CMSC 132: OBJECT-ORIENTED PROGRAMMING II

Java I/O – Part 2: Binary Files

Department of Computer Science
University of Maryland, College Park

Introduction to File I/O

- **What is a file?** A collection of data stored on disk.
- **Text vs. Binary files:**
  - **Text files**: Store data as human-readable characters (e.g., .txt).
  - **Binary files**: Store data as raw bytes, more compact and efficient.
- We learned about Text files in a previous lab session
- **Why use binary files?** Faster, more precise storage of structured data (e.g., images, serialized objects).
- **Streams in Java:** Sequential flow of data, similar to a pipeline.

TODAY WE WILL TALK ABOUT BINARY FILES

Understanding Streams in Java

- **Definition:** A sequence of data elements made available over time.
- **Two types of streams:**
  - **Byte Streams**: Process data in raw bytes (InputStream, OutputStream).
  - **Character Streams**: Process text data (Reader, Writer).
- **Why use byte streams?** Needed for handling binary files, images, and non-text data.
- **Direction of Streams:**
  - **Input Stream:** Reads data into a program.
  - **Output Stream:** Writes data from a program.

Writing to Binary Files with FileOutputStream

- **FileOutputStream** writes raw bytes to a file.
- **Features:**
  - Used for saving non-text data like images and audio files.
  - Can write individual bytes or byte arrays.
  - May require explicit flushing for efficient storage.
- **Potential Issues:**
  - Cannot write primitive types directly (use DataOutputStream).
    - It works at the **byte level** only.
    - It **does not understand Java primitive types** like int, double, boolean, char, etc.
  - Not buffered, which may cause performance overhead.
  - Think of FileOutputStream like a **raw pipe**. You can push raw bytes through it, but it doesn't know the format of your data.
    - See: `BinaryFileWriter and BinaryFileWriter1`

Enhancing Performance with BufferedOutputStream

- **Why use buffering?** Reduces I/O operations by grouping multiple bytes before writing.
- **How BufferedOutputStream helps:**
  - Speeds up writing operations by minimizing disk access.
  - Stores data in a temporary buffer before flushing to disk.
  - Works in combination with FileOutputStream.
- **Use Case:** Writing large binary files efficiently.

- See: `BufferedOutputDemo`

Note: When you run `BufferedOutputDemo` it will create a .bin file with size of 10, but when you open in the text editor you will see nothing because the bytes it writes correspond to non-printable characters !

Writing Primitive Data with DataOutputStream

- **Problem:** FileOutputStream writes only raw bytes, so primitive types need conversion.
- **Solution:** DataOutputStream converts primitive types (int, double, boolean, etc.) into byte format.
- **Key Features:**
  - Works on top of FileOutputStream.
  - Writes data in a structured binary format.
  - Can be read later using DataInputStream.
- **Common Uses:** Writing numerical data, structured records, game save files, etc.
    - See: `BinaryDataWriterExample`

Reading Binary Files with FileInputStream

- **FileInputStream** reads raw bytes from a file.
- **Features:**
  - Reads data as an array of bytes or one byte at a time.
  - Works for any binary file type (e.g., images, serialized data).
- **Potential Issues:**
  - Does not interpret bytes into meaningful types.
  - May be inefficient for large files (use buffering).

See: `FileInputStreamExample`

Improving Read Performance with BufferedInputStream

- **Why buffer input?** Reading a file one byte at a time is slow.
- **How BufferedInputStream helps:**
  - Reduces the number of disk accesses.
  - Speeds up reading by using an internal byte buffer.
- **Typical Usage:** Reading large binary files efficiently.

See: `BufferedInputStreamExample`

Reading Structured Data with DataInputStream

- **Problem:** FileInputStream reads only raw bytes.
- **Solution:** DataInputStream reads bytes and converts them into Java primitive types.
- **Key Features:**
  - Works with FileInputStream.
  - Reads int, double, boolean, String, etc., in binary format.
- **Common Uses:** Reading structured data, game saves, database files.

See: `DataInputStreamExample`

//run `BinaryDataWriterExample first to have .bin file`

Standard I/O in Java (System Class)

- **Standard Streams in Java (System class in java.lang)**
  - System.in → Standard input (keyboard, an InputStream).
  - System.out → Standard output (console, a PrintStream).
  - System.err → Standard error output (error messages, a PrintStream).
- **Why use standard streams?**
  - Redirect input/output (e.g., reading from a file instead of the keyboard).
  - Print formatted messages (System.out.println).
  - Debugging (System.err to print errors separately).

See: 2 examples in standard package