Input/Output

2301260 Programming Techniques

ผศ. ศศิภา พันธุวดีธร ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์

คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

File class

- An abstract representation of file and directory pathnames.
- Provides useful information about a file or directory
- Does not open files or process files
- Create a file object

  File in = new File (“input.txt”);
- Construct a FileReader from a File object
- FileReader reader = new FileReader(in);

File class

- Some interesting methods
  - String getName()
  - String getPath()
  - boolean isFile()
  - boolean isDirectory()
  - boolean exists()
  - boolean canRead()
  - boolean canWrite()
  - long length()

```
package fileclass;
import java.io.*;
public class Test {
   public static void main(String[] args) {
      File in = new File("C:\\Documents and Settings\\Sasipa\\My
   Documents\\NetBeansProjects\\FileClass\\input.txt");
      System.out.println("File name : " + in.getName());
      System.out.println("Path name : " + in.getPath());
      System.out.println("Is it a File : " + in.isFile());
      System.out.println("Is it a directory : " + in.isDirectory());
      System.out.println("Does the file exist : " + in.exists());
      System.out.println("Can read : " + in.canRead());
      System.out.println("Can write : " + in.canWrite());
      System.out.println("Data length : " + in.length());
   }
}                                          // input.txt contains This is my test file.
```

output

File name : input.txt

Path name : C:\Documents and Settings\Sasipa\My Documents\NetBeansProjects\FileClass\input.txt

Is it a File : true

Is it a directory : false

Does the file exist : true

Can read : true

Can write : true

Data length : 21

Obtaining file properties and manipulating file

java.io.File	
`+File(pathname: String)`	Creates a `File` object for the specified path name. The path name may be a directory or a file.
`+File(parent: String, child: String)`	Creates a `File` object for the child under the directory parent. The child may be a file name or a subdirectory.
`+File(parent: File, child: String)`	Creates a `File` object for the child under the directory parent. The parent is a `File` object. In the preceding constructor, the parent is a string.
`+exists(): boolean`	Returns true if the file or the directory represented by the `File` object exists.
`+canRead(): boolean`	Returns true if the file represented by the `File` object exists and can be read.
`+canWrite(): boolean`	Returns true if the file represented by the `File` object exists and can be written.
`+isDirectory(): boolean`	Returns true if the `File` object represents a directory.
`+isFile(): boolean`	Returns true if the `File` object represents a file.
`+isAbsolute(): boolean`	Returns true if the `File` object is created using an absolute path name.
`+isHidden(): boolean`	Returns true if the file represented in the `File` object is hidden. The exact definition of *hidden* is system-dependent. On Windows, you can mark a file hidden in the File Properties dialog box. On Unix systems, a file is hidden if its name begins with a period(.) character.
`+getAbsolutePath(): String`	Returns the complete absolute file or directory name represented by the `File` object.
`+getCanonicalPath(): String`	Returns the same as `getAbsolutePath()` except that it removes redundant names, such as "." and "..", from the path name, resolves symbolic links (on Unix), and converts drive letters to standard uppercase (on Windows).
`+getName(): String`	Returns the last name of the complete directory and file name represented by the `File` object. For example, new `File("c:\\book\\test.dat").getName()` returns `test.dat`.
`+getPath(): String`	Returns the complete directory and file name represented by the `File` object. For example, new `File("c:\\book\\test.dat").getPath()` returns `c:\book\test.dat`.
`+getParent(): String`	Returns the complete parent directory of the current directory or the file represented by the `File` object. For example, new `File("c:\\book\\test.dat").getParent()` returns `c:\book`.
`+lastModified(): long`	Returns the time that the file was last modified.
`+length(): long`	Returns the size of the file, or 0 if it does not exist or if it is a directory.
`+listFile(): File[]`	Returns the files under the directory for a directory `File` object.
`+delete(): boolean`	Deletes the file or directory represented by this `File` object.The method returns true if the deletion succeeds.
`+renameTo(dest: File): boolean`	Renames the file or directory represented by this `File` object to the specified name represented in dest. The method returns true if the operation succeeds.
`+mkdir(): boolean`	Creates a directory represented in this `File` object. Returns true if the the directory is created successfully.
`+mkdirs(): boolean`	Same as `mkdir()` except that it creates directory along with its parent directories if the parent directories do not exist.

Steps of Input/Output operation

- Input
  - Open file
  - While there are more data to read, Read data
  - Close file
- Output
  - Open file
  - While there are more data to write, Write file
  - Close file
- An IOException may occur during any I/O operation (checked exception)

Catch or Declare Checked Exceptions

Suppose p2 is defined as follows:

```
void p2() throws IOException {
  if (a file does not exist) {
     throw new IOException("File does not exist");
  }

  ...
}
```

Catch or Declare Checked Exceptions

Java forces you to deal with checked exceptions. If a method declares a checked exception (i.e., an exception other than Error or RuntimeException), you must invoke it in a try-catch block or declare to throw the exception in the calling method. For example, suppose that method p1 invokes method p2 and p2 may throw a checked exception (e.g., IOException), you have to write the code as shown in (a) or (b).

```
void p1() {
  try {
    p2();
  }
  catch (IOException ex) {
    ...
  }
}
```

(a)

```
void p1() throws IOException {

  p2();

}
```

(b)

Text I/O

- A File object encapsulates the properties of a file or a path, but does not contain the methods for reading/writing data from/to a file.
- To perform I/O, you need to create objects using appropriate Java I/O classes. The objects contain the methods for reading/writing data from/to a file.
- This section introduces how to read/write strings and numeric values from/to a text file using the Scanner and PrintWriter classes.

Writing Data Using PrintWriter

java.io.PrintWriter	
+PrintWriter(filename: String)	Creates a PrintWriter for the specified file.
+print(s: String): void	Writes a string.
+print(c: char): void	Writes a character.
+print(cArray: char[]): void	Writes an array of character.
+print(i: int): void	Writes an int value.
+print(l: long): void	Writes a long value.
+print(f: float): void	Writes a float value.
+print(d: double): void	Writes a double value.
+print(b: boolean): void	Writes a boolean value.
Also contains the overloaded println methods.	A println method acts like a print method; additionally it prints a line separator. The line separator string is defined by the system. It is \r\n on Windows and \n on Unix.
Also contains the overloaded printf methods.	The printf method was introduced in §3.6, "Formatting Console Output and Strings."

```
public class WriteData {
  public static void main(String[] args) throws Exception {
    java.io.File file = new java.io.File("scores.txt");
    if (file.exists()) {
      System.out.println("File already exists");
      System.exit(0);
    }

    // Create a file
    java.io.PrintWriter output = new java.io.PrintWriter(file);

    /* you can create PrintWriter objects for writing text to any file using print,  println, and printf **/
    // Write formatted output to the file

    output.print("John T Smith ");
    output.println(90);
    output.print("Eric K Jones ");
    output.println(85);

    // Close the file
    output.close();
  }
}
```

scores.txt

John T Smith 90

Eric K Jones 85

Reading Data Using Scanner

java.util.Scanner	
+Scanner(source: File)	Creates a Scanner object to read data from the specified file.
+Scanner(source: String)	Creates a Scanner object to read data from the specified string.
+close()	Closes this scanner.
+hasNext(): boolean	Returns true if this scanner has another token in its input.
+next(): String	Returns next token as a string.
+nextByte(): byte	Returns next token as a byte.
+nextShort(): short	Returns next token as a short.
+nextInt(): int	Returns next token as an int.
+nextLong(): long	Returns next token as a long.
+nextFloat(): float	Returns next token as a float.
+nextDouble(): double	Returns next token as a double.
+useDelimiter(pattern: String): Scanner	Sets this scanner's delimiting pattern.

ch11/lines/LineNumberer.java from BigJava textbook

```
 1  import java.io.File;
 2  import java.io.FileNotFoundException;
 3  import java.io.PrintWriter;
 4  import java.util.Scanner;
 5
 6  /**
 7     This program applies line numbers to a file.
 8  */
 9  public class LineNumberer
10  {
11     public static void main(String[] args) throws FileNotFoundException
12     {
13        // Prompt for the input and output file names
14
15        Scanner console = new Scanner(System.in);
16        System.out.print("Input file: ");
17        String inputFileName = console.next();
18        System.out.print("Output file: ");
19        String outputFileName = console.next();
20
```

Class LineNumber

just declare

No try-catch

```
21          // Construct the Scanner and PrintWriter objects for reading and writing
22
23          File inputFile = new File(inputFileName);
24          Scanner in = new Scanner(inputFile);
25          PrintWriter out = new PrintWriter(outputFileName);
26          int lineNumber = 1;
27
28          // Read the input and write the output
29
30          while (in.hasNextLine())
31          {
32             String line = in.nextLine();
33             out.println("/* " + lineNumber + " */ " + line);
34             lineNumber++;
35          }
36
37          in.close();
38          out.close();
39       }
40    }
```

```
Input file: x.txt
Output file: B.txt
Exception in thread "main" java.io.FileNotFoundException: x.txt (The system cannot find the file specified)
        at java.io.FileInputStream.open(Native Method)
        at java.io.FileInputStream.<init>(FileInputStream.java:138)
        at java.util.Scanner.<init>(Scanner.java:656)
        at linenumber.LineNumber.main(LineNumber.java:24)
Java Result: 1
```

Class LineNumber (use try-catch)

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

public class LineNumber  {
   public static void main(String[] args)   {      // was changed
      try   {                                      // was added
         Scanner console = new Scanner(System.in);
         System.out.print("Input file: ");
         String inputFileName = console.next();
         File inputFile = new File(inputFileName);
         Scanner in = new Scanner(inputFile);
```

```
        System.out.print("Output file: ");
        String outputFileName = console.next();
        PrintWriter out = new PrintWriter(outputFileName);
        int lineNumber = 1;
        while (in.hasNextLine())   {
            String line = in.nextLine();
            out.println("/* " + lineNumber + " */ " + line);
            lineNumber++;
        }
        in.close();                    // doesn't reach it if exception occurs
        out.close();                   // cause resource leak if exception occurs
    }
    catch (FileNotFoundException e)     {
        System.out.println (e);
    }
  }
}
```

Programmer calls toString()
Return the package of exception followed by the message that describes the exception

Class LineNumber (use try-catch-finally)

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

public class LineNumber    {
  public static void main(String[] args)    {
     Scanner console = new Scanner(System.in);
     System.out.print("Input file: ");
     String inputFileName = console.next();

     File inputFile = new File(inputFileName);
     Scanner in = null;
     PrintWriter out = null;
```

```
try      {
   in = new Scanner(inputFile);
   System.out.print("Output file: ");
   String outputFileName = console.next();
   out = new PrintWriter(outputFileName);
   int lineNumber = 1;

// Read the input and write the output
   while (in.hasNextLine())        {
      String line = in.nextLine();
      out.println("/* " + lineNumber + " */ " + line);
      lineNumber++;
   }
 }
 catch (FileNotFoundException e)      {
    System.out.println (e);
 }
```

```
        finally      {
          if (in != null)
            in.close();
          if (out != null)
            out.close();
        }
    }
}
```

Try-with-resources

- The try-with-resources statement
- It is a try statement that declares one or more resources.
- A *resource* is an object that must be closed after the program is finished with it.
- The try-with-resources statement ensures that each resource is closed at the end of the statement.

Try with resource

```
try ( ClassName theObject = new ClassName() )
{
  // use theObject here
}
catch ( Exception e )
{
  // catch exceptions that occur while using the resource
}
```

```
import java.io.*;
import java.util.Scanner;
public class TestWriteFile {
  public static void main(String[] args) {
        String str = null;
        Scanner in = new Scanner(System.in);
        try (PrintWriter output = new PrintWriter(new FileWriter("out.txt")))       {
                str = in.nextLine();
                while (!str.equals("quit")) {
                        output.println(str);
                str = in.nextLine();
                }
        }
        catch (IOException e)    {
             System.out.println(e);
        }
  }
}
```

exercise อ่านคะแนนสอบจากไฟล์ เพื่อแสดงข้อมูลคะแนน พร้อมทั้งหาค่าคะแนนเฉลี่ย แสดงทางจอภาพ

- Data in file “score.txt”

75.25

80

66.50

50

90

Output

Score

75.25

80.0

66.5

50.0

90.0

Average score is 72.35

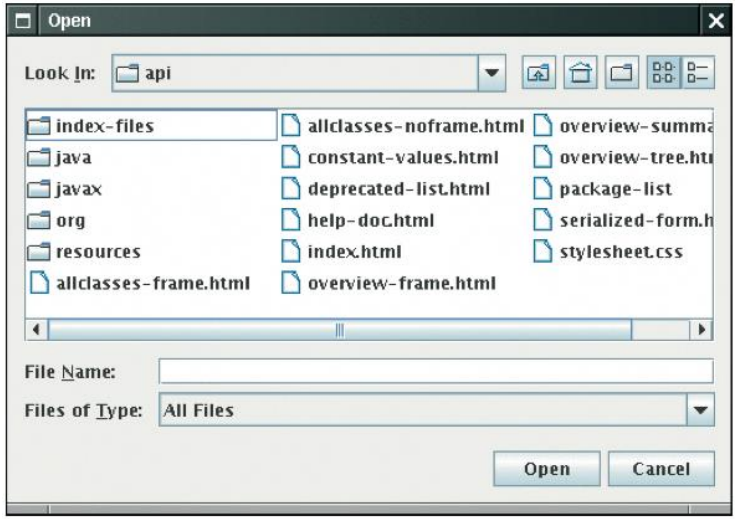
```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class AVGscoreFILE {
    public static void main(String[] args) throws FileNotFoundException {
        double sum = 0, score;
        int i = 0;
        File f = new File("score.txt");
        Scanner in = new Scanner(f);
        System.out.println("Score");
        while (in.hasNextLine()) {
            score = Double.parseDouble(in.nextLine());
            sum += score;
            System.out.println(score);
            i++;
        }
        System.out.println("Average score is " + (sum/i));
    }
}
```

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class AVGscoreFILE {
    public static void main(String[] args) throws FileNotFoundException {
        double sum = 0, score;
        int i = 0;
        File f = new File("score.txt");
        Scanner in = new Scanner(f);
        System.out.println("Score");
        while (in.hasNextDouble()) {
            score = in.nextDouble();
            sum += score;
            System.out.println(score);
            i++;
        }
        System.out.println("Average score is " + (sum/i));
    }
}
```

File Dialog Boxes

```
JFileChooser chooser = new JFileChooser();

if (chooser.showOpenDialog(null) ==
   JFileChooser.APPROVE_OPTION)
{
   File selectedFile = chooser.getSelectedFile();
   Scanner in = new Scanner(selectedFile);
   ...
}
```



A JFileChooser Dialog Box

Reading Text Input: Reading Words

- The `next` method reads a word at a time:

```
while (in.hasNext())
{
   String input = in.next();
   System.out.println(input);
}
```

- With our sample input, the output is:

```
Mary
had
a
little
lamb
…
```

- A *word* is any sequence of characters that is not white space

Reading Text Input: Processing Lines

- The `nextLine` method reads a line of input and consumes the newline character at the end of the line:

```
String line = in.nextLine();
```

- Example: process a file with population data from the CIA Fact Book with lines like this:

```
China  1330044605
India  1147995898
United States 303824646
...
```

- First read each input line into a string

Reading Text Input: Processing Lines

- Then use the `isDigit` and `isWhitespace` methods to find out where the name ends and the number starts. E.g. locate the first digit:

```
int i = 0;
while (!Character.isDigit(line.charAt(i))) { i++; }
```

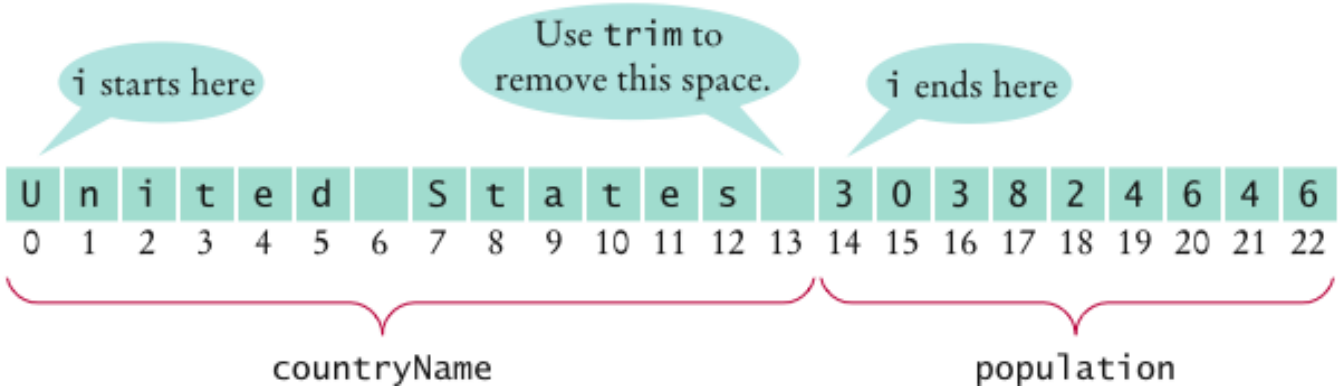
- Then extract the country name and population:

```
String countryName = line.substring(0, i);
String population = line.substring(i);
```

Reading Text Input: Processing Lines

- Use the `trim` method to remove spaces at the end of the country name:

```
countryName = countryName.trim();
```



- To convert the population string to a number, first trim it, then call the `Integer.parseInt` method:

```
int populationValue =
   Integer.parseInt(population.trim());
```

Reading Text Input: Processing Lines

- Occasionally easier to construct a new `Scanner` object to read the characters from a string:

```
Scanner lineScanner = new Scanner(line);
```

- Then you can use `lineScanner` like any other `Scanner` object, reading words and numbers:

```
String countryName = lineScanner.next();
while (!lineScanner.hasNextInt())
{
   countryName = countryName + " " +
  lineScanner.next();
}
int populationValue = lineScanner.nextInt();
```

Reading Text Input: Reading Numbers

- `nextInt` and `nextDouble` methods consume white space and the next number:

```
double value = in.nextDouble();
```

- If there is no number in the input, then a `InputMismatchException` occurs; e.g.

- To avoid exceptions, use the `hasNextDouble` and `hasNextInt` methods to screen the input:

```
if (in.hasNextDouble())
{
   double value = in.nextDouble();
   . . .
}
```

Reading Text Input: Reading Numbers

- `nextInt` and `nextDouble` methods do not consume the white space that follows a number
- Example: file contains student IDs and names in this format:

```
1729
Harry Morgan
1730
Diana Lin
. . .
```

- Read the file with these instructions:

```
while (in.hasNextInt())
{
   int studentID = in.nextInt();
   String name = in.nextLine();
   Process the student ID and name
}
```

Reading Text Input: Reading Numbers

- Initially, the input contains

- After the first call to `nextInt`, the input contains

- The call to `nextLine` reads an empty string! The remedy is to add a call to `nextLine` after reading the ID:

```
int studentID = in.nextInt();
in.nextLine(); // Consume the newline
String name = in.nextLine();
```

Reading Text Input: Reading Characters

- To read one character at a time, set the delimiter pattern to the empty string:

```
Scanner in = new Scanner(. . .);
in.useDelimiter("");
```

- Now each call to next returns a string consisting of a single character
- To process the characters:

```
while (in.hasNext())
{
   char ch = in.next().charAt(0);
   Process ch
}
```

Self Check 11.3

Suppose the input contains the characters `6,995.0`. What is the value of `number` and `input` after these statements?

```
int number = in.nextInt();
String input = in.next();
```

Answer: `number` is `6`, `input` is `",995.0"`.

Self Check 11.4

Suppose the input contains the characters `6,995.00 12`. What is the value of `price` and `quantity` after these statements?

```
double price = in.nextDouble();
int quantity = in.nextInt();
```

Answer: `price` is set to `6` because the comma is not considered a part of a floating-point number in Java. Then the call to `nextInt` causes an exception, and `quantity` is not set.

Self Check 11.5

Your input file contains a sequence of numbers, but sometimes a value is not available and marked as N/A. How can you read the numbers and skip over the markers?

Answer: Read them as strings, and convert those strings to numbers that are not equal to N/A:

```
String input = in.next();
if (!input.equals("N/A"))
{
   double value = Double.parseDouble(input);
   Process value
}
```

การเข้าถึงข้อมูลในไฟล์

1. Sequential access file

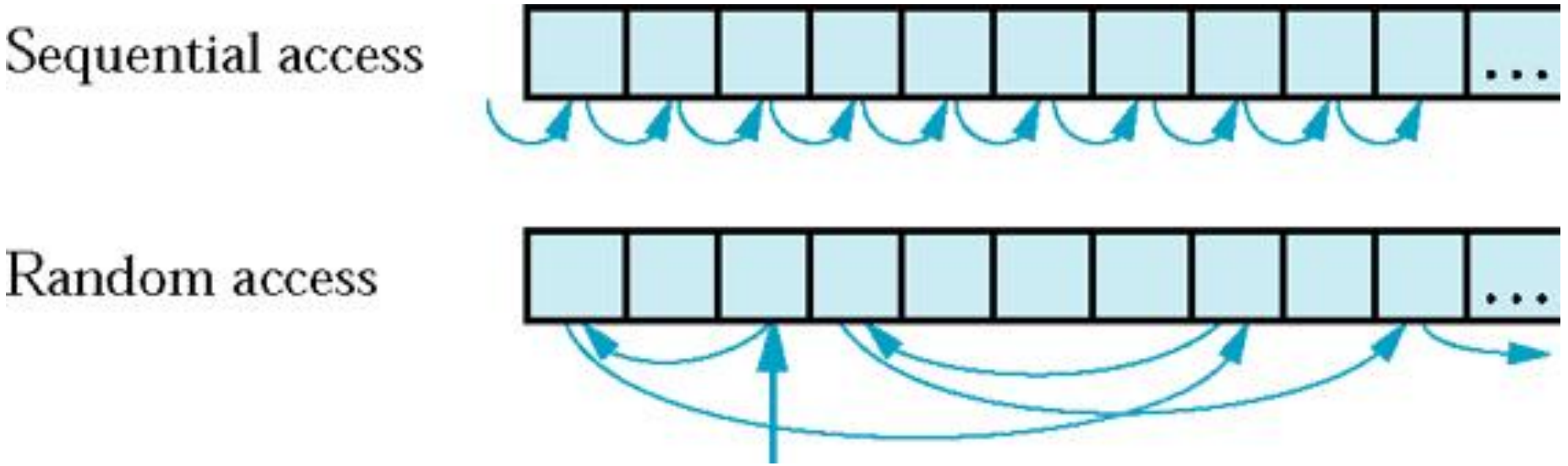
- อ่านไฟล์ เราจะใช้ Scanner
- เขียนไฟล์ให้เก็บเป็น text file ใช้ PrintWriter class

2. Random access file

- จะมี method สำหรับ read และ write ข้อมูลชนิดต่าง ๆ โดยที่เวลามันเก็บลงไฟล์เป็น byte - int กิน 4 ไบต์ double กิน 8 ไบต์ char กิน 2 ไบต์ เพราะเป็น unicode

Random and Sequential Access File

- Sequential access
  - a file is processed a byte at a time
- Random access
  - allows access at arbitrary locations in the file



Random Access File

- Random-access files
  - Access individual records directly and quickly
  - Use fixed length for every record
    - Easy to calculate record locations
  - To access a file randomly, open the file, seek a particular location, and read from or write to that file.

RandomAccessFile class

- Constructor
  - public RandomAcessFile(String name, String mode) throws FileNotFoundException;
  - public RandomAcessFile(File file, String mode) throws FileNotFoundException;
  - Mode “r” “w” “rw”

  RandomAccessFile f = new RandomAcessFile("bank.dat","rw");
- A file pointer is a position in a random-access file. Because files can be very large, the file pointer is of type long.

RandomAccessFile class

- Methods (all methods throw IOException)
  - seek() - To move the file pointer to a specific byte
    f.seek(n);
  - getFilePointer() - To get the current position of the file pointer.
    long n = f.getFilePointer();

    0 – the beginning of the file (the first byte)

    1 – the second byte
  - length() - To find the number of bytes in a file
    long fileLength = f.length();

RandomAccessFile class

- readByte() - To read a byte from the file (return byte)
- readChar() - To read a char from the file (return char)
- readInt() - To read an integer from the file (return int)
- readDouble() - To read a double from the file (return double)
- readLine() – To read a line of text from the file (return String)

RandomAccessFile class

- writeByte(int b) – To write a byte to the file
- writeInt(int b) – To write an integer to the file
- writeDouble(double b) – To write a double to the file
- writeChar(int b) – To write a char to the file
  if put a character as an argument, it will be converted to int
- writeChars(String b) – To write a string to the file
- close() – To close the stream

- All write methods return void

```
import java.io.*;
class RandomTest {
    public static void main(String[] args) {
        RandomAccessFile r = null;
        try    {
                r = new RandomAccessFile ("random.dat", "rw");
                r.writeChars("Good morning all students!\n");
                r.writeChars("7.5 + 8 = \n");
                r.writeDouble(7.5+8);
                r.writeChar('\n');
                r.writeChars("Total students are ");
                r.writeInt(63);
                r.writeChar('\n');
                r.writeChars("Bye\n");
                System.out.println("Total bytes used : " + r.length());
                r.seek(0);
                System.out.println(r.readLine());
                System.out.println("The current pointer is at " + r.getFilePointer());
                System.out.println(r.readLine());
                System.out.println(r.readDouble());
```

```
            System.out.println("The current pointer is at " + r.getFilePointer());
            int i;
            for (i=0; i<20; i++)
                        System.out.print(r.readChar());
             r.seek(r.getFilePointer());
             System.out.println(r.readInt());
             System.out.println(r.getFilePointer());
             r.seek(130);
             System.out.println(r.readLine());
      }
      catch (FileNotFoundException e)          {     System.out.println(e);       }
      catch (IOException e)        {     System.out.println(e);       }
      finally {
                        try  {      if (r != null)       r.close();              }
                        catch (IOException e)   {    System.out.println(e);      }
      }
    }
}
```

Output

```
Total bytes used : 138
 G o o d   m o r n i n g   a l l   s t u d e n t s !
The current pointer is at 54
 7 . 5   +   8   =
15.5
The current pointer is at 84


Total students are 63
128
 B y e
```

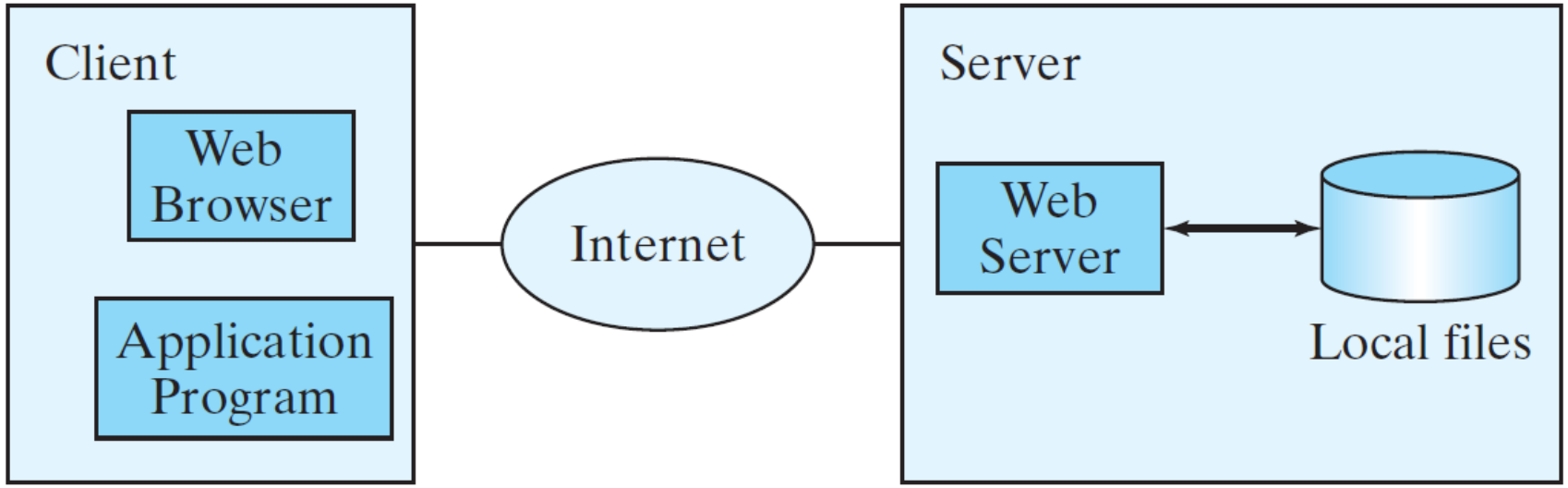
- ถ้าใช้ writeChar() จะเก็บเป็น unicode ซึ่งกิน 2 byte เวลาแสดงผลก็จะมี space ก่อน เพราะไบต์บนเป็น ศูนย์ หมด ถ้าอยากให้กิน byte เดียว ให้ใช้ writeByte(char c) , writeBytes(String s) จะเขียนสตริงได้ โดยแต่ละตัวอักขระจะเก็บด้วยไบต์เดียว

สรุป RandomAccessFile

- ใช้คลาส RandomAccessFile เพื่ออ่านข้อมูลจากไฟล์แบบสุ่ม
- มี file pointer บอกตำแหน่งปัจจุบันที่ pointer ชี้อยู่
- จะเข้าถึงข้อมูลที่ส่วนใด ให้คำนวณตำแหน่งแล้วเรียกใช้ seek() เพื่อย้าย file pointer ไปที่ตำแหน่งที่ต้องการเพื่อทำการอ่านเขียนข้อมูล
- int – 4 bytes
- double – 8 bytes
- char - 2 bytes (unicode)

Reading Data from the Web

Just like you can read data from a file on your computer, you can read data from a file on the Web.



Reading Data from the Web

URL url = **new** URL(**"www.google.com/index.html"**);

After a **URL** object is created, you can use the **openStream()** method defined in the **URL** class to open an input stream and use this stream to create a **Scanner** object as follows:

Scanner input = **new** Scanner(url.openStream());

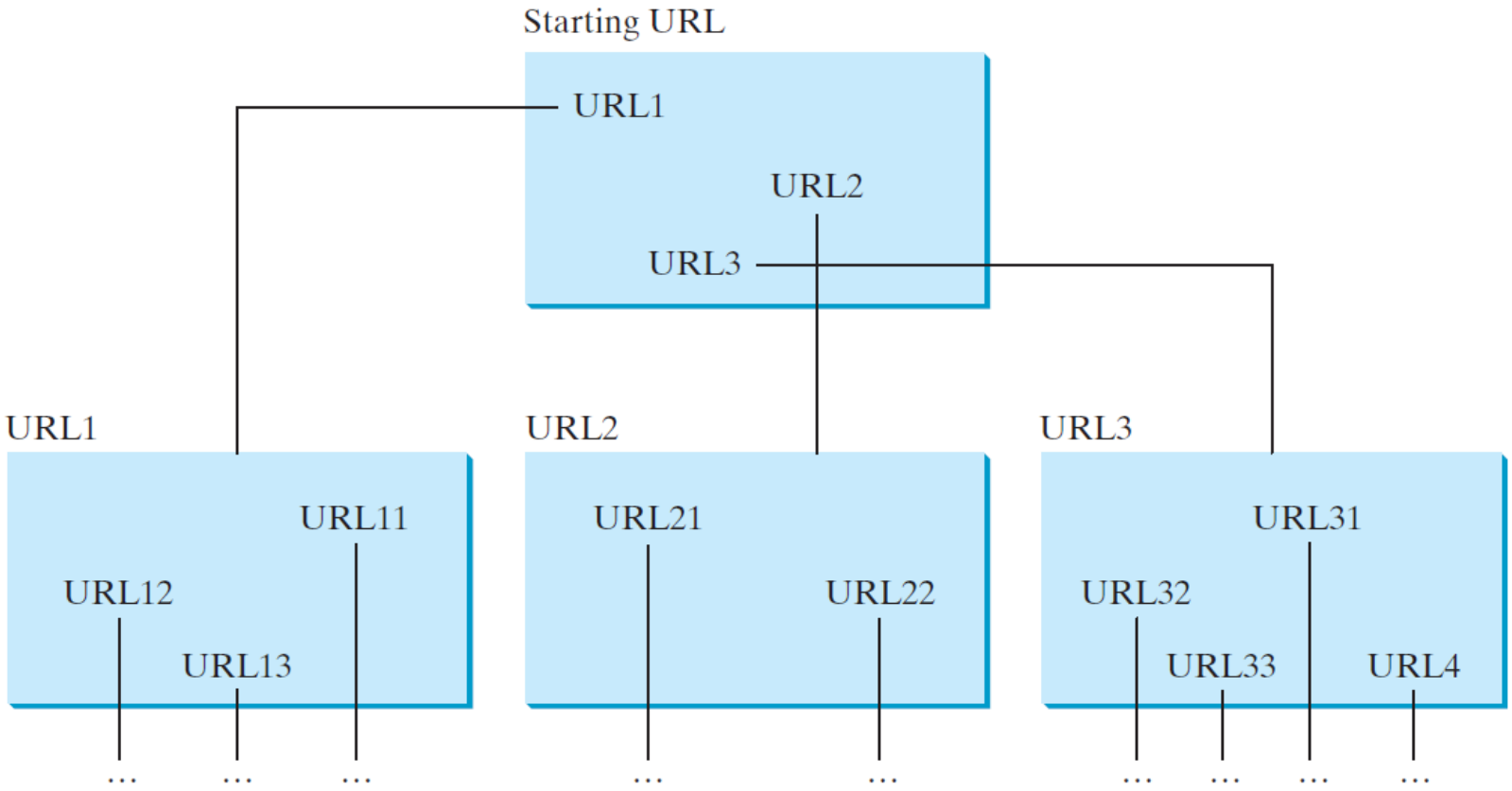
LISTING 12.17 ReadFileFromURL.java

```
import java.util.Scanner;
public class ReadFileFromURL {
        public static void main(String[] args) {
                System.out.print("Enter a URL: ");
                String URLString = new Scanner(System.in).next();
                try {
                        java.net.URL url = new java.net.URL(URLString);
                        int count = 0;
                        Scanner input = new Scanner(url.openStream());
                        while (input.hasNext()) {
                        String line = input.nextLine();
                        count += line.length();
                        }
                        System.out.println("The file size is " + count + " characters");
                }
                catch (java.net.MalformedURLException ex)       {   System.out.println("Invalid URL");   }
                catch (java.io.IOException ex)      {   System.out.println("I/O Errors: no such file");   }
        }
}
```

```
Enter a URL: http://www.yahoo.com
The file size is 190006 characters
```

Case Study: Web Crawler

This case study develops a program that travels the Web by following hyperlinks.



Case Study: Web Crawler

```
Add the starting URL to a list named listOfPendingURLs;
while listOfPendingURLs is not empty and size of listOfTraversedURLs <= 100 {
        Remove a URL from listOfPendingURLs;
        if this URL is not in listOfTraversedURLs {
          Add it to listOfTraversedURLs;
          Display this URL;
          Read the page from this URL and for each URL contained in the page {
                Add it to listOfPendingURLs if it is not is listOfTraversedURLs;
          }
    }
}
```

LISTING 12.18 WebCrawler.java

```
import java.util.Scanner;
import java.util.ArrayList;

public class WebCrawler {
    public static void main(String[] args) {
            java.util.Scanner input = new java.util.Scanner(System.in);
            System.out.print("Enter a URL: ");
            String url = input.nextLine();  //enter a URL
            crawler(url);                   // Traverse the Web from the a starting url (craw from this URL)
    }
```

```
public static void crawler(String startingURL) {
        ArrayList<String> listOfPendingURLs = new ArrayList<>();    //list of pending URLs
        ArrayList<String> listOfTraversedURLs = new ArrayList<>(); //list of traversed URLs

        listOfPendingURLs.add(startingURL);                 //add starting URL
        while (!listOfPendingURLs.isEmpty() &&
                        listOfTraversedURLs.size() <= 100) {
                        String urlString = listOfPendingURLs.remove(0);   //get the first URL
                        if (!listOfTraversedURLs.contains(urlString)) {
                                        listOfTraversedURLs.add(urlString);                 //URL traversed
                                        System.out.println("Craw " + urlString);

                                        for (String s: getSubURLs(urlString)) {
                                                        if (!listOfTraversedURLs.contains(s))
                                                                        listOfPendingURLs.add(s);                 //add a new URL
                                        }
                        }
        }
}
```

```
public static ArrayList<String> getSubURLs(String urlString) {
    ArrayList<String> list = new ArrayList<>();
    try {
        java.net.URL url = new java.net.URL(urlString);
        Scanner input = new Scanner(url.openStream());
        int current = 0;
        while (input.hasNext()) {
            String line = input.nextLine();              //read a line
            current = line.indexOf("http:", current);            //search for a URL
            while (current > 0) {                        //end of a URL
                int endIndex = line.indexOf("\"", current);
                if (endIndex > 0) { // Ensure that a correct URL is found     (URL ends with ")
                    list.add(line.substring(current, endIndex));        //extract a URL
                    current = line.indexOf("http:", endIndex);          //search for next URL
                }
                else
                    current = -1;
            }
        }
    }
    catch (Exception ex) {   System.out.println("Error: " + ex.getMessage());          }
    return list;                //return URLs
  }
}
```

References

- Deitel, H.M., and Deitel, P.J., *Java How to Program*, nineth edition, Prentice Hall, 2012.
- Horstmann, C., *Big Java*, John Wiley & Sons, 2009.
- Liang, Y. D., Introduction to Java Programming, tenth edition, Pearson Education Inc, 2015.