

Core Java

Day18 Agenda

- IO Framework

IO Framework

Chaining IO Streams

- Each IO stream object performs a specific task.
 - `FileOutputStream` -- Write the given bytes into the file (on disk).
 - `BufferedOutputStream` -- Hold multiple elements in a temporary buffer before flushing it to underlying stream/device. Improves performance.
 - `DataOutputStream` -- Convert primitive types into sequence of bytes. Inherited from `DataOutput` interface.
 - `ObjectOutputStream` -- Convert object into sequence of bytes. Inherited from `ObjectOutput` interface.
 - `PrintStream` -- Convert given input into formatted output.
 - Note that input streams does the counterpart of `OutputStream` class hierarchy.
- Streams can be chained to fulfil application requirements.

Primitive types IO

- `DataInputStream` & `DataOutputStream` -- convert primitive types from/to bytes
 - primitive type --> `DataOutputStream` --> bytes --> `FileOutputStream` --> file.
 - `DataOutput` interface provides methods for conversion - `writeInt()`, `writeUTF()`, `writeDouble()`, ...
 - primitive type <-- `DataInputStream` <-- bytes <-- `FileInputStream` <-- file.
 - `DataInput` interface provides methods for conversion - `readInt()`, `readUTF()`, `readDouble()`, ...

DataOutput/DataInput interface

- interface `DataOutput`
 - `writeUTF(String s)`
 - `writeInt(int i)`
 - `writeDouble(double d)`
 - `writeShort(short s)`
 - ...
- interface `DataInput`
 - `String readUTF()`
 - `int readInt()`
 - `double readDouble()`
 - `short readShort()`
 - ...

Serialization

- `ObjectInputStream` & `ObjectOutputStream` -- convert java object from/to bytes
 - Java object --> `ObjectOutputStream` --> bytes --> `FileOutputStream` --> file.
 - `ObjectOutput` interface provides method for conversion - `writeObject()`.
 - Java object <-- `ObjectInputStream` <-- bytes <-- `FileInputStream` <-- file.
 - `ObjectInput` interface provides methods for conversion - `readObject()`.
- Converting state of object into a sequence of bytes is referred as Serialization. The sequence of bytes includes object data as well as metadata.
- Serialized data can be further saved into a file (using `FileOutputStream`) or sent over the network (Marshalling process).
- Converting (serialized) bytes back to the Java object is referred as Deserialization.
- These bytes may be received from the file (using `FileInputStream`) or from the network (Unmarshalling process).

ObjectOutput/ObjectInput interface

- interface `ObjectOutput` extends `DataOutput`
 - `writeObject(obj)`
- interface `ObjectInput` extends `DataInput`
 - `obj = readObject()`

Serializable interface

- Object can be serialized only if class is inherited from `Serializable` interface; otherwise `writeObject()` throws `NotSerializableException`.
- `Serializable` is a marker interface.

transient fields

- `writeObject()` serialize all non-static fields of the class. If fields are objects, then they are also serialized.
- If any field is intended not to serialize, then it should be marked as "transient".
- The transient and static fields (except `serialVersionUID`) are not serialized.

serialVersionUID field

- Each serializable class is associated with a version number, called a `serialVersionUID`.
- It is recommended that programmer should define it as a static final long field (with any access specifier). Any change in class fields expected to modify this `serialVersionUID`.

```
private static final long serialVersionUID = 1001L;
```

- During deserialization, this number is verified by the runtime to check if right version of the class is loaded in the JVM. If this number mismatched, then `InvalidClassException` will be thrown.
- If a serializable class does not explicitly declare a `serialVersionUID`, then the runtime will calculate a default `serialVersionUID` value for that class (based on various aspects of the class described in the Java(TM) Object Serialization specification).

Buffered streams

- Each `write()` operation on `FileOutputStream` will cause data to be written on disk (by OS). Accessing disk frequently will reduce overall application performance. Similar performance problems may occur during network data transfer.
- `BufferedOutputStream` classes hold data into a in-memory buffer before transferring it to the underlying stream. This will result in better performance.
 - Java object --> `ObjectOutputStream` --> `BufferedOutputStream` --> `FileOutputStream` --> file on disk.
- Data is sent to underlying stream when buffer is full or `flush()` called explicitly.
- `BufferedInputStream` provides a buffering while reading the file.
- The buffer size can be provided while creating the respective objects.

PrintStream class

- Produce formatted output (in bytes) and send to underlying stream.
- Formatted output is done using methods `print()`, `println()`, and `printf()`.
- `System.out` and `System.err` are objects of `PrintStream` class.

Scanner class

- Added in Java 5 to get the formatted input.
- It is `java.util` package (not part of java io framework).

```
Scanner sc = new Scanner(inputStream);  
// OR  
Scanner sc = new Scanner(inputFile);
```

- Helpful to read text files line by line.

Character streams

- Character streams are used to interact with text file.
- Java char takes 2 bytes (unicode), however char stored in disk file may take 1 or more bytes depending on char encoding.
 - <https://www.w3.org/International/questions/qa-what-is-encoding>
- The character stream does conversion from java char to byte representation and vice-versa (as per char encoding).
- The abstract base classes for the character streams are the `Reader` and `Writer` class.
- `Writer` class -- write operation
 - `void close()` -- close the stream

- `void flush()` -- writes data (in memory) to underlying stream/device.
- `void write(char[] b)` -- writes char array to underlying stream/device.
- `void write(int b)` -- writes a char to underlying stream/device.
- Writer Sub-classes
 - `FileWriter`, `OutputStreamWriter`, `PrintWriter`, `BufferedWriter`, etc.
- Reader class -- read operation
 - `void close()` -- close the stream
 - `int read(char[] b)` -- reads char array from underlying stream/device
 - `int read()` -- reads a char from the underlying device/stream. Returns -1
- Reader Sub-classes
 - `FileReader`, `InputStreamReader`, `BufferedReader`, etc.

Assignments

1. Write a program that inputs 4 lines and stored them in a text file. Use `BufferedWriter` class.
2. Read a text file line by line and display on terminal. Hint: file --> `FileReader` --> `BufferedReader` --> ...
3. In menu-driven program of Books (with `ArrayList`) add two more menus. Save books in file and load books from file. Use `DataOutputStream` and `DataInputStream`.
4. In menu-driven program of Emp/Person (with `ArrayList`) add two more menus. Save books in file and load books from file. Use `ObjectOutputStream` and `ObjectInputStream`.