

NS4307

Network Programming

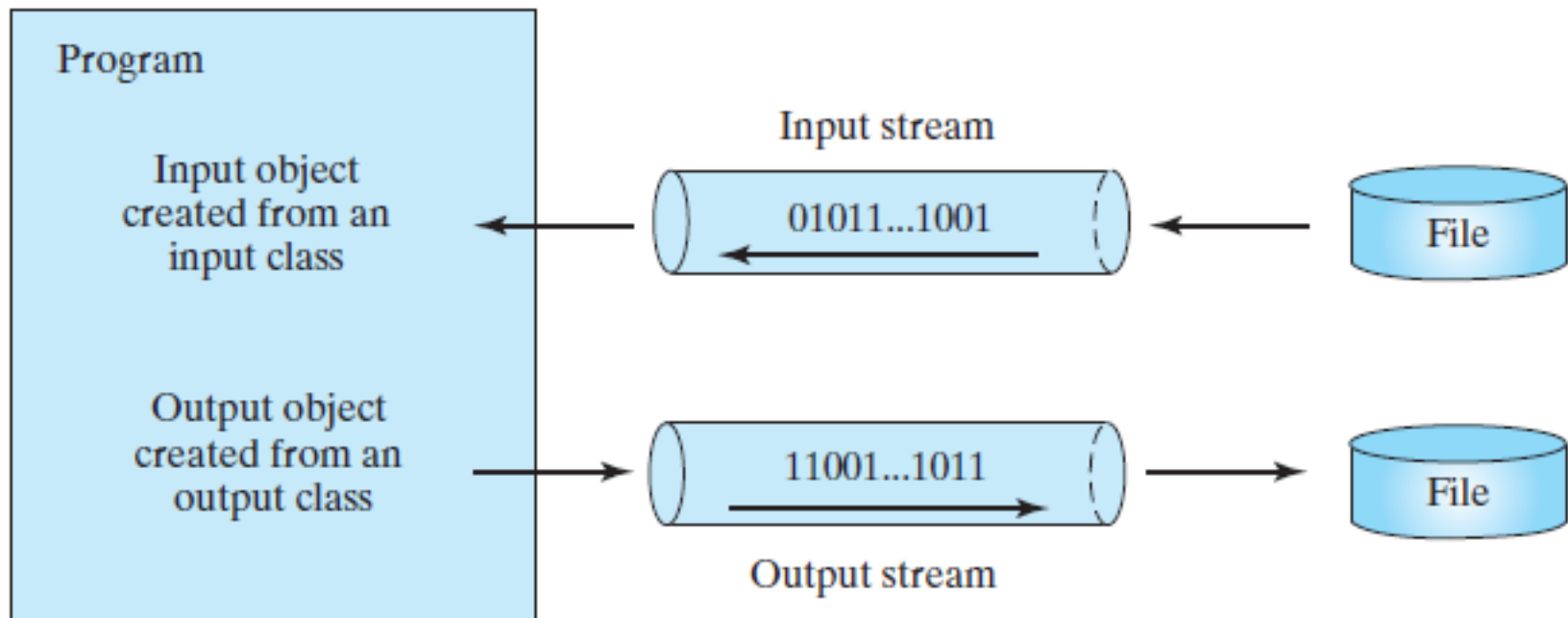
Java Input & Output (IO)

Introduction of Java I/O

- A large part of what network programs do is **simple input and output: moving bytes from one system to another.**
- Sending text to a client is not that different from writing a file.
- Java offers many classes for performing input and output where you need to create objects using appropriate Java I/O classes.

Introduction to Java I/O (cont.)

- I/O in Java is built on stream.
 - Input object **reads** a **stream of data** from a **file**
 - Output object **writes** a **stream of data** to a **file**

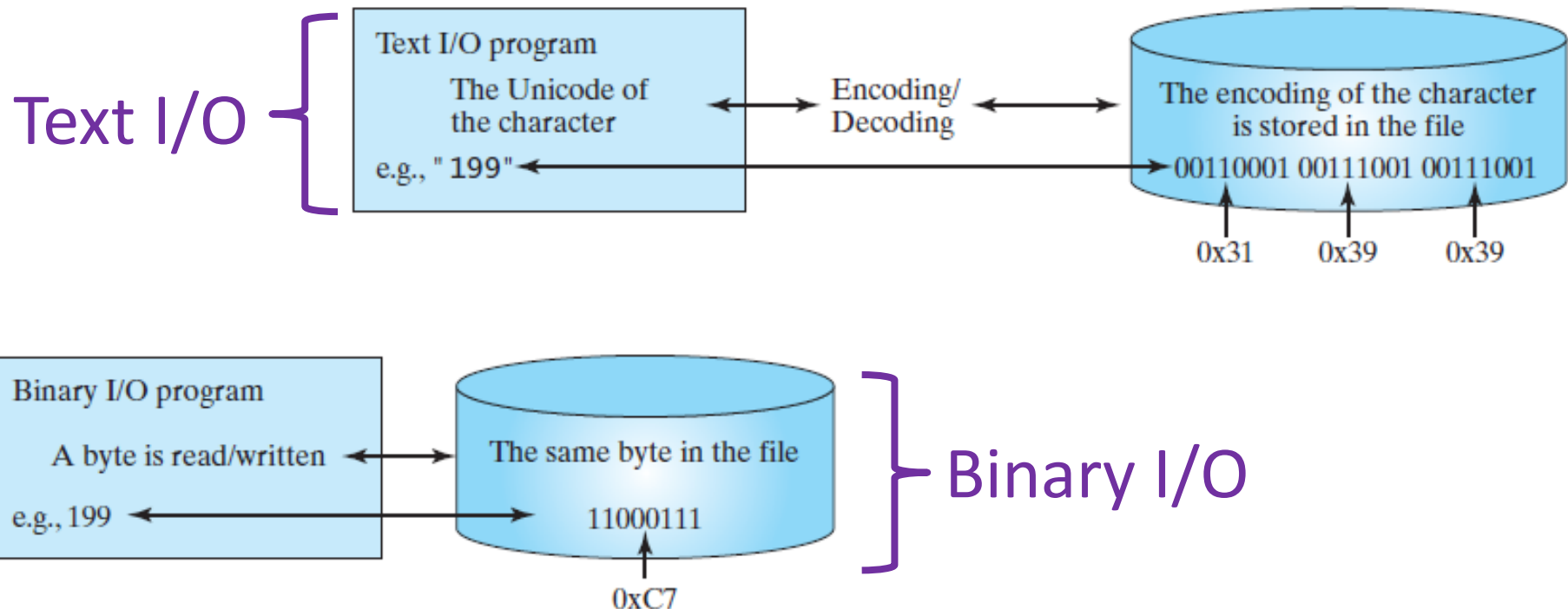


Java I/O Categories

- Java I/O can be categorised as:
 - text I/O classes
 - binary I/O classes
- We won't cover text I/O classes since you already learned them in Programming Application module.
- We will learn binary I/O classes.

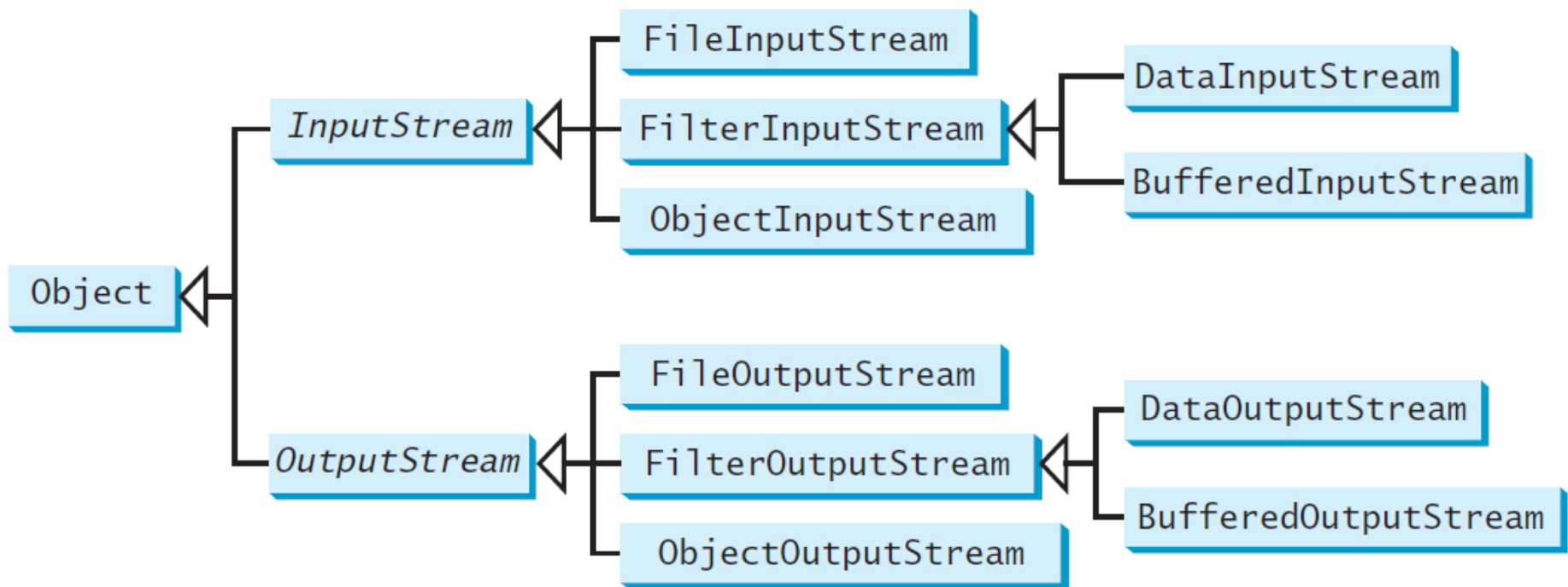
Binary I/O

- Binary I/O is **more efficient** than text I/O because it does not involve encoding or decoding the file.



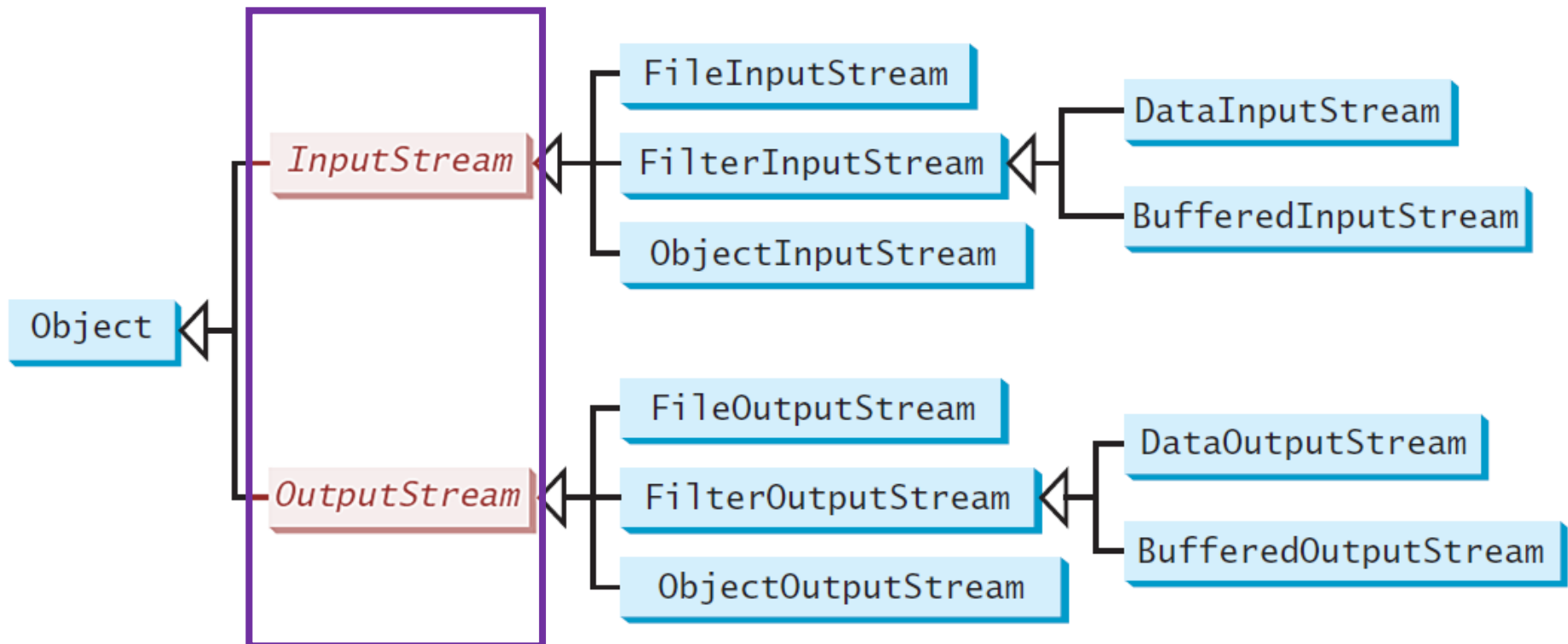
Binary I/O Classes

- InputStream, OutputStream, and their subclasses are for **performing binary I/O**.



InputStream/OutputStream

- Before going through the subclasses, you need to know the methods in InputStream and OutputStream class.



InputStream/OutputStream (cont.)

- InputStream/OutputStream are **abstract classes** where you **CANNOT** instantiate. Example:

```
InputStream input = new InputStream();  
OutputStream output = new OutputStream();
```

ERROR!



- You need to **instantiate into its subclasses**. Example:

```
InputStream input = new FileInputStream(...);  
OutputStream output = new FileOutputStream(...);
```


InputStream

- InputStream class provides **the fundamental methods** needed to read data as raw bytes.

Modifier and Type	Method and Description
int	available() Returns an estimate of the number of bytes that can be read (or skipped over) from this input stream without blocking by the next invocation of a method for this input stream.
void	close() Closes this input stream and releases any system resources associated with the stream.
abstract int	read() Reads the next byte of data from the input stream.
int	read(byte[] b) Reads some number of bytes from the input stream and stores them into the buffer array b.
int	read(byte[] b, int off, int len) Reads up to len bytes of data from the input stream into an array of bytes.
long	skip(long n) Skips over and discards n bytes of data from this input stream.

OutputStream

- OutputStream class provides the **fundamental methods** needed to write data.

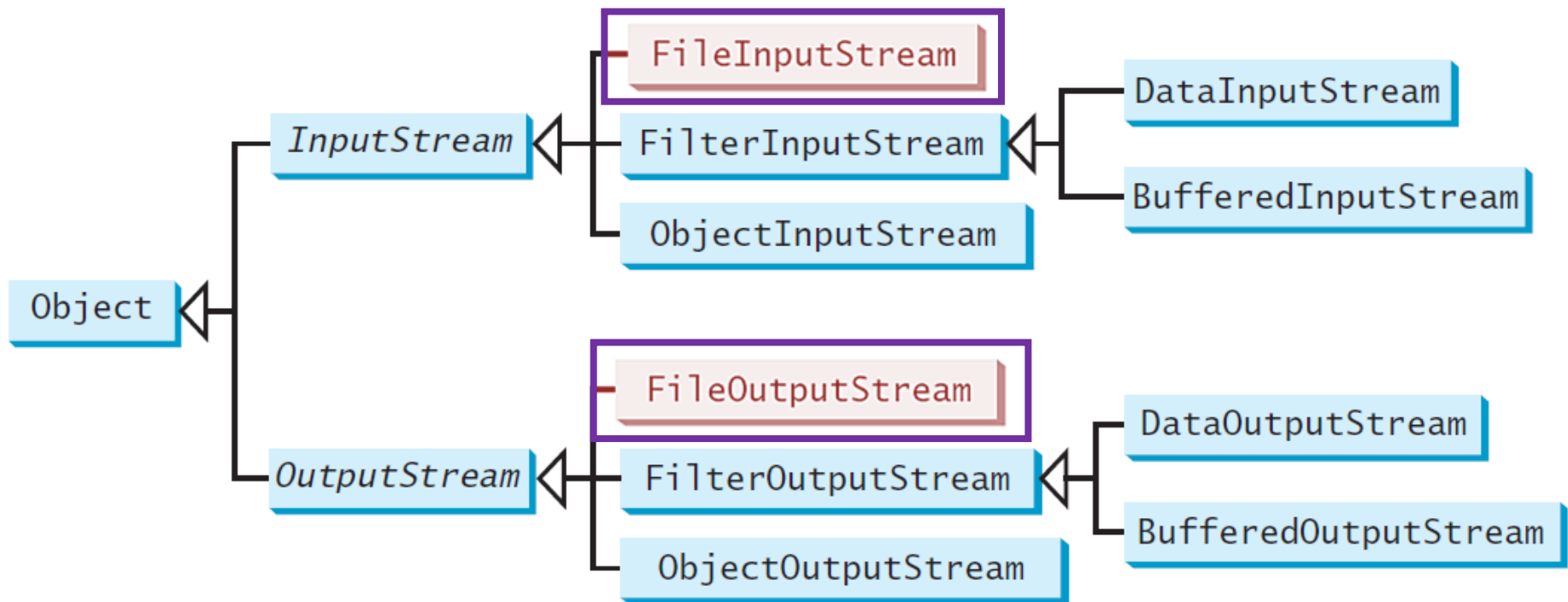
Modifier and Type	Method and Description
void	close() Closes this output stream and releases any system resources associated with this stream.
void	flush() Flushes this output stream and forces any buffered output bytes to be written out.
void	write(byte[] b) Writes b.length bytes from the specified byte array to this output stream.
void	write(byte[] b, int off, int len) Writes len bytes from the specified byte array starting at offset off to this output stream.
abstract void	write(int b) Writes the specified byte to this output stream.

InputStream/OutputStream (cont.)

- All methods stated **throws IOException**, you need to catch this using try and catch statement.
- You can access more in details on the classes and methods.
 - **InputStream:**
<https://docs.oracle.com/javase/8/docs/api/java/io/InputStream.html>
 - **OutputStream:**
<https://docs.oracle.com/javase/8/docs/api/java/io/OutputStream.htm>
!

FileInputStream/ FileOutputStream

- FileInputStream/FileOutputStream is a **subclass** for InputStream/OutputStream respectively.



FileInputStream/ FileOutputStream (cont.)

- FileInputStream/FileOutputStream is for **reading/writing bytes from/to files**.
- All methods in these classes **are inherited** from InputStream/OutputStream and it **does not introduce new methods**.

FileInputStream

- To **create an instance** of `FileInputStream`, there are two constructors available:

Constructor	Description
<code>FileInputStream(File file)</code>	Creates a <code>FileInputStream</code> from a <code>File</code> object.
<code>FileInputStream(String filename)</code>	Creates a <code>FileInputStream</code> from a file name.

- Example:

```
FileInputStream input = new FileInputStream(new File("path/testing.dat"));  
FileInputStream input = new FileInputStream("path/testing.dat");
```

- A **`FileNotFoundException`** will occur if you attempt to **create a `FileInputStream` with a nonexistent file**.
- <https://docs.oracle.com/javase/8/docs/api/java/io/FileInputStream.html>

FileOutputStream

- To **create an instance** of FileOutputStream, there are four constructors available:

Constructor	Description
<code>FileOutputStream(File file)</code>	Creates a FileOutputStream from a File object.
<code>FileOutputStream(String filename)</code>	Creates a FileOutputStream from a file name.
<code>FileOutputStream(File file, boolean append)</code>	Creates a FileOutputStream from a File object. If append is true, data are appended to the existing file.
<code>FileOutputStream(String filename, boolean append)</code>	Creates a FileOutputStream from a file name. If append is true, data are appended to the existing file.

- Example:

```
FileOutputStream output = new FileOutputStream("path/testing.dat");
```

```
FileOutputStream output = new FileOutputStream("path/testing.dat", true);
```

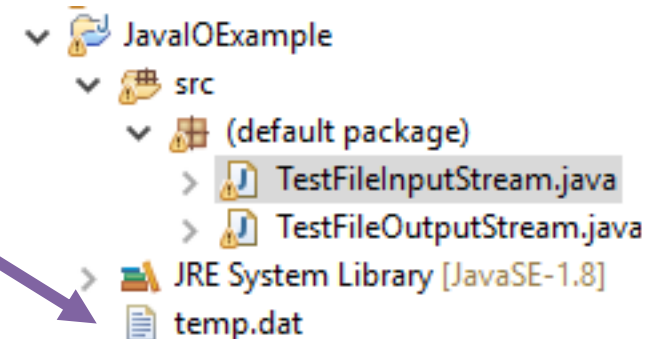
FileOutputStream (cont.)

- If the file does not exist, a new file will be created. If the file already exists, the first two constructors will delete the current content of the file.
- To retain the current content and append new data into the file, use the last two constructors and pass **true** to the **append** parameter.
- <https://docs.oracle.com/javase/8/docs/api/java/io/FileOutputStream.html>

Example:

TestFileOutputStream.java

- Lets create a **TestFileOutputStream.java**
- This program will **write Integer from 1 to 10 into a file “temp.dat”**.
- **Note:** This will save the file in Project Folder. You can specify the directory for your file if you want to save it at different folder.
 - “directory/temp.dat”




Example:

TestFileOutputStream.java (cont.)

```
public class TestFileOutputStream {  
  
    public static void main(String[] args) throws IOException {  
        try(FileOutputStream output = new FileOutputStream("temp.dat")) {  
            for(int i = 1; i <= 10; i++) {  
                output.write(i);  
            }  
        }  
    }  
}
```

content of temp.dat file (binary data) which cannot be read in text mode. The content might look differently for different text editor.



SOH STX ETX EOT ENQ ACK BEL BS

Example:

TestFileInputStream.java

- Lets create a **TestFileInputStream.java**
- This program will **read Integer from the file “temp.dat”** that we created from previous example.
- **Note:** This will read the file in Project Folder. You can specify the directory for your file if you want to read if it is at different folder.
 - “directory/temp.dat”

Example:

TestFileInputStream.java (cont.)

```
public class TestFileInputStream {  
  
    public static void main(String[] args) throws IOException {  
        try (FileInputStream input = new FileInputStream("temp.dat")) {  
            int value;  
            while((value = input.read()) != -1) {  
                System.out.println(value + " ");  
            }  
        }  
    }  
}
```

Example:

TestFileInputStream.java (cont.)

the output of the
application

```
<terminated> TestFileInputStream [Java Application]
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

Closing the stream

- The program uses the **try-with-resources** to declare and create input and output streams so that they will be automatically closed after they are used.
- The **InputStream** and **OutputStream** classes implement the **AutoClosable** interface where it defines the **close()** method that closes a resource.
- Any object of the **AutoClosable** type can be used with the **try-with-resources** syntax for automatic closing.



Closing the stream (cont.)

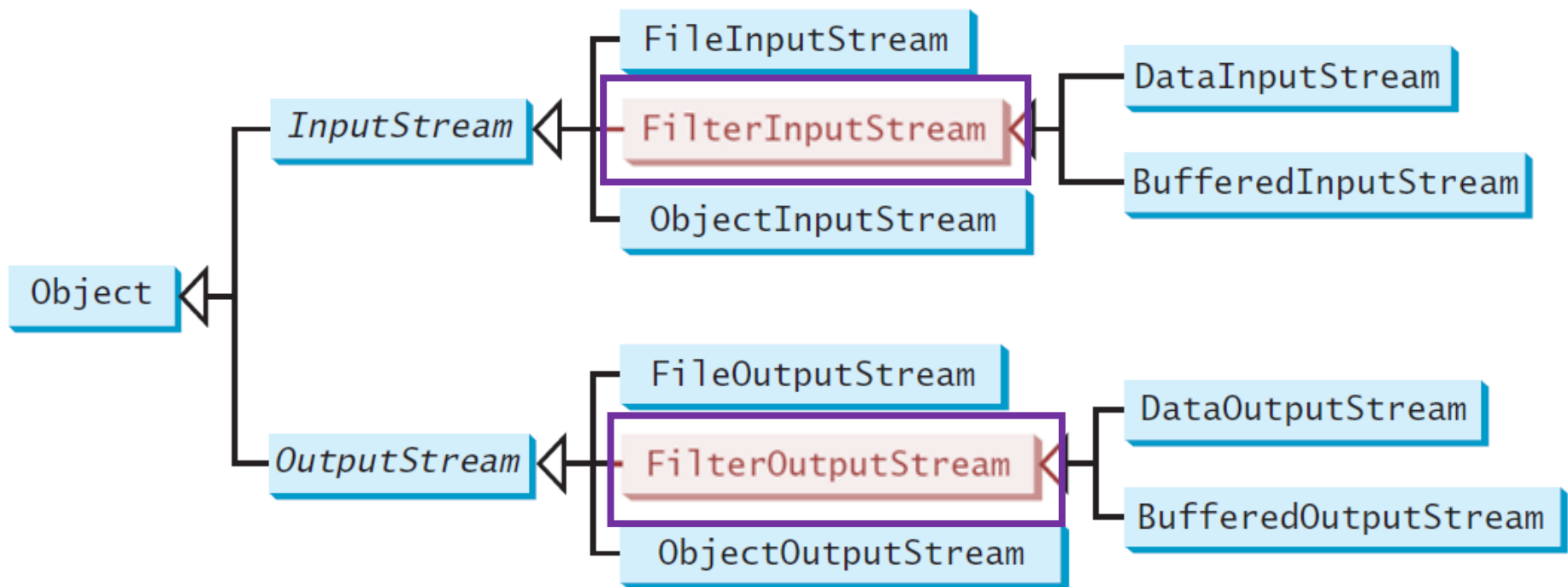
- **Note:** When a stream is no longer needed, always close it using the close() method or automatically close it using a try-with-resource statement. **Not closing streams may cause data corruption in the output file, or other programming errors.**

Example of not
using try-with-
resource
statement

```
public class TestFileOutputStream {  
  
    public static void main(String[] args) throws IOException {  
        FileOutputStream output = new FileOutputStream("temp.dat");  
        try {  
            for(int i = 1; i <= 10; i++) {  
                output.write(i);  
            }  
        } finally {  
            if(output != null) { output.close(); }  
        }  
    }  
}
```

FilterInputStream/ FilterOutputStream

- FilterInputStream/FilterOutputStream is a **subclass** for InputStream/OutputStream respectively.



FilterInputStream/ FilterOutputStream (cont.)

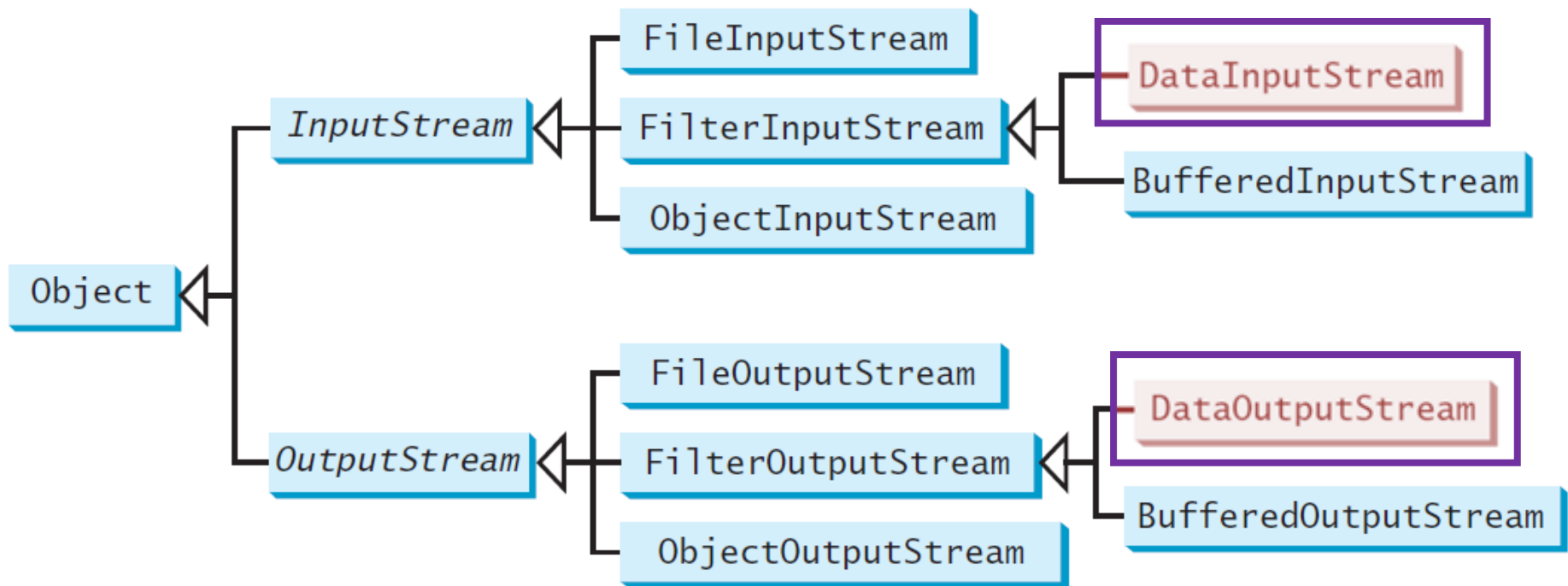
- Filter streams are streams that filter bytes for some purpose.
- The basic byte input stream provides a read method that can be used only for reading bytes.
 - If you want to read integers, doubles, or strings, you need a filter class to wrap the byte input stream.
 - Using a filter class enables you to read integers, doubles, and strings instead of bytes and characters.

FilterInputStream/ FilterOutputStream (cont.)

- FilterInputStream and FilterOutputStream are the base classes for filtering data.
- When you need to process primitive numeric types, use DataInputStream and DataOutputStream to filter bytes.

DataInputStream/ DataOutputStream

- DataInputStream/DataOutputStream is a **subclass** for FilterInputStream/FilterOutputStream respectively.



DataInputStream/ DataOutputStream (cont.)

- DataInputStream **reads bytes from the stream and converts them into appropriate primitive-type values or strings.**
 - It implements the methods defined in the DataInput Interface to read primitive data-type values and strings.
- DataOutputStream **converts primitive-type values or strings into bytes and outputs the bytes to the stream.**
 - It implements the methods defined in the DataOutput interface to write primitive data-type values and strings.
 - Primitive values are copied from memory to the output without any conversions.

DataInputStream

- The following are the methods implemented from DataInput interface.

Modifier and Type	Method and Description
boolean	readBoolean() Reads a boolean from the input stream.
byte	readByte() Reads a byte from the input stream.
char	readChar() Reads a character from the input stream.
float	readFloat() Reads a float from the input stream.
double	readDouble() Reads a double from the input stream.
int	readInt() Reads an int from the input stream.
long	readLong() Reads a long from the input stream.

DataInputStream (cont.)

Modifier and Type	Method and Description
short	readShort() Reads a short from the input stream.
String	readLine() Reads a line of characters from the input.
String	readUTF() Reads a string in UTF format.

<https://docs.oracle.com/javase/8/docs/api/java/io/InputStream.html>

DataOutputStream

- The following are the methods implemented from DataOutput interface.

Modifier and Type	Method and Description
void	writeBoolean(boolean b) Writes a Boolean to the output stream.
void	writeByte(int v) Writes the eight low-order bits of the argument v to the output stream.
void	writeBytes(String s) Writes the lower byte of the characters in a string to the output stream.
void	writeChar(char c) Writes a character (composed of 2 bytes) to the output stream.
void	writeChars(String s) Writes every character in the string s to the output stream, in order, 2 bytes per character.
void	writeFloat(float v) Writes a float value to the output stream.
void	writeDouble(double v) Writes a double value to the output stream.



DataOutputStream (cont.)

Modifier and Type	Method and Description
void	writeInt(int v) Writes an int value to the output stream.
void	writeLong(long v) Writes a long value to the output stream.
void	writeShort(short s) Writes a short value to the output stream.
void	writeUTF(String s) Writes s string in UTF format.

<https://docs.oracle.com/javase/8/docs/api/java/io/DataOutputStream.html>

Creating DataInputStream/ DataOutputStream

- DataInputStream/DataOutputStream implemented the following constructors.

```
public DataInputStream(InputStream instream)
```

```
public DataOutputStream(OutputStream outstream)
```

- The following is an example statements to create data streams.

```
DataInputStream input = new DataInputStream(new  
FileInputStream("in.dat"));
```

- This will creates an input stream for the file “in.dat”.

Creating DataInputStream/ DataOutputStream (cont.)

```
DataOutputStream output = new DataOutputStream(new  
    FileOutputStream("out.dat");
```

- This will creates an outputstream for the file **"out.dat"**.

Example:

TestDataOutputStream.java

```
public class TestDataOutputStream {  
  
    public static void main(String[] args) throws IOException {  
        try (DataOutputStream output = new DataOutputStream(new FileOutputStream("temp.dat"))); {  
            output.writeUTF("Jailani");  
            output.writeDouble(85.5);  
            output.writeUTF("Abdul");  
            output.writeDouble(70.3);  
            output.writeUTF("Rahman");  
            output.writeDouble(67.7);  
        }  
    }  
}
```

content of temp.dat file
(binary data) which cannot be
read in text mode. The
content might look differently
for different text editor.

1		0007	4a61	696c	616e	6940	5560	0000	0000
2		0000	0541	6264	756c	4051	9333	3333	3333
3		0006	5261	686d	616e	4050	eccc	cccc	cccd

Example:

TestDataInputStream.java

```
public class TestDataInputStream {  
  
    public static void main(String[] args) throws IOException {  
        try (DataInputStream input = new DataInputStream(new FileInputStream("temp.dat"));) {  
            System.out.println(input.readUTF() + " " + input.readDouble());  
            System.out.println(input.readUTF() + " " + input.readDouble());  
            System.out.println(input.readUTF() + " " + input.readDouble());  
        }  
    }  
}
```

the output of the
application

Jailani 85.5
Abdul 70.3
Rahman 67.7

DataInputStream/ DataOutputStream (cont.)

- DataInputStream and DataOutputStream read and write Java primitive-type values and strings in a machine-independent fashion.
- This will enable you to write a data file on one machine and read it on another machine that has a different operating system or file structure.
- An application uses a data output stream to write data that can later be read by a program using a data input stream.

DataInputStream/ DataOutputStream (cont.)



int, double, string ...

01000110011 ...



int, double, string ...

01000110011 ...

- Note: You have to read data in the order and format in which they are stored, since names are written in UTF-8 using **writeUTF**, you must read names using **readUTF**.

Detecting the End of a File

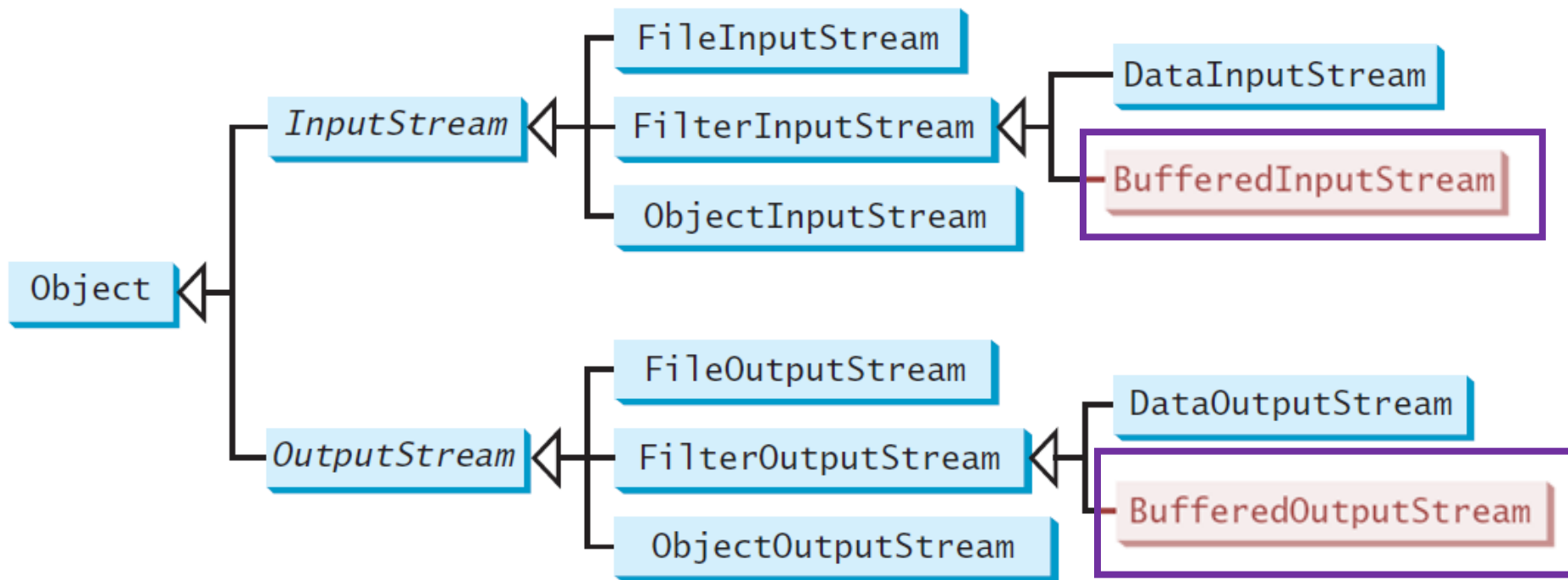
- If you keep reading data at the end of an **InputStream**, an **EOFException** will occur. This exception can be used to detect the end of a file.

```
public class TestEndOfFile {  
  
    public static void main(String[] args) {  
        try {  
            try (DataInputStream input = new DataInputStream(new FileInputStream("temp.dat"))) {  
                System.out.println(input.readUTF() + " " + input.readDouble());  
                System.out.println(input.readUTF() + " " + input.readDouble());  
                System.out.println(input.readUTF() + " " + input.readDouble());  
                System.out.println(input.readUTF() + " " + input.readDouble());  
            }  
        } catch (EOFException ex) {  
            System.out.println("End of File");  
        } catch (IOException ex) {  
            ex.printStackTrace();  
        }  
    }  
}
```

} EOFException is caught to indicate End of File.

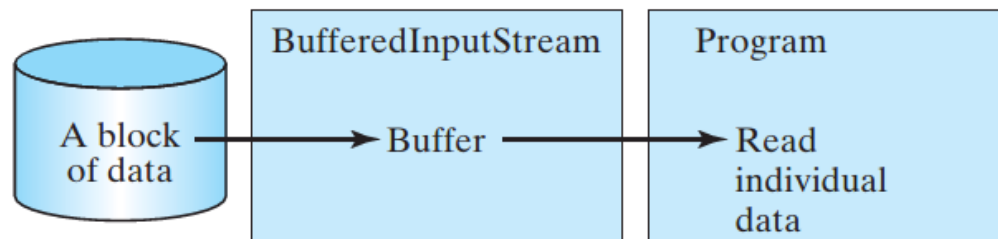
BufferedInputStream/ BufferedOutputStream

- BufferedInputStream/BufferedOutputStream is a **subclass** for FilterInputStream/FilterOutputStream respectively.



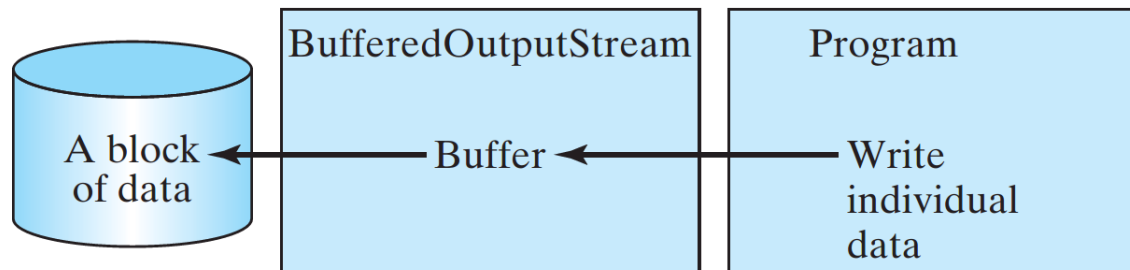
BufferedInputStream/ BufferedOutputStream (cont.)

- **BufferedInputStream/BufferedOutputStream** can be used to up input and output by **reducing the number of disk reads and writes**.
- **Using BufferedInputStream**, the whole block of data on the disk read into the buffer in the memory once.
 - The individual data are then delivered to your program from the buffer.



BufferedInputStream/ BufferedOutputStream (cont.)

- **Using BufferedOutputStream**, the individual data are first written to the buffer in the memory.
 - When the buffer is full, all data in the buffer are written to the disk once.



- BufferedInputStream/BufferedOutputStream **does not contain new methods.**

BufferedInputStream/ BufferedOutputStream (cont.)

- All methods in BufferedInputStream/BufferedOutputStream are inherited from InputStream/OutputStream classes.
- BufferedInputStream/BufferedOutputStream manages a **buffer behind the scene** and **automatically reads/writes data from/to disk** on demand.
- NOTE: You should always use buffered I/O to speed up input and output.
 - For small files, you may not notice performance improvements.
 - For large files (over 100MB), you will see substantial improvements using buffered I/O.

BufferedInputStream

- You can wrap a BufferedInputStream on any InputStream using the constructors below:

Constructor	Description
BufferedInputStream(InputStream in)	Creates a BufferedInputStream from an InputStream object.
BufferedInputStream(InputStream in, int bufferSize)	Creates a BufferedInputStream from an InputStream object with specified buffer size.

- Example:

```
DataInputStream input = new DataInputStream(new  
BufferedInputStream(new FileInputStream("temp.dat")));
```

<https://docs.oracle.com/javase/8/docs/api/java/io/BufferedInputStream.html>

BufferedOutputStream

- You can wrap a BufferedOutputStream on any OutputStream using the constructors below:

Constructor	Description
BufferedOutputStream(OutputStream out)	Creates a BufferedOutputStream from an OutputStream object.
BufferedOutputStream(OutputStream out, int bufferSize)	Creates a BufferedOutputStream from an OutputStream object with specified buffer size.

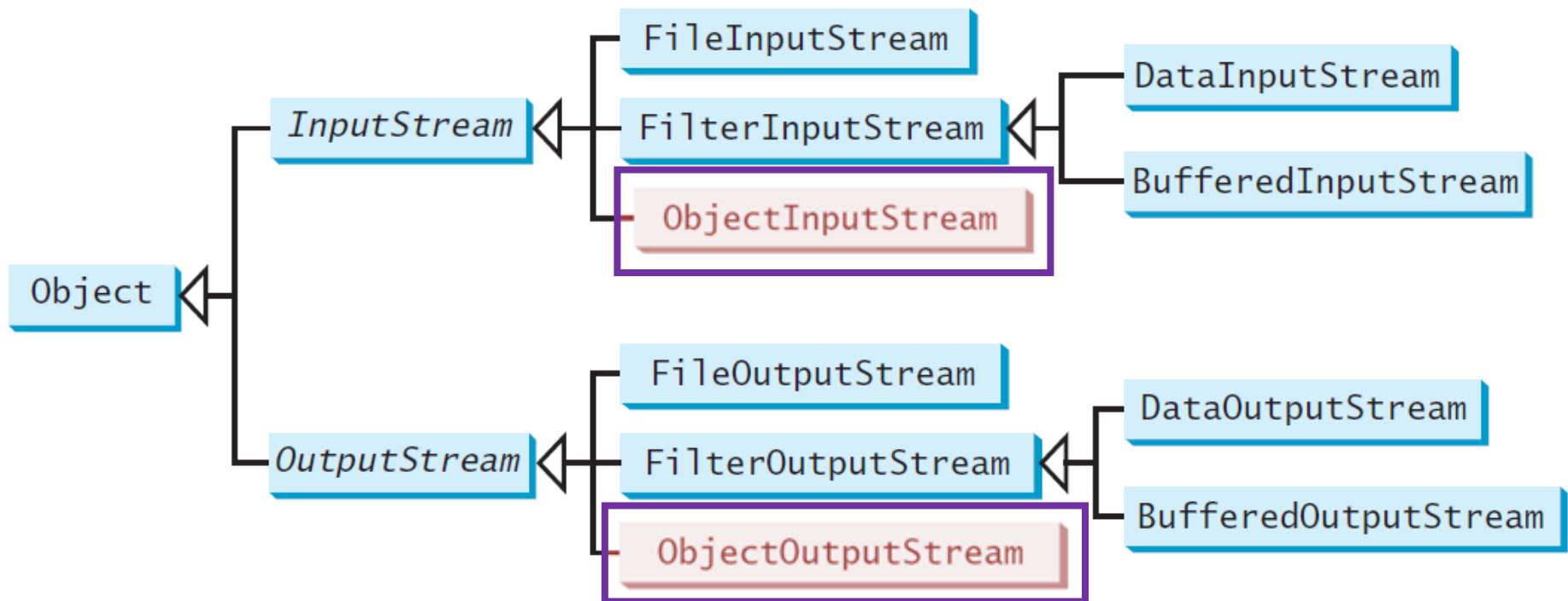
- Example:

```
DataOutputStream output = new DataOutputStream(new  
BufferedOutputStream(new FileOutputStream("temp.dat")));
```

<https://docs.oracle.com/javase/8/docs/api/java/io/BufferedOutputStream.html>

ObjectInputStream/ ObjectOutputStream

- ObjectInputStream/ObjectOutputStream is a **subclass** for InputStream/OutputStream respectively.



ObjectInputStream/ ObjectOutputStream (cont.)

- **ObjectInputStream/ObjectOutputStream** classes can be used to read/write serializable objects.
 - It enables you to perform I/O for objects in addition to primitive-type values and strings.
- Since **ObjectInputStream/ObjectOutputStream** contains all the functions of **DataInputStream/DataOutputStream**, you can replace DataInputStream/DataOutputStream completely with ObjectInputStream/ObjectOutputStream.

ObjectInputStream/ ObjectOutputStream (cont.)

- ObjectInputStream extends InputStream and implements ObjectInput and ObjectStreamConstants.
 - ObjectInput is a subinterface of DataInput.
- ObjectOutputStream extends OutputStream and implements ObjectOutput and ObjectStreamConstants.
 - ObjectOutput is a subinterface of DataOutput.

ObjectInputStream/ ObjectOutputStream (cont.)

- ObjectInputStream/ObjectOutputStream implemented the following constructors.

```
public ObjectInputStream(InputStream instream)
```

```
public ObjectOutputStream(OutputStream outstream)
```

- The following is an example statements to create object streams with buffer in the stream.

```
ObjectInputStream input = new ObjectInputStream(new  
BufferedInputStream(new FileInputStream("object.dat")));
```

- This will creates an input stream for the file **“object.dat”**.

Creating ObjectOutputStream/ ObjectOutputStream (cont.)

```
ObjectOutputStream output = new ObjectOutputStream(new  
BufferedInputStream(new FileOutputStream("object.dat")));
```

- This will creates an output stream for the file “**object.dat**”.

ObjectInputStream

- This class contains all the methods available to DataInputStream.
 - There is one (1) additional method available to ObjectInputStream.

Modifier and Type	Method and Description
Object	readObject() Reads an object.

- **This method may throw ClassNotFoundException**, because when the JVM restores an object, it first loads the class for the object if the class has not been loaded.

<https://docs.oracle.com/javase/8/docs/api/java/io/ObjectInputStream.html>

ObjectOutputStream

- This class contains all the methods available to DataOutputStream.
 - There is one (1) additional method available to ObjectOutputStream.

Modifier and Type	Method and Description
void	writeObject(Object o) Writes an object.

<https://docs.oracle.com/javase/8/docs/api/java/io/ObjectOutputStream.html>

Example:

TestObjectOutputStream

```
public class TestObjectOutputStream {

    public static void main(String[] args) throws FileNotFoundException, IOException {
        try(ObjectOutputStream output = new ObjectOutputStream(
            new BufferedOutputStream(new FileOutputStream("object.dat")))) {
            output.writeUTF("Jailani");
            output.writeDouble(85.5);
            output.writeObject(new Date());
            output.writeUTF("Rahman");
            output.writeDouble(100.0);
            output.writeObject(new Date());
        }
    }
}
```

content of object.dat file (binary data)
which cannot be read in text mode.
The content might look differently for
different text editor.

```
aced 0005 7711 0007 4a61 696c 616e 6940
5560 0000 0000 0073 7200 0e6a 6176 612e
7574 696c 2e44 6174 6568 6a81 014b 5974
1903 0000 7870 7708 0000 015a 636b 6593
7877 1000 0652 6168 6d61 6e40 5900 0000
0000 0073 7100 7e00 0077 0800 0001 5a63
6b65 9878 |
```

Example:

TestObjectInputStream

```
public class TestObjectInputStream {  
  
    public static void main(String[] args) throws FileNotFoundException, IOException, ClassNotFoundException {  
        try {  
            try(ObjectInputStream input = new ObjectInputStream(  
                new BufferedInputStream(new FileInputStream("object.dat")))) {  
                while(true) {  
                    String name = input.readUTF();  
                    double score = input.readDouble();  
                    Date date = (Date) input.readObject();  
                    System.out.println(name + " " + score + " " + date);  
                }  
            }  
        } catch (EOFException e) {  
            System.out.println("You have reached the end of file.");  
        }  
    }  
}
```

the output of the
application

```
Jailani 85.5 Wed Feb 22 09:24:27 SGT 2017  
Rahman 100.0 Wed Feb 22 09:24:27 SGT 2017  
You have reached the end of file.
```

Serializable Interface

- Not every object can be written to an output stream.
 - Objects that can be written are said to be serializable.
 - A serializable object is an instance of the Serializable interface, so the object's class must implement Serializable.
- The Serializable interface is a marker interface.
 - Since it has no methods, you don't need to add additional code in your that implements Serializable.
 - Implementing this interface enables the Java serialization mechanism to automate the process of storing objects and arrays.

Serializable Interface (cont.)

- Attempting to store an object that does not support the Serializable interface would cause a `NotSerializableException`.
- When a serializable object is stored, the class of the object is encoded.
 - This includes:
 - The class name
 - The signature of the class
 - The values of the object's instance variables
 - The closure of any other objects referenced by the object.
 - The values of the object's static variables are not stored.

Note

- **Noneserializable fields:** If an object is an instance of Serializable but contains nonserializable instance data fields, can it be serialized?
 - The answer is no.
- **Duplicate objects:** If an object is written to an object stream more than once, will it be stored in multiple copies?
 - No, it will not.
 - When an object is written for the first time, a serial number is created for it. The JVM writes the complete contents of the object along with the serial number into the object stream.