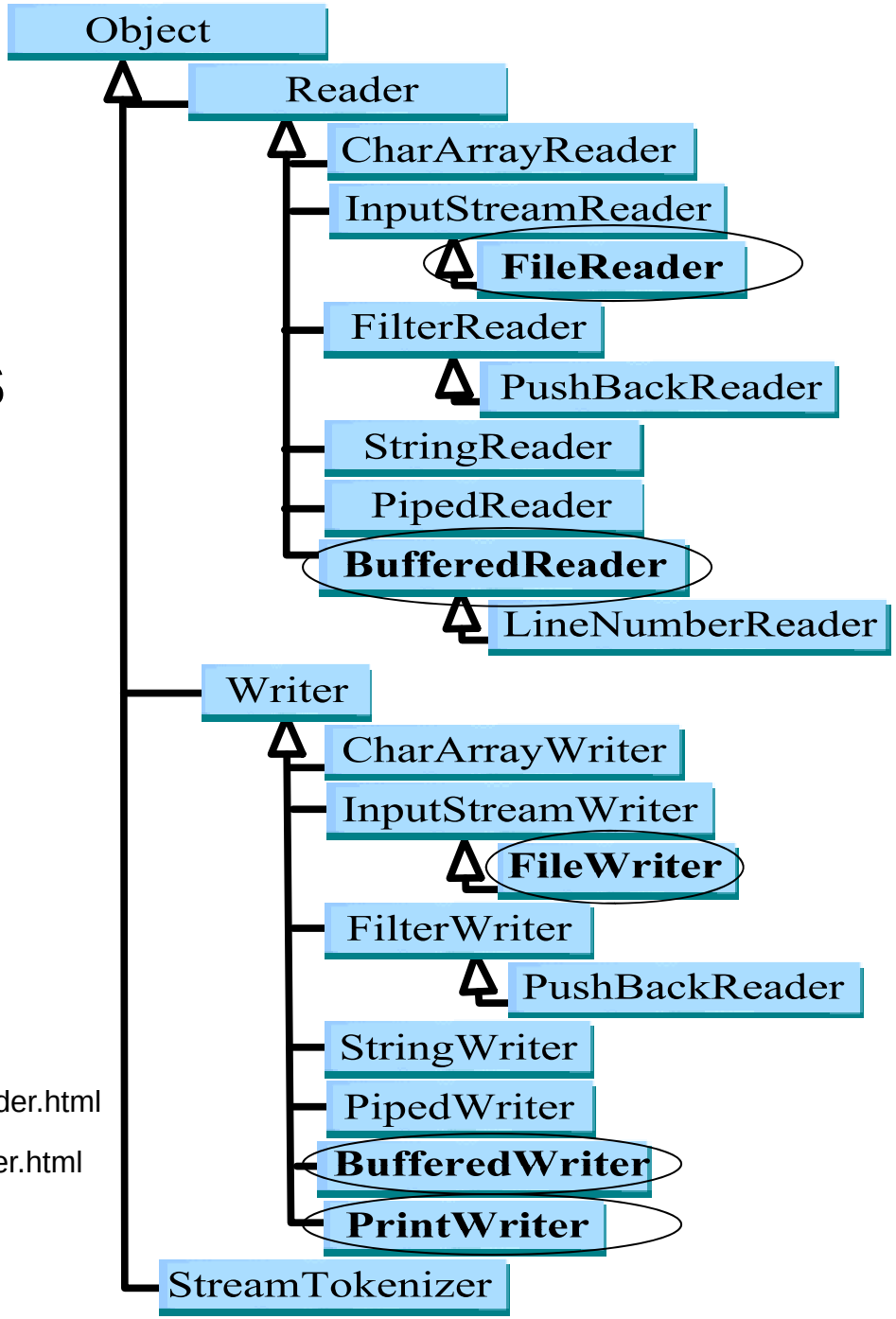
- Buffering is used in both input and output.
- I/O operations are expensive that may involve mechanical movement (inside your hard drives).
- A buffer is a sequence of memory locations that is used to store data temporarily between the program and an I/O device.
- Buffered I/O will be used in programming file I/O.

Chapter 14: File Input/Output

- Streams and Files
- File Stream Processing
- **Text File I/O**
- The File Class
- Binary File I/O

Text File I/O

Stream class hierarchy



http://java.sun.com/j2se/1.5.0/docs/api/java/io/Reader.html

http://java.sun.com/j2se/1.5.0/docs/api/java/io/Writer.html

Output Stream Classes for Writing Text Files

Class	Description	Methods
`FileWriter`	Provides the basic stream for text output.	**`write()`**, **`flush()`**, **`close()`**
`BufferedWriter`	Provides output buffering to improve performance.	**`write()`**, **`flush()`**, **`close()`**
`PrintWriter`	Provides a number of useful output methods for processing.	**`print(char)`**, **`print(String)`**, **`print(int)`**, **`print(float)`**, **`print(double)`**, **`print(boolean)`**, and the corresponding **`println()`** methods, **`flush()`**, **`close()`**

See http://java.sun.com/j2se/1.5.0/docs/api/java/io/FileWriter.html
In java IO package

Writing Text Files using a Buffered Stream Object

```
import java.io.*;
public class WritingTextFiles {
  public static void main( String[] args ) {
    try {
      FileWriter     fwStream = new FileWriter(    "data.txt" );
      BufferedWriter bwStream = new BufferedWriter( fwStream  );
      PrintWriter    pwStream = new PrintWriter(    bwStream  );
      int num ;
      for ( num = 0 ; num < 5 ; num++ )
        pwStream.println( "Number = " + num * 5 );
      pwStream.close();
    }
    catch ( IOException e ) {
       System.out.println( "IO Error!" + e.getMessage() );
       e.printStackTrace();
       System.exit( 0 );
    }
  }
}
```

Closing a file

Program Input and Output

(After execution, the file data.txt contains:)

```
Number = 0
Number = 5
Number = 10
Number = 15
Number = 20
```

Input Stream Classes for Reading Text Files

Class	Description	Methods
`FileReader`	Provides the basic stream for character-by-character input.	`read()`, `close()`
`BufferedReader`	Provides input buffering to improve performance. Provides both character-by-character and line-by-line input.	`read()`, `readLine()`, `close()`

See http://java.sun.com/j2se/1.5.0/docs/api/java/io/FileReader.html
In java IO package

Reading Text Files using the class BufferedReader

```
import java.io.* ;
public class ReadingTextFiles {
  public static void main( String[] args ) {
    try {
      FileReader     frStream = new FileReader(    "data.txt" );
      BufferedReader brStream = new BufferedReader( frStream  );
      String inputLine ;
      int    i         ;
      System.out.println( "The file contains:" );
      for ( i = 0 ; i < 5 ; i++ ) {
        inputLine = brStream.readLine(); // read in a line
        System.out.println( inputLine );
      }
      brStream.close();
    }
    catch ( FileNotFoundException e ) {
      System.out.println( "Error opening the input file!"
                          + e.getMessage() );
      System.exit( 0 );
    }
```

Reading Text Files using the class BufferedReader

```
    catch ( IOException e ) {
      System.out.println( "IO Error!" + e.getMessage() );
      e.printStackTrace();
      System.exit( 0 );
    }
  }
}
```

Program Input and Output

```
The file contains:
Number = 0
Number = 5
Number = 10
Number = 15
Number = 20
```

Using the StringTokenizer Class

```
import java.io.*;
import java.util.* ;  // needed for StringTokenizer
public class ReadingTextFiles2 {
  public static void main( String[] args ) {
    try {
      BufferedReader brStream
        = new BufferedReader( new FileReader( "data.txt" ) );
      String inputLine , str1 , str2 ;
      int    i , value ;
      System.out.println( "The file contains:" );
      for ( i = 0 ; i < 5 ; i++ ) {
        inputLine = brStream.readLine();
        StringTokenizer aString =
            new StringTokenizer( inputLine );
        str1  = aString.nextToken();
        str2  = aString.nextToken();
        value = Integer.parseInt( aString.nextToken() );
        System.out.println( "str1: "   + str1
                            + " str2: "  + str2
                            + " value: " + value );
      }
      brStream.close();
    }
```

Read in a line, and
Use StringTokenizer
to break it down
into words

Using the StringTokenizer Class

```
    catch ( FileNotFoundException e ) {
      System.out.println( "Error opening the input file!"
                          + e.getMessage());
      System.exit(0);
    }
    catch ( IOException e ) {
      System.out.println( "IO Error!" + e.getMessage() );
      e.printStackTrace();
      System.exit( 0 );
    }
  }
}
```

Program Input and Output

```
The file contains:
str1: Number str2: = value: 0
str1: Number str2: = value: 5
str1: Number str2: = value: 10
str1: Number str2: = value: 15
str1: Number str2: = value: 20
```

Reading File Names from the Keyboard

```
import java.util.Scanner ;
import java.io.* ;
public class ReadingTextFiles3 {
  public static void main( String[] args ) {
    String  fileName ;
    Scanner sc = new Scanner( System.in );
    System.out.println( "Enter the file name: " );
    fileName = sc.nextLine();
    try {
      BufferedReader brStream
        = new BufferedReader( new FileReader( fileName ) );
      String inputLine ;
      int    i ;
      System.out.println( "The file contains:" );
      for ( i = 0 ; i < 5 ; i++ ) {
        inputLine = brStream.readLine();
        System.out.println( inputLine );
      }
      brStream.close();
    }
```

```
    catch ( FileNotFoundException e ) {
      System.out.println( "Error opening the input file!"
                          + fileName );
      System.exit( 0 );
    }
    catch ( IOException e ) {
      System.out.println( "IO Error!" + fileName );
      e.printStackTrace() ;
      System.exit( 0 );
    }
  }
}
```

Program Input and Output

```
Enter the file name:
data.txt
The file contains:
Number = 0
Number = 5
Number = 10
Number = 15
Number = 20
```

Testing for End of File

- We use end-of-file indicator (EOF) to test the end of a file.

- When using the method – readLine()
  - We test the special value *null*

- When using the method – read()
  - We test the special value *-1*

- These two methods will not throw EOFException

Unbuffered I/O And End of File Testing

```
import java.util.Scanner ;
import java.io.* ;
public class FileCopying {
  public static void main( String[] args )
    throws IOException
  {
    int     ch ;
    Scanner sc = new Scanner( System.in ) ;
    System.out.println( "Enter the input file name: " );
    String fileName1 = sc.nextLine();
    System.out.println( "Enter the output file name: " );
    String     fileName2 = sc.nextLine() ;
    FileReader frStream  = new FileReader( fileName1 );
    FileWriter fwStream  = new FileWriter( fileName2 );
    System.out.println( "The file contains:" );
    while ( ( ch = frStream.read() ) != -1 ) {
      System.out.print( (char) ch );
      fwStream.write(   (char) ch );
    }
    frStream.close();
    fwStream.close();
} }
```

When using read, test for -1

When using readLine, test for null

```
Program Input and Output
Enter the input file name:
data.txt
Enter the output file name:
output.txt
The file contains:
Number = 0
Number = 5
Number = 10
Number = 15
Number = 20
```

Example: Using readLine

```
import java.util.Scanner ;
import java.io.* ;
public class FileCopying2 {
  public static void main( String[] args ) throws IOException {
    int            ch ;
    Scanner        sc        = new Scanner( System.in );
    System.out.println( "Enter the input file name:" );
    String         fileName1 = sc.nextLine();
    System.out.println( "Enter the output file name:" );
    String         fileName2 = sc.nextLine();
    BufferedReader brStream  =
        new BufferedReader( new FileReader( fileName1 ) );
    PrintWriter    pwStream  =
        new PrintWriter( new BufferedWriter(
                                 new FileWriter( fileName2 ) ) );
    System.out.println( "The file contains:" );
    String aString = brStream.readLine();
    while ( aString != null ) {
      System.out.println( aString );
      pwStream.println(   aString );
      aString = brStream.readLine();
    }
    brStream.close();
    pwStream.close();
} }
```

Program Input and Output

```
Enter the input file name:
data.txt
Enter the output file name:
output.txt
The file contains:
Number = 0
Number = 5
Number = 10
Number = 15
Number = 20
```

Using Scanner Class

```
try
{
  Scanner scStream = new Scanner( new File( "data.txt" ) );
  String inputLine
  System.out.println( "The file contains:" );
  while ( scStream.hasNext() )
  {
    inputLine = scStream.nextLine();
    System.out.println( inputLine );
  }
  scStream.close();
}
catch ( FileNotFoundException e )
{
  System.out.println( "File Error!" + e.getMessage() );
  System.exit( 0 );
}
catch ( IOException e )
{
  System.out.println( "IO Error!" + e.getMessage() );
  e.printStackTrace();
  System.exit( 0 );
}
```

Chapter 14: File Input/Output

- Streams and Files
- File Stream Processing
- Text File I/O
- **The File Class**
- Binary File I/O

The File Class

- Obtaining information about a file or a directory

Method	Description
`exists()`	Returns a boolean `true` if the file exists.
`isDirectory()`	Returns a boolean `true` if the file is a directory.
`isFile()`	Returns a boolean `true` if the file is a file.
`canRead()`	Returns a boolean `true` if the file can be read from.
`canWrite()`	Returns a boolean `true` if the file exists and can be written to.
`delete()`	Deletes the designated file. Returns a boolean `true` if the delete operation is successful.
`getName()`	Returns the file name of the file from the given pathname.
`getAbsolutePath()`	Returns the complete absolute file or directory name represented by the `File` object.
`getParent()`	Returns the pathname of the parent directory of the file from the given pathname.
`length()`	Returns the size of the file, in bytes.
`renameTo()`	Renames the file. This method returns a boolean `true` if the operation is successful.

Using the File Class

```
import java.io.* ;
import java.util.Scanner ;
public class UsingFileClass {
  public static void main( String[] args ) {
    Scanner sc = new Scanner( System.in );
    System.out.println( "Enter the file name: " );
    String fileName = sc.nextLine();
    File   aFile    = new File( fileName );
    System.out.println( "Test exists      = " + aFile.exists()     );
    System.out.println( "Test isDirectory = " + aFile.isDirectory());
    System.out.println( "Test isFile      = " + aFile.isFile()     );
    System.out.println( "Test canRead     = " + aFile.canRead()    );
    System.out.println( "Test canWrite    = " + aFile.canWrite()   );
    System.out.println( "Test getName     = " + aFile.getName()    );
    System.out.println( "Test getAbsPath  = "
                        + aFile.getAbsolutePath() );
    System.out.println( "Test getParent   = " + aFile.getParent()  );
    System.out.println( "Test length      = " + aFile.length()
                                               + " bytes" );
   }
}
```

```
Program Input and Output
Enter the file name:
Java.txt
Test exists      = false
Test isDirectory = false
Test isFile      = false
Test canRead     = false
Test canWrite    = false
Test getName     = Java.txt
Test getAbsPath  = C:\Java.txt
Test getParent   = null
Test length      = 0 bytes
```

http://java.sun.com/j2se/1.5.0/docs/api/java/io/File.html

Checking Input File Name

```
import java.io.*;
import java.util.Scanner;
public class CheckFileReading {
  public static void main( String[] args )
  {
    Scanner sc = new Scanner( System.in );
    System.out.println( "Enter the file name: " );
    String fileName = sc.nextLine();
    File   inFile   = new File( fileName );
    while ( ( ! inFile.exists()  )
         || ( ! inFile.canRead() ) )
    {
      if ( ! inFile.exists() )
        System.out.println( "File not exist!" );
      else
      if ( ! inFile.canRead() )
        System.out.println( "File can’t be read!" );

      System.out.println( "Enter the file name again: " );
      fileName = sc.nextLine();
      inFile = new File( fileName );
    }
```

Checking Input File Name

```
    try {
      BufferedReader brStream =
          new BufferedReader( new FileReader( fileName ) );
      String inputLine ;
      int    i ;

      System.out.println( "The file contains: " );
      for ( i = 0 ; i < 5 ; i++ )
      {
        inputLine = brStream.readLine();
        System.out.println( inputLine );
      }

      brStream.close();
    }
    catch ( IOException e ) {
      System.out.println( "IO Error!" + e.getMessage() );
      e.printStackTrace();
      System.exit( 0 );
    }
  }
}
```

Program Input and Output

```
Enter the file name:
file1.txt
File not exist!
Enter the file name again:
output.txt
The file contains:
Number = 0
Number = 5
Number = 10
Number = 15
Number = 20
```

Checking Output File Name

```
import java.io.*;
import java.util.Scanner ;
public class CheckFileWriting {
  public static void main( String[] args )
  {
    System.out.println( "Enter the file name: " );
    Scanner sc       = new Scanner( System.in );
    String  fileName = sc.nextLine();
    File    outFile  = new File( fileName );

     // check existence
    if ( outFile.exists() )
    {
      System.out.println( "File " + fileName
                          + " currently exists." );
      System.out.print( "Overwrite the file? (y/n): " );
      String inputAns = sc.next();
      char ans = inputAns.charAt( 0 );
      if ( ans == 'n' )
        System.exit( 0 );  // exit
    }
```

```
    try {
      PrintWriter pwStream
        = new PrintWriter( new BufferedWriter(
                                 new FileWriter( fileName ) ) );
      int num ;
      for ( num = 0 ; num < 5 ; num++ )
      {
        pwStream.println( "Number = " + num * 5 );
      }

      pwStream.close();
    }
    catch ( IOException e ) {
      System.out.println( "IO Error!" + e.getMessage() );
      System.exit(0);
    }
  }
}
```

Program Input and Output

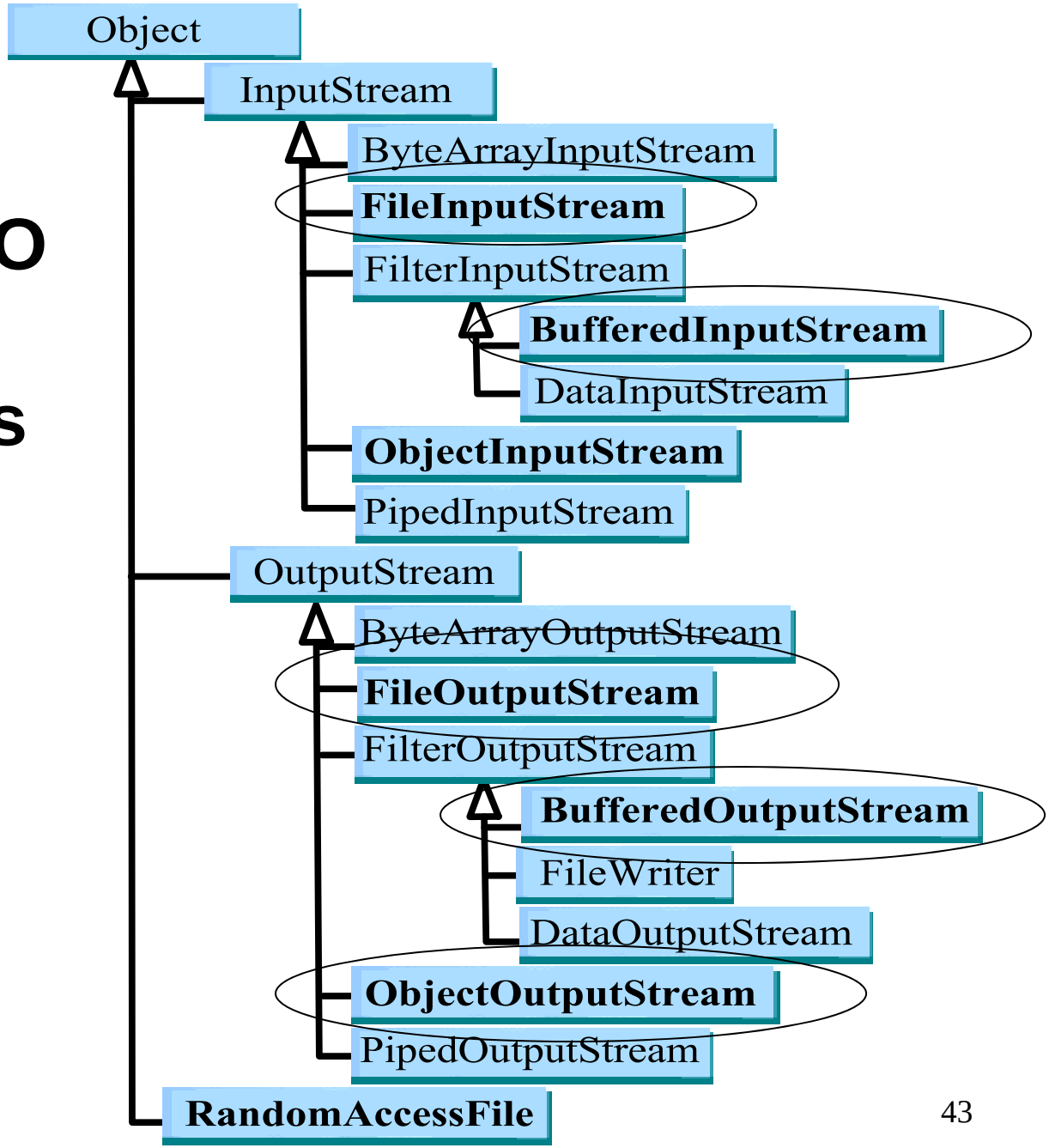
```
Enter the file name:
output.txt
File output.txt currently exists.
Overwrite the file? (y/n): n

Enter the file name:
output.txt
File output.txt currently exists.
Overwrite the file? (y/n): y
(After execution, the file output.txt contains)
Number = 0
Number = 5
Number = 10
Number = 15
Number = 20
```

Chapter 14: File Input/Output

- Streams and Files
- File Stream Processing
- Text File I/O
- The File Class
- **Binary File I/O**

Binary File I/O

Stream class hierarchy



Output Stream Classes for Writing Binary Files

Class	Description	Methods
`FileOutputStream`	Provides the basic stream for text output.	`write()`, `flush()`, `close()`
`BufferedOutputStream`	Provides output buffering to improve performance.	`write()`, `flush()`, `close()`
`ObjectOutputStream`	Provides a number of useful output methods for processing.	`writeByte()`, `writeBoolean()`, `writeBytes()`, `writeChar()`, `writeChars()`, `writeDouble()`, `writeFloat()`, `writeInt()`, `writeLong()`, `writeShort()`, `writeUTF()`, `flush()`, `close()`

http://java.sun.com/j2se/1.5.0/docs/api/java/io/package-tree.html

Using the Class ObjectOutputStream to Write Binary Data

```
import java.io.* ;
import java.util.Scanner ;
public class WritingBinaryData {
  public static void main( String[] args ) {
    String fileName = " " ;
    try {
      String name   ;
      int    age    ;
      double height ;
      System.out.println( "Enter the file name: " );
      Scanner sc = new Scanner( System.in );
      fileName   = sc.nextLine();

      FileOutputStream     foStream
            = new FileOutputStream(     fileName );
      BufferedOutputStream boStream
            = new BufferedOutputStream( foStream );
      ObjectOutputStream   doStream
            = new ObjectOutputStream(   boStream );

      int i ;
```

```
    for ( i = 0 ; i < 3 ; i++ ) {
      System.out.print( "Enter name: " );
      name   = sc.next();
      System.out.print( "Enter age: " );
      age    = sc.nextInt();
      System.out.print( "Enter height: " );
      height = sc.nextDouble();
      doStream.writeUTF(    name   );   // name
      doStream.writeInt(    age    );   // age
      doStream.writeDouble( height );   // height
    }
    System.out.println( "Writing completed!" );
    doStream.close();
  }
  catch ( FileNotFoundException e ) {
    System.out.println( "IOError: File not found!" + fileName );
    System.exit( 0 );
  }
  catch ( IOException e ) {
    System.out.println( "File IO Error!" + e.getMessage() );
    System.exit( 0 );
  }
}
}
```

Program Input and Output

```
Enter the file name:
binary.dat
Enter name: Junrong
Enter age: 16
Enter height: 1.76
Enter name: ChooEng
Enter age: 17
Enter height: 1.77
Enter name: MayLing
Enter age: 18
Enter height: 1.78
Writing completed!
```

ObjectOutputStream Methods

Method	Description
`writeByte(int b)`	Writes the **`byte b`** to the data output stream.
`writeBoolean(boolean b)`	Writes the **`boolean b`** to the data output stream.
`writeBytes(String s)`	Writes the **`String s`** to the data output stream as a sequence of bytes.
`writeChar(int c)`	Writes the **`char c`** to the data output stream as a 2-byte Unicode value.
`writeChars(String s)`	Writes the **`String s`** to the data output stream as a sequence of characters. Each character is written as 2-byte Unicode value.
`writeDouble(double d)`	Writes the **`double d`** to the data output stream using 8 bytes.
`writeFloat(float f)`	Writes the **`float f`** to the data output stream using 4 bytes.
`writeInt(int i)`	Writes the **`int i`** to the data output stream using 4 bytes.
`writeLong(long l)`	Writes the **`long l`** to the data output stream using 8 bytes.
`writeShort(short s)`	Writes the **`short s`** to the data output stream using 2 bytes.
`writeUTF(String s)`	Writes the **`String s`** to the data output stream. **`UTF`** is a particular encoding method for the string.
`flush()`	Flushes the stream. This will write any buffered output bytes and flush through to the underlying stream.
`close()`	Closes the file output stream connection.

http://java.sun.com/j2se/1.5.0/docs/api/java/io/ObjectOutputStream.html

Further study: hardware level flush/sync, see java.io.FileDescriptor

The Unicode Text Format (UTF)

- Unicode
  - Uses a 16-bit representation, and is used to represent the ASCII character set and a large variety of Asian and other international language characters.
- ASCII
  - Uses a 8-bit representation.
- Unicode encoding is not an efficient scheme, for programs that use only the standard ASCII characters, in which one byte (out of two) is wasted when the characters are saved in Unicode.
- UTF stands for Unicode Text Format
  - An alternative coding scheme for ASCII characters.
- When we do not need to use other Unicode characters, the UTF coding scheme could be used for representing the ASCII character set.

Input Stream Classes for Reading Binary Files

Class	Description	Methods
`FileInputStream`	Provides the basic stream for byte-based input.	`read()`, `close()`
`BufferedInputStream`	Provides input buffering to improve performance.	`read()`, `close()`
`ObjectInputStream`	Provides a number of useful input methods for processing.	`readByte()`, `readBoolean()`, `readChar()`, `readDouble()`, `readFloat()`, `readInt()`, `readLong()`, `readShort()`, `readUTF()`, `close()`

Using the Class ObjectInputStream to Read Binary Data

```
import java.util.Scanner ;
import java.io.* ;
public class ReadingBinaryData1 {
  public static void main( String[] args ) {
    String fileName = " " ;
    try {
      String  name   ;
      int     age    ;
      double  height ;
      Scanner sc = new Scanner( System.in );

      System.out.println( "Enter the file name: " );
      fileName = sc.nextLine();

      FileInputStream     fiStream
            = new FileInputStream(     fileName );
      BufferedInputStream biStream
            = new BufferedInputStream( fiStream );
      ObjectInputStream   diStream
            = new ObjectInputStream(   biStream );

      int i ;
```

Using the Class ObjectInputStream to Read Binary Data

```
      for ( i = 0 ; i < 3 ; i++ )
      {
        System.out.print( "Name: " );
        System.out.println( name   = diStream.readUTF()    );
        System.out.print( "Age: " );
        System.out.println( age    = diStream.readInt()    );
        System.out.print( "Height: " );
        System.out.println( height = diStream.readDouble() );
      }
      diStream.close();
    }
    catch ( FileNotFoundException e ) {
      System.out.println( "IOError: File not found!" + fileName );
      System.exit( 0 );
    }
    catch ( IOException e ) {
      System.out.println( "File IO Error!" + e.getMessage() );
      System.exit( 0 );
    }
  }
}
```

Program Input and Output

```
Enter the file name:
binary.dat
Name: Junrong
Age: 16
Height: 1.76
Name: ChooEng
Age: 17
Height: 1.77
Name: MayLing
Age: 18
Height: 1.78
```

ObjectInputStream Class Methods

Method	Description
`readByte()`	Reads a `byte` from the data input stream.
`readBoolean()`	Reads a `boolean` from the data input stream.
`readChar()`	Reads a `char` (2 bytes) from the data input stream.
`readDouble()`	Reads a `double` (8 bytes) from the data input stream.
`readFloat()`	Reads a `float` (4 bytes) from the data input stream.
`readInt()`	Reads an `int` (4 bytes) from the data input stream.
`readLong()`	Reads a `long` (8 bytes) from the data input stream.
`readShort()`	Reads a `short` (2 bytes) from the data input stream.
`readUTF()`	Reads a `String` value from the data input stream.
`close()`	Closes the file input stream connection.

Reading Binary Data with End of File Testing

```
import java.io.* ;
import java.util.Scanner ;
public class ReadingBinaryData2 {
  public static void main( String[] args ) {
    String fileName = " " ;
    try {
      String  name   ;
      int     age    ;
      double  height ;
      Scanner sc = new Scanner( System.in );

      System.out.println( "Enter the file name: " );
      fileName = sc.nextLine();

      FileInputStream     fiStream
            = new FileInputStream(     fileName );
      BufferedInputStream biStream
            = new BufferedInputStream( fiStream );
      ObjectInputStream   diStream
            = new ObjectInputStream(   biStream );

      int i ;
```

When reading beyond the file, an end-of-file Exception (EOFException) is thrown
Use end-of-file exception for testing end-of-file

```
1:  try {
      while ( true ) {
2:      System.out.print( "Name: " );
        System.out.println( name   = diStream.readUTF() );
        System.out.print( "Age: " );
        System.out.println( age    = diStream.readInt() );
        System.out.print( "Height: " );
        System.out.println( height = diStream.readDouble() );
      }
    }
3:  catch ( EOFException e ) {}  // or IOException
    diStream.close();
  }
  catch ( FileNotFoundException e ) {
    System.out.println( "IOError: File not found!" + fileName );
    System.exit( 0 );
  }
  catch ( IOException e ) {
    System.out.println( "File IO Error!" + e.getMessage() );
    System.exit( 0 );
  }
}
}
```

to catch the end-of-file exception

Note more specific catch blocks first, then the general one

Program Input and Output

```
Enter the file name:
binary.dat
Name: Junrong
Age: 16
Height: 1.76
Name: ChooEng
Age: 17
Height: 1.77
Name: MayLing
Age: 18
Height: 1.78
```

Further Reading

- Read Chapter 14 on “File Input/Output” of the textbook.

http://java.sun.com/j2se/1.5.0/docs/api/java/io/package-tree.html

Thank you !!!