10.2 Text File I/O

- Important classes for text file **output** (to the file)
  - » **`PrintWriter – filter class`**
  - » **`FileOutputStream`**
- Important classes for text file **input** (from the file):
  - » **`BufferedReader – filter class`**
  - » **`FileReader`**
- Note that **`FileOutputStream`** and **`FileReader`** are used only for their constructors, which can take file names as arguments.

- To use these classes your program needs a line like the following:

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

Listing 10.1 Sending Output to a Text File - TextFileOutputDemo.java

```
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class TextFileOutputDemo
{
   public static void main(String[] args)
   {
      String fileName = "out.txt"; //The name could be read from
                                   //the keyboard.
      PrintWriter outputStream = null;
      try
      {
         outputStream = new PrintWriter(fileName);
      }
```

```
public PrintWriter(String fileName)
            throws FileNotFoundException

public PrintWriter(File file)
            throws FileNotFoundException
```

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

        System.out.println("Enter three lines of text:");
        Scanner keyboard = new Scanner(System.in);
        for (int count = 1; count <= 3; count++)
        {
            String line = keyboard.nextLine( );
            outputStream.println(count + " " + line);
        }
        outputStream.close( );
        System.out.println("Those lines were written to " +
                           fileName);
    }
}
```

C:₩WINDOWS₩system32₩cmd.exe

```
Enter three lines of text:
aaaaa
bbbbb
ccccc
Those lines were written to out.txt.
계속하려면 아무 키나 누르십시오 . . .
```

out.txt - 메모장

파일(F) 편집(E) 서식(O) 보기(

```
1 aaaaa
2 bbbbb
3 ccccc
```

FileOutputStream()

» PrintWriter : useful that supports writing values of various data types by converting them to their string representation.

```
import java.io.PrintWriter;
import java.io.FileOutputStream;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class TextFileOutputDemo2
{
   public static void main(String[] args)
   {
      String fileName = "out.txt"; //The name could be read from
                                   //the keyboard.
      PrintWriter outputStream = null;
      try
      {
         outputStream = new PrintWriter(new FileOutputStream(fileName));
      }
```

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

        System.out.println("Enter three lines of text:");
        Scanner keyboard = new Scanner(System.in);
        for (int count = 1; count <= 3; count++)
        {
            String line = keyboard.nextLine( );
            outputStream.println(count + " " + line);
        }
        outputStream.close( );
        System.out.println("Those lines were written to " +
                            fileName);
    }
}
```

Every File Has Two Names

- The code to open the file creates two names for an output file
  - » the name used
    - `out.txt` in the example
  - » e (that is connected to the file)
    - `outputStream` in the example

- Java programs use the *stream* name
  - » `outputStream` in the example
  - » Serves as a temporary name for the file that is used within the your program.

Text File Output

- To open a text file for output: connect a text file to a stream for writing
  - » create a stream of the class `PrintWriter` and connect it to a text file

  For example:

```
PrintWriter outputStream =
   new PrintWriter(new FileOutputStream("out.txt");
```

- Then you can use `print` and `println` to write to the file

```
outputStream.println(count + " " + line);
```

- The text lists some other useful `PrintWriter` methods

java.io

Class PrintWriter

```
java.lang.Object
  └ java.io.Writer
      └ java.io.PrintWriter
```

All Implemented Interfaces:

Closeable, Flushable, Appendable

```
public class PrintWriter
extends Writer
```

Print formatted representations of objects to a text-output stream. This class implements all of the `print` methods found in `PrintStream`. It does not contain methods for writing raw bytes, for which a program should use unencoded byte streams.

Unlike the `PrintStream` class, if automatic flushing is enabled it will be done only when one of the `println`, `printf`, or `format` methods is invoked, rather than whenever a newline character happens to be output. These methods use the platform's own notion of line separator rather than the newline character.

Methods in this class never throw I/O exceptions, although some of its constructors may. The client may inquire as to whether any errors have occurred by invoking `checkError()`.

Since:

JDK1.1

Constructor Summary

PrintWriter(File file) Creates a new PrintWriter, without automatic line flushing, with the specified file.
PrintWriter(File file, String csn) Creates a new PrintWriter, without automatic line flushing, with the specified file and charset.
PrintWriter(OutputStream out) Create a new PrintWriter, without automatic line flushing, from an existing OutputStream.
PrintWriter(OutputStream out, boolean autoFlush) Create a new PrintWriter from an existing OutputStream.
PrintWriter(String fileName) Creates a new PrintWriter, without automatic line flushing, with the specified file name.
PrintWriter(String fileName, String csn) Creates a new PrintWriter, without automatic line flushing, with the specified file name and charset.
PrintWriter(Writer out) Create a new PrintWriter, without automatic line flushing.
PrintWriter(Writer out, boolean autoFlush) Create a new PrintWriter.

메소드의 개요

PrintWriter	append (char c) 지정된 문자를 이 라이터에 추가합니다.
PrintWriter	append (CharSequence csq) 지정된 문자 순서를 이 라이터에 추가합니
PrintWriter	append (CharSequence csq, int start, int 지정된 문자 순서의 서브 순서를 이 라이터
boolean	checkError () 스트림이 닫혀지지 않은 경우는, 그 스트림
protected void	clearError () 이 스트림의 에러 상태를 해제합니다.
void	close () 스트림을 닫아, 거기에 관련하는 모든 syste
void	flush () 스트림을 플래시 합니다.
PrintWriter	format (Locale l, String format, Object 지정된 서식 캐릭터 라인 및 인수를 사용하다.
PrintWriter	format (String format, Object ... args) 지정된 서식 캐릭터 라인 및 인수를 사용하다.
void	print (boolean b) boolean 형의 값을 출력합니다.
void	print (char c) 문자를 출력합니다.
void	print (char[] s) 문자의 배열을 출력합니다.
void	print (double d) 배정밀도의 부동 소수점수(실수)를 출력합

void	print (float f) 부동 소수점수(실수)를 출력합니다.
void	print (int i) 정수를 출력합니다.
void	print (long l) long 정수를 출력합니다.
void	print (Object obj) 객체를 출력합니다.
void	print (String s) 캐릭터 라인을 출력합니다.
PrintWriter	printf (Locale l, String format, Object . 지정된 서식 캐릭터 라인 및 인수를 사용해, 편리한 메소드입니다.
PrintWriter	printf (String format, Object ... args) 지정된 서식 캐릭터 라인 및 인수를 사용해, 편리한 메소드입니다.
void	println () 행의 단락 캐릭터 라인을 기입하는 것으로,
void	println (boolean x) boolean 치를 출력해, 행을 종료시킵니다.
void	println (char x) 문자를 출력해, 행을 종료시킵니다.
void	println (char[] x) 문자의 배열을 출력해, 행을 종료시킵니다.
void	println (double x) 배정밀도 부동 소수점수(실수)를 출력해, 행
void	println (float x) 부동 소수점수(실수)를 출력해, 행을 종료시
void	println (int x) 정수를 출력해, 행을 종료시킵니다.
void	println (long x) long 형의 정수치를 출력해, 행을 종료시킵니
void	println (Object x) Object 를 출력해, 행을 종료시킵니다.
void	println (String x) 캐릭터 라인을 출력해, 행을 종료시킵니다.
protected void	setError () 에러가 발생한 것을 나타냅니다.
void	write (char[] buf) 문자의 배열을 기입합니다.
void	write (char[] buf, int off, int len) 문자의 배열의 일부를 기입합니다.
void	write (int c) 단일의 문자를 기입합니다.
void	write (String s) 캐릭터 라인을 기입합니다.
void	write (String s, int off, int len) 캐릭터 라인의 일부를 기입합니다.

TextFileOutputDemo

Part 1

```
public static void main(String[] args)
{
    PrintWriter outputStream = null;
    try
    {
        outputStream =
            new PrintWriter(new FileOutputStream("out.txt"));
    }
    catch(FileNotFoundException e)
    {
        System.out.println("Error openin
        System.exit(0);
    }
```

A `try`-block is a block: `outputStream` would not be accessible to the rest of the method if it were declared inside the `try`-block

Opening the file

Creating a file can cause the `FileNotFound-Exception` if the new file cannot be made.

java.io

Class FileOutputStream

```
java.lang.Object
  └ java.io.OutputStream
      └ java.io.FileOutputStream
```

All Implemented Interfaces:

Closeable, Flushable

```
public class FileOutputStream
extends OutputStream
```

A file output stream is an output stream for writing data to a `File` or to a `FileDescriptor`. Whether or not a file is available or may be created depends upon the underlying platform. Some platforms, in particular, allow a file to be opened for writing by only one `FileOutputStream` (or other file-writing object) at a time. In such situations the constructors in this class will fail if the file involved is already open.

`FileOutputStream` is meant for writing streams of raw bytes such as image data. For writing streams of characters, consider using `FileWriter`.

Since:

JDK1.0

See Also:

`File`, `FileDescriptor`, `FileInputStream`

Constructor Summary
FileOutputStream(File file) Creates a file output stream to write to the file represented by the specified File object.
FileOutputStream(File file, boolean append) Creates a file output stream to write to the file represented by the specified File object.
FileOutputStream(FileDescriptor fdObj) Creates an output file stream to write to the specified file descriptor, which represents an existing connection to an actual file in the file system.
FileOutputStream(String name) Creates an output file stream to write to the file with the specified name.
FileOutputStream(String name, boolean append) Creates an output file stream to write to the file with the specified name.

FileOutputStream

```
public FileOutputStream(String  name)
                 throws FileNotFoundException
```

지정된 이름의 파일에 기입하기 위한 파일 출력 스트림을 작성합니다. 이 파일 접속을 나타내기 위해서 (때문에), 새로운 `FileDescriptor` 객체가 생성됩니다.

우선, 시큐리티 매니저가 존재하는 경우,`checkWrite` 메소드가 `name` 를 인수로서 불려 갑니다.

파일은 존재하지만, 보통 파일은 아니고 디렉토리인 경우, 파일은 존재하지 않고 작성도 할 수 없는 경우, 또는 하등의 이유로써 열릴 수가 없는 경우는,`FileNotFoundException` 가 throw 됩니다.

파라미터:

`name` - 시스템에 의존하는 파일명

예외:

`FileNotFoundException` - 파일은 존재하지만, 보통 파일은 아니고 디렉토리인 경우, 파일은 존재하지 않고 작성도 할 수 없는 경우, 또는 하등의 이유로써 열릴 수가 없는 경우

`SecurityException` - 시큐리티 매니저가 존재해,`checkWrite` 메소드가 파일에의 기입해 액세스를 거부하는 경우

관련 항목:

`SecurityManager.checkWrite(java.lang.String)`

Methods of FileOutputStream

Method Summary	
void	**close**() Closes this file output stream and releases any system resources associated with this stream.
protected void	**finalize**() Cleans up the connection to the file, and ensures that the `close` method of this file output stream is called when there ar no more references to this stream.
FileChannel	**getChannel**() Returns the unique `FileChannel` object associated with this file output stream.
FileDescriptor	**getFD**() Returns the file descriptor associated with this stream.
void	**write**(byte[] b) Writes `b.length` bytes from the specified byte array to this file output stream.
void	**write**(byte[] b, int off, int len) Writes `len` bytes from the specified byte array starting at offset `off` to this file output stream.
void	**write**(int b) Writes the specified byte to this file output stream.

TextFileOutputDemo

Part 2

```
System.out.println("Enter three lines of text:");
Scanner keyborad = new Scanner(System.in);
   for (int count = 1; count <= 3; count++)
   {
       String line = keyboard.readLine();
       outputStream.println(count + " " + line);
   }
   outputStream.close();
   System.out.println("... written to out.txt.");
}
```

Writing to the file

Closing the file

The `println` method is used with two different streams: `outputStream` and `System.out`

Gotcha: Overwriting a File

- Opening a file creates an empty file
- Opening a file creates a new file if it does not already exist
- Opening a file that already exists eliminates the old file and creates a new, empty one
  - » data in the original file is lost
- To see how to check for existence of a file, see the section of the text that discusses the `File` class (and a later slide).

Java Tip: Appending to a Text File

- To add to a file instead of replacing it, use a different constructor for **`FileOutputStream`**:

```
outputStream =
   new PrintWriter(new FileOutputStream("out.txt",      );
```

Second parameter indicates that file should not be replaced if it already exists.

- Data written to file will be added to the end of the file.

- Sample code for letting user tell whether to replace or append:

```
System.out.println("A for append or N for new file:");
String ans = keyboard.nextLine();
boolean append =(ans.equalsIgnoreCase("A"));
outputStream = new PrintWriter(
      new FileOutputStream("out.txt", append));
```

true if user enters 'A'

append

```
import java.io.PrintWriter;
import java.io.FileOutputStream;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class TextFileOutputDemo3
{
   public static void main(String[] args)
   {
      String fileName = "out.txt"; //The name could be read from
                                   //the keyboard.
      PrintWriter outputStream = null;
      Scanner keyboard = new Scanner(System.in);
      try
      {
         System.out.println("A for append or N for new file:");
         String ans = keyboard.nextLine();
         boolean append = (ans.equalsIgnoreCase("A"));
         outputStream = new PrintWriter(new FileOutputStream("out.txt",
append));
      }
```

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

    System.out.println("Enter three lines of text:");

    for (int count = 1; count <= 3; count++)
    {
        String line = keyboard.nextLine( );
        outputStream.println(count + " " + line);
    }
    outputStream.close( );
    System.out.println("Those lines were written to " +
                       fileName);
  }
}
```

Closing a File

- An output file should be closed when you are done writing to it (and an input file should be closed when you are done reading from it).

- Use the `close` method of the class `PrintWriter` (`BufferedReader` also has a `close` method).

- For example, to close the file opened in the previous example:

```
outputStream.close();
```

- If a program ends normally it will close any files that are open.

FAQ: Why Bother to Close a File?

If a program automatically closes files when it ends normally, why close them with explicit calls to `close`?

<u>Two reasons:</u>

1. To make sure it is closed if a program ends (it could get damaged if it is left open).

2. A file open for writing must be before it can be opened for reading.
   - Although Java does have a class that opens a file for both reading and writing, it is not used in this text.

Text File Input with BufferedReader

- To open a text file for input: connect a text file to a stream for reading
  - » use a stream of the class `BufferedReader` and connect it to a text file
  - » use the `FileReader` class to connect the `BufferedReader` object to the text file
  - » that support buffered character-based I/O
- For example:

```
BufferedReader inputStream =
  new BufferedReader(new FileReader("data.txt"));
```

- Then:
  - » read lines **(`Strings`)** with `readLine`
  - » `BufferedReader` has no methods to read numbers directly, so read numbers as `String`s and then convert them
  - » read **a `char`** with `read`

Listing 10.2 Reading Data from a Text File

```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
public class TextFileInputDemo
{
   public static void main (String [] args)
   {
      String fileName = "out.txt";
      Scanner inputStream = null;
      System.out.println ("The file " + fileName +
            "\ncontains the following lines:\n");
```

```
        try
        {
            inputStream = new Scanner (new File (fileName));
        }
        catch (FileNotFoundException e)
        {
            System.out.println ("Error opening the file " +
                    fileName);
            System.exit (0);
        }
        while (inputStream.hasNextLine ())
        {
            String line = inputStream.nextLine ();
            System.out.println (line);
        }
        inputStream.close ();
    }
}
```

```
The file out.txt
contains the following lines:

1 A tall tree
2 in a short forest is like
3 a big fish in a small pond.
```

Reading from a Text File

- Figure 10.3 Additional methods in class `Scanner`

Scannner_Object_Name`.hasNext()` Returns true if more input data is available to be read by the method `next`.
Scannner_Object_Name`.hasNextDouble()` Returns true if more input data is available to be read by the method `nextDouble`.
Scannner_Object_Name`.hasNextInt()` Returns true if more input data is available to be read by the method `nextInt`.
Scannner_Object_Name`.hasNextLine()` Returns true if more input data is available to be read by the method `nextLine`.

Use toString for Text-File Output Display ## The Species Class with a toString Method - Species.java

```
import java.io.Serializable;
import java.util.Scanner;

/**
 Serialized class for data on endangered species.
 Includes a main method.
*/
public class Species
{
   private String name;
   private int population;
   private double growthRate;

   public Species( )
   {
      name = null;
      population = 0;
      growthRate = 0;
   }

   public Species(String initialName, int initialPopulation,
                  double initialGrowthRate)
   {
```

```
public String toString()
  {
     return ("Name = " + name + "\n"
          + "Population = " + population + "\n"
          + "Growth rate = " + growthRate + "%");
  }

  public void readInput( )
  {
     Scanner keyboard = new Scanner(System.in);
     System.out.println("What is the species' name?");
     name = keyboard.nextLine( );

     System.out.println(
                "What is the population of the species?");
     population = keyboard.nextInt( );
     while (population < 0)
     {
        System.out.println("Population cannot be negative.");
        System.out.println("Reenter population:");
        population = keyboard.nextInt( );
     }

     System.out.println("Enter growth rate (% increase per year):");
    growthRate = keyboard.nextDouble( );
  }
……
```

Display ## Using toString to Output an Object. - TextFileObjectOutputDemo.java

- Use to String for Text-File Output

```
import java.io.PrintWriter;
import java.io.FileNotFoundException;

public class TextFileSpeciesOutputDemo
{
   public static void main(String[] args)
   {
      PrintWriter outputStream = null;
      try
      {
         outputStream =
            new PrintWriter("species.records");
      }
      catch(FileNotFoundException e)
      {
         System.out.println("Error opening species.records.");
         System.exit(0);
      }
```

```
Species oneRecord =
            new Species("Calif. Condor", 27, 0.02);


    outputStream.println[redacted]));  // equivalent
    outputStream.println( );
    outputStream.printlr[redacted]d); // equivalent
    outputStream.close( );
    System.out.println("End of Program.");
  }
}
```

```
tsW@@@@@@@jvWch09Wspecies.records]
Macros   Configure   Window   Help

Name = Calif. Condor
Population = 27
Growth rate = 0.02%

Name = Calif. Condor
Population = 27
Growth rate = 0.02%
```

끝