File I/O

5 Feb 2010
CMPT166
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

java.io classes

- Object holding pathname information: File
- Formatted text I/O:
  - Scanner, PrintWriter
- Byte-based streams:
  - FileInputStream, FileOutputStream
- Object-based I/O (Serializable):
  - ObjectInputStream, ObjectOutputStream
- Standard streams:
  - System.in (an InputStream), System.out, System.err (both PrintStreams)

File methods

- File is essentially a wrapper around a filename string. Constructor:
  - **File oFile = new File(“output.txt”);**
- Check if exists, can read/write:
  - **if (oFile.exists() && oFile.canRead())**
- Check file type:
  - **If (oFile.isFile() || oFile.isDirectory())**
- Get parent directory:
  - **oFile.getParent()**
- Get just the filename: oFile.getName()

Formatted text stream I/O

- java.io.PrintWriter: output formatted text
  - **PrintWriter output =**
    new PrintWriter(oFile);
  - **output.println(“Hello, World!”);**
    - Methods as with System.out
- java.util.Scanner: read text from stream
  - **Scanner input =**
    new Scanner(new File(“in.txt”));
    // or: new Scanner(System.in);
  - **id = input.nextInt();**
- Remember to close() when you're done

File I/O exceptions

- An instance of the class FileNotFoundException is raised if the file cannot be opened:

```
try {
    out = new PrintStream( “out.txt” );
} catch ( FileNotFoundException e ) {
    System.err.println( “No write permissions!” ); }
```

- Scanner raises InputMismatchException if wrong type, or NoSuchElementException if input is exhausted.
- EOFException when the end of file is reached
- These are subclasses of IOException

Object-based I/O

- Use FileInputStream / FileOutputStream to open a file for binary I/O
  - **fos = new FileOutputStream(“output.db”)**
- Wrap the stream in an ObjectInputStream / ObjectOutputStream to use object serialization
  - **oos = new ObjectOutputStream(fos);**
- Use readObject/writeObject to do the I/O:
  - **oos.writeObject(myobj);**
  - readObject() returns a generic Object:
    - Cast it back to the original type
    - **myobj = (MyObj) ios.readObject();**

Serializable objects

- Serialization is converting an object to a representation that can be written to a stream
- The Serializable interface is a tag:
  - Interface with no methods
  - Used to identify what objects are serializable
- Primitive types are serializable
- Arrays of serializable objects are serializable
- A class can be tagged as serializable if all its non-transient instance variables are serializable
    - Vars declared transient are skipped in serialization

Customizing serialization

- Serializable objects: just tag as Serializable
  - all the work of reading/writing is done for you
- Methods writeObject() / readObject()
  - Specify format to use in writing out
  - Can call defaultWriteObject() to use default functionality
  - Or use your own writeInt(), etc. to write out non-serializable fields
- See CustomDataExample.java

Summary of I/O classes

- Formatted text I/O:
  - Create a File object (pathname)
  - Write: create a PrintWriter, call .print()
  - Read: create a Scanner, call .next*()
- Object-based I/O:
  - Create a File object (pathname)
  - Write: create a FileOutputStream
    - Create ObjectOutputStream: .writeObject()
  - Read: create a FileInputStream
    - Create ObjectInputStream: .readObject()

Random-access files

- Sequential files are hard to modify in-place
  - Must erase and rewrite entire file
- Random-access files:
    - file = new RandomAccessFile(“user.db”, “rw”);
- Can be used in place of FileInputStream / FileOutputStream, e.g., to do object-based I/O
- File position pointer:
    - file.seek(num_bytes);
  - Seek to position relative to start