Core Java
Java IO

Java : Accessing File System

Supported by java.io package

Accessing File System using java.io

- **When to use**
  - Used to check the pathname that it encapsulates against the physical file system and see whether it corresponds to a real file or directory.
  - Can be used to create file stream objects.

- **Creating File Object**
  - File(String path)
  - File(File dir,File file)

- **Portable Path Consideration**
  - char separatorChar
  - String separator
  - char pathSeparatorChar
  - String pathSeparator

Testing & Checking File Objects

- getName()
  - Returns the name of the file or the directory without path.
- getPath()
  - Returns a String object containing the path for the File object—including the file or directory name.
- isAbsolute()
  - Returns true if the File object refers to an absolute pathname, and false otherwise.
- getParent()
  - Returns a String object containing the name of the parent directory of the file or directory represented by the current File object.
- ***Note : When a file object is created using relative path, getParent() return null.***

Testing & Checking File Objects

- getParentFile()
  - Returns the parent directory as a File object, or null if this File object does not have a parent.
- equals(File)
  - Returns true if both File objects refers to the same file.
- exists()
  - Returns true if the file or directory referred to by the File object exists and false otherwise.
- isDirectory()
  - Returns true if the File object refers to an existing directory and false otherwise.
- isFile()

Testing & Checking File Objects

- isHidden()

- canRead()
- canWrite()
  - Both methods can throw SecurityException if user hasn't read or write access.

- getAbsolutePath()

- getAbsoluteFile()
  - Returns a File object containing the absolute path for the directory or file referenced by the current File object.

Getting File List:

- String[] list()
  - The method will throw an exception of type SecurityException if access to the directory is not authorized.
- String[] list(FilenameFilter)
- File[] listFiles(FileFilter)
- File[] listFiles(FilenameFilter)
- File[] listRoots()
  - returns an array of File objects referring to root directories in the current file system.
- long lastModified()
  - return value represents the time that the directory or file was last modified.

Creating & Modifying Files & Folders

- boolean renameTo(File)
- boolean setReadOnly( )
- boolean mkdir( )
- boolean mkdirs( )
- boolean createNewFile()
  - Creates a new empty file with the pathname defined by the current File object if and only if the specified directory already exists.

FileFilter & FilenameFilter

Interface java.io.FileFilter

boolean accept(File)

Interface java.io.FilenameFilter

boolean accept(File directory, String filename)

```
public static void main(String[]args)
{
        File f = new File("d:/sudi");
        if( f.isDirectory())
        {
           File files[] = f.listFiles(new MyFilter());
           for( File file : files )
              System.out.println(file);
        }
}
```

```
class MyFilter implements FileFilter
{
  public boolean accept(File f)
  {
     if( f.getName().endsWith(".jpg"))
        return true;
     else
         return false;
  }
}
```

Java : Input Output

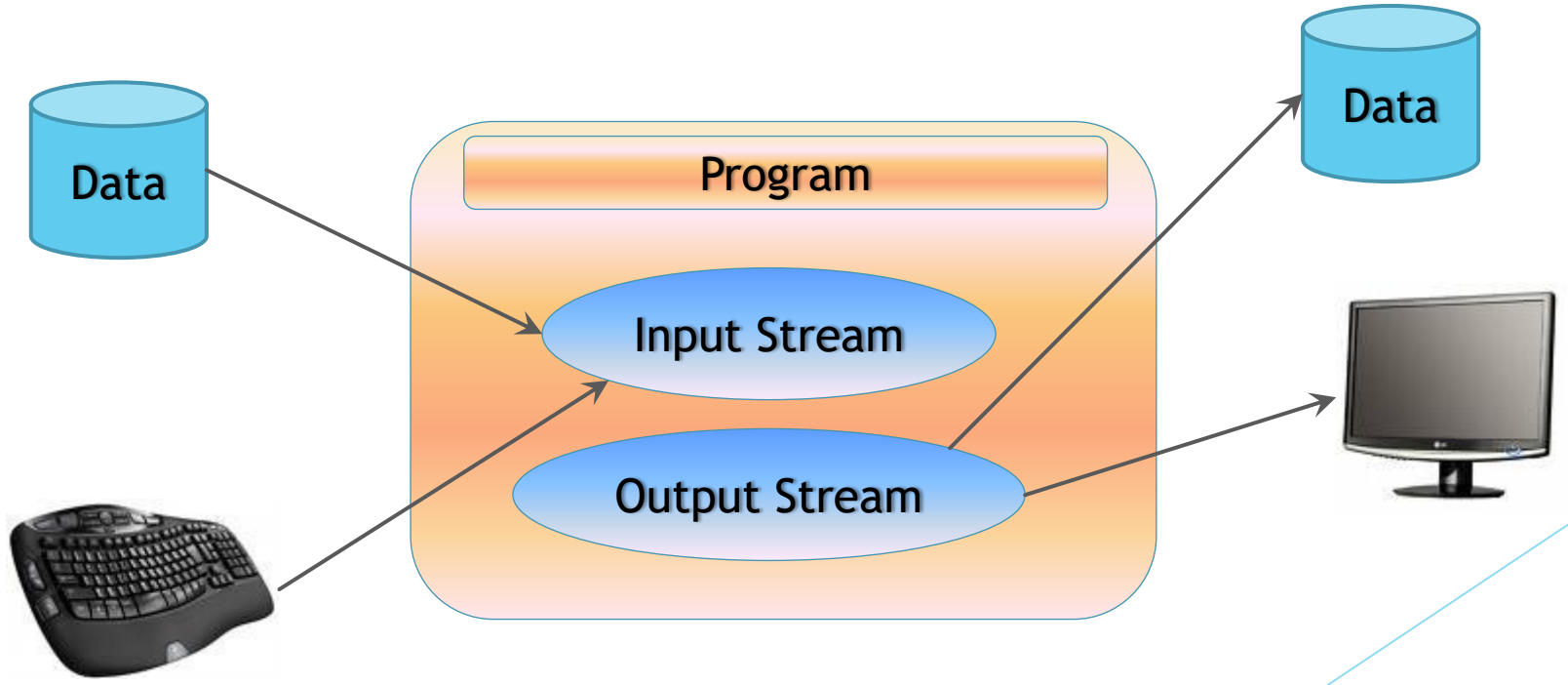
Supported by java.io package

Java IO

- Java programs perform Input Output through **streams**.
- A **stream** is simply a sequence of bytes that flows into or out of our program.
- Java implements stream within class hierarchies define in the **java.io** package.

Stream

- **Input Stream :** Any stream, the data can be read from.
- **Output stream** : Any stream, the data can be written to.



Types Of Stream

- **Types Of Stream:**
  - Byte or Binary Stream.
  - Character Stream.

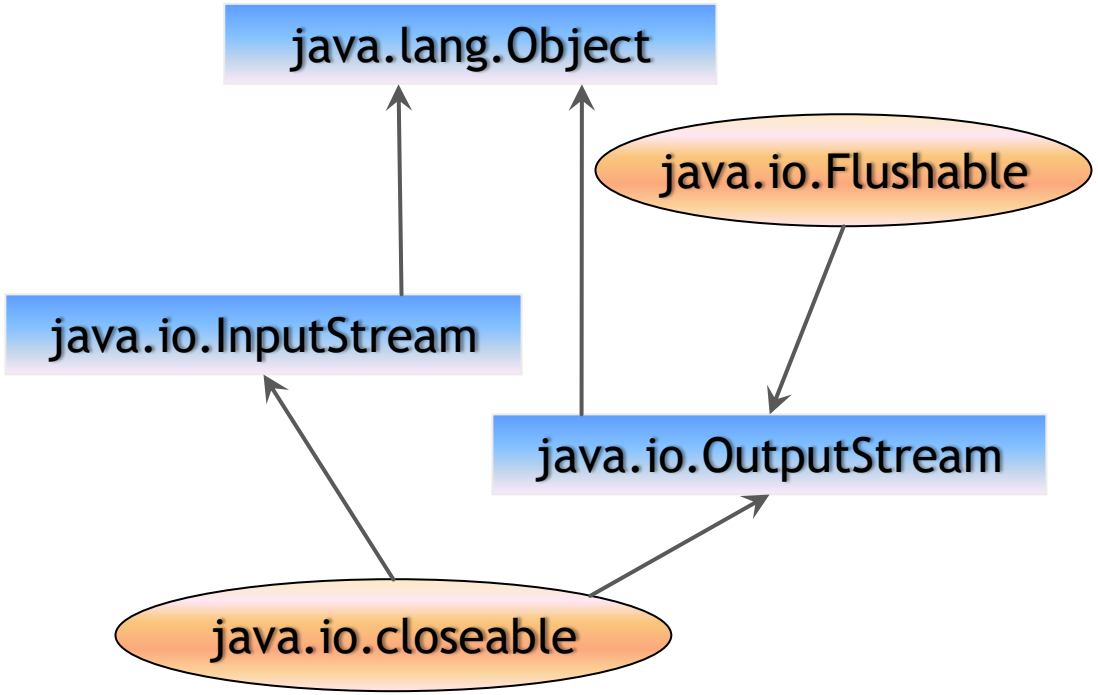
Binary/Byte Stream:

- Byte stream provide a convenient means for handling IO of bytes. Byte streams are used for example, when reading and writing binary data.

- Byte streams are defined by using two class hierarchies. At the top are two abstract class: **java.io.InputStream** and **java.io.OutputStream**.

- The abstract classes **InputStream** and **OutputStream** several key methods, that the other stream classes implement. Two of the most important methods are **read()** and ***write()***, which respectively read and write bytes of data.

InputStream & OutputStream



Closeable & Flushable

- **Added by jdk 5.**
- **Offers a uniform way of specifying that a stream can be closed or flushed.**

- **java.io.Closeable**

  public abstract void close( ) throws java.io.IOException

  - close( ) closes the stream and release any resources that the stream object is holding.

- **java.io.Flushable**

  public abstract void flush( ) throws java.io.IOException

  - Flushing a stream causes the buffered output to be physically written to the underlying device.

Basic Input Stream Operations

- **int read() throws IOException**

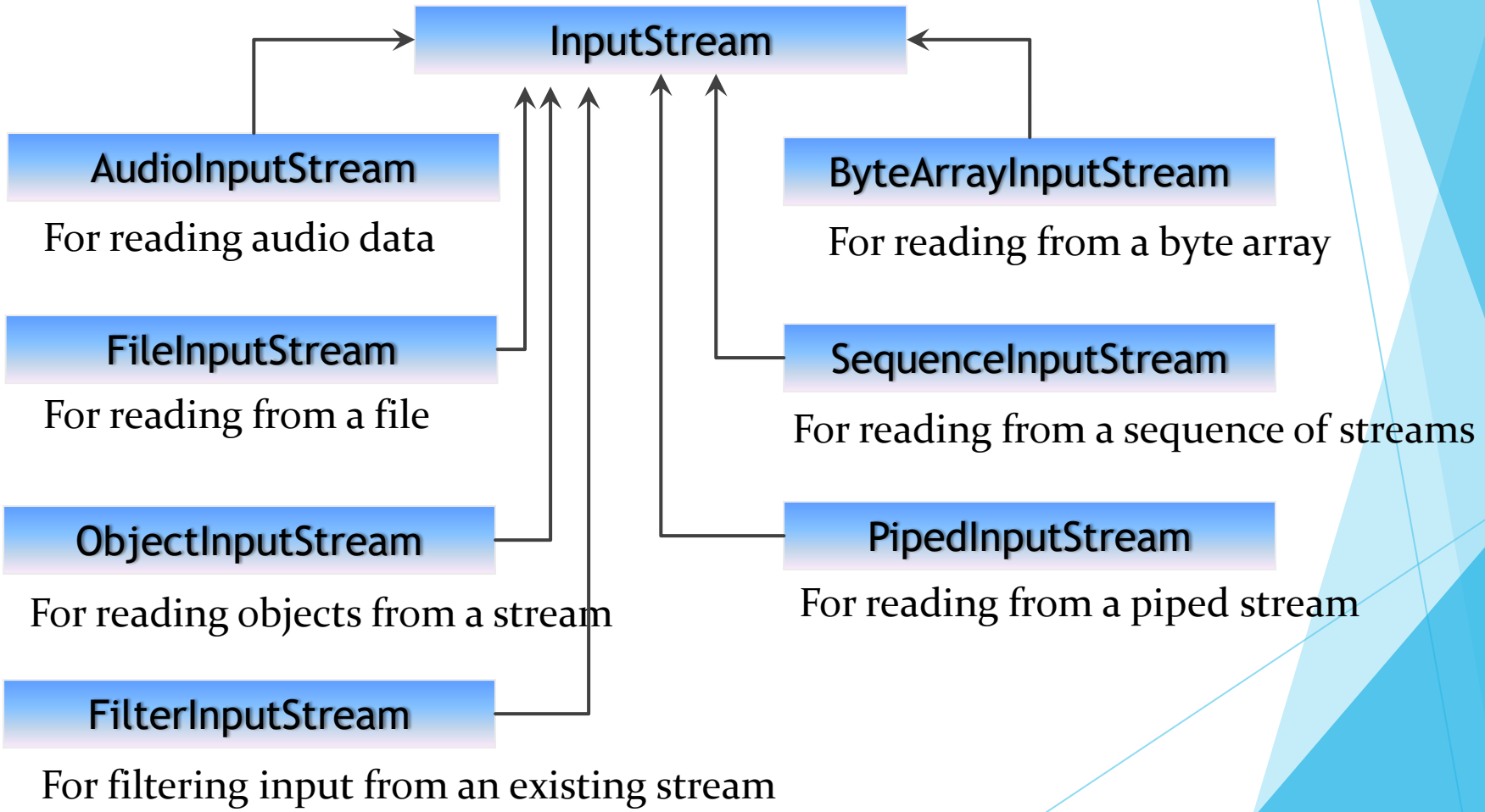
  Returns the next byte available from the stream as type int. If the end of the stream is reached, the method will return the value -1.

- **int read(byte[] array) throws IOException**

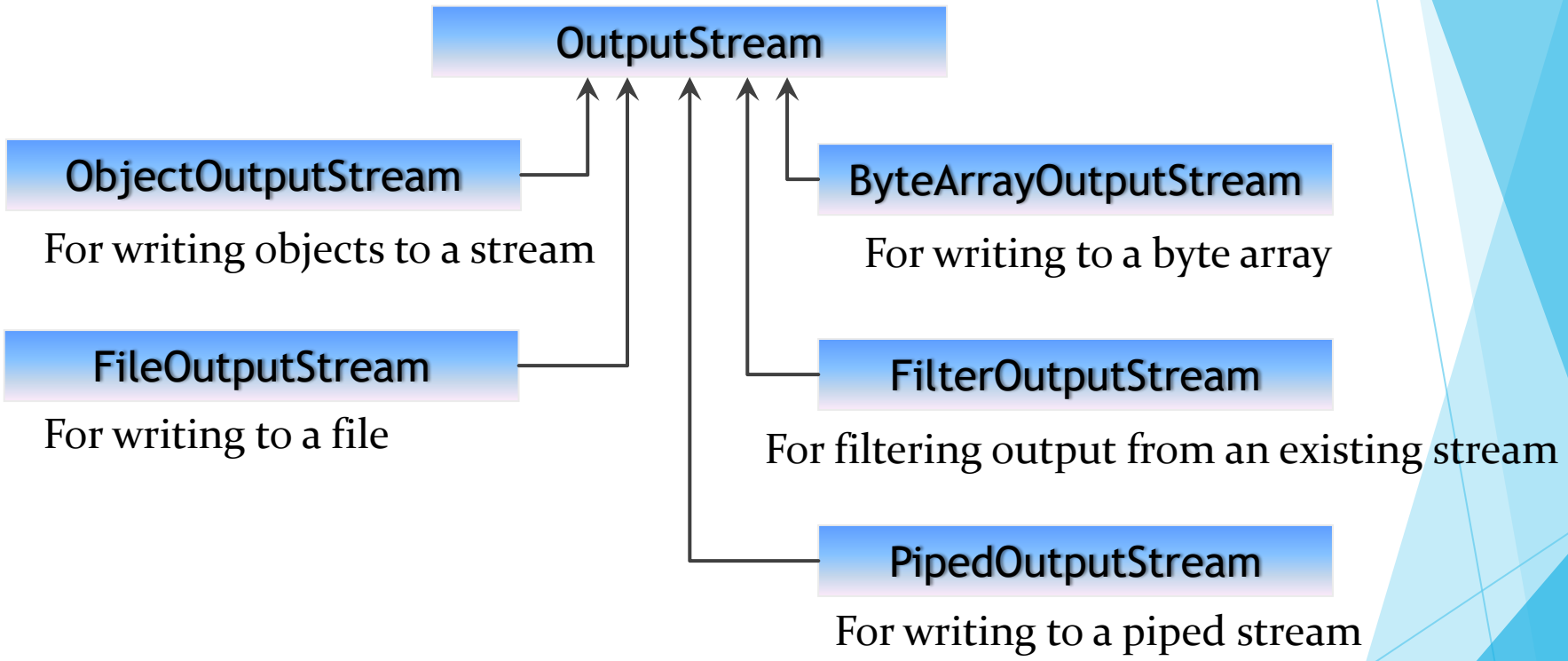
  This method reads bytes from the stream into successive elements of array. The maximum of array.length bytes will be read.

InputStream

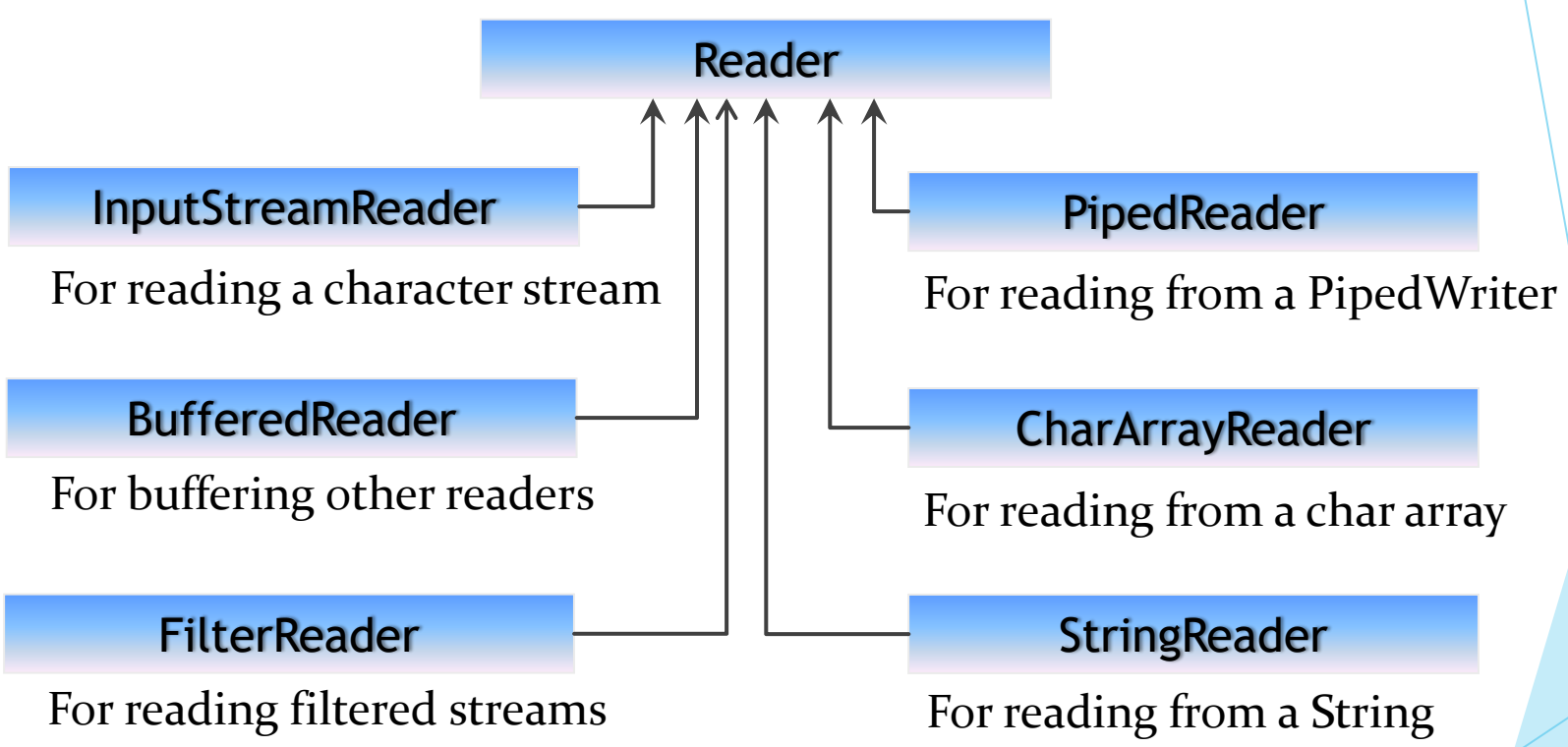
Direct Subclasses : 7



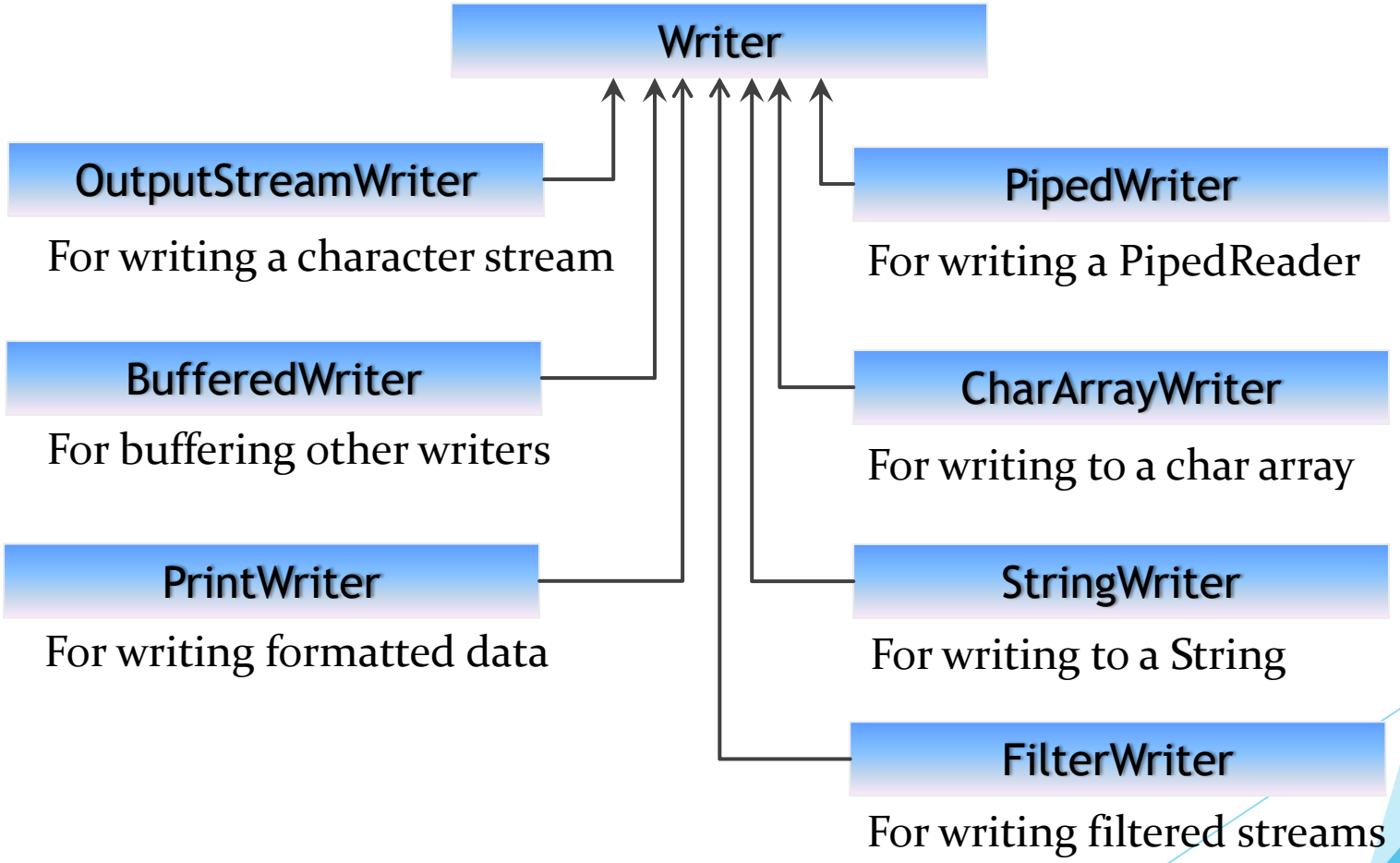
OutputStream



Stream Readers



Stream Writers



Standard Streams

Operating system typically defines three standard streams that are accessible through members of the System class.

- **Standard Input Stream :**

  Keyboard by default. Accessible by **System.in**.

  Type: InputStream

- **Standard Output Stream :**

  Corresponds to output on the command line. Accessible by **System.out**.

  Type: PrintStream

- **Standard Error Output Stream**

  for error messages that usually maps to the command-line output by default. Accessible **by System.err**.

  Type: PrintStream

FileInputStream

Constructors :

FileInputStream(File) throws FileNotFoundException

FileInputStream(String) throws FileNotFoundException

FileInputStream(FileDescriptor)

```
try{
        FileInputStream fis = new FileInputStream( "Test.java" );
        while( fis.available()>0 ){
                System.out.print((char)fis.read());
        }
}
catch(Exception e){
/* do something */
}
```

FileOutputStream

Constructors

FileOutputStream(String) throws FileNotFoundException

FileOutputStream(String, boolean) throws FileNotFoundException

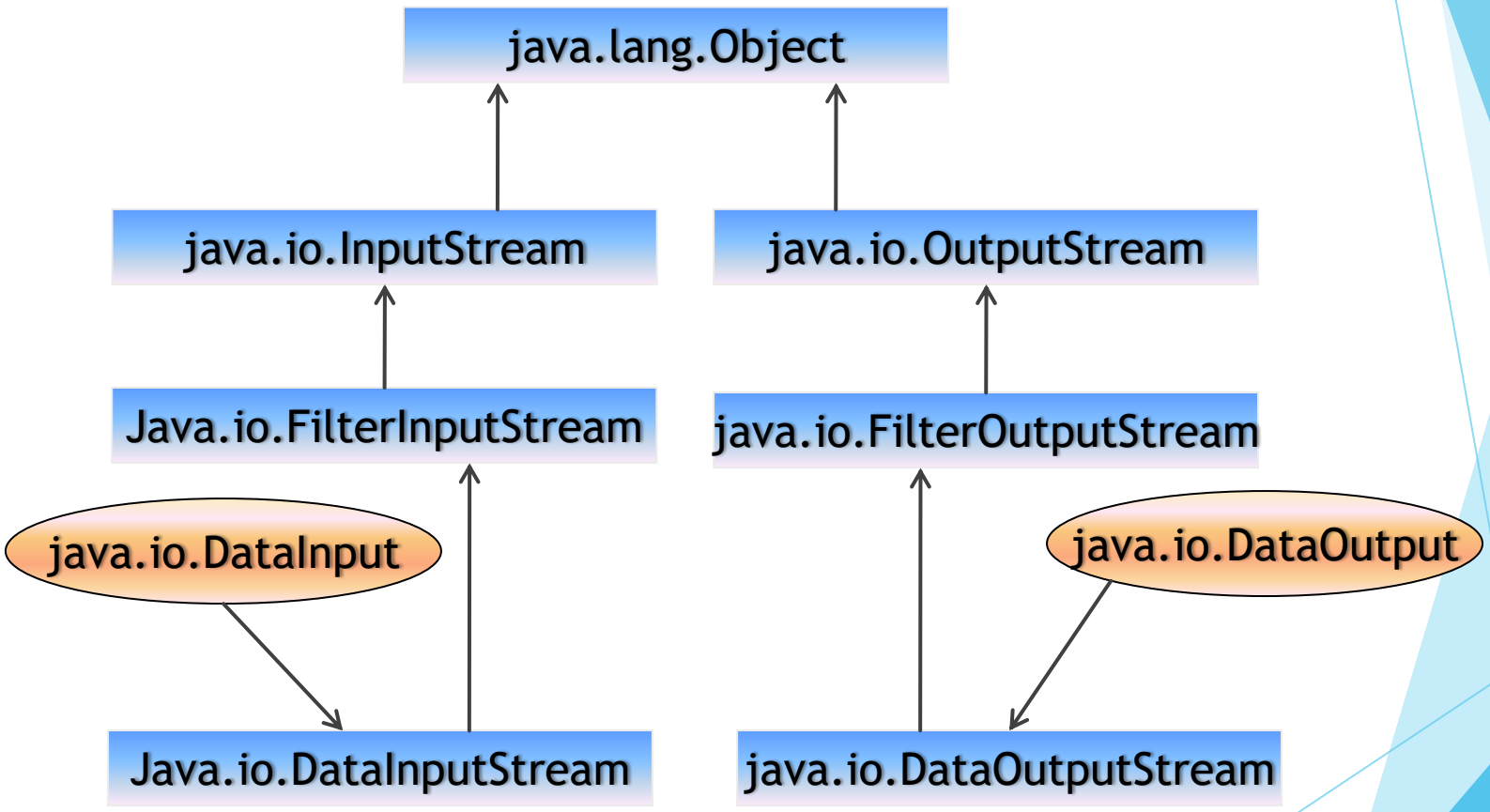
FileOutputStream(File) throws FileNotFoundException

FileOutputStream(File, boolean) throws FileNotFoundException

FileOutputStream(FileDescriptor)

A FileDescriptor object represents an existing connection to a file, so since the file must exist, this constructor does not throw an exception of type FileNotFoundException.

DataInputStream & DataOutputStream



DataInputStream & DataOutputStream : Example

```
try{
        FileOutputStream fos = new FileOutputStream("primitives.txt");
        DataOutputStream dos = new DataOutputStream( fos );
                dos.writeInt(50);
                dos.writeBoolean(true);
                dos.writeFloat(6.7f);
                dos.close();

        FileInputStream fis = new FileInputStream("primitives.txt");
        DataInputStream dis = new DataInputStream(fis );
                System.out.println( dis.readInt() );
                System.out.println( dis.readBoolean() );
                System.out.println( dis.readFloat() );
                dis.close();
}
catch(IOException ioe){
 ioe.printStackTrace(System.err);
}
```

Serialization

- **Introduction:** The process of writing state of an object to a file is called serialization, but strictly speaking it is the process of converting an object from java supported form into either file supported form or n/w supported form.
- Object Graph in Serialization
- Customized Serialization
- Serialization in Inheritance
- Externalization
- SerialVersionUID

ObjectInputStream & ObjectOutputStream contd.

```
class MyClass implements Serializable{
        int a;
        transient float f;
        MyClass(int x, float y){
                a = x;
                f = y;
        }
        void show(){
        System.out.println( a+" : "+f );
        }
}
```

```
MyClass mc = new MyClass(7,5.6f);
try{
    FileOutputStream fos = new FileOutputStream("object.txt");
    ObjectOutputStream oos = new ObjectOutputStream( fos );

    oos.writeObject( mc );
    oos.close();

    FileInputStream fis = new FileInputStream("object.txt");
    ObjectInputStream ois = new ObjectInputStream( fis );
    ( (MyClass)ois.readObject( ) ).show();

    ois.close();
}
catch(Exception ioe){
ioe.printStackTrace(System.err);
}
```

Object Graph Example

```
//Object Graph means: if a class
contains object of another class

class Test implements Serializable {
Demo d = new Demo();
}

class Demo implements Serializable {
Example e = new Example();
}

class Example implements Serializable {
int i = 100;
}

// In the above scenario, if we want to
serialize the Test object, it's
mandatory every object is serialized
```

```
public class ObjectGraphInSerialization {

public static void main(String[] args) {
Test t = new Test();
try {
FileOutputStream fos = new FileOutputStream("object_graph.txt");
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(t);

oos.close();
fos.close();
FileInputStream fis = new FileInputStream("object_graph.txt");
ObjectInputStream ois = new ObjectInputStream(fis);

Test d1 = (Test) ois.readObject();

System.out.println(d1.d.e.i);

ois.close();
} catch (Exception ioe) {
ioe.printStackTrace(System.err);
}

}

}
```

Customized Serialization

- **Transient data members will not be serialized, its default value will be stored, but there is requirement that it should be serialized but with some extra security (encryption), so for that we can do custom serialization.**

- **Can be achieve by using 2 methods**

1. private void writeObject(ObjectOutputStream oos) throws Exception:

This method will be executed automatically at the time of serialization, hence at the time serialization if we want to perform any activity, we have to define that in this method only.

2. private void readObject(ObjectInoutStream ois) throws Exception

- This method will be executed automatically at the time of de-serialization, hence at the time of de-serialization if we want to perform any activity, we must define that in this method only.

- **NOTE: above methods are callback methods because these are executed automatically by the JVM.**

Serialization in Inheritance

- Every serializable parent class, all the child class will be by default serializable.
- It is not mandatory that for serializable child parent should be serialized.
- At the time of de-serialization JVM will check is any parent class non-serializable or not, if any parent class is non-serializable then JVM will execute instance control flow in every non-serializable parent and share its instance variable to the current object.
- For every non-serializable parent class, it is mandatory that class should have default constructor.

Externalization

- In serialization, all the property will be serialized by-default by JVM.
- Suppose there are thousand of property of a class, but we want to store only few property then we go for externalization. This is control by the user.
- For externalization implements Externalizable, and it is child of Serializable from JDK 1.1.
- The are two methods to implement
  - readExternal
  - writeExternal
- In externalization, since all the property are not going to save, so in place of writing object, it saves the property.
- At the time of de-serializable, JVM will call public no-argument constructor, so it is mandatory to define default constructor in every externalizable class.

SerialVersionUID

- In serialization, it is not mandatory that serialization and de-serialization happening at the same machine, location etc..

Java : RandomAccessFile

RANDOM /read/write operations on file

- Java RandomAccessFile provides facility to both **read** and **write** data to a file. RandomAccessFile works with file as large array of bytes stored in the file system and a cursor using which we can move the file pointer position.

- RandomAccessFile class is part of Java IO.

- While creating the instance of RandomAccessFile in java, we need to provide the mode to open the file.
- **Constructor :**
  - *RandomAccessFile(String, String) throws FileNotFoundException;*
  - *RandomAccessFile(File,String) throws FileNotFoundException;*

- **Mode:**

  to open the file for read only mode we have to use **“r”** and for read-write operations we have to use **“rw”.**

- Using file pointer, we can read or write data from random access file at any position. To get the current file pointer, you can call **getFilePointer()** method and to set the file pointer index, you can call **seek(int i)** method.

- If we write data at any index where data is already present, it will override it.

Example :

- **Java RandomAccessFile read example:**

```
RandomAccessFile raf = new RandomAccessFile("file.txt", "r");
raf.seek(1);
byte[] bytes = new byte[5];
raf.read(bytes);
raf.close();
System.out.println(new String(bytes));
```

Example :

- **Java RandomAccessFile write example:**

```
RandomAccessFile raf = new RandomAccessFile("file.txt", "rw");
raf.seek(5);
raf.write("Data".getBytes());
raf.close();
```

java.io.StreamTokenizer

Gives a way to look for patterns in an input stream.

Constructors

StreamTokenizer(InputStream)

StreamTokenizer(Reader)

Constants : 4

TT_EOF : indicates end of stream has been read

TT_EOL : indicates end of line has been read

TT_NUMBER : indicates a no. has been read

TT_WORD : indicates a word token has been read.

Instance Variables : 3

ttype : indicates the token type that has just been read by nextToken().

nval : holds the no.

sval : holds the word.

Using StreamTokenizer

```
StreamTokenizer st = new StreamTokenizer(new FileInputStream("x.txt"));
int n=0,w=0;

while( st.nextToken()!=st.TT_EOF )
{
        if( st.ttype==st.TT_NUMBER)
        n++;
        else if( st.ttype == st.TT_WORD)
        w++;
}
System.out.println("Numbers in the file : "+n);
System.out.println("Words in the file : "+w);
```

Using StreamTokenizer contd.

- **resetSyntax()**

  Resets the state of the tokenizer object so no characters have any special significance. This has the effect that all characters are regarded as ordinary and are read from the stream as single characters. The value of each character will be stored in the ttype field.

- **ordinaryChar(int ch)**

  Sets the character ch as an ordinary character. An ordinary character is a character that has no special significance.

- **ordinaryChars(int low,int high)**

  Causes all characters from low to hi inclusive to be treated as ordinary characters.

- **wordChars(int start, int end)**

  Used to specify the range of characters that can be used in words.

- **whitespaceChars(int start,int end)**

  Used to specify the range of whitespace characters.

Formatted output to Command Line

Using printf( ) defined in PrintStream & PrintWriter.

- **printf(String format, Object ... args)**

  Outputs the values of the elements in args according to format specifications in format. An exception of type NullPointerException will be thrown if format is null.

- The format parameter is a string that should contain at least one format specification for each of the argument values that follow the format argument.

  General Form :

  %[argument_index$][flags][width][.precision]conversion

Formatted output to Command Line contd.

Conversion : This is a single character specifying how the argument is to be presented.

Conversion Character	Meaning
d, o, and x	Apply to integers and specify the no. system to be used to output the integers.
f, g, and a	Apply to floating-point values and specify that the output representation should be decimal notation, scientific notation (with an exponent), or hexadecimal with an exponent, respectively.
C	specifies that the argument value is a character and should be displayed as such.
S	specifies that the argument is a string.
B	specifies that the argument is a boolean value, so it will be output as “true” or “false”.
n	specifies the platform line separator so “%n” will have the same effect as “\n”.

Formatted output to Command Line

contd.

- **argument_index :** A decimal integer that identifies one of the arguments that follow the format string by its sequence number, where “1$” refers to the first argument, “2$” refers to the second argument, and so on.

- **Flags** : a set of flag characters that modify the output format. The flag characters that are valid depend on the conversion that is specified. Commonly used flags are :

  ‘-’ and ‘^’ apply to anything and specify that the output should be left justified and uppercase, respectively.

  ‘+’ forces a sign to be output for numerical values.

  ‘0’ forces numerical values to be zero-padded.

- **Width**
- **Precision**

Formatted output to Command Line contd.

```
int a = 5, b = 15, c = 255;
double x = 27.5, y = 33.75;
System.out.printf("x = %2$f y = %1$a", x, y);
System.out.println();
System.out.printf("a = %3$d b = %1$x c = %2$o", a, b, c);
```

Output

x = 33.750000 y = 0x1.b8p4
a = 255 b = 5 c = 17

String str = “The quick brown fox.”;
System.out.printf(“%nThe string is:%n%s%n%1$25s”, str);

Output

The string is:
The quick brown fox.
The quick brown fox.

Formatting Data into a String

Note :-

- *Printf() method produces the string that is output by using an object of type java.util.Formatter, and it is the Formatter object that is producing the output string from the format string and the argument values.*

- *A Formatter object is also used by a static method format() that is defined in the String class.*

```
double x = 27.5, y = 33.75;
String outString = String.format("x = %15.2f y = %14.3g", x, y);
System.out.print(outString);
```

Output:

x = 27.50 y = 33.750

Formatting Data into a StringBuffer

java.util.Formatter object can directly be used to format data also.

```
StringBuffer buf = new StringBuffer();
java.util.Formatter formatter = new java.util.Formatter(buf);

double x = 27.5, y = 33.75;
formatter.format("x = %15.2f y = %14.3g", x, y);

System.out.print(buf);
```

Points to remember :

A Formatter object can format data and transfer it to destinations other than StringBuilder and StringBuffer objects.