Unit 08

Files, Streams and Object Serialization

CMPS 251, Fall 2020, Dr. Abdulaziz Al-Ali

Check point (for previous unit)

- What are layout managers? What do they do?
- Describe these:
  - GridPane
  - FlowPane
  - BorderPane

Objectives

- Reading and writing text files
- Reading and writing objects in binary files

Overview

- Data stored in your program is temporary
  - Arrays, variables, etc
- When the program ends, all the data is lost
- To store data between program runs, we use files
- Java has MANY ways to read and write files
  - In these slides we'll only use a few of them

Files and Streams

- Java programs perform file processing by using classes from package **java.io**.
- Includes definitions for stream classes
  - **FileInputStream** (for byte-based input from a file)
  - **FileOutputStream** (for byte-based output to a file)
  - **FileReader** (for character-based input from a file)
  - **FileWriter** (for character-based output to a file)
- You open a file by creating an object of one of these stream classes. The object's constructor opens the file.

Writing to a Text File: Formatter Class

- Formatter outputs formatted Strings to the specified stream
- Constructors:
  - One argument of type **File**
  - …

Writing to a Text File: Example

```
import java.util.Formatter;
import java.io.File;
import java.io.IOException;

public class SimpleFormatter {
          Formatter out;

          public SimpleFormatter() {
                    try {
                              out = new Formatter(new File("MyOutFile.txt"));
                    } catch (IOException ioe) {
                              System.out.println("File is not opened. Exception occurred..\n"
                                                  + ioe);
                              return;
                    }

                    out.format("Hi there, this is being written to a file.\n");

                    for(int i=0; i < 10; i++) {
                              out.format("%d\n", i);
                    }

                    out.close();
          }
}
```

We give it a File object

format() method writes the content into the file.

Writing to a Text File: Exceptions

- A **SecurityException** occurs if the user does not have permission to write data to the file.
- A **FileNotFoundException** occurs if the file does not exist and a new file cannot be created.

Reading From a Text File: Scanner Class

- Class **Scanner** can be used to retrieve data sequentially from a file

- Constructors:
  - One argument **System.in** which we have been using to read from the standard input device of your system, the keyboard by default.
  - One argument a **File** object resulting in the methods of scanner to be invoked on the file specified.
  - …

- Browse the **API doc** files of the **Scanner** class for more explanations.

Reading From a Text File: Example

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class SimpleScanner {
        Scanner in1;
        public SimpleScanner() {
                try {
                        in1 = new Scanner(new File("myData.txt"));
                } catch (FileNotFoundException fnfe) {
                        System.out.println(fnfe);
                        return;
                }
                // Read and print out all lines in the file
                while (in1.hasNextLine()) {
                        System.out.println(in1.nextLine());
                }
                if (in1 != null) in1.close();
        }
}
```

We give it a File object instead of System.in

Checks whether we have more stuff to read from the file

Reading From a Text File: Comments

- You can use any of the scanner methods you'd like
  - `nextInt(), nextFloat()`, etc.

- Frequently you just read in lines from the file and process them as strings

Demo

- Formatter (see JavaFormatter.java)
- Scanner (see JavaScanner.java)

Warm up

- Why do we need to output information to files?
- How many ways can we write to files?
- Which class do we use to write formatted text files?
- When does Java actually try to open a file?

File Class: Constructors

- File has four constructors:

File(File parent, String child) Creates a new **File** instance from a parent abstract pathname and a child pathname string.
File(String pathname) Creates a new **File** instance by converting the given pathname string into an abstract pathname.
File(String parent, String child) Creates a new **File** instance from a parent pathname string and a child pathname string.
File(URI uri) Creates a new **File** instance by converting the given file: URI into an abstract pathname.

- **A Uniform Resource Identifier (URI)** is a more general form of the **Uniform Resource Locators (URLs)** that are used to locate websites.

Note: abstract pathname = File object

File Class: Some Methods

boolean	delete()Deletes the file or directory denoted by this abstract pathname.
boolean	exists()Tests whether the file or directory denoted by this abstract pathname exists.
String	getAbsolutePath()Returns the absolute pathname string of this abstract pathname.
String	getName()Returns the name of the file or directory denoted by this abstract pathname.
String	getParent()Returns the pathname string of this abstract pathname's parent, or null if this pathname does not name a parent directory.
File	getParentFile()Returns the abstract pathname of this abstract pathname's parent, or null if this pathname does not name a parent directory.
String	getPath()Converts this abstract pathname into a pathname string.
boolean	isDirectory()Tests whether the file denoted by this abstract pathname is a directory.
boolean	isFile()Tests whether the file denoted by this abstract pathname is a normal file.
long	lastModified()Returns the time that the file denoted by this abstract pathname was last modified.
long	length()Returns the length (size in Bytes) of the file denoted by this abstract pathname.
String[]	list()Returns an array of strings naming the files and directories in the directory denoted by this abstract pathname.
File[]	listFiles()Returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname.
File[]	listFiles(FilenameFilter filter)Returns an array of abstract pathnames denoting the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.
boolean	mkdir()Creates the directory named by this abstract pathname.

Note: "abstract pathname" = *File* object.

Separator Characters

A separator character is used to separate directories and files in the path.

- On Windows, it is a backslash (\).
  - Example: C:\Users\dana\MyDocuments\...
- On Linux/UNIX, it is a forward slash (/).
  - Example: /home/dana/MyDocuments/…
- Java processes both characters identically
- When building Strings that represent path information, you can use `File.separator` to obtain the local computer's proper separator.
  - This constant returns a String consisting of one character—the proper separator for the system.

New Line Character

Different platforms use different line-separator characters.

- On UNIX/Linux/Mac OS X, it is a newline (`\n`)
- On Windows, it is a combination of a carriage return and a line feed (`\r\n`)
- You can use the %n format specifier in a format control string to output a platform-specific line separator
- Method `System.out.println` outputs a platform-specific line separator after its argument.
- Java can read either one transparently

Check point

- When is a file opened by a Java program?

- Which of the following is used in Windows and which is used in Linux?
  - /
  - \
  - \n
  - \r\n

- Which method inside the File class checks if it is a directory?

Demo

- File class usage (see FileTester.java)
- FileReader/Writer (see FileReaderWriter.java) - optional

Question!

- How would you store information about a Person class?

- Suppose we only have two members:
  - String name
  - int age

- Suppose every Person has a Pet., then what else should we include in our file?

- What if every Pet has a favorite Meal?

Object Serialization

- Java object serialization is the ability to read, or write, an entire object from or to a file.
- A serialized object is represented as a sequence of bytes that includes the object's data and its type information
- After a serialized object has been written into a file, it can be read from the file and deserialized to recreate the object in memory
- Objects of classes that implement interface Serializable can be serialized and deserialized with ObjectOutputStreams and ObjectInputStreams

```
import java.io.Serializable;
class Person implements Serializable{
  //same body of the class Person as you used to code
  it
}
```

Writing Objects to Binary Files

```
public class BinaryObjectWriter {
        public BinaryObjectWriter() {
                ObjectOutputStream out;
                FileOutputStream fos;
                Person p1, p2;

                try {
                        fos = new FileOutputStream("myfile.obj");
                        out = new ObjectOutputStream(fos);

                        p1 = new Person("Ahmed");
                        p2 = new Person("Hind");

                        out.writeObject(p1);
                        out.writeObject(p2);
                        out.writeObject(null);
                        out.close();
                } catch (IOException ioe) {
                        System.out.println(ioe);
                }
        }
}
```

Writing Person *objects* directly!

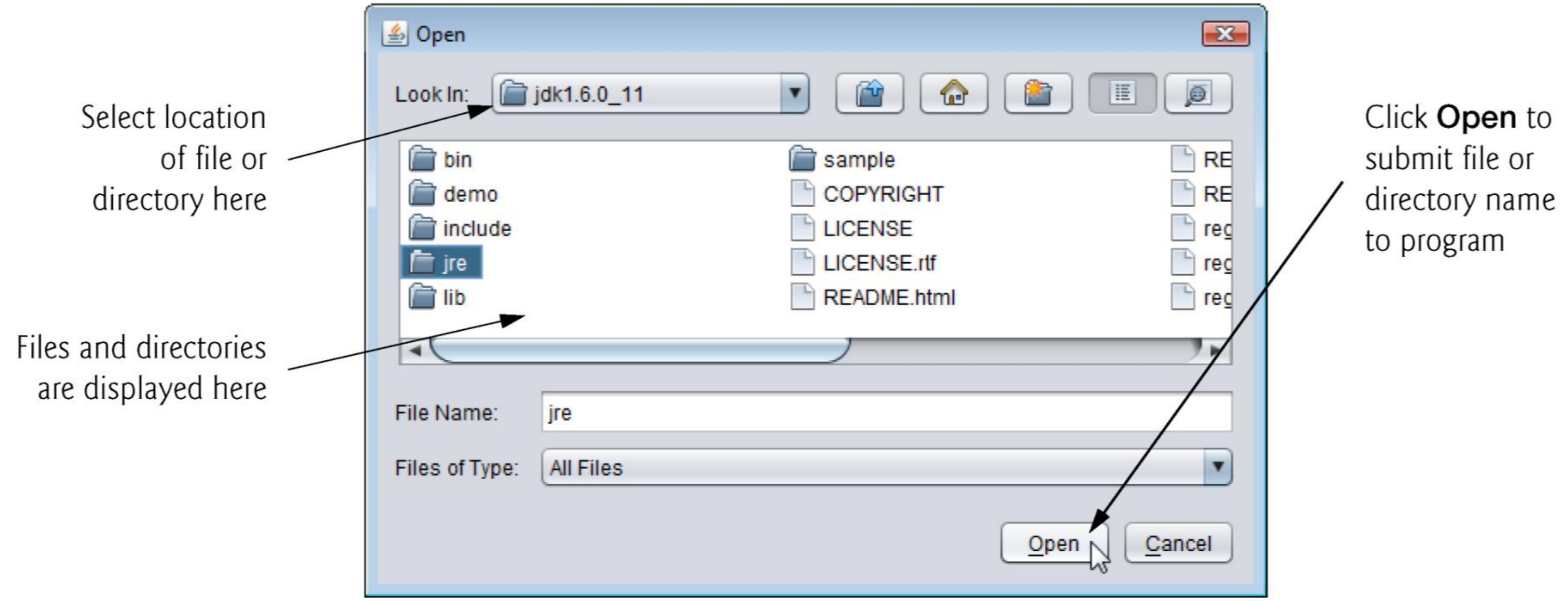
Reading Objects from Binary Files

```
public class BinaryObjectReader {
         public BinaryObjectReader() {
                  ObjectInputStream in;
                  FileInputStream fis;
                  try {
                           Object obj;
                           Person p;

                           fis = new FileInputStream(“myfile.obj");
                           in = new ObjectInputStream(fis);

                           while ((obj = in.readObject()) != null) {
                                    p = (Person) obj;
                                    System.out.println(p.getInfoLong());
                           }

                           in.close();
                  } catch (IOException ioe) {
                           System.out.println(ioe);
                  } catch (ClassNotFoundException e) {
                           System.out.println(e);
                  }
         }
}
```

Reading Person *objects* directly! $\longrightarrow$

For Fun: JFileChooser

- Class JFileChooser displays a dialog that enables the user to easily select files or directories.



- See sample code for an example

For Fun: `JFileChooser`

- Two methods to keep in mind:

- showOpenDialog(..)
  - This method displays the chooser to the user
  - returns JFileChooser.APPROVE_OPTION
  - Or JFileChooser.CANCEL_OPTION

What type are these returned values?!

- getSelectedFile()
  - Returns the selected File abstract pathname.

Demo

- See SerializedObjects.java

Wisdom check

- What are the good and bad things about writing files as text?
- And as binary?

Summary

- Files
- Reading and writing text files
- Reading and writing objects from binary files