Java I/O (serialization) and stream

CSE-220

Covered Topics

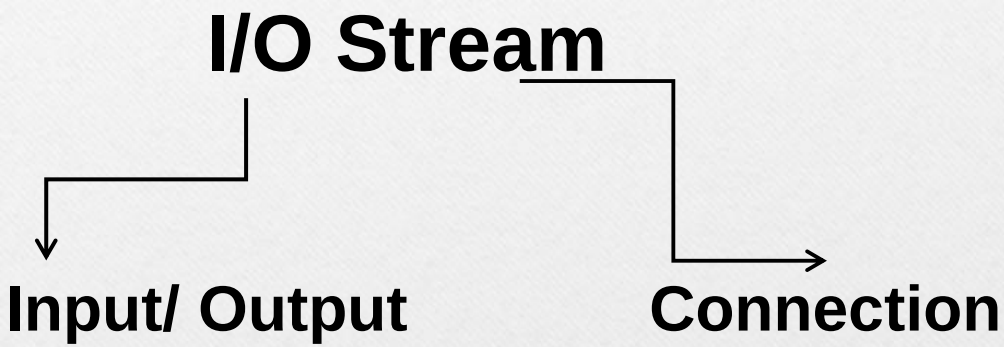
- I/O Stream.
- Writing and reading data from a file.
- Serialization and Deserialization.

I/O stream why used?

- Store data permanently
- Data available or not?
- Persistence operation of a file

I/O Stream (Java.io)

Not only performing file I/O operation but also contain classes for reading and writing data from **keyboard** / **console.**



I/O Streams

Stream: an object that either delivers data to its destination (screen, file, etc.) or that takes data from a source (keyboard, file, etc.)

it acts as a buffer between the data source and destination

Input stream: a stream that provides input to a program

`System.in` is an input stream

Output stream: a stream that accepts output from a program

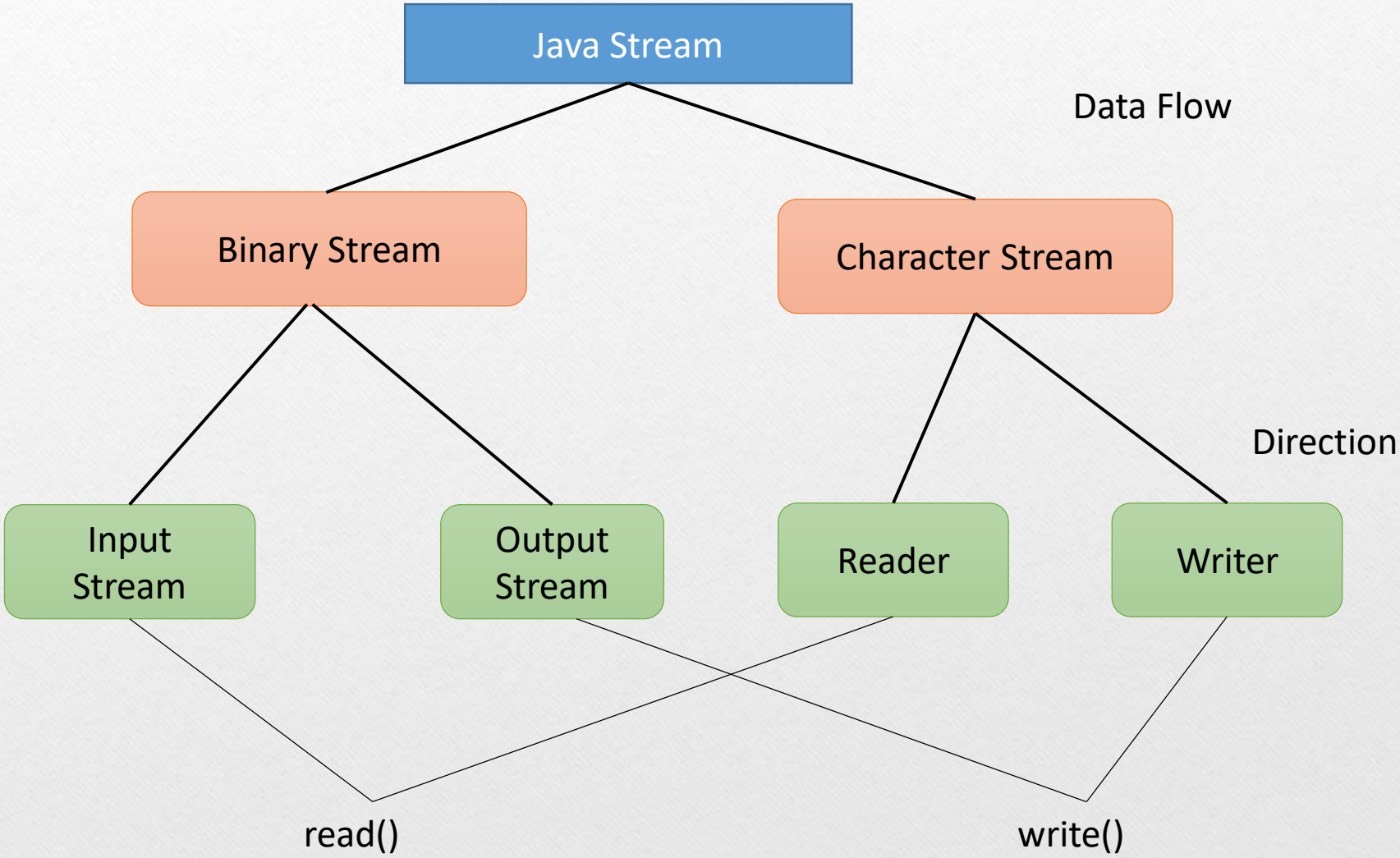
`System.out` is an output stream

A stream connects a program to an I/O object

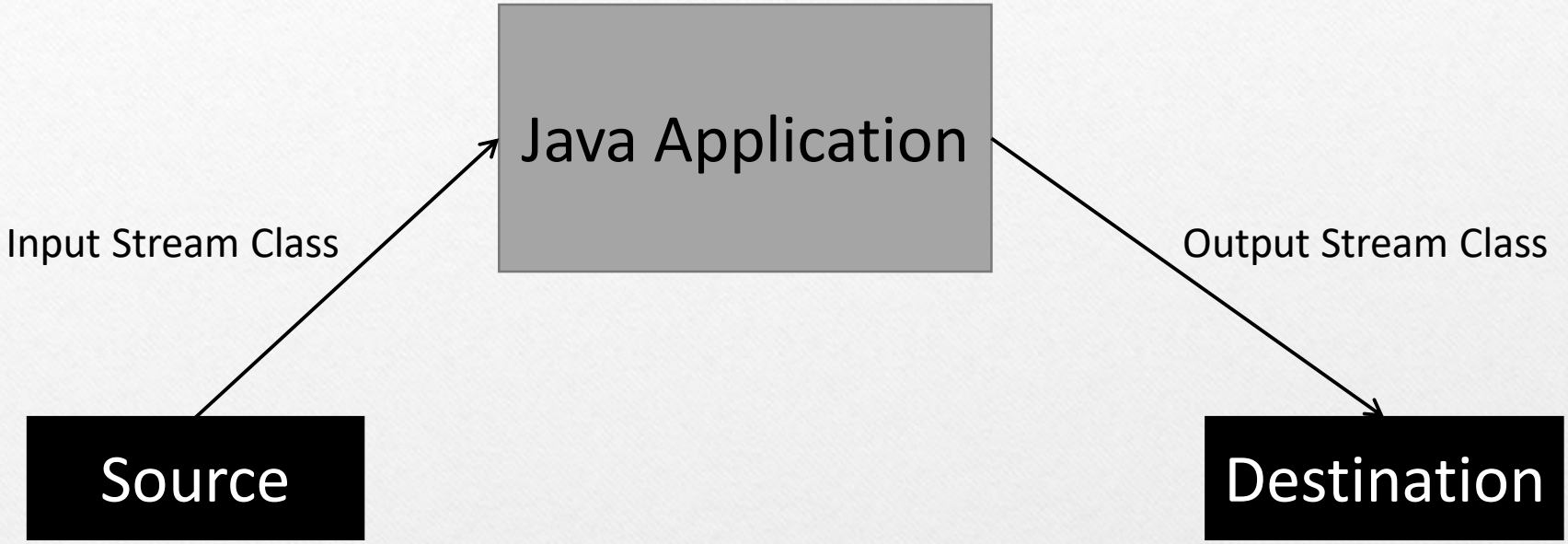
`System.out` connects a program to the screen

`System.in` connects a program to the keyboard

I/O Streams & Types of Streams



Input Stream Class



Keyboard, mouse, file, socket, Database, object, array, string, stringbuffer etc.

Monitor(console), file, socket, Database, object, array, String, Stringbuffer etc.

Input Stream Class: Subclasses

- Input stream class is a **abstract class**.
- 9 methods are there.
- Changing operation is only “Read” remaining all operation logic is same.

Input Stream Class: Subclasses

FileInputStream

ByteArrayInputStream

FilterInputStream

- **Data Input Stream**
- Buffered Input Stream

ObjectInputStream

Piped Input Stream

Sequence Input Stream

String Buffer Input Stream

Input Stream Class: Subclasses

- Input stream is connection based input stream class.
- Need to close the connection after completion.
- Special interface 'closeable' is used.
- Input stream class is a implementation class of *closeable* interface.
- *Closeable* interface has special method called *close.*

- In Java 1.7 version, special concept called 'Try with resource' is used.
  - Call the close method automatically.
  - A special interface called 'auto closeable' is used to support the try with resource.
  - Auto closeable have a special method 'close' which is inherited to closeable interface.

Input Stream Class: Subclasses

- Data Input & object input interface are the special input interface.
- Object input interface is a sub interface of data input interface.
- DataInputStream is a implementation class of a data input interface.
- ObjectInputStream is a implementation class of a object input interface.

Output Stream Class: Subclasses

- Output stream class is a **abstract class**.
- 8 methods are there.
- Changing operation is only “Write” remaining all operation logic is same.

Output Stream Class: Subclasses

FileOutputStream

ByteArrayOutputStream

FilterOutputStream

- **Data Output Stream**
- Buffered Output Stream
- **Print Stream**

ObjectOutputStream

Piped Output Stream

Output Stream Class: Subclasses

- Data Output & object Output interface are the special output interface.
- Object output interface is a sub interface of data output interface.
- DataOutputStream is a implementation class of a data output interface.
- ObjectOutputStream is a implementation class of a object output interface.
- Another special interface **flushable** is used.

Output & Input Stream Class: Subclasses

- File Input/Output Stream Class
- Data Input/Output Stream Class
- Object Input/Output Stream Class
- Sequence Input Class
- Print Stream Class

Input Stream Class: Methods

- Public int available() throws IOException
- Public void close() throws IOException
- Public abstract int read() throws IOException
- Public int read (byte[] b) throws IOException
- Public int read (byte[] b, int offset, int len) throws IOException
- Public boolean markSupported()
- Public void mark (int readlimit)
- Public void reset () throws IOException
- Public long skip (long n) throws IOException

Output Stream Class: Methods

- Public abstract void write(int b) throws IOException
- Public void write (byte[] b)

Writing Data into a File

- Add import java.io.* statement
- Create FileOutputStream object
- Invoke fos.write(data) [*Store data*]
- Invoke fos.flush() [*Flush data into file*]
- Invoke fos.close() [*Close fos connection*]
- Handle exceptions

Reading Data from a File

- Add import java.io.* statement
- Create FileInputStream object
- Invoke fis.read() [*Read data*]
- Invoke fis.close() [*Close fis connection*]
- Handle exceptions

File Output Stream Class and its constructor

- To create file output stream object, we need to create file output stream constructor.
- In file output stream class there is no null parameter constructor.
- Parameterized constructor must accept file name as input.

```
FileOutputStream(String name)
                    throws FileNotFoundException
```

```
FileOutputStream(File file)
                    throws FileNotFoundException
```

```
FileOutputStream(FileDescriptor file)
```

File Output Stream Class and its constructor

- File output stream constructor not supporting appending operation, but supporting overriding.

FileOutputStream(String name, boolean append)
throws FileNotFoundException

FileOutputStream(File file, boolean append)
throws FileNotFoundException

File Input Stream Class and its constructor

- To create file output stream object, we need to create file input stream constructor.
- In file input stream class there is no null parameter constructor.
- Parameterized constructor must accept file name as input.

```
FileInputStream(String name)
                throws FileNotFoundException
```

```
FileInputStream(File file)
                throws FileNotFoundException
```

```
FileInputStream(FileDescriptor file)
```

Reading Data From a File

Steps in reading data:

- Add import java.io.*
- Handle FNFE and IOException
- Create FIS class object by passing file name as argument
- Invoke fis.read() method and assign it to int type variable
- Display or use the byte data return from the file
- Close stream after usage by using the method fis.close()

Reading Data From a File (Example)

Writing Data To a File

Steps in writing data:

- Add import java.io.*
- Handle FNFE and IOException
- Create FOS class object by passing file name as argument
- Invoke fis.write(data) method
- Close stream after usage by using the method fos.close()

Writing Data To a File (Example)

File Copy Program (Example)

Serialization and Deserialization

- **Serialization in Java** is a mechanism of *writing the state of an object into a byte-stream*
- The reverse operation of serialization is called *deserialization* where byte-stream is converted into an object.
- Any class that is trying to implement serialization or deserialization concept should implement the interface called 'java.io.Serializable' or 'java.io.Externalizable' interface.
- This process is platform-independent.
- For serializing the object, we call the **writeObject()** method of *ObjectOutputStream class*
- For deserialization we call the **readObject()** method of *ObjectInputStream* class.

Serialization and Deserialization (Example)

Transient Keyword

- Data member defined as transient, will not be serialized.

Example:

Thank You