Java Course
Lecture 9 - Java IO

PRAGMATIC IT Learning & Outsourcing Center

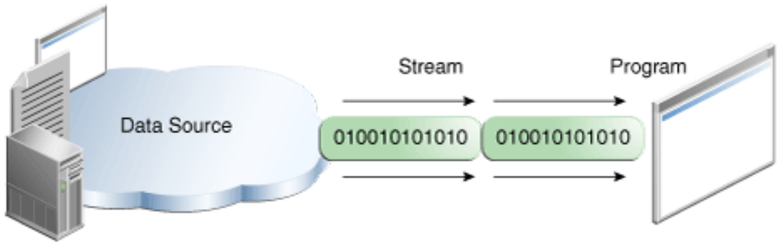
www.pragmatic.bg

Summary or Agenda?.. Lets call it Agenda…

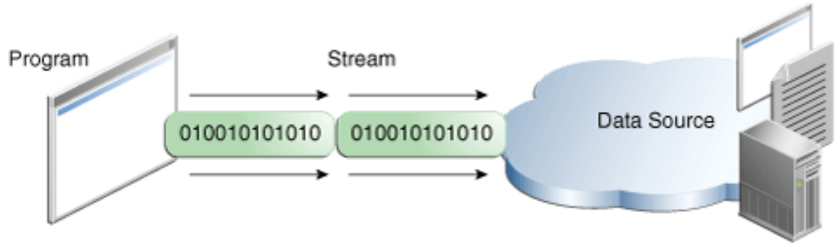
- I/O
- Streams
  - Binary
  - Text
- Files and Directories
  - java.util.File
  - java.nio.Files
- Serialization
  - Default serialization
  - Custom serialization

Java IO

- What's IO
  - The communication between an information processing system (such as a computer) and the outside world
    - ➢ Wikepedia
- What's IO to a computer program
  - I (Input) – that which goes in the program



  - O (output) – that which is outputted by the program



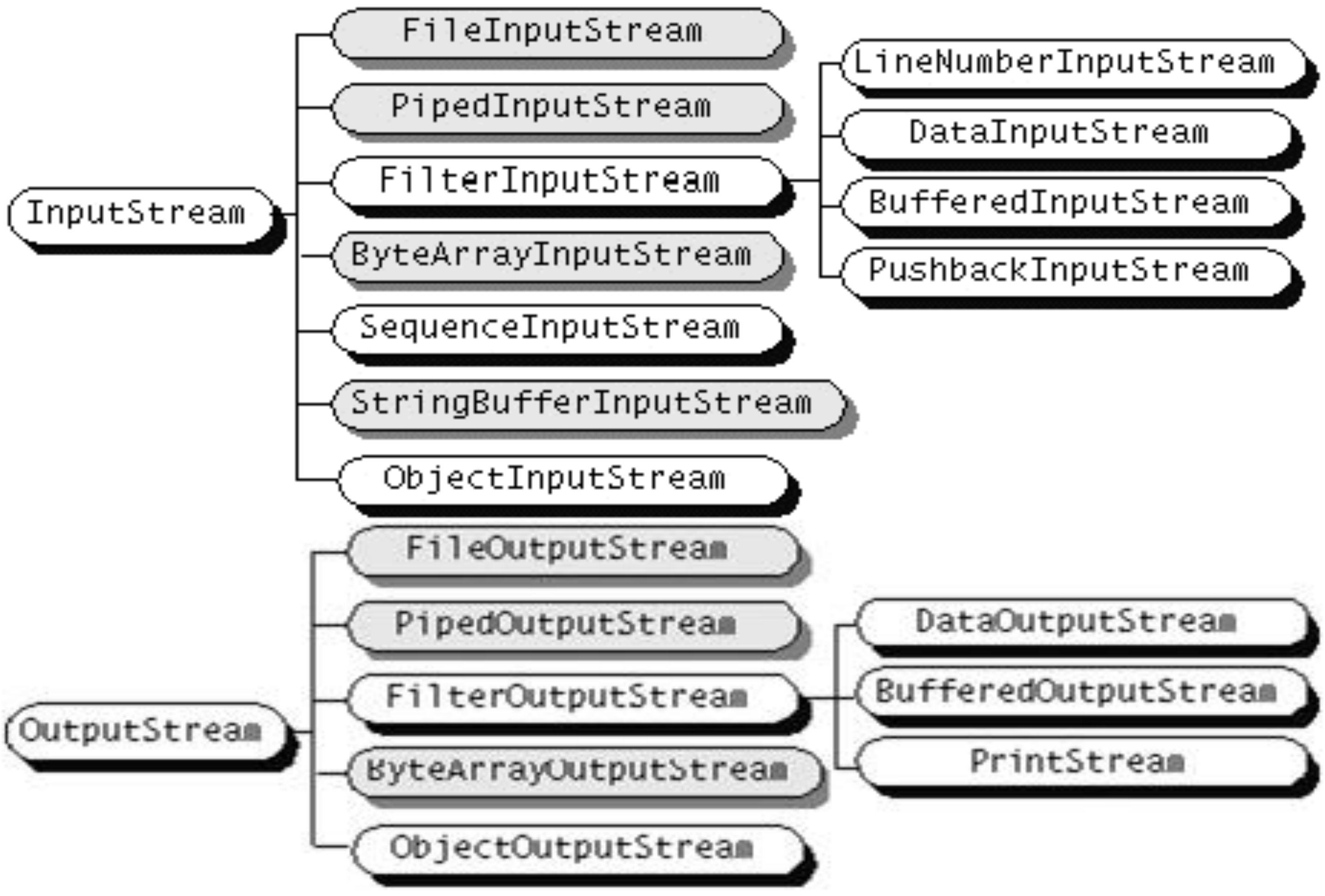
Streams

- What's a stream ?
  - Sequence of data elements made available over time
    - Wikipedia
- Types of streams
  - Binary – non human readable
  - Character –human readable (xml, html , json)

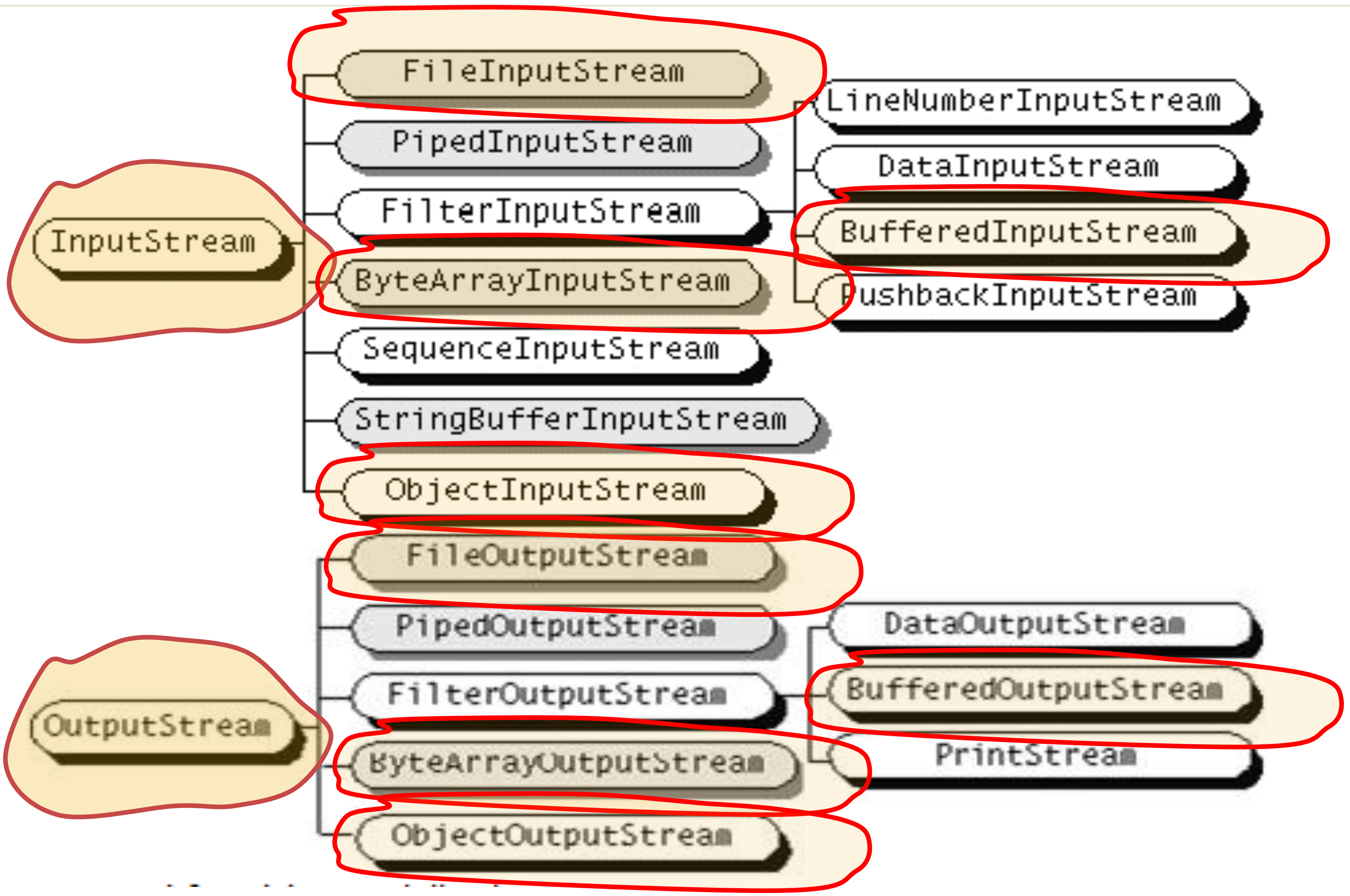
Binary Data

- Non human readable stuff
  - Example : image files, exe files, dlls, .class files
- Simple interpretation of the raw bytes as meaningful data doesn't yield results
- There must be a good specification on how to interpret the binary data otherwise it is completely meaningless to the reader
- In java raw binary data is handled by the Input/ OutputStream classes

I/O Streams


InputStream
FileInputStream
PipedInputStream
FilterInputStream
LineNumberInputStream
DataInputStream
BufferedInputStream
PushbackInputStream
ByteArrayInputStream
SequenceInputStream
StringBufferInputStream
ObjectInputStream
OutputStream
FileOutputStream
PipedOutputStream
FilterOutputStream
DataOutputStream
BufferedOutputStream
PrintStream
ByteArrayOutputStream
ObjectOutputStream

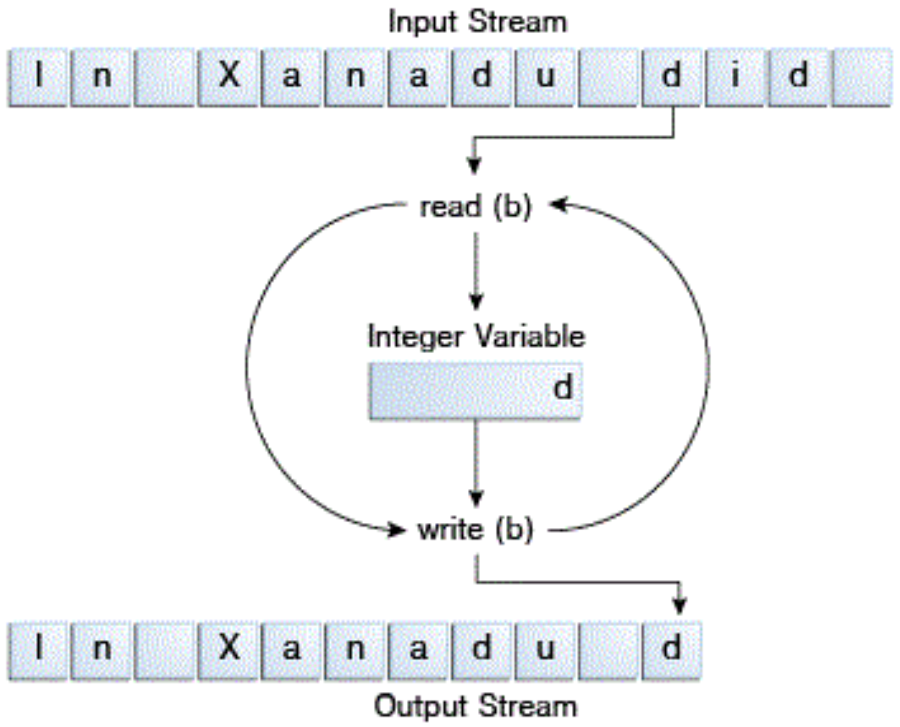
Commonly used ones:



Predefined System streams

- System.in (InputStream)
  - Standard input (stdin)
  - Use for taking input
  - Preferable than files for console tools
- System.out (PrintStream)
  - Standard output (stdout)
  - Use for displaying results and user interaction
- System.err (PrintStream)
  - Standard error (stderr)
  - Use for displaying errors (like exception stack traces)
  - The default output stream for Exception.printStackTrace()

So how was it working?

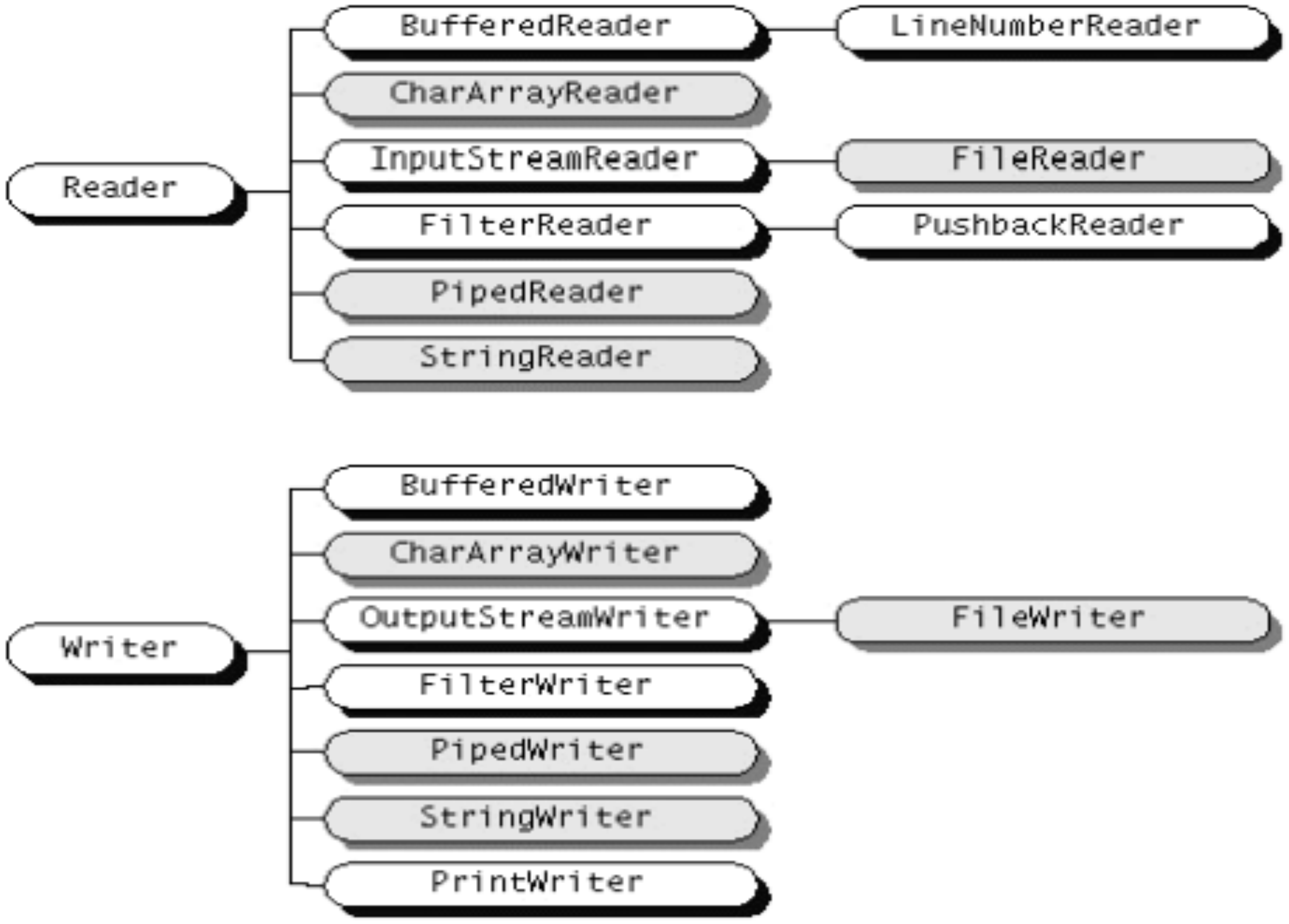


- Note this seems like a normal program, but it actually represents a kind of low-level I/O that you should avoid. Since *.txt contains character data, the best approach is to use character streams

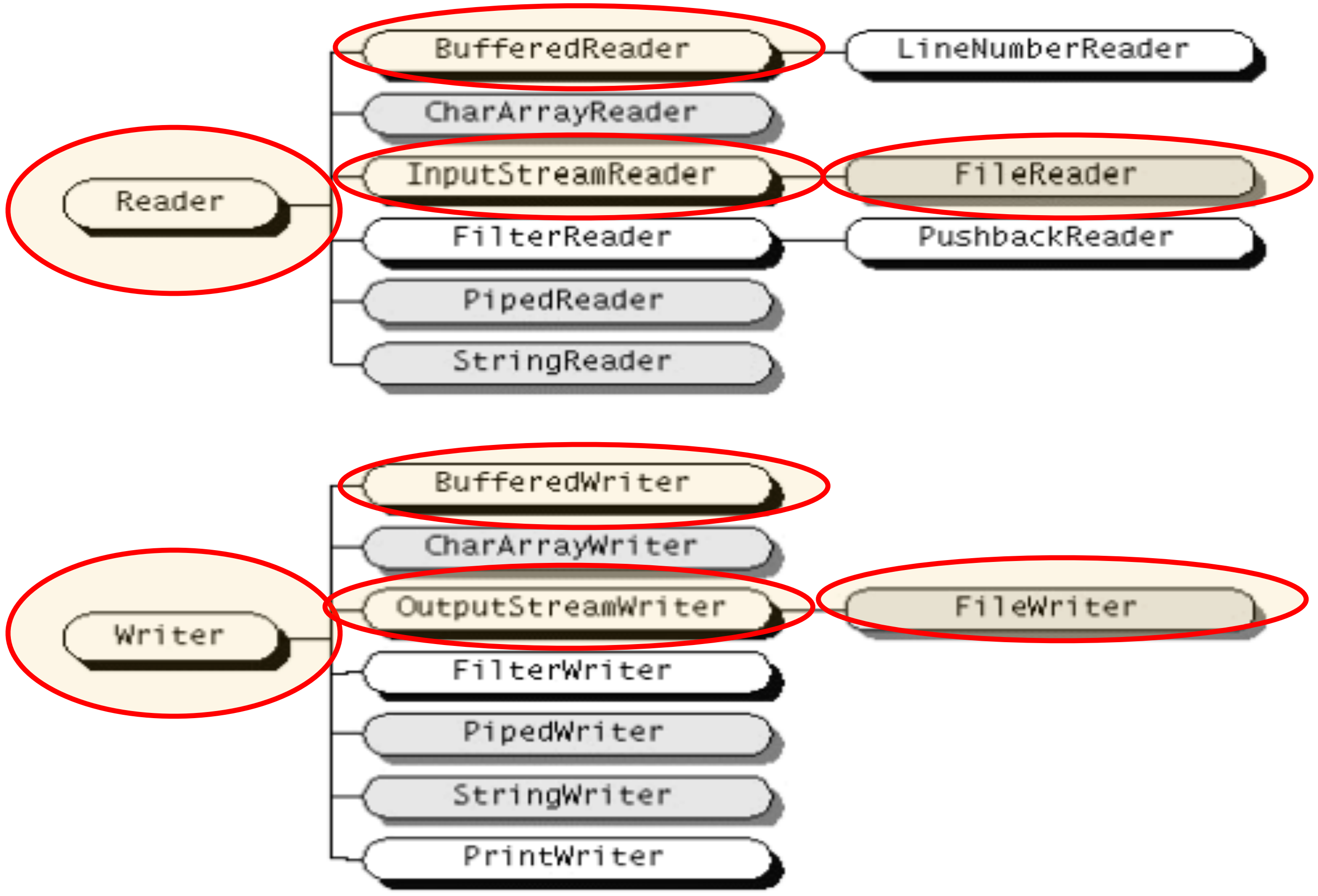
Character Data Streams

- Reader / Writer
  - Character sets (symbol-to-number mapping)
  - Encodings (number-to-bits/bytes mapping)
  - "Character encoding" = character set + encoding
  - Normal human readable text files
    - For example a simple .txt file

Reader/ Writer



Commonly used ones

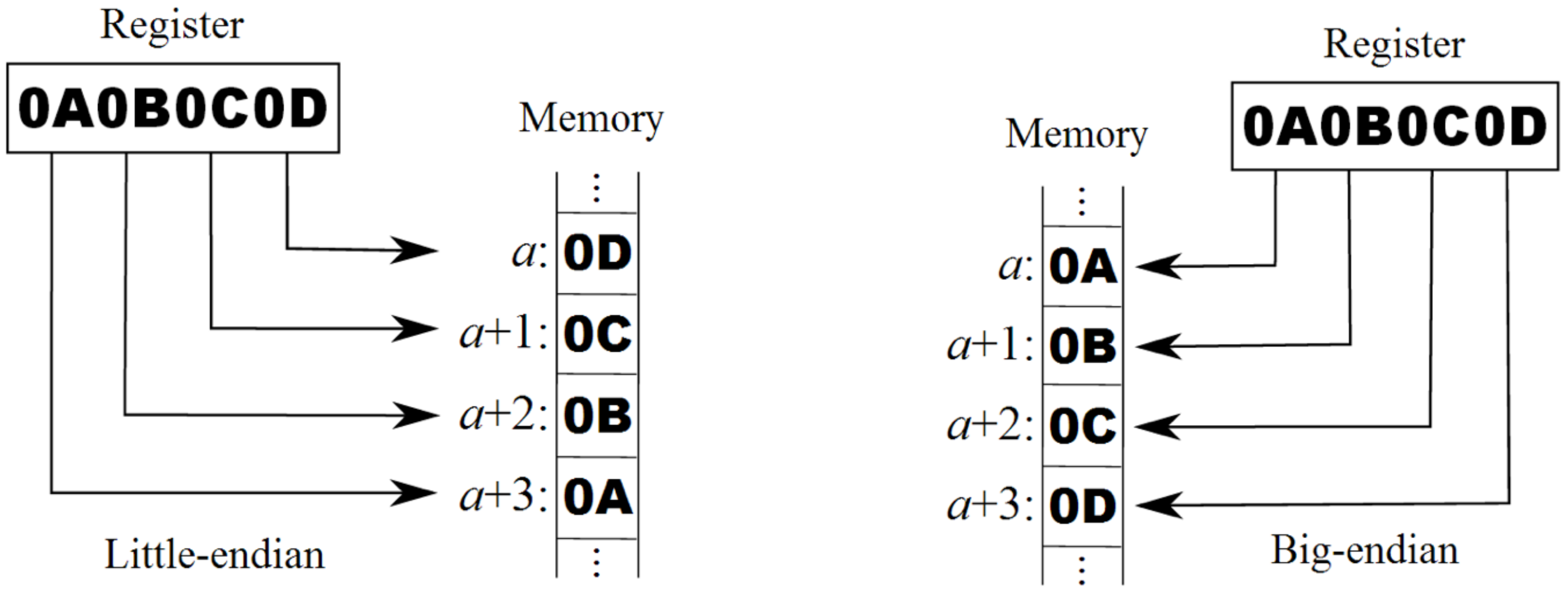


Character sets

- Most common character set – ASCII
  - ASCII has 127 symbols and it represents the latin alphabet
  - For more info on charsets go to http://en.wikipedia.org/wiki/Category:Character_sets
  - Java uses Unicode since version 1.0

Encoding

- Little Endian/Big Endian



- @See http://en.wikipedia.org/wiki/Endianness

File Streams

- Text Files:
  - FileReader
  - FileWriter
- Binary Files:
  - FileInputStream
  - FileOutputStream
  - *Read from / write to* a file on the file system

File Streams

- FileInputStream and FileOutputStream are to be used with binary data.
  - Using them means that whatever file you have you will be managing it bit by bit
- FileReader and FileWriter are used with text files.
  - Never try to read a binary file using a Reader or a Writer since that might cause data corruption.
  - Java will try to interpret the binary stream as a character data steam

Buffered Streams

- Character:
  - BufferedReader
  - BufferedWriter
- Byte:
  - BufferedInputStream
  - BufferedOutputStream
    - Reducing the number of accesses required on the original data Use for better performance
    - If you use FileInputStream and FileOutputStream a file with 10000 bytes will require file to be accessed 10000 times. Buffered classes however use a buffer so they work in chunks (512 bytes by default)

java.util.Scanner

- Not a member of the stream API per say
- Regular expression based tokenizing
- Can be constructed from String, InputStream, File…
- Easily scan all scalar types, strings and big numbers
- Supports looking ahead
- Support for getting tokens matching given regular expression

PrintStream/ PrintWritter

- Classes PrintStream and PrintWriter
  - PrintStream wraps OutputStream
  - PrintWriter wraps Writer
- Can be constructed by File, file name, OutputStream
  - PrintWriter – using a Writer
- Most used methods are:
  - print(), println() – arguments are concatenated
  - printf(), format() – the new format strings.. in a way..new..
  - Support automatic or manual flushing
  - Throws IO exceptions only in constructors

printf method (added mostly for C lovers)

- Two overloads:
  - PrintStream printf(Locale l, String format, Object...args)
  - PrintStream printf(String format, Object... args)
- Example :
  - System.out – is a PrintWritter .Try using the printf function and output something to the console.

Difference between PrintStream and PrintWriter?

- PrintStream prints to an OutputStream, and PrintWriter prints to a Writer.
- And the difference between an OutputStream and a Writer?
- Both are streams, with the primary difference being a OutputStream is a stream of bytes while a Writer is a stream of characters.
- PrintStream is not so commonly used because methods like print(String) uses the default system encoding which leads to bugs. With writer you can specify the encoding you want to be used.

Random Access Files

- Random accessing files is likely when:
  - You are working with binary files
  - You have large record based files
  - You want to both read and write into a file
  - You want to access just part of the information of a file (a single entry from an archive file)
  - You want to implement an in-file database

Random access files

- In package java.io
- Implements DataInput and DataOutput
- Accepts four open modes in constructor:
  - r – for reading only
  - rw – for reading and writing
  - rwd – for flushing on each operations
  - rws – for flushing including the metadata
- Provides:
  - int skipBytes(int)
  - void seek(long)
  - long getFilePointer()

java.io.File

- java.io.File
  - path to a file system object – file or directory
  - Provides platform independent file system access
  - Can be constructed by URI, path, or parent and name of a child
  - Provides basic file operations and queries
  - Provides getting directory items
  - Provides directory creation
  - Provides creation of a temporary files
  - To create a child of an item use the constructor!
  - Lacks advanced file system operations !

java.nio.Files

- New API introduced with java 7
- A new class that has a plethora of utility methods
- Created to ease the pain with handling native filesystem operations.
- Allows for better control over the common tasks associated with native files
    - Copy/ paste/ delete and so on

java.nio.Path

- Very similar to the old java.io.File class
- It represents a file path
- java.nio.Paths – a utility class that has a lot of methods for creating new path objects by using a string or an URI object

Serialization

- The problem:
  - All reusable Java objects exist only as long as the Java virtual machine remains running
  - What if we want our objects to exist beyond the lifetime of the virtual machine?
- Solution:
  - Object serialization provides an ability to read and write a whole object tree to and from a byte stream
  - Java Serialization API is small and easy to use
  - It provides a standard mechanism for developers to handle object serialization

Default Serialization

- Just implement `java.io.Serializable`
- Serializable interface is just marker interface, it doesn't contain any methods
- However there is a requirement:
  - All fields of the class have to be either primitive data types, or represent other serializable objects
  - Non-serialized fields should be marked as `transient` and their values will not be serialized. The values of `static` fields are also not serialized

```
import java.io.Serializable;
import java.util.Date;

class Person implements Serializable {
     String name = "Peter";
     int age = 28;
     Date birthDate = new Date();
  }
```

How to invoke serialization?

- Use an output stream
  - For example `FileOutputStream`
- Chain it with the `ObjectOutputStream`
- Call the method `writeObject()` providing the instance of a `Serializable` object as an argument
- Don't forget to close the streams

Serialization example

```
public static void serialize(Person person,
    String fileName) throws IOException {
  FileOutputStream fileOut =
    new FileOutputStream(fileName);
  ObjectOutputStream objOut =
    new ObjectOutputStream(fileOut);
  try {
    objOut.writeObject(person);
  } finally {
    objOut.close();
    fileOut.close();
  }
}
```

How to invoke deserialization

- Open an input stream
- Chain it with the `ObjectInputStream`
- Call the method `readObject()` and cast the returned object to the class that is being deserialized
- Again don't forget to close all streams
- During the process of deserialization all transient variables will be initialized with default values according to their type

Deserialization example

```
public static Person deserialize (String
fileName)
throws IOException, ClassNotFoundException {
   FileInputStream fileIn =
      new FileInputStream(fileName);
   ObjectInputStream objIn =
      new ObjectInputStream(fileIn);
   try {
      Person person = (Person) objIn.readObject();
      return person;
   }
   finally {
      objIn.close();
      fileIn.close();
   }
}
```

Custom serialization

- Java provides also am explicit control of the serialization process
- To use that you need to **implement** interface `Externalizable` which itself extends `Serializable` and adds two methods:

```
public void writeExternal(ObjectOutput out)
    throws IOException;
 public void readExternal(ObjectInput in) throws
    IOException, java.lang.ClassNotFoundException
```

Example

```
class SomePerson implements java.io.Externalizable
{
   String name;
   int age;
   java.util.Date someDate;

   public void writeExternal(ObjectOutput stream)
        throws java.io.IOException {
      stream.writeUTF(name);
      stream.writeInt(age);
   }

   public void readExternal(ObjectInput stream)
         throws java.io.IOException {
      name = stream.readUTF();
      age = stream. readInt();
   }
}
```

What else to read

- If you want to know more … : http://docs.oracle.com/javase/tutorial/essential/io/

Q and A ?

www.pragmatic.bg

PRAGMATIC IT Learning & Outsourcing Center

PRAGMATIC IT Learning & Outsourcing Center

Problems ?!

1. What's the difference between Writers and OutputStreams
2. When is a good idea to use a buffered stream ?
3. What class would you use to read a few pieces of data that are at known positions near the end of a large file?
4. What object represents a file from the file system.
5. Write a program that deletes a file ?
   - @see File.delete method
6. Write the same program but this time using the new Files class ?
7. Write a program that finds all files within a directory and prints their name and size?

Problems(2)

1. Create a class `Client` containing first and last name. `Client` should also have a bank account and address, that has to be modeled by separate classes. Make the whole hierarchy serializable. Implement serialization and deserialization of `Client` objects to and from a file.
2. Use the above class hierarchy and customize the serialization in a way that the bank account information is not persisted.