Lecture 10

Files

References:

1. Tony Gaddis, Chapters 4 and 11, Starting out with Java: From Control Structures through Objects, 7 edition

1

Chapter Topics

- Introduction to File Input and Output
- Advanced Topics:
  - Binary Files
  - Random Access Files
  - File Pointer
  - Object Serialization

2

File Input and Output

- The data saved to a file
  - *Input files* or *output files*
- Files
  - Files opened
  - Data written to the file
  - The file closed prior to program termination
- Two types of files
  - binary
  - text

3

Writing Text To a File

- To open a file for text output
  - Need to create an instance of the `PrintWriter` class

```
PrintWriter outputFile = new PrintWriter("StudentData.txt");
```

Pass the name of the file that you wish to open as an argument to the `PrintWriter` constructor.

Warning: if the file already exists, it will be erased and replaced with a new file.

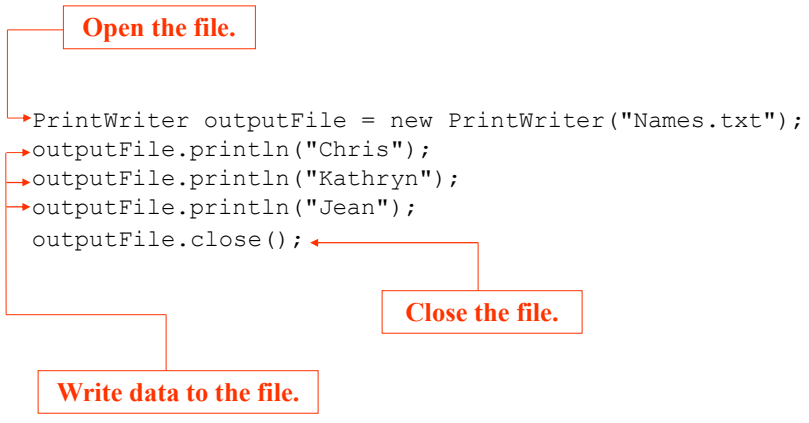
4

The `PrintWriter` Class (1 of 3)

- Write data to a file using the `print` and `println` methods
- the `println` method of the `PrintWriter` class
  - Place a newline character after the written data
- The `print` method
  - Writes data without writing the newline character

5

The `PrintWriter` Class (2 of 3)



6

The `PrintWriter` Class (3 of 3)

- To use the `PrintWriter` class,

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

- E.g.,FileWriteDemo.java

7

Exceptions

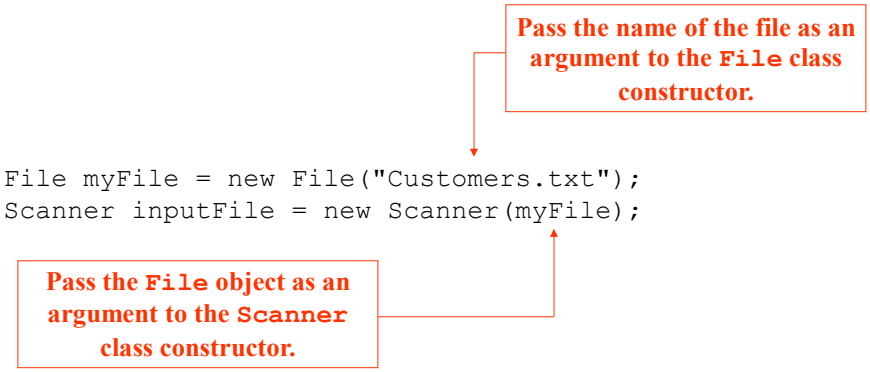
- `PrintWriter` objects
  - Can throw an `IOException`

```
public static void main(String[] args)
  throws IOException
```


8

Reading Data From a File (1 of 2)

- The `File` class and the `Scanner` class to read data from a file:



9

Reading Data From a File (2 of 2)

- Once an instance of `Scanner` created

```
// Open the file.
File file = new File("Names.txt");
Scanner inputFile = new Scanner(file);
// Read a line from the file.
String str = inputFile.nextLine();
// Close the file.
inputFile.close();
```

- Example: FileReadDemo.java

10

Binary Files (1 of 6)

- Data can be stored in a file in its raw binary format.
- A file that contains binary data is often called a *binary file*.
- Storing data in its binary format is more efficient than storing it as text.
  - Low space and speedy processing

11

Binary Files (2 of 6)

- Binary files cannot be opened in a text editor such as Notepad.
- To write data to a binary file you must create objects from the following classes:
  - **`FileOutputStream`** - allows you to open a file for writing binary data. It provides only basic functionality for writing bytes to the file.
  - **`DataOutputStream`** - allows you to write data of any primitive type or String objects to a binary file. Cannot directly access a file. It is used in conjunction with a `FileOutputStream` object that has a connection to a file.

12

Binary Files (3 of 6)

- A `DataOutputStream` and a `FileOutputStream` object to write data to a binary file

```
FileOutputStream fstream = new
    FileOutputStream("MyInfo.dat");
DataOutputStream outputFile = new
    DataOutputStream(fstream);
```

- If the file that you are opening with the `FileOutputStream` object already exists, it will be erased and an empty file by the same name will be created.

13

Binary Files (4 of 6)

- These statements can be combined into one.

```
DataOutputStream outputFile = new
 DataOutputStream(new
    FileOutputStream("MyInfo.dat"));
```

- Once the DataOutputStream object has been created, you can use it to write binary data to the file.
- Example: WriteBinaryFile.java

14

Binary Files (5 of 6)

- To open a binary file for input

```
FileInputStream fstream = new
  FileInputStream("MyInfo.dat");
DataInputStream inputFile = new
  DataInputStream(fstream);
```

- These two statements can be combined into one.

```
DataInputStream inputFile = new
  DataInputStream(new
     FileInputStream("MyInfo.dat"));
```


15

Binary Files (6 of 6)

- Once the `DataInputStream` object has been created, you can use it to read binary data from the file.
- Example:
  - ReadBinaryFile.java

16

Writing and Reading Strings (1 of 2)

- To write a string to a binary file, use the `DataOutputStream` class's `writeUTF` method.
  - UTF stands for Unicode Text Format.
- The `DataInputStream` class's `readUTF` method reads from the file.

17

Writing and Reading Strings (2 of 2)

- To write a string to a file:

```
String name = "Chloe";
outputFile.writeUTF(name);
```

- To read a string from a file:

```
String name = inputFile.readUTF();
```

- Example:
  - WriteUTF.java
  - ReadUTF.java

18

Random Access Files (1 of 4)

- Text files and the binary files previously shown use *sequential file access*.
- With sequential access:
  - The first time data is read from the file
  - As the reading continues, the file's read position advances sequentially through the file's contents.
- Sequential file access is useful in many circumstances.
- If the file is very large, it can take a long time.

19

Random Access Files (2 of 4)

- Java allows a program to perform *random file access*.
  - May immediately jump to any location in the file.
- To create and work with random access files in Java, you use the `RandomAccessFile` class.

```
RandomAccessFile(String filename, String mode)
```

  - *`filename`*: the name of the file.
  - *`mode`*: a string indicating the mode in which you wish to use the file.
    - "r" = reading
    - "rw" = for reading and writing.

20

Random Access Files (3 of 4)

```
// Open a file for random reading.
RandomAccessFile randomFile = new
  RandomAccessFile("MyData.dat", "r");
// Open a file for random reading and writing.
RandomAccessFile randomFile = new
  RandomAccessFile("MyData.dat", "rw");
```

- When opening a file in "r" mode where the file does not exist, a `FileNotFoundException` will be thrown.
- Opening a file in "r" mode and trying to write to it will throw an `IOException`.

21

Random Access Files (4 of 4)

- A file that is opened or created with the **`RandomAccessFile` class is treated as a binary file**.
- The `RandomAccessFile` class has:
  - writeChar(int v)
  - readChar()
- Example: WriteLetters.java

22

The File Pointer (1 of 5)

- The `RandomAccessFile` class treats a file as a stream of bytes.
- The bytes are numbered:
  - the first byte is byte 0.
  - The last byte's number is one less than the number of bytes in the file.
- Internally, the `RandomAccessFile` class keeps a long integer value known as the *file pointer*.

23

The File Pointer (2 of 5)

- The *file pointer* holds the byte number of a location in the file.
- When a file is first opened, the file pointer is set to 0.
- When an item is read from the file, it is read from the byte that the file pointer points to.
- Reading also causes the file pointer to advance to the byte just beyond the item that was read.

24

The File Pointer (3 of 5)

- An `EOFException` is thrown when a read causes the file pointer to go beyond the size of the file.
- Writing also takes place at the location pointed to by the file pointer.
- If the file pointer points to the end of the file, data will be written to the end of the file.
- If the file pointer holds the number of a byte within the file, at a location where data is already stored, a write will **overwrite** the data at that point.

25

The File Pointer (4 of 5)

- The seek method is used to move the file pointer.

```
rndFile.seek(long position);
```

- The argument is the number of the byte to move the file pointer to.

26

The File Pointer (5 of 5)

```
RandomAccessFile file = new
  RandomAccessFile("MyInfo.dat", "r");
file.seek(99);
byte b = file.readChar();
```

- Example: ReadRandomLetters.java

27

Object Serialization (1 of 6)

- When an object containing other types of objects as fields,
  - Saving its contents complicated
  - Java: ***serialize* objects to** save objects to a file (then deserialize it)
  - C++: serialization/unserialization
- Serialization
  - Object's data converted into a series of bytes
    - The other objects contained as fields automatically serialized as well
  - Saved to a file for later retrieval

28

Object Serialization (2 of 6)

- For an object to be serialized
  - The class must implement the `Serializable` interface.
- The `Serializable` interface
  - has no methods or fields
  - let the Java compiler know that objects of the class might be serialized
- If a class contains objects of other classes as fields
  - Those classes must also implement the `Serialzable` interface

29

Object Serialization (3 of 6)

- FileOutputStream and ObjectOutputSteam classes
  - To write the bytes to a file, need two steps

```
FileOutputStream outStream = new
  FileOutputStream("Objects.dat");
ObjectOutputStream objectOutputFile =
  new ObjectOutputStream(outStream);
```


30

Object Serialization (4 of 6)

- To serialize an object and write it to the file, t
  - The `ObjectOutputStream` class's `writeObject` method used

```
BankAccount account = new
        BankAccount(25000.0);
objectOutputFile.writeObject(account);
```

- Deserialization
  - The process of reading a serialized object's bytes and constructing an object from them

31

Object Serialization (5 of 6)

- To deserialize an object an `ObjectInputStream` object is used in conjunction with a `FileInputStream` object.

```
FileInputStream inStream = new
  FileInputStream("Objects.dat");
ObjectInputStream objectInputFile = new
  ObjectInputStream(inStream);
```

- To read a serialized object from the file, the `ObjectInputStream` class's `readObject` method used

```
BankAccount account;
account = (BankAccount)
  objectInputFile.readObject();
```


32

Object Serialization (6 of 6)

- The `readObject` method returns the deserialized object
  - must cast the return value to the desired class type

- Examples:
  - SerializeObjects.java
  - DeserializeObjects.java

33