Object Oriented Programming in Java

Er.Sital Prasad Mandal

BCA- 2nd sem
Mechi Campus
Bhadrapur, Jhapa, Nepal
(Email : info.sitalmandal@gmail.com)
https://ctaljava.blogspot.com/

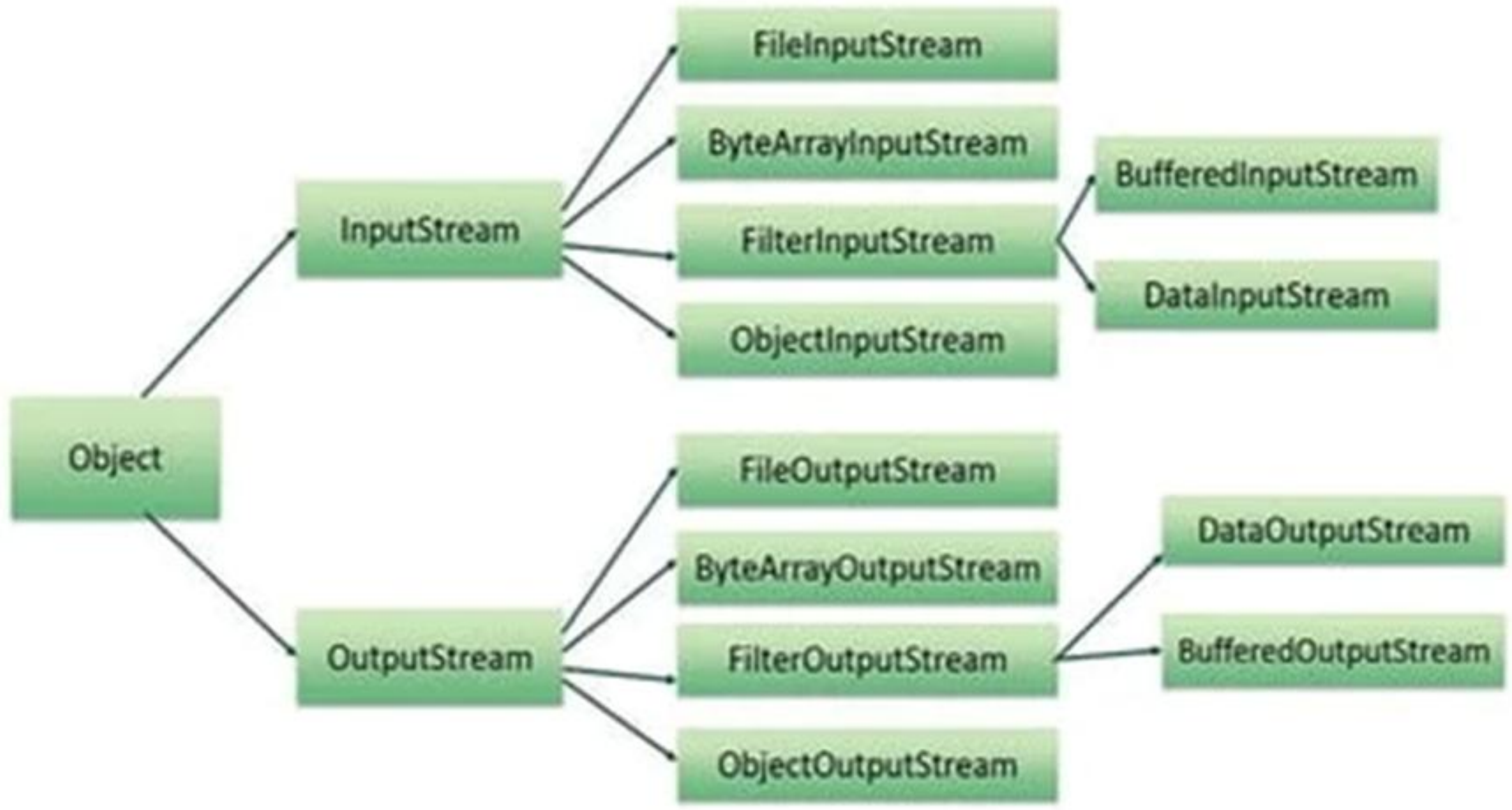
Text Book

1. Deitel & Dietel. -Java: How to-program-. 9th Edition. TearsorrEducation. 2011, ISBN: 9780273759168
2. Herbert Schildt. "Java: The CoriviaeReferi4.ic e 61 Seventh Edition. McGraw -Hill 2006, 1SBN; 0072263857

8. I/O and Streams

1. java.io package, Files and directories,
2. Streams and Character Streams;
3. Reading/Writing Console Input/Output,
4. Reading and Writing files,
5. The Serialization Interface,
6. Serialization & Deserialization.

Stream hierarchy



Input/Output in Java

● Programs read inputs from data sources (e.g., keyboard, file, network, memory buffer, or another program) and write outputs to data sinks (e.g., display console, file, network, memory buffer, or another program).

● Java I/O (Input and Output) is used to process the input and produce the output.

● Java uses the concept of stream to make I/O operation fast.

The java.io package contains all the classes required for input and output operations.

Stream

● A stream is a sequential and contiguous one-way flow of data (just like water or oil flows through the pipe).

● Java does not differentiate between the various types of data sources or sinks (e.g., file or network) in stream I/O. They are all treated as a sequential flow of data.

Java I/O (Input and Output)

● The Java program receives data from a source by opening an input stream, and sends data to a sink by opening an output stream. All Java I/O streams are one-way (except the RandomAccessFile).

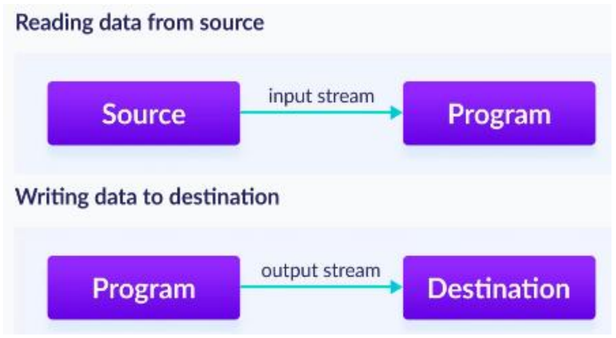
● If your program needs to perform both input and output, you have to open two streams - an input stream and an output stream.

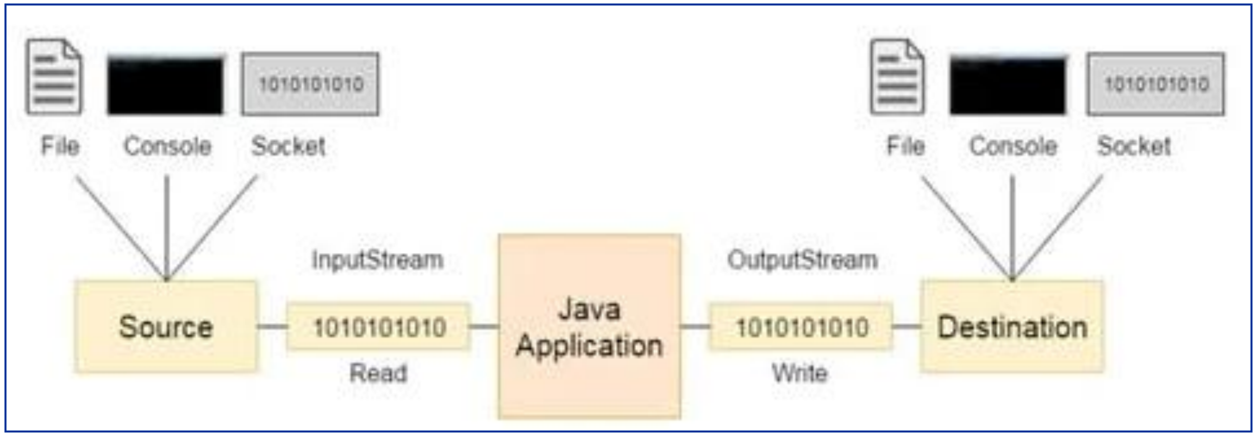
8. I/O and Streams

Stream

- Stream I/O operations involve three steps:

1. Open an input/output stream associated with a physical device (e.g., file, network, console/keyboard), by constructing an appropriate I/O stream instance.
2. Read from the opened input stream until "end-of-stream" encountered, or write to the opened output stream (and optionally flush the buffered output).
3. Close the input/output stream.





https://ctaljava.blogspot.com/

8. I/O and Streams

Stream

Types of Streams

Depending upon the data a stream holds, it can be classified into:

1. Byte Stream
2. Character Stream

Byte stream

- ❖ **Byte stream** is used to read and write a single byte (8 bits) of data.
- ❖ All byte stream classes are derived from base abstract classes at top of hierarchy, they are called **InputStream** and **OutputStream**.

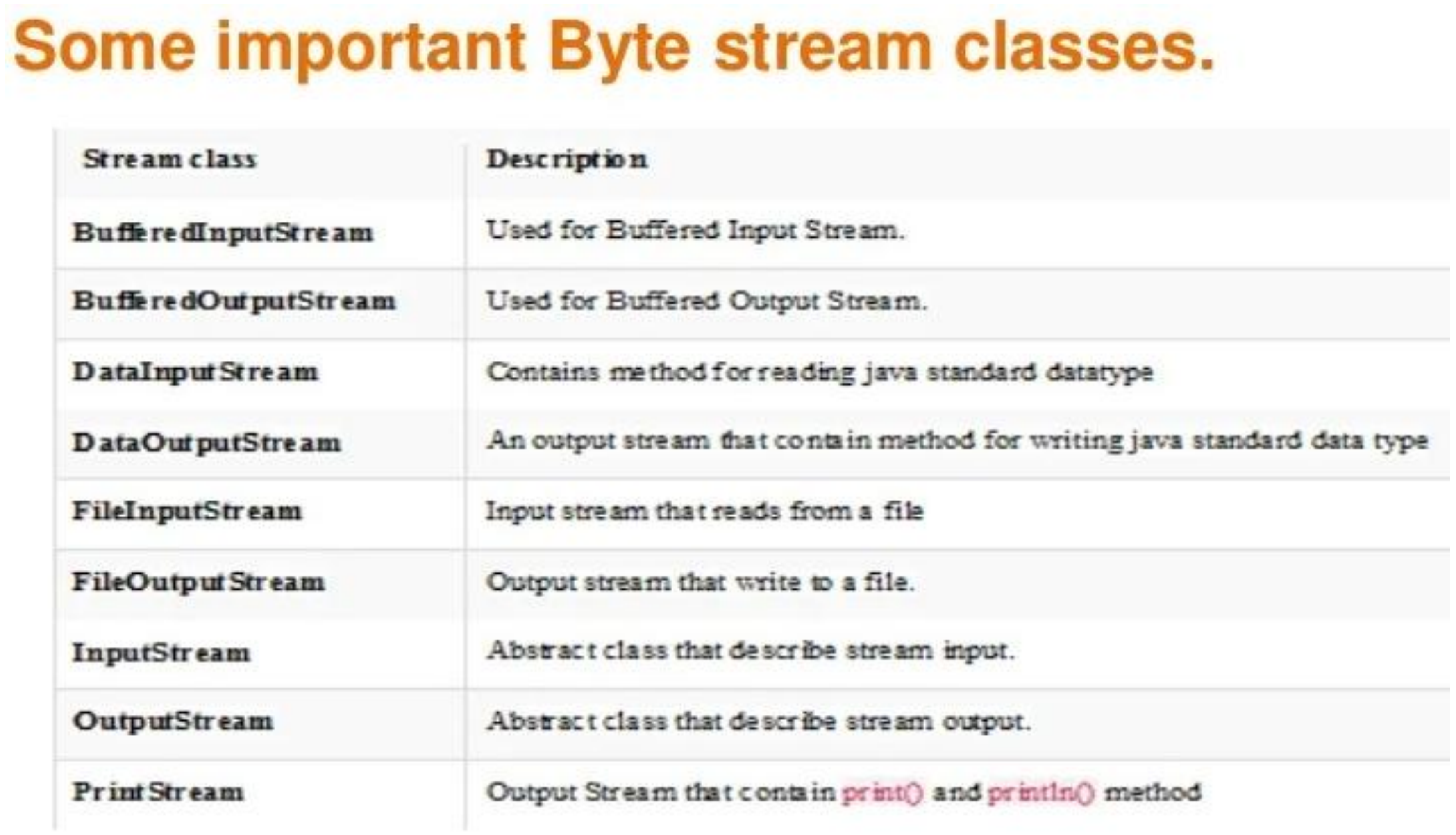
Two most important are:

- ○ **read():** reads byte of data.
- ○ **write():** writes byte of data.

Character Stream

- ❖ Character stream is used to read and write a single character of data.
- ❖ All the character stream classes are derived from base abstract classes at the top of hierarchy, they are **Reader** and **Writer**.

Stream

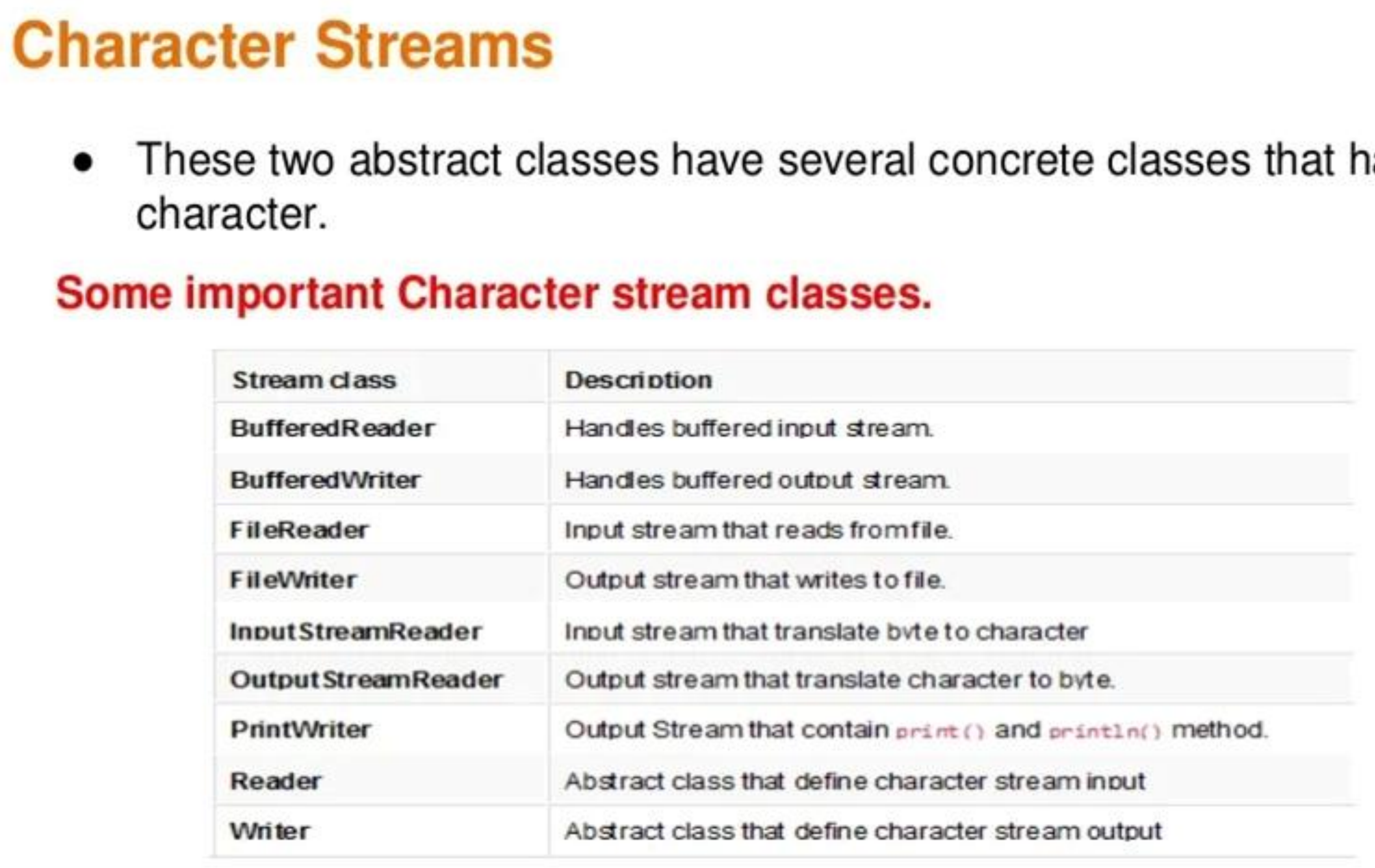


Some important Byte stream classes.

Stream class	Description
BufferedInputStream	Used for Buffered Input Stream.
BufferedOutputStream	Used for Buffered Output Stream.
DataInputStream	Contains method for reading java standard datatype
DataOutputStream	An output stream that contain method for writing java standard data type
FileInputStream	Input stream that reads from a file
FileOutputStream	Output stream that write to a file.
InputStream	Abstract class that describe stream input.
OutputStream	Abstract class that describe stream output.
PrintStream	Output Stream that contain print() and println() method

8. I/O and Streams

Stream



Character Streams

- These two abstract classes have several concrete classes that handle unicode character.

Some important Character stream classes.

Stream class	Description
BufferedReader	Handles buffered input stream.
BufferedWriter	Handles buffered output stream.
FileReader	Input stream that reads from file.
FileWriter	Output stream that writes to file.
InputStreamReader	Input stream that translate byte to character
OutputStreamReader	Output stream that translate character to byte.
PrintWriter	Output Stream that contain `print()` and `println()` method.
Reader	Abstract class that define character stream input
Writer	Abstract class that define character stream output

Assignment Streams

Stream A stream is a sequence of data.In Java a stream is composed of bytes.

A stream is a way of sequentially accessing a file and contiguous one- way flow of data. It's called a stream because it is like a stream of water or oil that continues to flows through the pipe.

There are two kinds of Streams:

OutputStream

Java application uses an output stream to write data to a destination, it may be a file, an array, peripheral device or socket.

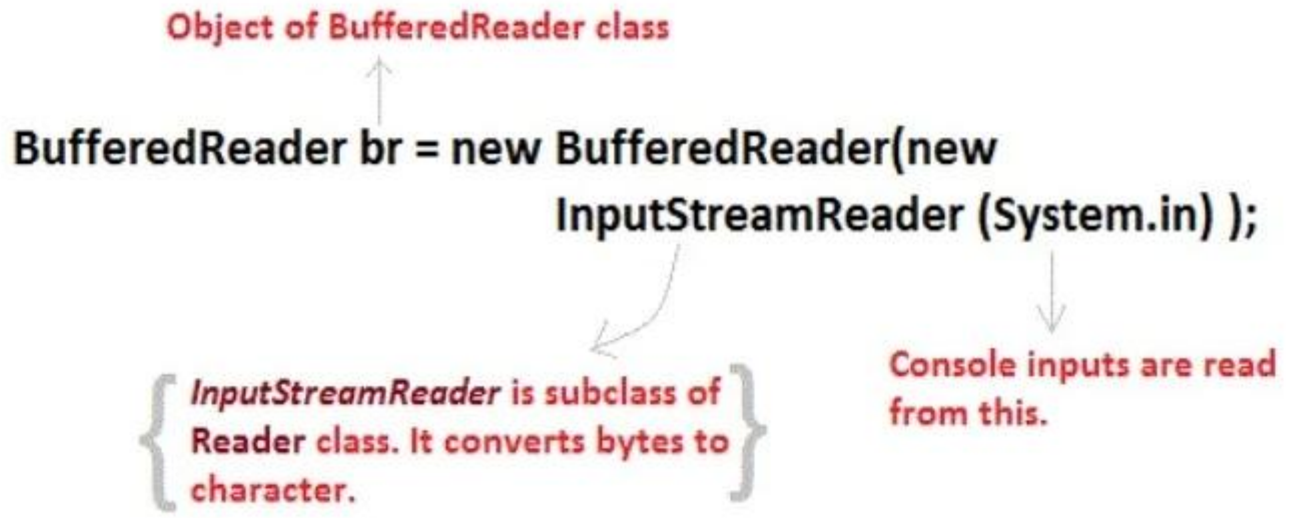
InputStream

Java application uses an input stream to read data from a source, it may be a file, an array, peripheral device or socket.

Reading/Writing Console Input/Output

Reading Console Input

- **Example**: We use the object of BufferedReader class to take inputs from the keyboard.



Object of BufferedReader class

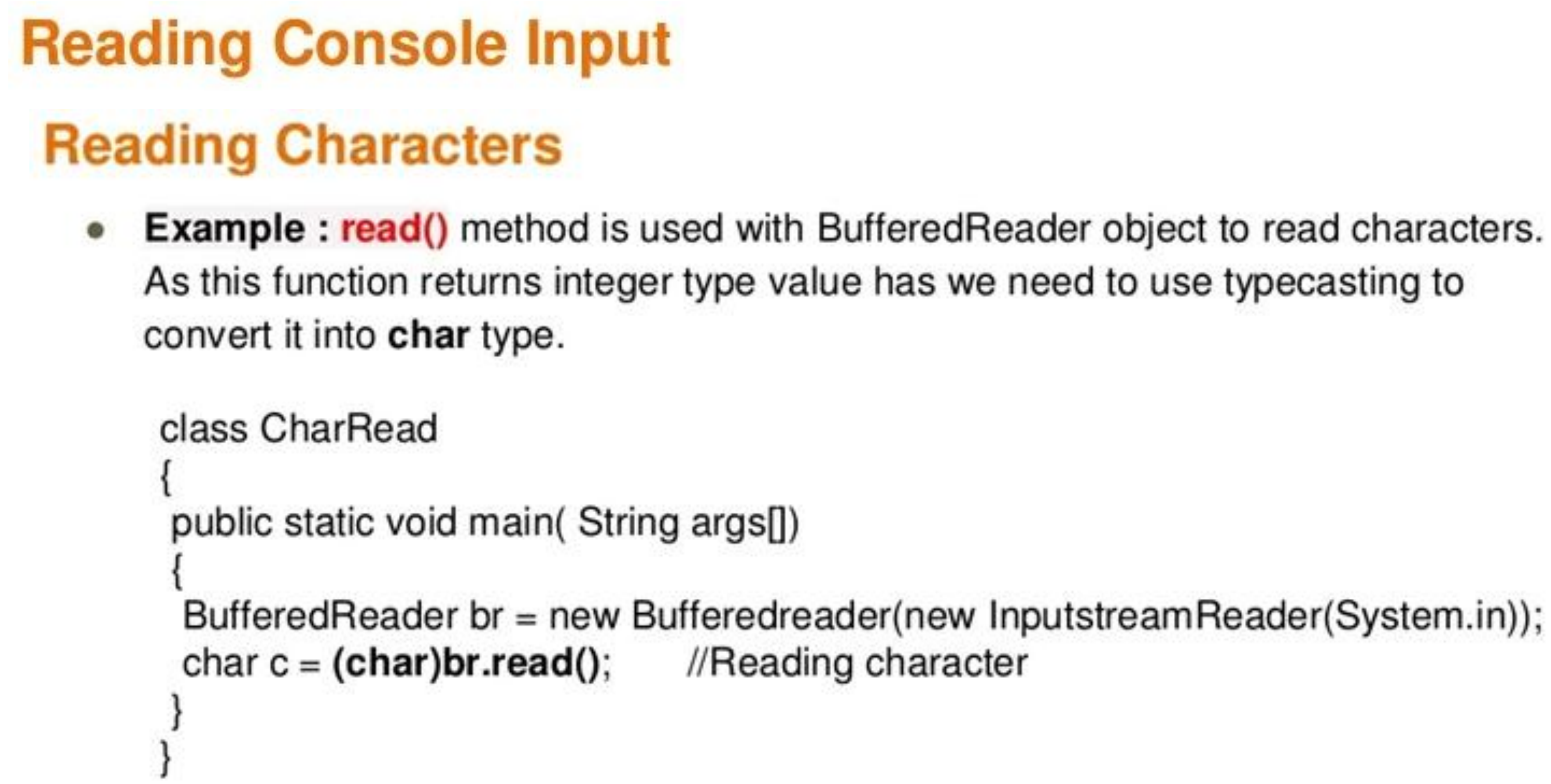
```
BufferedReader br = new BufferedReader(new
                        InputStreamReader (System.in) );
```

InputStreamReader is subclass of Reader class. It converts bytes to character.

Console inputs are read from this.

8. I/O and Streams

Reading/Writing Console Input/Output



Reading Console Input

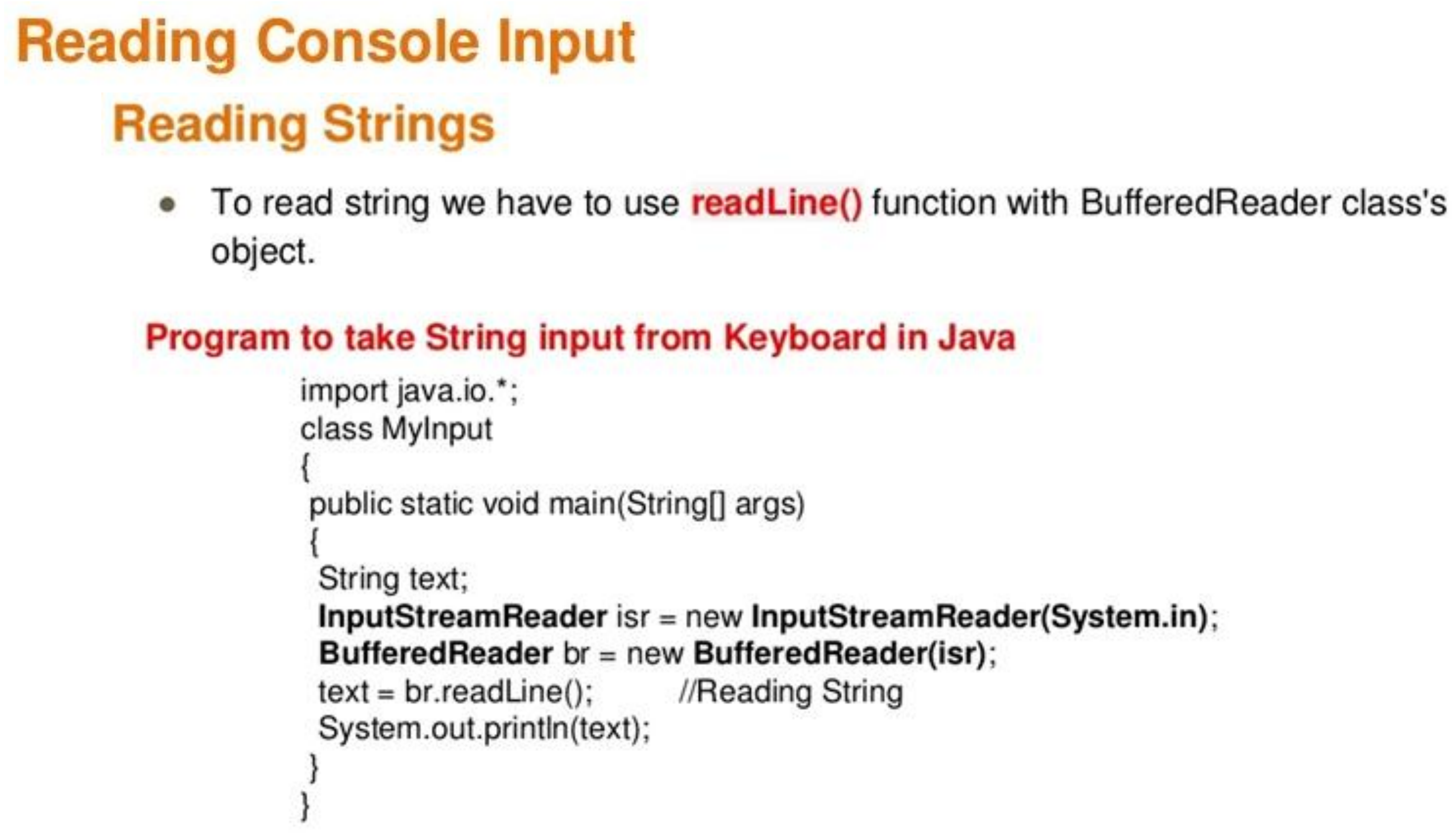
Reading Characters

- **Example :** read() method is used with BufferedReader object to read characters. As this function returns integer type value has we need to use typecasting to convert it into **char** type.

```
class CharRead
{
 public static void main( String args[])
 {
  BufferedReader br = new Bufferedreader(new InputstreamReader(System.in));
  char c = (char)br.read();       //Reading character
 }
}
```

8. I/O and Streams

Reading/Writing Console Input/Output



Reading Console Input

Reading Strings

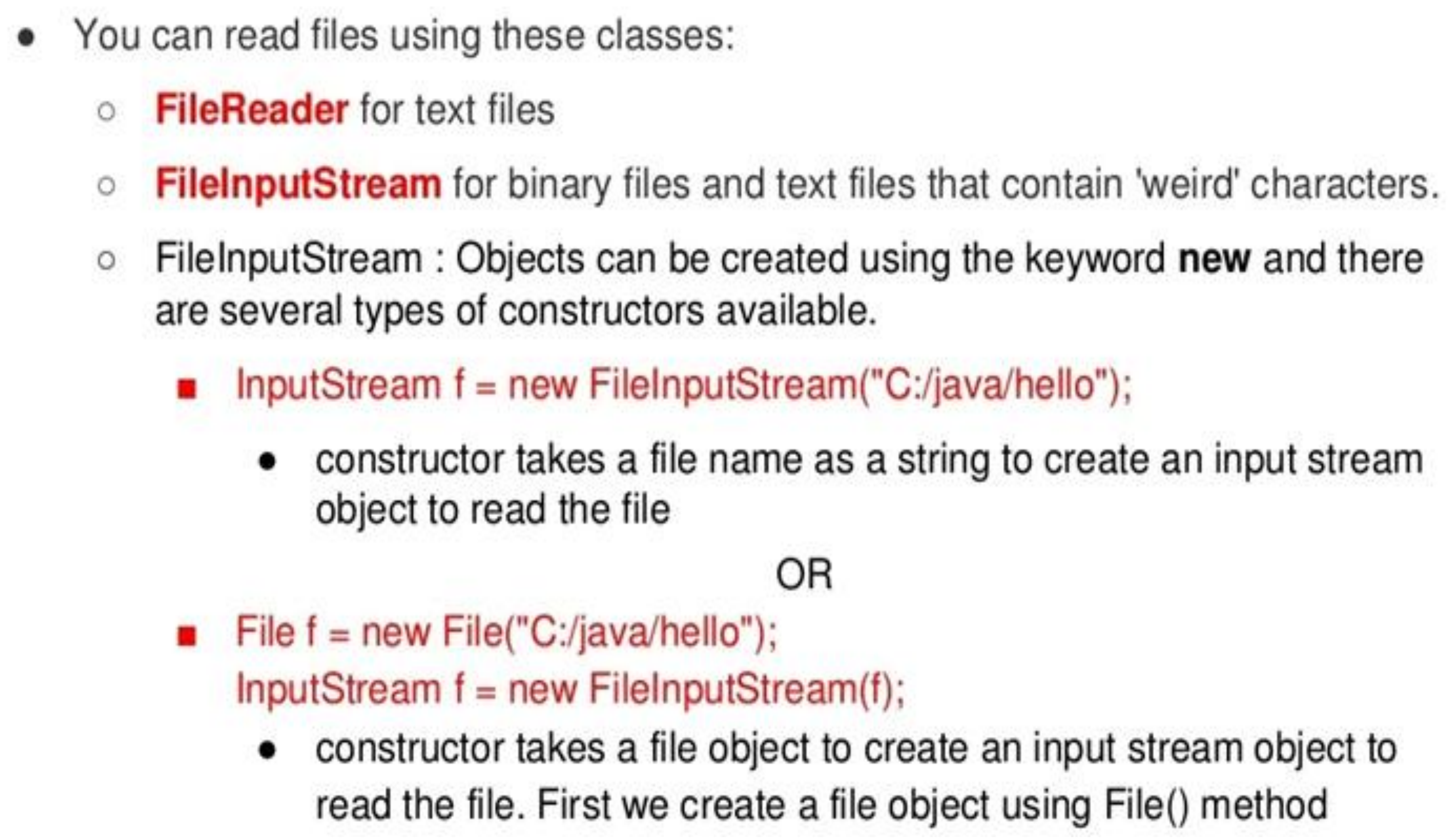
- To read string we have to use **readLine()** function with BufferedReader class's object.

Program to take String input from Keyboard in Java

```
import java.io.*;
class MyInput
{
 public static void main(String[] args)
 {
  String text;
  InputStreamReader isr = new InputStreamReader(System.in);
  BufferedReader br = new BufferedReader(isr);
  text = br.readLine();        //Reading String
  System.out.println(text);
 }
}
```

8. I/O and Streams

Reading files

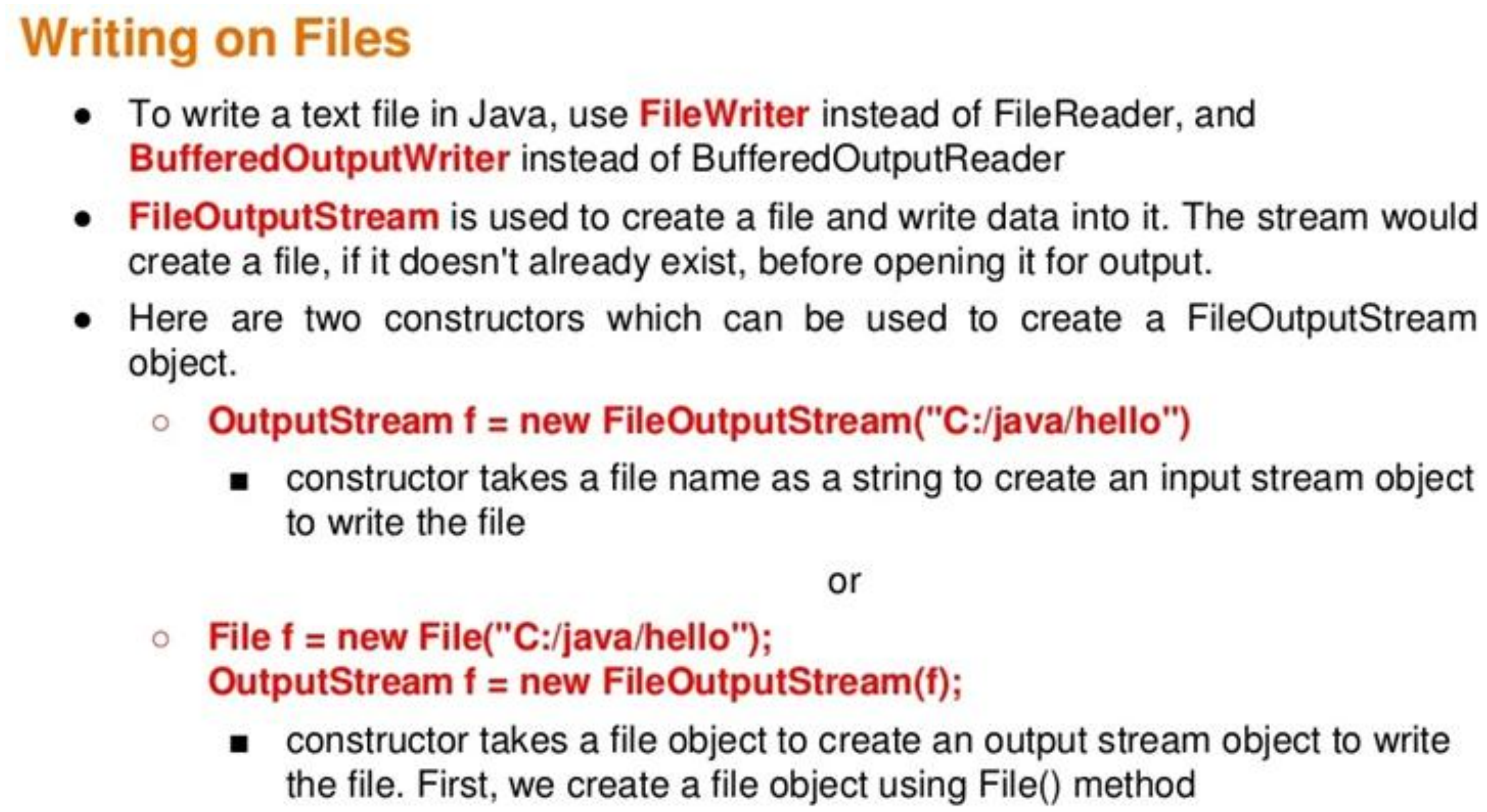


- You can read files using these classes:
  - **FileReader** for text files
  - **FileInputStream** for binary files and text files that contain 'weird' characters.
  - FileInputStream : Objects can be created using the keyword **new** and there are several types of constructors available.
    - InputStream f = new FileInputStream("C:/java/hello");
      - constructor takes a file name as a string to create an input stream object to read the file

      OR
    - File f = new File("C:/java/hello");
      InputStream f = new FileInputStream(f);
      - constructor takes a file object to create an input stream object to read the file. First we create a file object using File() method

8. I/O and Streams

Writing files



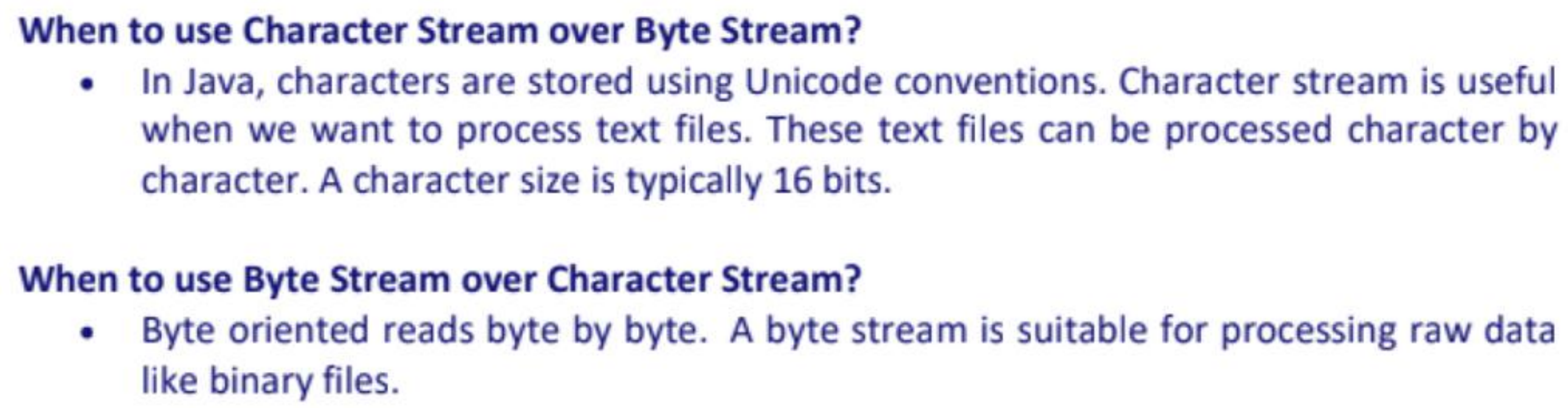
Writing on Files

- To write a text file in Java, use **FileWriter** instead of FileReader, and **BufferedOutputWriter** instead of BufferedOutputReader
- **FileOutputStream** is used to create a file and write data into it. The stream would create a file, if it doesn't already exist, before opening it for output.
- Here are two constructors which can be used to create a FileOutputStream object.
  - **OutputStream f = new FileOutputStream("C:/java/hello")**
    - constructor takes a file name as a string to create an input stream object to write the file

or

  - **File f = new File("C:/java/hello");**
    OutputStream f = new FileOutputStream(f);
    - constructor takes a file object to create an output stream object to write the file. First, we create a file object using File() method

Concept Assignment



When to use Character Stream over Byte Stream?

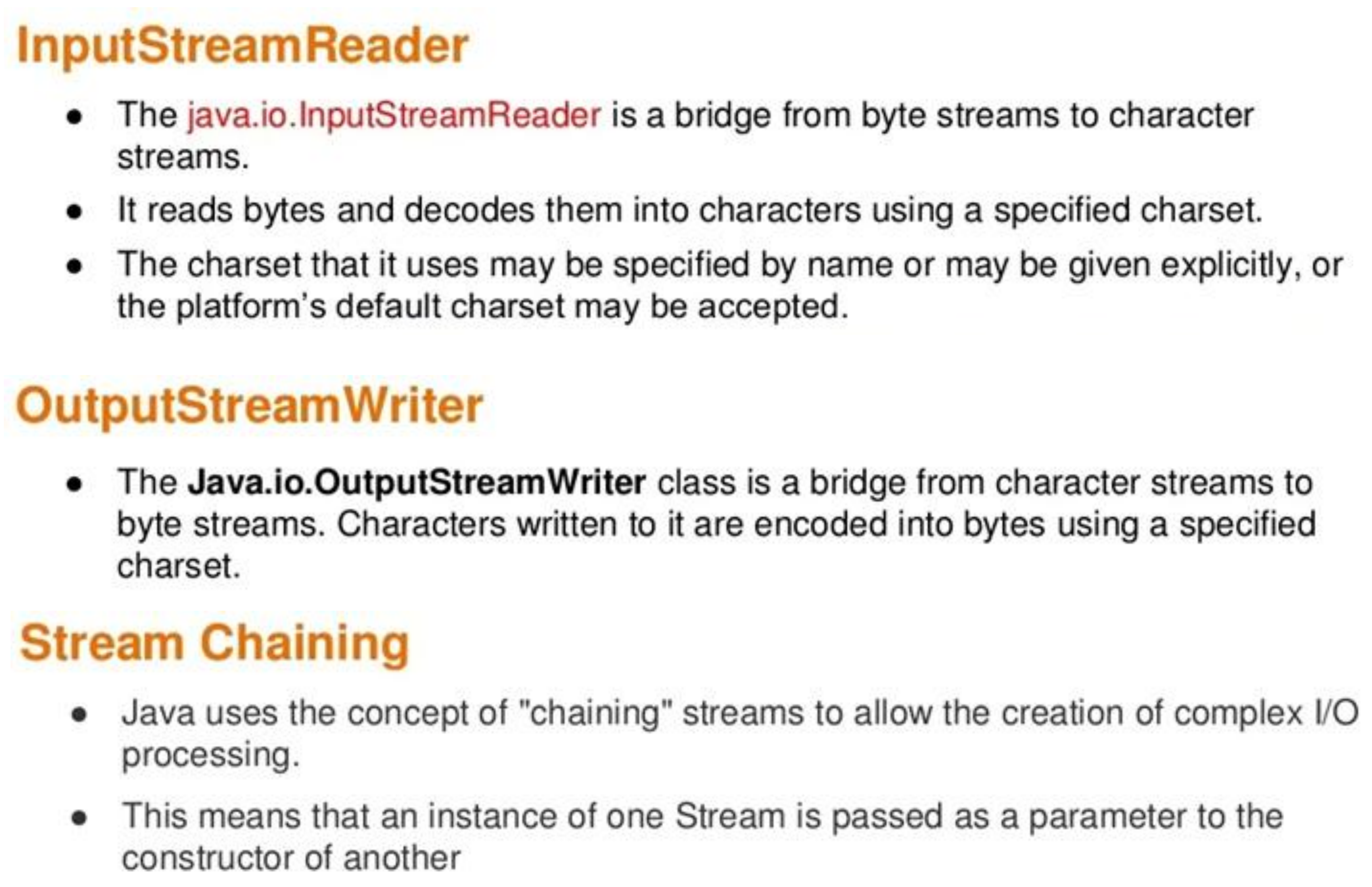
- In Java, characters are stored using Unicode conventions. Character stream is useful when we want to process text files. These text files can be processed character by character. A character size is typically 16 bits.

When to use Byte Stream over Character Stream?

- Byte oriented reads byte by byte. A byte stream is suitable for processing raw data like binary files.

8. I/O and Streams

Concept Assignment



InputStreamReader

- The java.io.InputStreamReader is a bridge from byte streams to character streams.
- It reads bytes and decodes them into characters using a specified charset.
- The charset that it uses may be specified by name or may be given explicitly, or the platform's default charset may be accepted.

OutputStreamWriter

- The **Java.io.OutputStreamWriter** class is a bridge from character streams to byte streams. Characters written to it are encoded into bytes using a specified charset.

Stream Chaining

- Java uses the concept of "chaining" streams to allow the creation of complex I/O processing.
- This means that an instance of one Stream is passed as a parameter to the constructor of another

Serializable interface

How to Serialize (Store) and De-serialize (Re-store back) Objects?

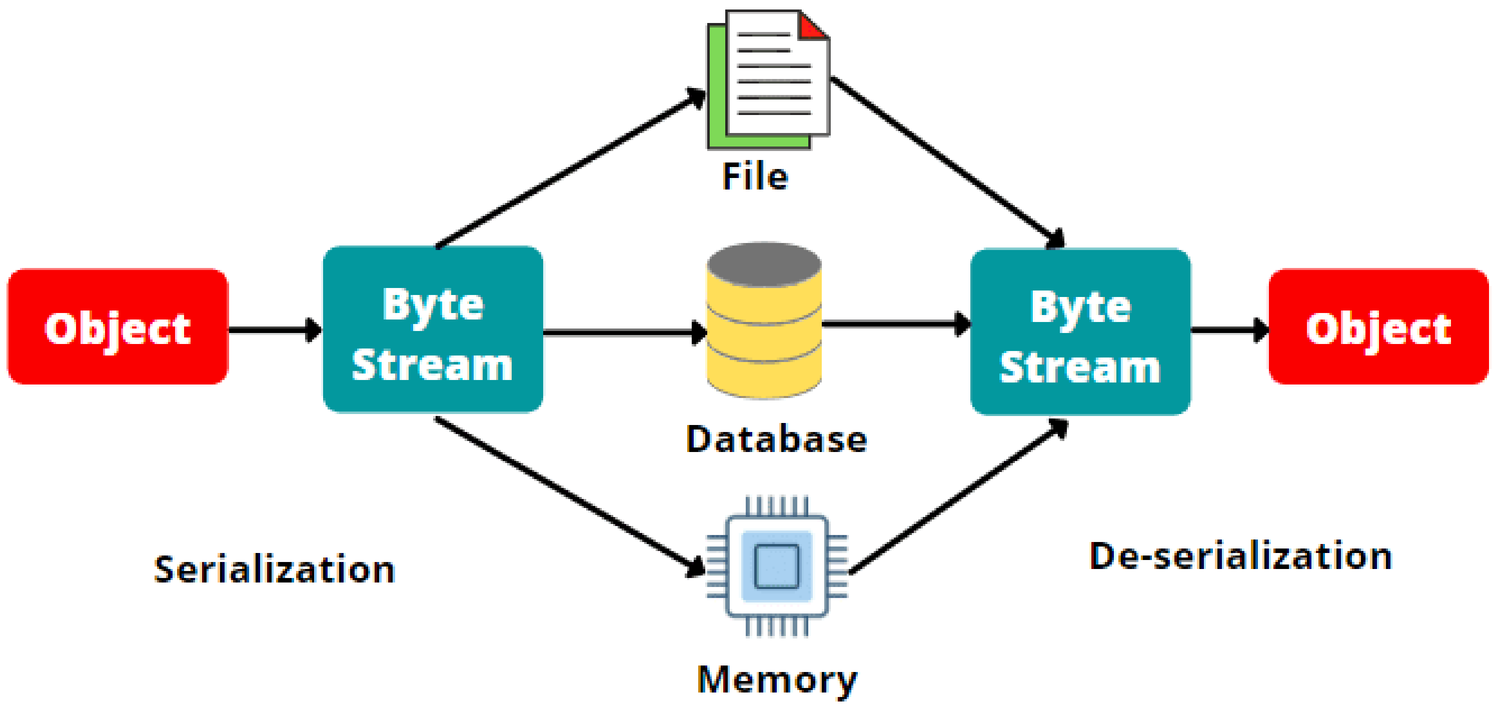


Fig: Serialization vs De-serialization process

Serializable interface

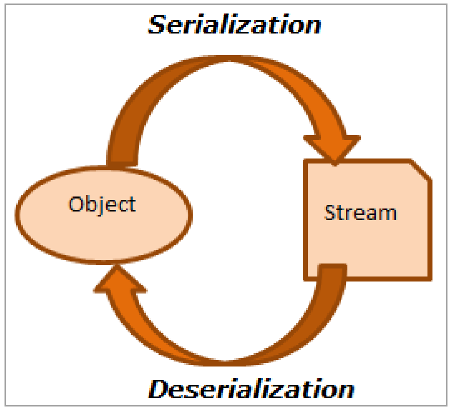
Marker Interface In Java

A marker interface in Java is an empty interface that has no fields or methods. This marker interface tells the compiler that the objects of the class that implement the marker interface are different and that they should be treated differently.
Each marker interface in Java indicates that it represents something special to JVM or compiler.

Serializable interface: Serializable is a marker interface present in the java.io package. We can serialize objects using this interface i.e. save the object state into a file.

Serializable interface

- A serializable interface is a marker interface that defines no members. It does not have any method also. Since it has no methods, we do not need to add additional code in the class that implements Serializable interface.
- Serializable interface is used to mark class objects as serializable so that they can be written into a file. If a class is serializable then all of its subclasses are also serializable.
- By default, String class and all the wrapper classes implement Serializable interface.
- So, if we want to send the state of an object over a network or file, we must implement Serializable interface.
- If serializable interface is not implemented by a class, storing that class objects into a file will generate NotSerializableException.



Serializable interface

Advantage:

- Used for marshaling(writing) (traveling the state of an object on the network)
- To persist or save an object’s state
- JVM independent
- Easy to understand and customize

Serialization & Deserialization.

```java
import java.io.*;
import java.io.Serializable;
//Class Student implements
class Student implements Serializable{
    int id;
    String name;
    public Student(int id, String name) {
        this.id = id;
        this.name = name;
    }
}

class SerializationDeSerialDemo{
    public static void main(String args[]){
        try{
            //Create the object of student class
            Student s1 =new Student(27,"James Gosling");
            //Write the object to the stream by creating a output stream
            FileOutputStream fout=new FileOutputStream("James.txt");
            ObjectOutputStream out=new ObjectOutputStream(fout);
            out.writeObject(s1);
            out.flush();
            //close the stream
            out.close();
            System.out.println("Object successfully written to the file");

            //Create a stream to read the object
            ObjectInputStream in=new ObjectInputStream(new FileInputStream("James.txt"));
            Student s=(Student)in.readObject();
            //print the data of the deserialized object
            System.out.println("Student object: " + s.id+" "+s.name);
            //close the stream
            in.close();

        }catch(Exception e){System.out.println(e);}
    }
}
```

```
Object successfully written to the file
Student object: 27 James Gosling
```

Serialization & Deserialization.

```java
import java.io.Serializable;
//Class Student implements
class Student1 implements Serializable{
    String name;
    String email;
    transient String password;
    public Student1(String name, String email, String password) {
        this.name = name;
        this.email = email;
        this.password=password;
    }
}

public class BasicSerializationExample {
    static final String filePath = "Student.txt";

    static void serialize(Student1 user) {
        try {
            FileOutputStream fos = new FileOutputStream(filePath);
            ObjectOutputStream outputStream = new
                                                ObjectOutputStream(fos);
            outputStream.writeObject(user);
            outputStream.close();
        } catch (IOException ex) {
            System.err.println(ex);
        }
    }
```

```java
static Student1 deserialize() {
        Student1 savedUser = null;

        try {
            FileInputStream fis = new FileInputStream(filePath);
            ObjectInputStream inputStream = new  -
                                                 ObjectInputStream(fis);
            savedUser = (Student1) inputStream.readObject();
            inputStream.close();
        } catch (IOException | ClassNotFoundException ex) {
            System.err.println(ex);
        }

        return savedUser;
    }

    public static void main(String[] args) {
        String name = "James";
        String email = "info@codejava.net";
        String password = "secret";
        Student1 newUser = new Student1(name, email, password);

        serialize(newUser);

        Student1 s = deserialize();
        //print the data of the deserialized object
        System.out.println("Student object: " + s.name+"
-                            "+s.email+s.password);
        //close the stream
    }
  }
```

```
Student object: James info@codejava.net
                                null
```

Concept Assignment

In Java, we have three interfaces that are Marker interfaces as shown below:
#1) Serializable interface: Serializable is a marker interface present in the java.io package. We can serialize objects using this interface i.e. save the object state into a file.
#2) Cloneable interface: The cloneable interface is a part of the java.lang package and allows the objects to be cloned.
#3) Remote interface: The remote interface is a part of the java.RMI package and we use this interface to create RMI applications. This interface mainly deals with remote objects.

#4) Serialization of Static and Transient Variables
Any static and transient variables declared inside a class cannot be serialized. *For example:*

```
static int x = 30;
transient String str = "myPassword";
```

These variables cannot be stored in a file. Static is the part of class, not object.

Motivate

We fall. We fail. We break.
But then, we rise. We heal.
We overcome.
@successpictures