CISC 3120

C17: I/O Streams and File I/O

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Outline

- Recap and issues
    - Review your progress
    - Assignments: Practice, CodeLab, and Project
- Exception Handling
- Introduction to Paths and Files
- File Input/Output and Input/Output Streams
- A few related concepts
    - Character and character encoding
    - Formatted I/O and unformatted I/O
    - Buffered I/O and unbuffered I/O
    - Sequential and random access
- Assignment

Path and File

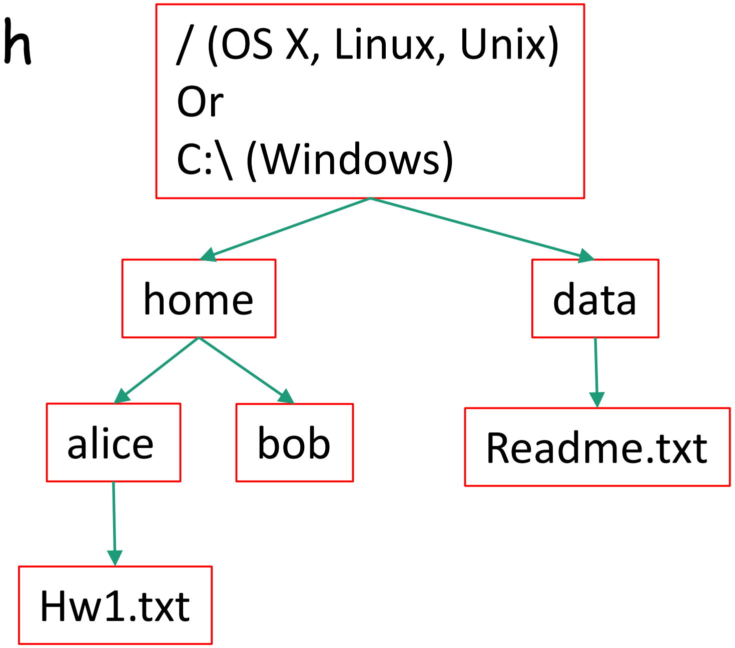
- Concept of path in OS
- The Path interface and Paths helper class
- The File and Files classes

File System Trees

- A file system stores and organizes files on some form of media allowing easy retrieval
- Most file systems in use store the files in a tree (or hierarchical) structure.
  - Root node at the top
  - Children are files or directories (or folders in Microsoft Windows)
  - Each directory/folder can contain files and subdirectories

Path

- Identify a file by its *path* through the file system tree, beginning from the root node
  - Example: identify Hw1.txt
  - OS X
    - /home/alice/Hw1.txt
  - Windows
    - C:\home\alice\Hw1.txt
  - Delimiter
    - Windows: "\"
    - Unix-like: "/"

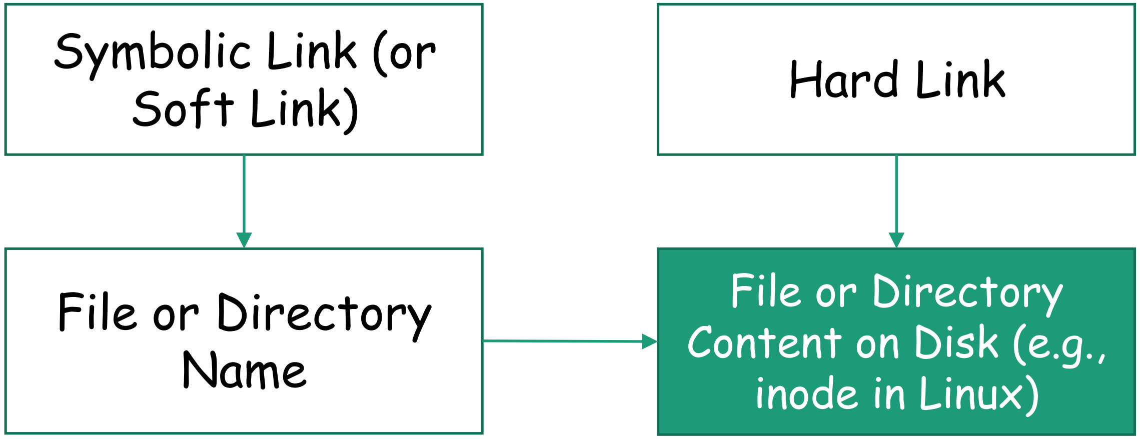


Relative and Absolute Path

- Absolute path
  - Contains the root element and the complete directory list required to locate the file
    - Example: /home/alice/Hw1.txt or C:\home\alice\Hw1.txt
- Relative path
  - Needs to be combined with another path in order to access a file.
  - Example
    - alice/Hw1.txt or alice\Hw1.txt, without knowing where alice is, a program cannot locate the file
  - "." is the path representing the current working directory
  - ".." is the path representing the parent of the current working directory

Symbolic Link and Hard Link

- A file-system object (source) that points to another file system object (target).
    - Symbolic link (soft link): an "alias" to a file or directory name
    - Hard link: another name of a file or directory


Symbolic Link (or Soft Link)
Hard Link
File or Directory Name
File or Directory Content on Disk (e.g., inode in Linux)

Transparency to Users

- Links are transparent to users
  - The links appear as normal files or directories, and can be acted upon by the user or application in exactly the same manner.
- Create symbolic links from the Command Line
  - Unix-like: ln
  - Windows: mklink

Unix-like OS: Example

- Unix-like (e.g., Linux, OS X): “#” leads a comment. do the following on the terminal,
  - echo “hello, world!” > hello.txt   # create a file, the content is “hello, world!”
  - ln -s hello.txt hello_symlink.txt   # create a soft link to hello.txt
  - ls -l hello_symlink.txt   # list the file, what do we observe?
  - cat hello_symlink.txt   # show the content using the symbolic link, what do we observe?
  - ln hello.txt hello_hardlink.txt   # create a hard link
  - ln -l hello_hardlink.txt   # observation?
  - cat hello_hardlink.txt   # observation?
  - mv hello.txt hello2.txt   # rename hello.txt
  - ls -l hello_symlink.txt   # observation?
  - ln -l hello_hardlink.txt   # observation?
  - cat hello_symlink.txt   # observation?
  - cat hello_hardlink.txt   # observation

Window: Example

- On Windows, it requires elevated privilege to create file symbolic link. Do not type the explanation in “()”.
  - echo “hello, world!” > hello.txt (create a file, the content is “hello, world!”)
  - mklink hello_symlink.txt hello.txt (create a soft link to hello.txt)
  - dir hello_symlink.txt (list the file, what do we observe?)
  - more hello_symlink.txt (show the content using the symbolic link, what do we observe?)
  - mklink /h hello_hardlink.txt hello.txt (create a hard link to hello.txt)
  - dir hello_hardlink.txt (observation?)
  - more hello_hardlink.txt (observation?)
  - move hello.txt hello2.txt (rename hello.txt)
  - dir hello_symlink.txt (observation?)
  - dir hello_hardlink.txt (observation?)
  - more hello_symlink.txt (observation?)
  - more hello_hardlink.txt (observation?)

Questions?

- Concept of file system trees
- Concept of paths
  - Traversal of file system trees
  - Absolute path
  - Relative path
- Symbolic link and hard link

The Path Interface

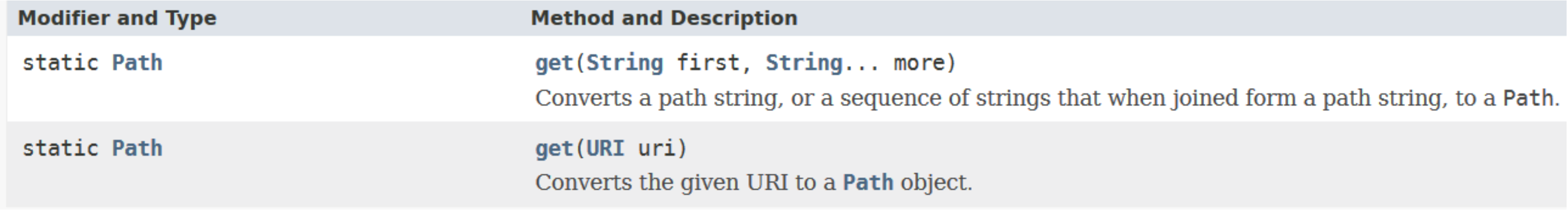
- A programmatic representation of a path in the file system.
- Use a Path object to examine, locate, manipulate files
    - It contains the file name and directory list used to construct the path.
- Reflect underlying file systems, is system-dependent.
- The file or directory corresponding to the Path might not exist.

Path Operations: Example

- Creating a Path
- Retrieving information about a Path
- Removing redundancies from a Path
- Converting a Path
- Joining two Paths
- Creating a relative Path of two Paths
- Comparing two Paths
- PathDemoCLI in the Sample Programs repository

Obtain an Instance of Path

- The Paths helper class



Modifier and Type	Method and Description
static Path	get(String first, String... more) Converts a path string, or a sequence of strings that when joined form a path string, to a Path.
static Path	get(URI uri) Converts the given URI to a Path object.

- Examples
  - Path p1 = Paths.get(‘alice/hw1.txt’);
  - Path p1 = Paths.get(‘alice’, ‘hw1.txt’);
  - Path p3 = Paths.get(URI.create("file://C:\\home\\alice\\Hw1.txt"));
- Paths.get methods is equivalent to FileSystems.getDefault().getPath methods

Retrieve Information about a Path

- Use various methods of the Path interface

```
// On Microsoft Windows use:
Path path = Paths.get("C:\\home\\alice\\hw1.txt");
// On Unix-like OS (Mac OS X) use:
// Path path = Paths.get("/home/alice/hw1.txt");
System.out.format("toString: %s%n", path.toString());
System.out.format("getFileName: %s%n", path.getFileName());
System.out.format("getName(0): %s%n", path.getName(0));
System.out.format("getNameCount: %d%n", path.getNameCount());
System.out.format("subpath(0,2): %s%n", path.subpath(0,2));
System.out.format("getParent: %s%n", path.getParent());
System.out.format("getRoot: %s%n", path.getRoot());
```

More about Path

- Normalize a Path and remove redundancy
- Convert a Path
  - To a URI
  - To absolute Path
  - To real Path
- Join two Paths
- Creating a relative Path of two Paths
- Compare two Paths, iterate Path

Convert to Real Path

- The Path.toRealPath method returns the real path of an existing file.
  - If true is passed to this method and the file system supports symbolic links, this method resolves any symbolic links in the path (thus, the real path)
  - If the Path is relative, it returns an absolute path.
  - If the Path contains any redundant elements, it returns a path with those elements removed.
- The method checks the existence of the Path
  - It throws an exception if it does not exist or cannot be accessed.

Compare Two Paths, Iterate Path

- Equals: test two paths for equality
- startsWith and endsWith: test whether a path begins or ends with a particular string
- Iterator: iterate over names of a Path
- Comparable: compare Path, e.g., for sorting

File and Legacy File I/O

- Path & File in Java: evolving in Java
  - java.nio.file.Path since version 1.7
  - java.io.file since version 1.0
- Generally, the Path interface can do everything the File class (legacy) can do
  - Implication
    - Use Path for new applications

Limitation of Legacy File I/O

- Many methods don't throw exceptions when they fail
- The rename method does not work consistently across platforms
- Support for symbolic links is limited
- Support for file system meta data is limited (file permissions, ownership, and other access control attributes)
- Access to file meta data is inefficient
- Many File's methods do not scale to large file systems
- Dealing with file system tree that has circular symbolic links is difficult and unreliable

From File to Path

- The File class has a toPath method
  - Example
    - File file = …
    - Path fp = file.toPath();
  - We can now take advantage of what Path is to offer
    - Example: delete a file
      - Path fp = file.toPath();
      - Files.delete(fp);
    - Instead of
      - file.delete();

Mapping Legacy I/O to New I/O Functionality

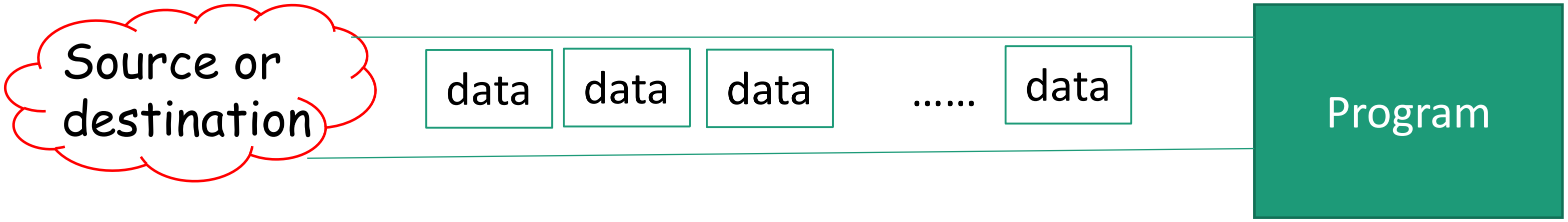
- See Oracle's Java tutorial at,
  - https://docs.oracle.com/javase/tutorial/essential/io/legacy.html
- where you should examine the mapping table closely

Questions?

- Recommendation: use java.nio instead of java.io whenever possible
- With Java Path interface and Paths utility class
  - Concept of path
  - Creating a Path
  - Retrieving information about a Path
  - Removing redundancies from a Path
  - Converting a Path
  - Joining two Paths
  - Creating a relative Path of two Paths
  - Comparing two Paths
- A file (both file and directory) corresponding to a Path object may not exist, how do we know if it exists, and its state?

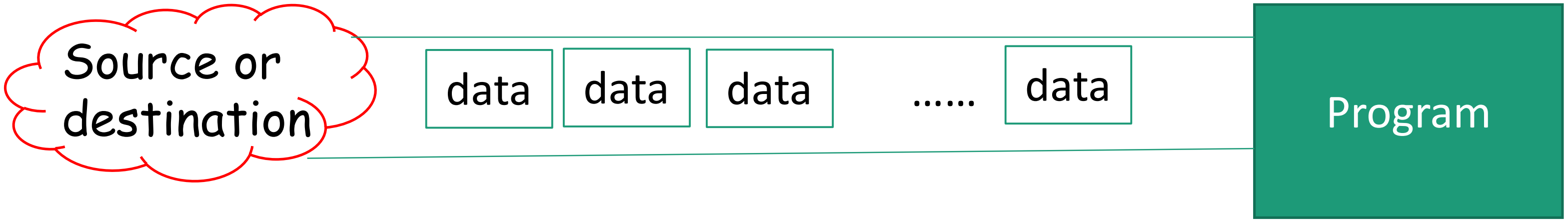
Input and Output Streams

- A stream is a sequence of data associated with an input source or an output destination.
  - Input source or output destination
    - Files, network end point, standard I/O, memory array, programs
  - A program uses an *input stream* to read data from a source, one item at a time
  - A program uses an *output stream* to write data to a destination, one item at time



Sequence of Data

- What are the data? What kind of data?
  - Sequence of bytes: byte streams
  - Sequence of characters: character streams
  - Sequence of values of any primitive data type: data streams
  - Sequence of Objects: object streams

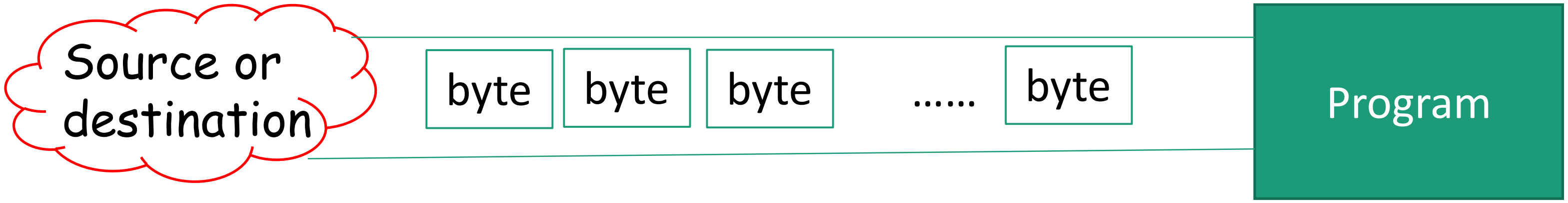


Questions?

- Concept of I/O streams
- Different types of I/O streams

Byte Stream

- Programs use *byte streams* to perform input and output of 8-bit bytes.
    - Read or write one or more bytes at a time
- Most basic streams
    - A value of any other type of data can be considered as a sequence of one or more types
- Two abstract classes: InputStream, OutputStream
    - Use concrete subclasses



Use Byte Streams

- Low-level, you may have better options
- Instantiate concrete subclass of the InputStream or OutputStream class
  - Common types of sources or destinations: files, byte arrays, audio input, and others
  - Example: using files as sources or destinations of streams
    - FileInputStream, FileOutputStream
- Must close streams to release resources
- I/O may cause errors, deal with exceptions
  - Example: cannot create streams, cannot read or write to streams

Byte Stream: Examples

- Use try-catch-finally
  - The finally block will be executed regardless

```
try { // initialize resources }

catch (...) { …}

finally { // close stream }
```

- Use try-with-resources
  - If the resource is autoclosable (e.g., a stream), we may be better off using the try-with-resources

```
try (// initialize resources)

catch(…) {}
```

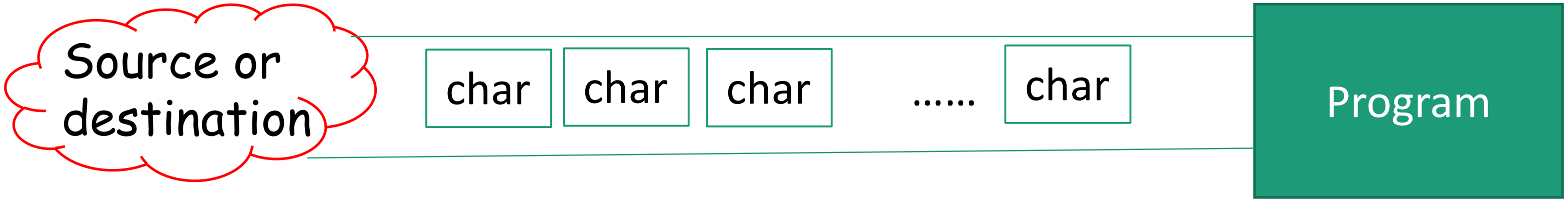
  - Resources closed automatically in the reverse order they are initialized.
- See the examples in the Sample Programs repository

Questions?

- Concept of byte streams
- InputStream and OutputStream

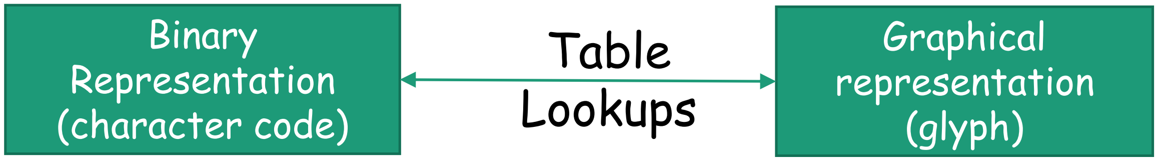
Character Stream

- Programs use *character streams* to perform input and output of Unicode characters
  - Read or write one or more characters at a time
  - A character is a 16-bit Unicode
- Two abstract classes: Reader, Writer
  - Use concrete subclasses



Characters

- Basic units to form written text
  - Each language has a set of characters
  - Generally, a character is a code (a binary number)
  - A character can have many different glyphs (graphical representation)
    - The 1st letter in the English Alphabet
      - Character "a": a, **a**, a, *a*, …



Binary Representation (character code)

Table Lookups

Graphical representation (glyph)

Unicode

- A single coding scheme for written texts of the world's languages and symbols
- Each character has a code point
    - Originally 16-bit integer (0x0000 – 0xffff), extended to the range of (0x0 – 0x10ffff), *e.g.*, U+0000, U+0001, …, U+2F003, …, U+FF003, …, U+10FFFF
- All the codes form the Unicode code space
    - Divided into planes, each plane is divided into blocks
        - Basic Multilingual Plane (BMP), the 1st plane, where a language occupies one or mote blocks
- Encoding schemes
    - Express a code point in bytes: in UTF-8, use 1 to 4 bytes (grouped into code units) to represent a code point (space saving, backward comparability with ASCII)
    - Code units

Encoding Scheme: Code Point and Code Units: Examples

- All code units are in hexadecimal.

Unicode code point	U+0041	U+00DF	U+6771	U+10400
Representative glyph	A	β	東	∂
UTF-32 code units	00000041	000000DF	00006771	00010400
UTF-16 code units	0041	00DF	6771	D801 DC00
UTF-8 code units	41	C3 9F	E6 9D B1	F0 90 90 80

Characters in the Java Platform

- Original design in Java
  - A character is a 16-bit Unicode
    - A Unicode 1.0 code point is a 16-bit integer
    - Java predates Unicode 2.0 where a code point was extended to the range (0x0 – 0x10ffff).
    - Example: U+0012: '\u0012'
- Evolved design: a character in Java represents a UTF-16 code unit
  - The value of a character whose code point is no above U+FFFF is its code point, a 2-byte integer
  - The value of a character whose code point is above U+FFFF are 2 code units or 2 2-byte integers ((high surrogate: U+D800 ~ U+DBFF and low surrogate: U+DC00 to U+DFFF)
- In Low-level API: Use code point, a value of the int type (e.g., static methods in the Character class)

Use Character Streams

- On a higher level than byte stream
    - Generally, for human consumption since a character is a character in a natural langauge
- Instantiate concrete subclass of the Reader or Writer class
    - Common sources or destinations: files, character arrays, strings, byte streams, and others
    - Example: using files as sources or destinations
        - FileReader, FileWriter: use default character encoding only
        - InputStreamReader, OutputStreamWriter: can specify character encoding
- Must close streams to release resources
- I/O may cause errors, deal with exceptions
    - Example: cannot create streams, cannot read or write to streams

Character Stream: Examples

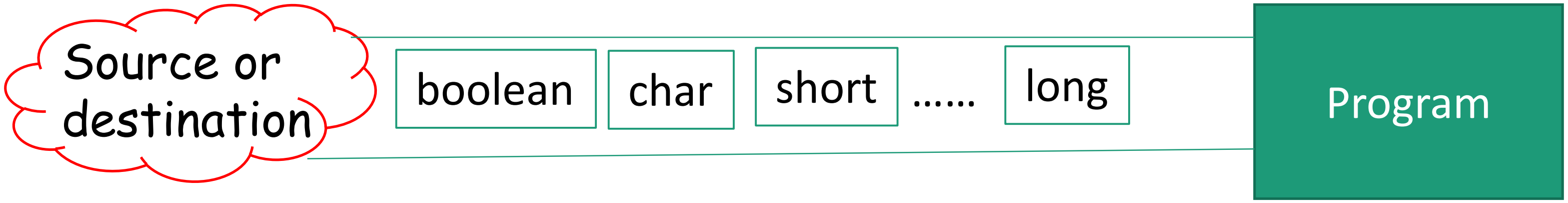
- Use try-catch-finally
- Use try-with-resources
- Important question: which character encoding is in use?
  - Reader and Writer's encoding scheme should match.
- Examples in the Sample Programs repository
  - CharFileCopier: uses default character encoding
  - CharFileStreamCopier: uses user provided character encoding

Questions

- Character, character encoding
- Unicode, Unicode code unit, Unicode code point
- Characters in the Java platform
- Character streams
  - Using default character encoding (what is the default encoding)?
  - Using user provided character encoding

Data Streams

- Data streams represents sequences of primitive data type values and String values in their internal representation (raw types)
  - boolean, char, byte, short, int, long, float, and double as well as String values in internal (called raw or binary representation) representation
  - Unformatted I/O
- Two interfaces: DataInput and DataOutput



Data Streams: Example

- Use DataInputStream (implementing DataInput)
  - Read primitive Java data type values from an underlying input stream in a portable way (machine-independent way)
  - Work with files: construct a FileInputStream first
- Use DataOutputStream (implementing DataOutput)
  - write primitive Java data type values to an output stream in a portable way
  - Work with files: construct a FileOutputStream first
- Use a Hex editor to examine file content
  - In Eclipse, install a Hex Editor from Eclipse Marketplace

Questions?

- Data streams
  - read boolean, char, byte, short, int, long, float, double, and String values
  - write boolean, char, byte, short, int, long, float, double, and String values

Formatted and Unformatted I/O

- Unformatted I/O
  - Transfers the internal (binary or raw) representation of the data directory between memory and the file
  - Example: read or write binary files
    - with DataInputStream, DataOutputStream
- Formatted I/O
  - Converts the internal (binary or raw) representation to characters before transferring to file
  - Converts to the internal binary representation from characters when transferring from a file
  - Example: read and write text files
    - with Scanner, PrintWriter (and PrintStream)

Formatted Input: Example

- Use Scanner
- Scanner breaks down inputs into tokens using a delimiter pattern
  - Delimiter pattern is expressed in Regular Expressions
  - Default delimiter pattern is whitespace
- The tokens may then be converted into values of primitive types or Strings using the various next methods.

Formatted Output: Example

- Use PrintWriter and PrintStream
- System.out and System.err are PrintStream objects
- Formatting
  - PrintWriter and PrintStream support formatting
  - String also supports formatting
    - One may use character streams to do formatted I/O
  - Similar to C/C++'s printf-family functions

Standard Streams

- Many operating systems have Standard Streams.
- By default, they read input from the keyboard and write output to the display.
- Standard Output Streams
    - Standard output: System.out, a PrintStream object
    - Standard error: System.err, a PrintStream object
- Standard Input Streams
    - Standard input: System.in, a byte stream

The Console Class

- Access the character-based console device associated with current JVM
- Not every JVM has a console
  - If it has one, obtain it via System.console()
  - If it doesn't, System.console() returns null
- A few read and write methods

Formatted or Unformatted?

	Formatted	Unformatted
Example	Text files	Binary files
Efficiency	Slower	Faster
Space	Larger	Smaller
Fidelity	Not exact	Exact
Portability	More	Less
Human Readability	More	Less

Culture and Formatted I/O

- Formatted I/O are often used for humans
- Our culture influences how we write and read, and how we format text
- Example

Language (Region)	Formatted Numbers
German (Germany)	123.456,789
German (Switzerland)	123'456.789
English (United States)	123,456.789

- Do you have any other examples?
  - How about numbers, currency, date, calendar, …

Locale

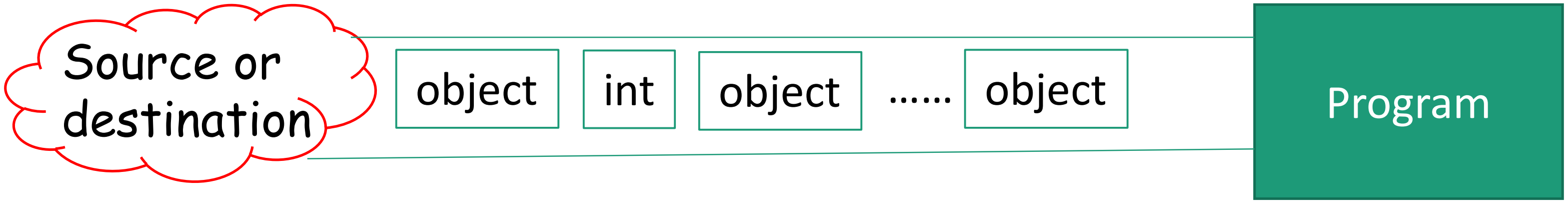
- Language and geographic environment are two important influences on our culture
- Locate in a computer system is to represent this concept
    - Language and geographical region (e.g., country)
- Java
    - java.util.Locale
- When use formatted I/O, we should always consider
    - Locale
    - Character encoding

Questions

- Concepts of Formatted I/O and Unformatted I/O
- When to use Formatted I/O and Unformatted I/O?
- How to use Formatted I/O and Unformatted I/O?
- Examples?
  - PrintWriter, and Scanner
  - Locale, character encoding

Object Streams

- Object streams represent a sequence of graphs of Java objects and primitive data type values
  - Objects must be serializable (corresponding class implements the java.io.Serializable interface)
  - An object may reference another object, forming a graph of objects



Object Streams: Example

- ObjectInputStream, ObjectOutputStream
  - They implement ObjectInput and ObjectOutput
  - ObjectInput is a sub-interface of DataInput
  - ObjectOutput is a sub-interface of DataOutput
  - Object streams are data streams of serialized objects and primitive type values
- More when we discuss I/O via computer networks

Questions?

- Concept of object streams
- Serialized objects
- Example application

Memory Buffer for Streams

- A memory buffered may be allocated with a stream to increase I/O efficiency
- Unbuffered streams
- Buffered streams

Buffered and Unbuffered Streams

- *Unbuffered* I/O and Streams
  - Each read or write request is handled directly by the underlying OS.
  - Each request often triggers disk access, network activity, or others
- Buffered I/O and streams.
    - Buffered input streams read data from a memory area known as a *buffer;* the native input API is called only when the buffer is empty.
    - Similarly, buffered output streams write data to a buffer, and the native output API is called only when the buffer is full
  - Buffered I/O streams are generally more efficient.

Buffered Streams

- Byte streams
  - BufferedInputStream
  - BufferedOutputStream
- Character streams
  - BufferedReader
  - BufferedWriter

Use Buffered Streams

- Wrap an unbuffered streams with a buffered stream
  - Example
    - reader = new BufferedReader(new FileReader(“kaynmay.txt"));
    - writer = new BufferedWriter(new FileWriter(" kaynmay.txt "));
    - in = new BufferedInputStream(new FileInputStream(“keynmay.bin”));
    - out = new BufferedOutputStream(new FileOutputStream(“keynmay.bin”));
- Flushing buffered output streams
  - Write out a buffer at critical points, without waiting for it to fill.
  - To flush a buffered output stream manually, invoke its flush method.
  - Some buffered output classes support autoflush
    - Example: an autoflushable PrintWriter object flushes the buffer on every invocation of println or format.

Questions

- Concept of buffered I/O
- Buffered streams
- Flush buffered streams
- How about unbuffered streams?
- Examples
  - We can easily revise the example application discussed to use buffered streams

Random Access and Sequential Access

- Sequential access
  - Read sequentially, byte or character or other units are read sequentially one after another
  - Generally, streams must be read sequentially
- Random access
  - Behaves like a large array of bytes stored in the file system.
  - Any byte or character or other units can be read without having to read anything before it first
  - RandomAccessFile

Random Access File

- Generally, provides
  - Length
    - the size of the file
  - File pointer/Cursor
    - an index into the implied array, pointing to the byte next read reads from or next write writes to.
    - Each read or write results an advancement of the pointer
    - The file pointer can be obtained
  - Seek:
    - Set the file pointer
  - Generally, unformatted files (binary files)

When to use Random Access Files?

- For applications
  - only interested in a portion of the file.
  - read and write large files that cannot be load into the memory

Random Access File: Example

- The File and Files classes
- RandomAccessFile
- Use RandomAccessFile
  - Must be opened in a mode
    - “r”: read-only
    - “rw”: read & write
    - “rws”: read, and write content & metadata synchronously
    - “rwd”: read, and write content synchronously

Questions

- Concept of sequential and random access
- Use random access files

More about Files and Paths

- File, Path, Files, and Paths
- File operations use system resources
  - Release resources, one of the two, if permissible
    - close() method to release resources explicitly
    - Try-with-resources blocks (if implemented Autocloseable)
- File operations often throws exceptions
  - Catch or specify requirement for the exceptions
- Method chaining
- Link awareness
- Variable arguments (varargs)
- Some file operations are "atomic"
- Two methods of Files accept Glob parameter

Questions

- Check a file or a directory
- Delete, copy, or move a file or directory
- Manage metadata, i.e., file and file store attributes
- Read, write, and create files; create and read directories; read, write, and create random access file
- Create and read directories
- Deal with Links
- Walk a file tree, find files, watch a directory for changes

Assignments

- Practice assignments include bonus assignments