I/O in Java

Binary files, utilities

Review

What is the difference between byte streams and character streams?

a) They're the same, just different names

b) The first reads single bytes, the second reads two bytes at a time

c) The first reads UTF-16 and the second reads UTF-32

d) The first is buffered and the second is unbuffered

Review

What is the difference between buffered streams and unbuffered streams?

a) They're the same, just different names

b) The first reads single bytes, the second reads two bytes at a time

c) The first reads data in larger chunks and does subsequent reads from there; the second always reads directly from the file

d) The first uses text files and the second uses binary files

Objectives

- Topics
  - Binary Java I/O classes
  - Reading/writing classes
  - Useful file utilities
- Goals: after this lecture, you will be able to
  - describe why binary files are used
  - read and write binary files
  - use the File and Path classes

Binary Data

- There are two ways to store data in binary form: as objects or using random access files
  - Technically, a text file can be random access, but it is a really bad idea to treat it that way: the text fields are almost always different sizes, so moving around in the file is error-prone
- ***Object streams*** store entire objects to a file
- ***Random Access*** files store records in binary format to a file

Object Streams

- The ***ObjectInputStream*** and ***ObjectOutputStream*** allow entire objects to be read/written, instead of reading/writing their fields and using setters/getters
- To do this, a class must implement the **`Serializable`** interface
  - This interface has no methods, it just identifies the class as being serializable
- Then use `writeObject(yourObject)` and `(YourClass)readObject(yourObject)` to write/read the *binary* (not text) file

```
public class Student implements Serializable {
   private String name;
   private int id;
   private double gpa;
// etc.
}


Student student = new Student("Jones", 123, 3.75);

try (ObjectOutputStream oos = new ObjectOutputStream(
      new FileOutputStream("student.bin"))) {

    oos.writeObject(student);

} catch (IOException e) {
    System.out.println(e.getMessage());
}
```

wrap a FileInputStream

```
try ( ObjectInputStream ois = new ObjectInputStream( new
      FileInputStream("student.bin"))) {

    student = (Student)ois.readObject();

} catch (IOException e) {
    System.out.println(e.getMessage());
} catch (ClassNotFoundException e) {
    System.out.println(e.getMessage());
}
```

cast to your Class type

Object Streams, cont.

- ObjectInputStream doesn't have a way to peek ahead or say "end of file", so using a loop to read a file can stop when ...
  - `readObject( )` throws an exception – this isn't the right way to handle normal processing
  - the `available( )` method in FileInputStream returns a value less than zero –you have to declare a variable of that type instead of just wrapping it. This is the better solution.

```
ObjectInputStream ois = null;
FileInputStream fis = null;
try {
      fis = new FileInputStream("student.bin");
      ois = new ObjectInputStream(fis);

      while (fis.available() > 0) {
            student = (Student)ois.readObject();
            list.append(student);
      }

} catch (IOException e) {
    System.out.println(e.getMessage());
} catch (ClassNotFoundException e) {
    System.out.println(e.getMessage());
}
```

Use FileInputStream.available(); it returns a positive integer if there's more to read

Random Access Files

- ***RandomAccessFile*** allows you to move the file pointer around at will with the seek(byteNumber) method – but you need to know what argument to use as the byte number
  - seek(0) takes you back to the beginning of the file
  - That's why you don't do this with text files
- And you usually open this type of file for "rw" access

Random Access Files, cont.

- Usually, these files are homogeneous: every entry is one type of data
  - Or a series of several data items (a record), repeated
  - Java lacks an easy way to group data into records, so this method isn't as common as in C++

Random Access Files, cont.

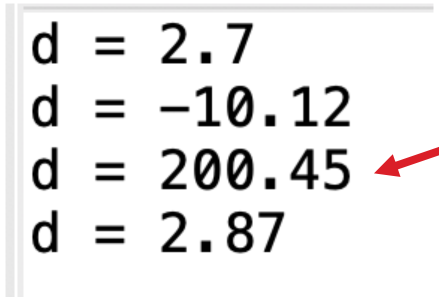
- To use seek( ), you need to compute the record number and the size of the record
- seek( ) takes a `long` parameter
  - For primitive data:
    - char is 2 bytes
    - int is 4 bytes
    - double is 8 bytes

```
public class RandomAccessStuff {
    public static void main(String[] args) {
      double[] data = {2.7, -10.12, 9.5, 2.87};
      try ( RandomAccessFile raf = new
RandomAccessFile("binary.bin", "rw") )   {

          for (double d: data) {
            raf.writeDouble(d);      // Write out the data
          }

          long b = raf.length()/4;   // Or just set it to 8

          raf.seek(2*b);             // Seek to record #2
          raf.writeDouble(200.45);

// continued ->
```

or just "r" or "w"

8 bytes for a float

```
            raf.seek(0);           // Seek to beginning of file
            for (int i = 0; i < 4; i++ ) {
              data[i]=raf.readDouble();
            }
            raf.close();

      } catch (IOException e) { System.out.println(e.getMessage()); }
      for (double d: data) { System.out.println("d = " + d);}
  }
```



```
d = 2.7
d = -10.12
d = 200.45
d = 2.87
```

changed from 9.5

nio: Path

- The "n" is for new, but it's no longer so new
- ***Path*** is an interface that models the underlying OS idea of directory path in the file system
- ***Paths*** is a helper class for doing Path stuff, with utility method get()

```
Path path = Paths.get("myfile.txt");
System.out.println(path.toAbsolutePath()); // Absolute path to myfile.txt
```

File

- ***File*** is an older, concrete class
- We've used it before: wrapped inside a Scanner
- methods include: getAbsolutePath(), delete(), mkdir(), length(), exists(), renameTo(File anotherFile)

```
File path = File("myfile.txt");
if (file.exists()) {
    System.out.println( file.getAbsolutePath() );
}
```

nio Files

- ***Files*** is a helper class for doing File stuff
- Most take a Path parameter

```
// Using path from earlier slide
List<String> lines = Files.readAllLines(path);
long size = Files.size(path);
Files.delete(path);
```

Files, cont.

- createFile(Path) – optional attributes parameter
- createDirectory(Path) – optional attributes parameter
- copy(Path source, Path target) – optional CopyOption
- move(Path source, Path target) – renames a file
- delete(Path)
- exists(Path)
- isDirectory(Path), isExecutable(Path), isReadable(Path), isWriteable(Path)
- readAllBytes(Path), readAllLines(Path)

Files, cont.

```
Path path = Paths.get("/Users/marty/java/myfile.txt");
try {
      Path file = Files.createFile(path);
      System.out.println("File created: " + file)
} catch(IOException e) {
      System.out.println(e.getMessage);
}
```