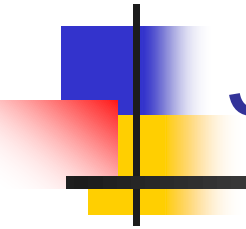
Fpt University



Java Input-Output

Object Oriented Programming with Java

Chapter 7

FPTU Da Nang – IT Department

Objectives - Contents

- **File**
- **RandomAccessFile**
- **Streams**
  - Low-Level Streams
  - High-Level Filter Streams
- **Readers, and Writers**
  - Low-level reader and writer
  - High-level reader or writer
- **Object Streams and Serialization**

General introduction

- Java Input and output classified
  - Access mechanism
    - Random Access – RandomAccessFile
      - allow simultaneous read and write
    - Sequential access - using stream
      - Separate class for read and write operation
  - Data format
    - Text :
      - Reader : for reading text data
      - Writer: for writing text data
    - Binary
      - InputStream: stream of binary data to read
      - OutputStream: stream of binary data to write

General introduction

- Access level
  - Low level
    - read and write directly on the input/output device
    - read or write 1 unit of data (1 byte – 1 character)
  - High level
    - Using cache for reading and writing block of data
- Utility I/O class
  - InputStreamReader, OutputStreamWriter : convert binary to text
  - Scanner : helpful class for input from console
- Ex: FileInputStream, FileOutputStream BufferedReader, BufferedWriter, FileReader..
- Class for interact with OS File System
  - File

File – access OS file system

- **java.io.File**
  - Used to represent file or directory names on the OS's file system
  - Not for reading and writing data
- Constructor
  - File(String pathname);
  - File(String dir, String subpath);
  - File(File dir, String subpath);
    - File f1 = new File("C:\\a");
    - File f2 = new File(f1, "Xyz.java");

    Note: new File(): No files are created on disk.

File – methods

- **boolean exists()**: check whether file name or directory exists?
- **String getAbsolutePath()**: absolute path.
- **String getName()**: returns the file or directory name.
- **String getParent()**: parent folder name
- **boolean isDirectory()**: is the directory?
- **boolean isFile()**: is the file?
- **String[] list()**: list namesAsianc files & subdirectories of the directory

File – methods

- **boolean canRead()**
- **boolean canWrite()**
- **boolean delete()** delete files and folders
- **long length()**
- **boolean mkdir()** create folder
- **boolean renameTo(File newname)** rename file or folder

RandomAccessFile

- RandomAccessFile
  - random access, using cursor for seeking read/write data at any position on file
  - Can read | write block of data.
- Constructor
  - RandomAccessFile(String file, String mode)
  - RandomAccessFile(File file, String mode)
    + reading mode "r" - read write “rw.”
- Support read and write data in primitive type
  - Each read/write operation automatically moves the cursor to the new position
  - long length() throws IOException
  - void seek(long position) throws IOException
    move cursor to specific position

Fpt University

RandomAccessFile..

Read Method	Write Method
boolean readBoolean()	void writeBoolean(boolean b)
byte readByte()	void writeByte(int b)
short readShort()	void writeShort(int s)
char readChar()	void writeChar(int c)
int readInt()	void writeInt(int i)
long readLong()	void writeLong(long l)
float readFloat()	void writeFloat(float f)
double readDouble()	void writeDouble(double d)
int readUnsignedByte()	None
int readUnsignedShort()	None
String readLine()	None
String readUTF()	void writeUTF(String s)

Fpt University

Unicode & UTF

- Java uses two representations of text
  - Unicode encoding for text representation in memory
  - UTF for input and output
- Unicode uses 16 bits to represent 1 character.
- UTF
  - stands for "UCS Transformation Format,"
  - UCS: “Universal Character Set.”
  - compressed 16-bit Unicode character into encoding of 7-8 bit pattern : UTF-7, UTF-8
  - UTF encodes characters with enough bits to represent the character

Handle Input – output data

- Always use high-level I/O class for I/O tasks
- High-level I/O classes always use low-level I/O object to read- write data from IO device
  - The constructor of high-level I/O classes always accepts a low-level I/O object as an argument
- Using utility class for converter or simple.
- Ex:

```
PrintWriter pw= new PrintWriter(new FileWriter("data.txt"));

DataInputStream ds= new DataInputStream( new
FileInputStream("data.dat"));

BufferedReader br= new BufferedReader( new
InputStreamReader( new FileInputStream("data.txt")));
```

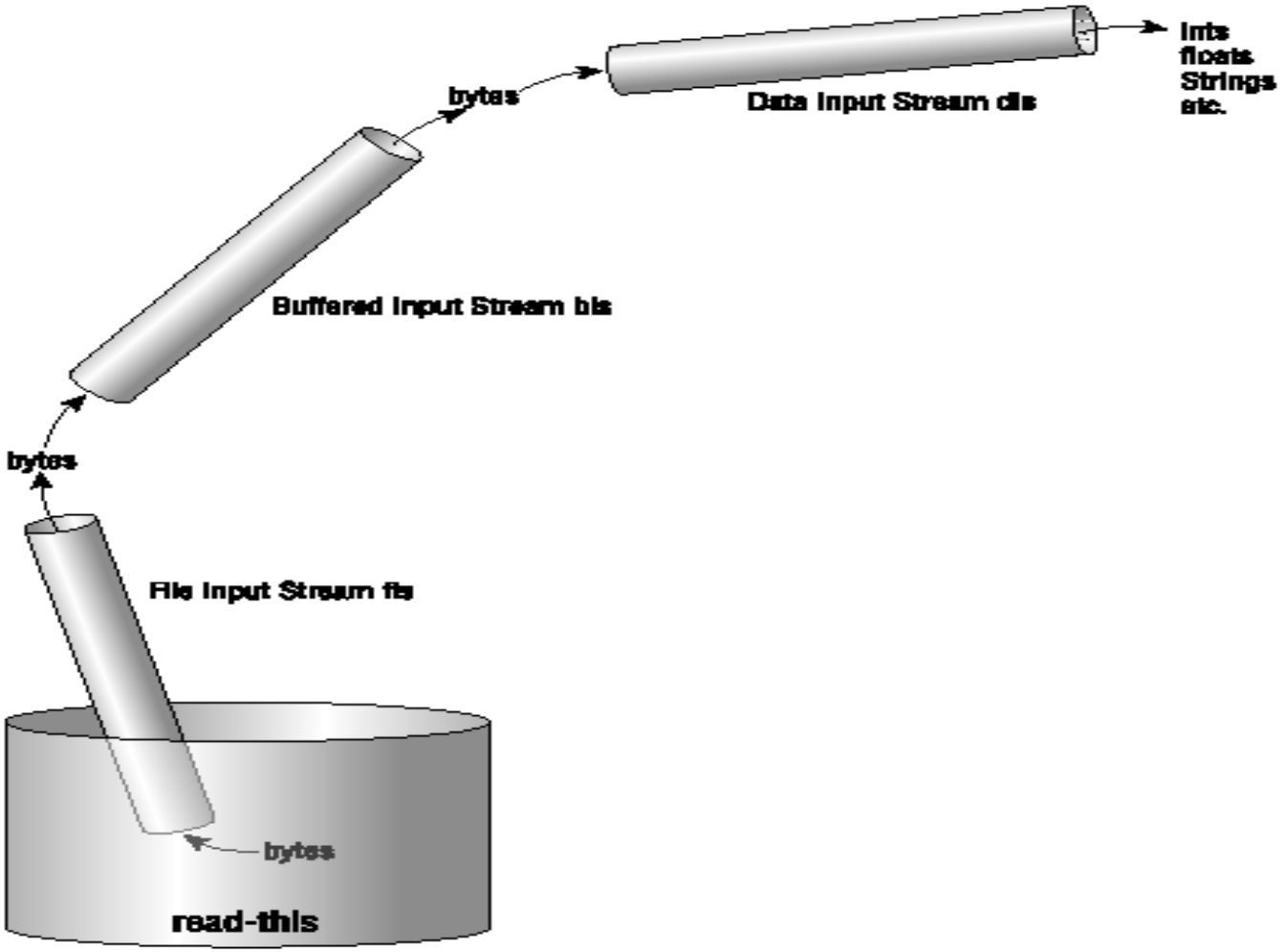
Text I/O – Reader and Writer

- ***Text IO* classes** all extend from the abstract super class Reader and Writer
  - FileReader(String pathname)
  - FileReader(File file)
  - FileWriter(String pathname)
  - FileWriter(File file)
  - CharArrayReader and CharArrayWriter
  - StringReader and StringWriter
  - PipedReader and PipedWriter
- High level text IO classes
  - BufferedReader, BufferedWriter
  - PrintWriter

Binary stream IO

- Low-level binary IO streams
  - InputStream & OutputStream are super class for all IO binary stream class
  - FileInputStream, FileOutputStream
- High-level binary IO classes
  - DataInputStream
    - read data from low level inputstream, return formatted data
  - DataOutputStream, PrintStream
    - receives data in primitive format, writes bytes to a low-level binary output stream
  - BufferedInputStream & BufferedOutputStream

High-Level Filter Streams



High-Level Streams (2)

- **DataInputStream methods**
  - boolean readBoolean() throws IOException
  - byte readByte() throws IOException
  - char readChar() throws IOException
  - double readDouble() throws IOException
  - float readFloat() throws IOException
  - int readInt() throws IOException
  - long readLong() throws IOException
  - short readShort() throws IOException
  - String readUTF() throws IOException

High-Level Streams (3)

- DataOutputStream is the symmetric version of DataInputStream.
- DataOutputStream(OutputStream ostream)
  - void writeBoolean(boolean b) throws IOException
  - void writeByte(int b) throws IOException
  - void writeBytes(String s) throws IOException
  - void writeChar(int c) throws IOException
  - void writeDouble(double d) throws IOException
  - void writeFloat(float b) throws IOException
  - void writeInt(int i) throws IOException
  - void writeLong(long l) throws IOException
  - void writeShort(int s) throws IOException
  - void writeUTF(String s) throws IOException

High-Level Streams (4)

- **BufferedInputStream** & **BufferedOutputStream**:
  - use buffer memory to read and write blocks of data, increasing read and write performance
- **PrintStream**:
  - Can write text or primitives.
  - The primitive type will be converted to bytes
  - System.out & System.err are instance variables

Object Streams and Serialization

- Object Streams are used for reading and writing objects
- *Serialization* is the process of decomposing an object and converting it to a contiguous sequence of bytes - for data transmission in the distributed network enviroment.
  - Serialized data only – not class definitions
  - Snapshot of the object in memory.
  - Static data does not serialize.
  - Declare **transient** for non-serializable variables

Object Streams & Serialization

- Example:

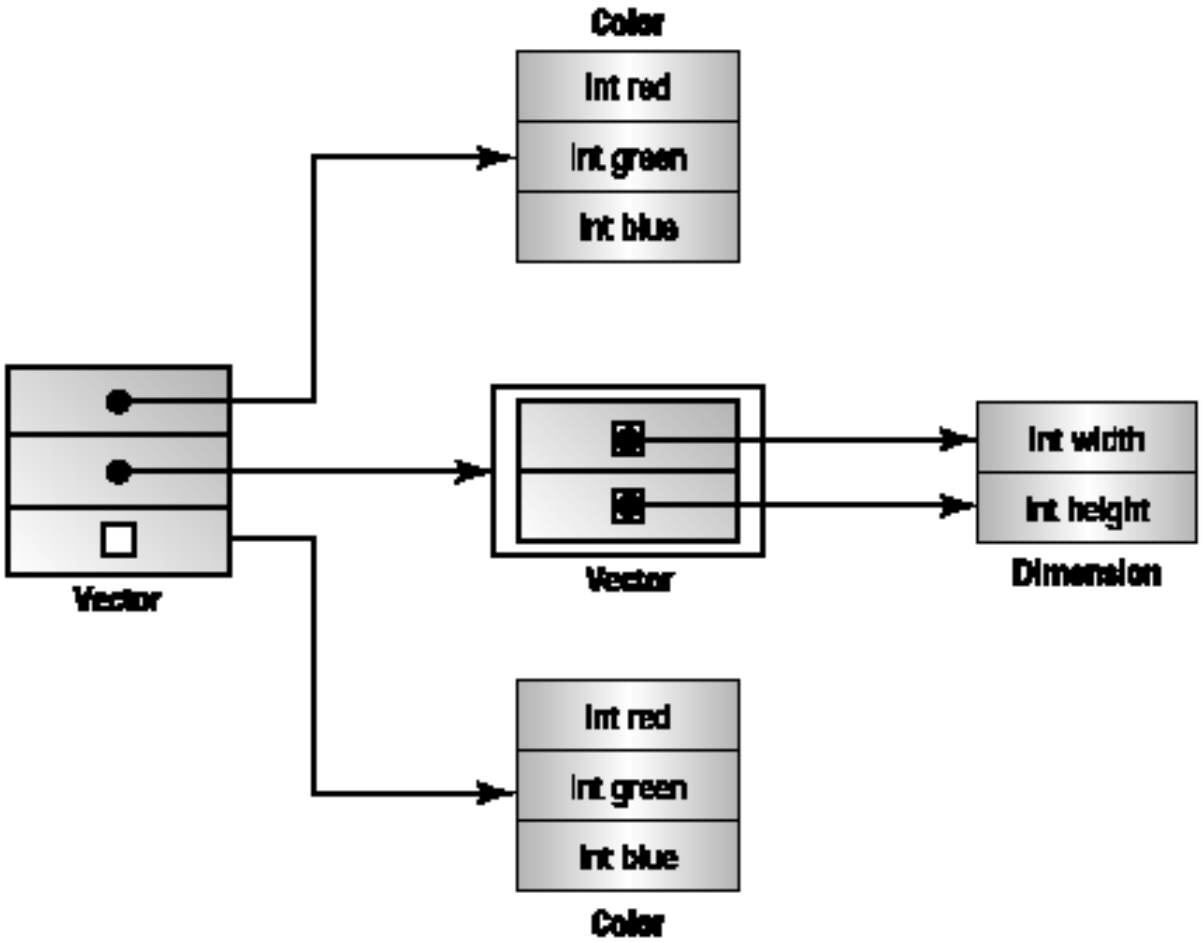
```
try {
FileOutputStream fos =new FileOutputStream("xx.ser");
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(myVector);
oos.close();
fos.close();
}
   catch (IOException e) { }
```

Object Streams and Serialization

- Cont..

```
try {
 FileInputStream fis =new FileInputStream (“xx.ser”);
 ObjectInputStream ois = new ObjectInputStream(fis);
 Vector vector = (Vector)(ois.readObject());
 int b = ((Color)(vec.elementAt(2))).getBlue();
 ois.close();
 fis.close();
 }
 catch (IOException e) { }
```

Object Streams and Serialization



The Serializable interface

- Class declared **implements Serializable** to make it possible for serialization.
- **Serializable** is a tag interface
  - there is no method to override
- public class Account implements Serializable{
  private transient Date lastAccess;
  private int accNo;

  …
  //
  }

Exercise

- Write an application that reads its own source file and prints out the statistics of characters frequency appear in the code
- Add “saving function” for Student Management Application so that when the program ends, save the data to a file, when running the program will read the file again to recover old data.

Constructive questions

- Why I/O in Java only used separate classes for read-write task but not for both?
- Declare and initialize a variable of the I/O class to read a text file.
- Want to read and write Student's score to a file, but can't find any method to read and write GPA score in FileReader/FileWriter or FileInputStream/FileOutputStream classes. Explain and suggest solutions.
- Proposing a solution to save program running state and all data.. Without using DBMS.

Constructive questions

- How to quickly identify a class in the java.io library used for what I/O purposes?
- Why don't high level I/O classes have default constructor.. as is common in other classes?
- Game applications often have a Save function that can save the state of the game and Load to reload an unfinished game. Suggest an I/O solution for this purpose.
- How to exchange data between 2 computers in distributed applications in a network environment? An example of an online game.
- What does the implements Serializable declaration mean?
- What is the transient declaration used for? Give an example..