Lecture Six

Exploring Input-Output Operations

© **Dr. Mohammad Mahfuzul Islam, PEng**
Professor, Dept. of CSE, BUET

Template Class

Template Class: A template class works with any type of data, allowing for code reusability and flexibility.

C++ I/O system is build upon two template class hierarchies:

- ✓ **basic_streambuf** (associated 8 bit class: **streambuf**)
- ✓ **basic_ios** (associated 8 bit class: **ios**)

basic_streambuf: supplies low level I/O operations and provides underlying supports for entire I/O system. Used for advanced I/O programming.

basic_ios: A high level class that provides formatting, error checking and status information related to stream I/O.

An Example of Template Class

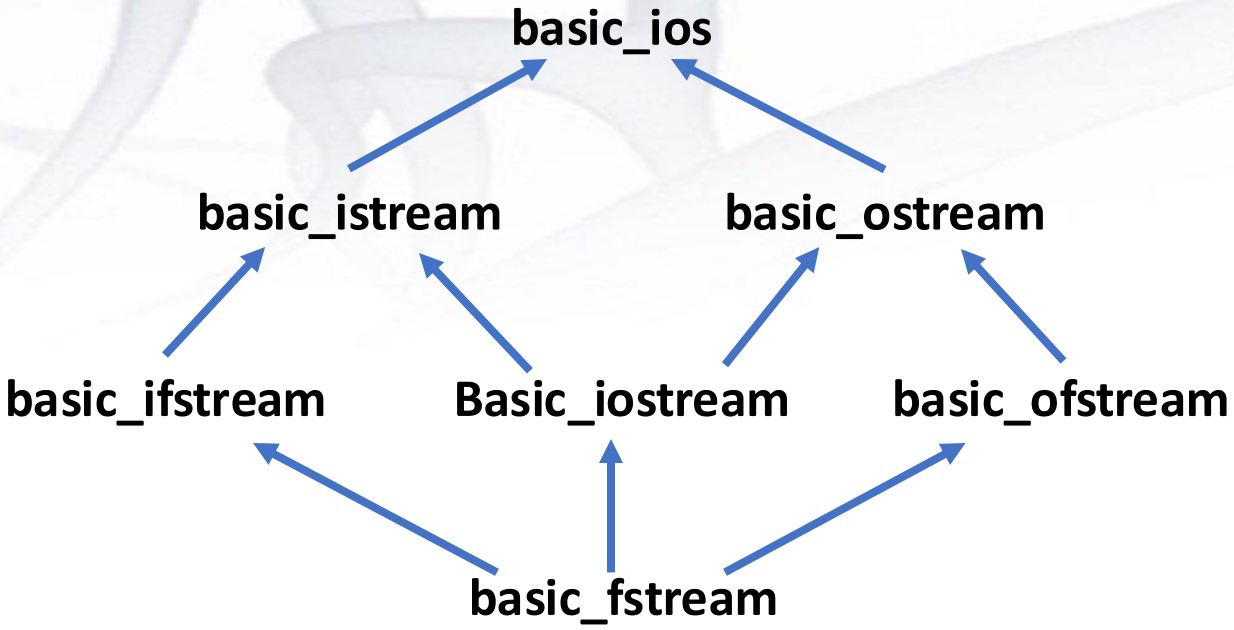
```
#include <iostream>
using namespace std;

template <class T>
class Gen{
    T val;
public:
    Gen(T v){ val = v;}
    T getval(){ return val;}
};

int main(){
    Gen<int> iob(88);
    Gen<double> dob(22.22);
    Gen<char> cob('X');

    cout << "Value: " << iob.getval() << endl;
    cout << "Value: " << dob.getval() << endl;
    cout << "Value: " << cob.getval() << endl;

    return 0;
}
```

OUTPUT:

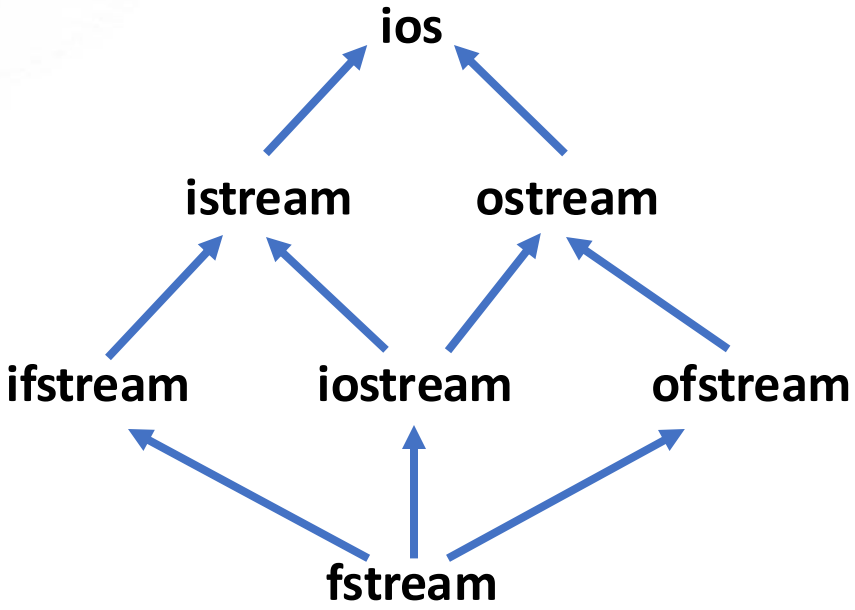
```
Value: 88
Value: 22.22
Value: X
```

Template classes for C++ I/O

Template Class



8-bit character based Class



Stream

Stream:

- ✓ A stream is a logical device that either produces or consumes information.
- ✓ A stream is linked to a physical device by the C++ I/O system.
- ✓ All streams behave in the same manner, although actual physical devices differ.

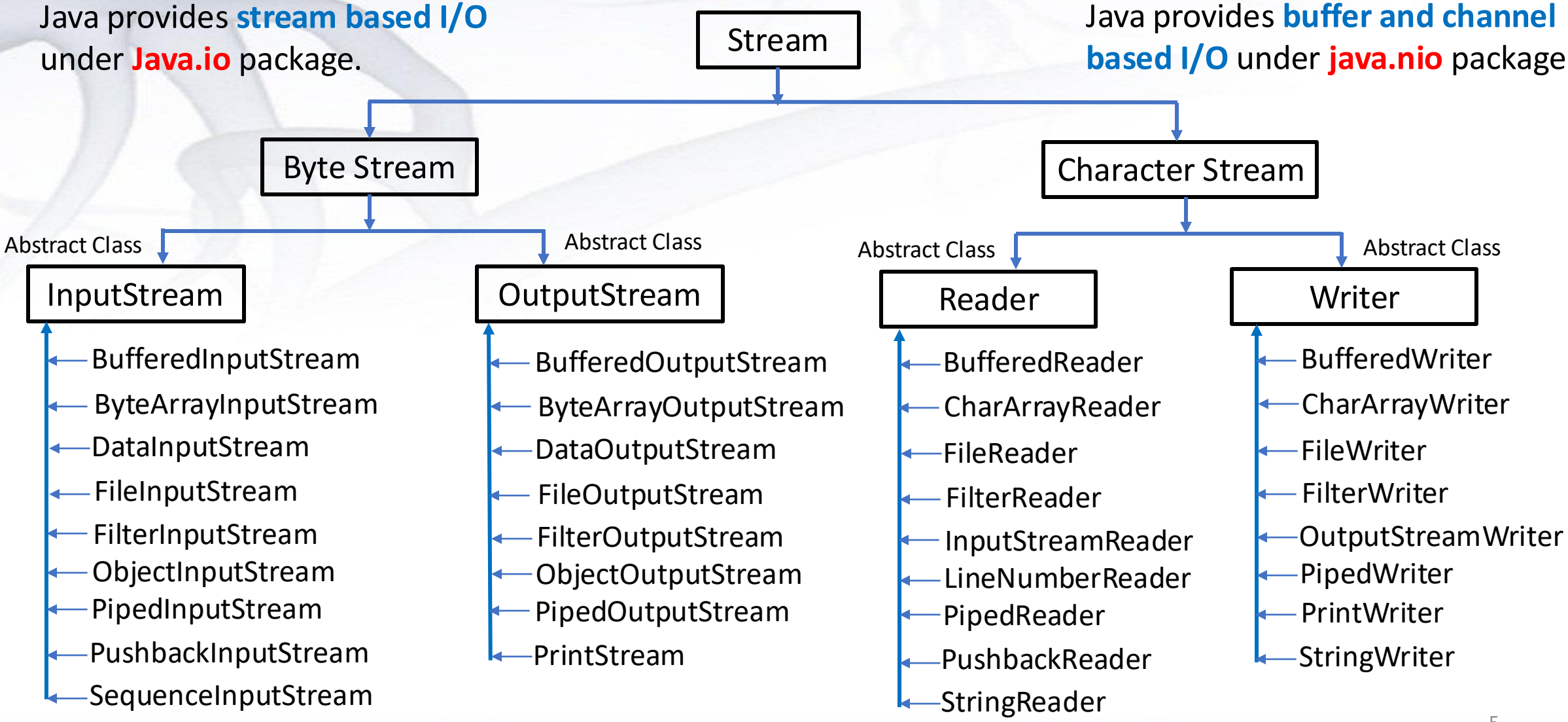
C / C++ Streams:

C Stream	C++ Stream (8-bit)	C++ Stream (Wide Character)	Default Device	Meaning
stdin	cin	wcin	Keyboard	Standard input
stdout	cout	wcout	Screen	Standard output
stderr	cerr	wcerr	Screen	Standard error
-	clog	wclog	Screen	Buffered version of standard error

Java Stream

Java provides **stream based I/O** under **Java.io** package.

Java provides **buffer and channel based I/O** under **java.nio** package.



Basic I/O Operations

C++

- C++ performs most I/O operations using **cin** and **cout** stream under std namespace.
- Some members of **ios** class for formatting I/O operations in C++:
  - ✓ **Bitmask enumeration** called **fmtflags**
  - ✓ **Width()**
  - ✓ **Precision()**
  - ✓ **Fill()**
  - ✓ Manipulator
  - ✓ Operator overloading (**Inserter** & **Extractor**)

Java

- Java performs most I/O operations using **System** class defined under **java.lang** package.
- All java programs automatically imports **java.lang** package.
- **System** contains three predefined stream variables: **in**, **out** and **err**. These fields are declared as **public**, **static** and **final** within System.
- **System.in** is an object of type **InputStream**. **System.out** and **System.err** are objects of type **PrintStream**

fmtflags Enumeration in C++

- Each stream has associated with a set of format flags that control the way of formatting information.
- The following values are defined to set or clear format flags.

Collective Value	Individual Value	Purpose
Adjustified	**left**	Output is left justified.
	right	Output is right justified **(Default setting)**
	internal	A numeric value is padded to fill a field by inserting spaces between any sign or base character.
Basefield	**oct**	Display output in octal.
	dec	Display output in decimal. **(Default setting)**
	hex	Display output in hexadecimal.
Floatfield	**scientific**	Display floating point values in scientific notation.
	fixed	Display floating point values in normal notation **(Default setting.)**

The **dec** flag **overrides** the other flags, so it is necessary to turn it off when turning on either hex or oct.

fmtflags Enumeration in C++

Flag	Purpose
boolalpha	Boolean input or output using the keyword true or false.
skipws	Leading whitespace characters (spaces, tabs and newlines) are discarded when input is processed in a stream.
showbase	The base of numeric values are displayed. For example, if the conversion base is hexadecimal, 1F is displayed as 0x1F.
showpoint	A decimal point and tailing zeros to be displayed for all floating point output - whether needed or not.
showpos	A leading plus sign (+) to be displayed before positive values. This only affects on decimal output.
unitbuf	The buffer is flashed after each insertion operation.
uppercase	The character are displayed uppercase.

Members of ios class for handling flags:

fmtflags flags(): Returns the current setting of each flag format.

fmtflags flags(fmtflags f): Set all format flags.

fmtflags setf(fmtflags flags): Set one or more format flags.

void unsetf(fmtflags flags): Clear one or more format flags.

fmtflags Enumeration in C++

```
#include <iostream>
using namespace std;

int main(){
    ios::fmtflags f = cout.flags();
    cout << 123.45 << " Hello " << -10 << " ” << 100.0 << " " << 100 << endl;

    cout.unsetf(ios::dec);
    cout.setf(ios::hex | ios::showbase | ios::showpos);
    cout << 123.45 << " Hello " << -10 << " ” << 100.0 << " " << 100 << endl;

    ios::fmtflags f2 = ios::scientific | ios::showpoint | ios::uppercase;
    cout.setf(f2);
    cout << 123.45 << " Hello " << -10 << " ” << 100.0 << " " << 100 << endl;

    if (f & ios::dec) cout << "Dec is set" << endl;
    cout.flags(f);
    cout << 123.45 << " Hello " << -10 << " ";
    cout << 100.0 << " " << 100 << endl;

    return 0;
}
```

OUTPUT:

123.45 Hello -10 100 100
+123.45 Hello 0xfffffff6 +100 0x64
+1.234500E+02 Hello 0XFFFFFFF6 +1.000000E+02 0X64
Dec is set
123.45 Hello -10 100 100

Using width(), precision() and fill()

Prototypes in **ios** class:

Prototype	Default
streamsize width(streamsize w);	Minimum size
streamsize precision(streamsize p);	6 digits
char fill(char ch);	spaces

- Each method returns the **old value**.
- **Streamsize** is defined in ios as some form of integer.

OUTPUT:

```
 Hello
123.457
      Hello
*****Hello
123.45679
123.45679%%%
   2     1.414    4
   3     1.732    9
   4       2     16
   5     2.236    25
```

```
#include <iostream>
#include <cmath>
using namespace std;

int main(){
    cout << " Hello " << endl;
    cout << 123.456789 << endl;
    cout.width(10); cout << "Hello" << endl;
    cout.width(10);
    cout.fill('*'); cout << "Hello" << endl;

    cout.precision(8); cout << 123.456789 << endl;
    cout.width(12);
    cout.fill('%'); cout.setf(ios::left);
    cout << 123.456789 << endl;
    cout.fill(' ');
    cout.setf(ios::right);
    cout.precision(4);
    for( double x =2.0; x<=5; x++){
        cout.width(7); cout << x;
        cout.width(12); cout << sqrt(x);;
        cout.width(7); cout << x*x << endl;
    }
    return 0;
}
```

Manipulator in C++

➢To access manipulators that takes parameters, **<iomanip>** must be included. This is not necessary if the manipulator does not take arguments.

➢ If the manipulator does **not** take an **argument**, it is **not followed by parenthesis**, because it is the **address of the manipulator** that is passed to the overloaded << operator.

List of Manipulators:

Manipulator	Purpose	Stream
boolalpha	Turn on boolalpha flag	I/O
dec	Turn on dec flag	I/O
endl	Output a newline character and flushes the output stream	Output
ends	Output a null	Output

Manipulator	Purpose	Stream
fixed	Turn on fixed flag	Output
flush	Flushes a stream	Output
hex	Turn on hex flag	I/O
internal	Turn on internal flag	Output
left	Turn on left flag	Output
noboolslpha	Turn off boolalpha flag	I/O
noshowbase	Turn off showbase flag	Output
noshowpoint	Turn off showpoint flag	Output
noshowpos	Turn off showpos flag	Output
noskipws	Turn off skipws flag	Input
nounitbuf	Turn off unitbuf flag	Output
nouppercase	Turn off uppercase flag	Output
oct	Turn on oct flag	I/O

Manipulator in C++

Manipulator	Purpose	Stream
resetiosflags(fmtflags f)	Turn off the flags specified in f	I/O
right	Turn on right flag	Output
scientific	Turn on scientific flag	Output
setbase(int base)	Set the number base to base	I/O
setfill(int ch)	Set the fill character to ch	Output
setiosflags(fmtflags f)	Turn on the flags specified in f	I/O
setprecision(int p)	Set number of digits of precision	Output
setw(int w)	Set the field width to w	Output
showbase	Turn on showbase flag	Output
showpoint	Turn on showpoint flag	Output
showpos	Turn on showpos flag	Output
skipws	Turn on skipws flag	Input

Manipulator	Purpose	Stream
unitbuf	Turn on unitbuf flag	Output
uppercase	Turn on uppercase flag	Output
ws	Skips leading white space	Input

```
#include <iostream>
#include <iomanip>
using namespace std;

int main(){
    cout << hex << 100 << endl;
    cout << oct << 10 << endl;
    cout << setfill('*') << setw(10);
    cout << 100 << " hi"<< endl;

    return 0;
}
```

OUTPUT:

64
12
*******144 hi

Creating Own Manipulator in C++

Two **reasons** for creating custom manipulator:

- ✓ Consolidate a sequence of several separate I/O operations. This **simplifies** the source code and **prevents accidental errors**; and
- ✓ When need to perform I/O operations on a **nonstandard device**.

OUTPUT:

Enter a hexadecimal number: c2d
You entered: *****0XC2D

```
#include <iostream>
#include <iomanip>
using namespace std;

ostream& setup(ostream& os) {
    os << hex << showbase << uppercase;
    os.width(10);
    os.fill('*');
    return os;
}

istream& hex_input(istream& is) {
    is >> hex;
    return is;
}

int main() {
    int number;

    cout << "Enter a hexadecimal number: ";
    cin >> hex_input >> number;
    cout << "You entered: " << setup << number << endl;

    return 0;
}
```

Operator Overloading: Inserter & Extractor

Inserter

- **Inserter** inserts objects into output stream, the **output operation** is called an **insertion** and the **<<** is called the **insertion operator**.
- **Inserter accepts two parameters. The first parameter is a reference to output stream and the second parameter is the object to be displayed. The function returns an output stream.**
- **An inserter cannot be a member of a class. An inserter can be a friend function.**

Extractor

- **Extractor extracts objects from input stream, the input operation is called an extraction and the >> is called the extraction operator.**
- **The first parameter is a reference to input stream and the second parameter is the object to be displayed. The function returns an input stream.**
- **An extractor cannot be a member of a class. An extractor can be a friend function**

Operator Overloading: Inserter & Extractor

```
#include <iostream>
using namespace std;

class Inventory{
      char itemName[20];
      int itemNumber;
      double price;
public:
      Inventory(char* name, int num, double cost){
            strcpy(itemName, name);
            itemNumber = num;
            price = cost;
      }
      friend ostream& operator<<(ostream& os, Inventory& inv);
      friend istream& operator>>(istream& is, Inventory& inv);
};

ostream& operator<<(ostream& os, Inventory& inv){
      os << "Item: " << inv.itemName << endl;
      os << "Item Number: " << inv.itemNumber << endl;
      os << "Price: " << inv.price << endl;
      return os;
}
```

```
istream& operator>>(istream& is, Inventory& inv){
      cout << "Enter item name: ";
      is >> inv.itemName;
      cout << "Enter item number: ";
      is >> inv.itemNumber;
      cout << "Enter price: ";
      is >> inv.price;
      return is;
}

int main(){
      Inventory item1("hammer", 1234, 19.95);
      Inventory item2("wrench", 4567, 8.75);
      cout << item1 << item2;
      cin >> item1 >> item2;
      cout << item1 << item2;
      return 0;
}
```

OUTPUT:

```
Item: hammer
Item Number: 1234
Price: 19.95
Item: wrench
Item Number: 4567
Price: 8.75
Enter item name: Table
Enter item number: 5
Enter price: 150
Enter item name: Chair
Enter item number: 5
Enter price: 80
Item: Table
Item Number: 5
Price: 150
Item: Chair
Item Number: 5
Price: 80
```

Basic Java I/O

- ✓ **System** class of java provides facilities like standard input, standard output and standard error streams. *System class can't be instantiated*.
- ✓ Java **Scanner** is a utility class to read user input or process simple regex-based parsing of file or string source

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner myObj = new Scanner(System.in);

        System.out.print("Enter the first number: ");
        int a = myObj.nextInt();
        System.out.print("Enter the second number: ");
        int b = myObj.nextInt();

        int sum = a + b;
        System.out.println("The sum is "+ sum+ ".");
    }
}
```

Scanner Methods:

- `nextBoolean()`
- `nextByte()`
- `nextDouble()`
- `nextFloat()`
- `nextInt()`
- `nextLine`
- `nextLong()`
- `nextShort()`

OUTPUT:

```
Enter the first number: 23
Enter the second number: 43
The sum is 66.
```

Java Console

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;

public class Main {
   public static void main(String[] args) throws IOException {
      int a, i = 0;
      char ch;
      String[] str = new String[100];
      BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
      PrintWriter pw = new PrintWriter(System.out, true);

      a = System.in.read();
      System.out.write(a);

      System.out.println("\nEnter 'q' to quit");
      do{
         ch = (char) br.read();
         pw.println(ch);
      } while (ch != 'q');
      System.out.println("Enter a line and write 'stop' to quit.");
      do{
         str[i++] = br.readLine();
      }while (!str[i-1].equals("stop"));

      for(int j = 0; j < i-1; ++j) pw.println(str[j]);
   }
}
```

OUTPUT:

```
A
A
Enter 'q' to quit
CS10rq
C
S
1
0
r
q
Enter a line and write 'stop' to quit.
Testing PrintWriter class
instead of System.out
stop

Testing PrintWriter class
instead of System.out
```

Java Console

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

- **System.in** is an object of type **InputStream** and reads bytes from keyboard.
- **InputStreamReader** converts bytes to characters.

```
InputStreamReader(InputStream inputStream);
```

- Constructors of **BufferedReader**:

```
BufferedReader(Reader inputReader);
```

- `int read() throws IOException` reads a character from input stream and returns it as an integer value. Return -1 at the end of stream.
- `String readLine() throws IOException` reads a String from input stream.

PrintWriter pw = new PrintWriter(System.*out*, true);

- Constructors of **PrintWriter**:

```
PrintWriter(OutputStream outputStream,
            Boolean flushingOn);
```

- `void write(int byteval)` is used to write a character to the console.

System.out is an object of type **PrintStream**. Console output is mostly accomplished with `print()` and `println()` under **`System.out`** rathe than using **`PrintWriter`**.

File I/O

File Operations in C++

- **Header:** <fstream>
- **Classes:** ifstream, ofstream, fstream
- **Opening** a file:
  - ✓ ofstream out;

    out.open("test.txt"); //default

    out.open("test.txt", ios::out | ios::trunc);
  - ✓ ifstream in("test.txt");
- **Closing** a file: in.close();
- **Checking** whether a file is open:

  if(in.is_open()) cout << "file is open";
- **Reading** a character from file: in.get(ch);
- **Writing** a character to a file: out.put(ch);

- **openmode** is an **enumeration** defined by **ios**. Values of openmode are as follows:

ios::in	Open file to read.
ios::out	Open file to write
ios::app	All output to be appended at the of the file
ios::ate	Open with marker at the end of the file
ios::binary	Open file in binary mode
ios::trunc.	Truncate file to zero length
ios::nocreate	Open if only exists, don't create

Two or more can be combined using | operator.

- **Checking** end of file:

  while (!in.eof()) {.....}

File Operations in C++

```
#include <iostream>
#include <fstream>
using namespace std;

int main(){
   try{
      ofstream out("Inventory", ios::app);
      if( !out ) throw "Cannot open output file";
      out << "Radios " << 39 << endl;
      out << "Toasters " << 21 << endl;
      out << "Mixers " << 17 << endl;

      out.close();
   } catch(const char* message){
      cout << message << endl;
      return 1;
   }

   try{
      ifstream in("Inventory");
      if(!in) throw "Cannot open input file";

      char item[20];
      int quantity;
      in >> item >> quantity;
      while(!in.eof()){
          cout << item << ' ' << quantity << endl;
          in >> item >> quantity;
      }
      in.close();
   } catch(const char* message){
      cout << message << endl;
      return 1;
   }

   ifstream in("Inventory");
   ofstream out("Temp");
   char ch;

   in >> ch;                    // >> operator eats WS
   while(!in.eof()){
      out << ch;
      in >> ch;
   }

   in.close();
   out.close();
   return 0;
}
```

Sequential Access of Binary File in C++

Method	Return Type	Description
get(char &ch)	istream	Read a character from input stream.
put(char ch)	ostream	Write a character to output stream.
read(char *buf, streamsize num)	istream	Read ***num*** characters from input stream and put them to ***buf***.
write(char *buf, streamsize num)	ostream	Write ***num*** characters from ***buf*** to output stream.
gcount()	streamsize	Returns number of characters read by last input operation.
getline(char *buf, streamsize num)	istream	Reads ***num-1*** characters from input stream and put them into ***buf***. Reading stops when a newline or end of file is encountered.
getline(char *buf, streamsize num, char delim)	Istream	Reads ***num-1*** characters from input stream and put them into ***buf***. Reading stops when the ***delim*** or end of file is encountered.
ignore(streamsize num = 1, int_type delim = EOF)	istream	Read and ignore ***num*** (default 1) characters or the character ***delim*** (default EOF) is encountered.
peek() & putback()	int/ istream	peek() returns next character from the stream. putback() ?

get() also provide similar functionalities of ***getline()***, except doesn't remove delimiter from input stream.

Sequential Access of Binary File in C++

```
#include <fstream>
#include <iostream>
using namespace std;

int main(){
   ifstream in;
   ofstream out;
   fstream out2("test3.cpp", ios::out | ios::app);
   char ch;

   try{
      in.open("test.cpp");
      if(!in) throw "Cannot open input file";
      out.open("test2.cpp", ios::out);
      if(!out) throw "Cannot open output file";
   } catch(const char* message){
      cout << message << endl;
      return 1;
   }
```

```
    in.get(ch);
    while(!in.eof()){
        out.put(ch);
        out2.put(ch);
        in.get(ch);
    }

    if (in.is_open())
      cout << ”file already open\n";

    in.close();
    out.close();
    out2.close();

    return 0;
}
```

Random Access of Binary File in C++

There are two pointers for file:

- ✓ ***get*** pointer
- ✓ ***put*** pointer

seekdir is an enumeration of ***ios*** with following values:

Ios::beg (beginning of file), **ios::cur** (current), **ios::end**

Method	Description
istream &seekg(off_type offset, seekdir origin)	Move the get pointer offset number of characters from the origin.
ostream &seekp(off_type offset, seekdir origin)	Move the put pointer offset number of characters from the origin.
post_type tellg()	Current location of get pointer.
post_type tellp()	Current location of put pointer.
iostate rdstate()	Returns current status of error flags
bool good(), bool bad(), bool eof(), bool fail()	Another way of checking errors.
void clear(iostate flags = ios::goodbit)	When goodbit is set, all others are cleared.

iostate is an ***ios*** enumeration with following values:

ios::goodbit	//set 1 for no error
ios::eofbit	//1 for EOF encountered
ios::failbit	// 1 for non-fatal I/O error
ios::badbit	// 1 for fatal I/O error

Random Access of Binary File in C++

```
#include <iostream>
#include <fstream>
using namespace std;

void showState(ios::iostate state){
     if(state & ios::goodbit) cout << "goodbit is set" << endl;
     if(state & ios::eofbit) cout << "eofbit is set" << endl;
     if(state & ios::failbit) cout << "failbit is set" << endl;
     if(state & ios::badbit) cout << "badbit is set" << endl;
}

int main(){
    char *str = new char[80];

    ifstream in;
   ofstream out;
   try{
        in.open("test.cpp");
        if(!in) throw "Cannot open i
        out.open("test2.cpp");
        if(!out) throw "Cannot oper
   } catch(const char* message){
        cout << message << endl;
        return 1;
   }
```

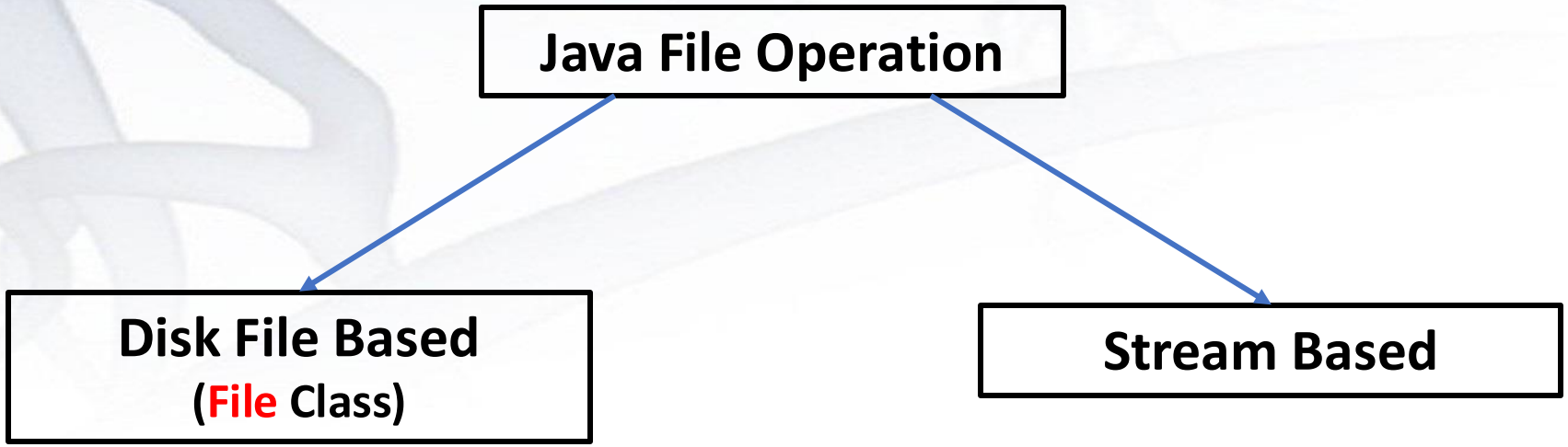
```
#include <iostream>
using namespace std;

int main(){
float length = 40.5;
#include <iostream>
using namespace std;

int main(){
float length = 40.5;
```

```
    cout << "getPointer: " << in.tellg() << " putPointer: " << out.tellp() << endl;

    in.read(str, 80);
    out.write(str, 80);
    cout << "getPointer: " << in.tellg() << " putPointer: " << out.tellp() << endl;

    in.seekg(0, ios::end);
    out.seekp(0, ios::end);
    cout << "getPointer: " << in.tellg() << " putPointer: " << out.tellp() << endl;

    in.read(str, 80);
    cout << in.gcount() << " characters read" << endl;
    out.write(str, 80);
    cout << "getPointer: " << in.tellg() << " putPointer: " << out.tellp() << endl;

    ios::iostate state = in.rdstate();
    showState(state);

    in.clear();
    state = in.rdstate();
    showState(state);

    return 0;
}
```

OUTPUT:

```
getPointer: 0 putPointer: 0
getPointer: 80 putPointer: 80
getPointer: 287 putPointer: 80
0 characters read
getPointer: -1 putPointer: 160
eofbit is set
failbit is set
```

File Operation in Java



File Class:

- A **File** object is used to **obtain** or **manipulate** the information associated with a disk file, such as the ***permissions***, ***time***, ***date***, and ***directory path***, and to navigate subdirectory hierarchies.
- A **directory** in Java is treated simply as a **File** with one additional property

File Class

Constructors:

File(String *directoryPath*)
File(String *directoryPath*, String *filename*)
File(File *dirObj*, String *filename*)
//dirObj is File object that represents a directory
File(URI *uriObj*) //URI object that describes a file

Examples:

```
File f1 = new File("/");
File f2 = new File("/", "autoexec.bat");
File f3 = new File(f1, "autoexec.bat");
```

Some Methods:

```
getName()
getPath()
getAbsolutePath()
getParent()
exists()
canWrite()
canRead()
isHidden()
isDirectory()
isFile()
isAbsolute()
lastModified()
length()
```

lastModified() returns time in milliseconds since January 1, 1970, Coordinated Universal Time (UTC)

Some more Methods:

```
long getFreeSpace()
long getTotalSpace()
long getUsableSpace()
boolean delete()
void deleteOnExit()
boolean setLastModified(long ms)
boolean setReadOnly()

boolean renameTO(File)
String[] list()
String[] list(FilenameFilter)

boolean mkdir()
boolean mkdirs()
```

listFiles() is alternative methods fof list().

File Class

```
import java.io.File;

public class Main {
    static void print(String s){
        System.out.println(s);
    }
    public static void main(String[] args) {
        File f1 = new File("/", "tmp");
        if(f1.exists()){
            print(f1.getName() + " exists");
        } else {
            print(f1.getName() + " does not exist");
        }
        File f2 = new File("/");

        print("FileName: " + f1.getName());
        print("Path: "+ f1.getPath());
        print("Absolute Path: " + f1.getAbsolutePath());
        print("Parent: " + f1.getParent());
        print("Size: " + f1.length());
        print("Last Modified: " + f1.lastModified());
        print("Is Directory: " + f1.isDirectory());
        print("Is File: " + f1.isFile());
        print("FreeSpace: " + f1.getFreeSpace());
        print("Usable Space: " + f1.getUsableSpace());
        String [] files = f2.list();
        for (String file : files) {
            print(file);
        }
    }
}
```

OUTPUT:

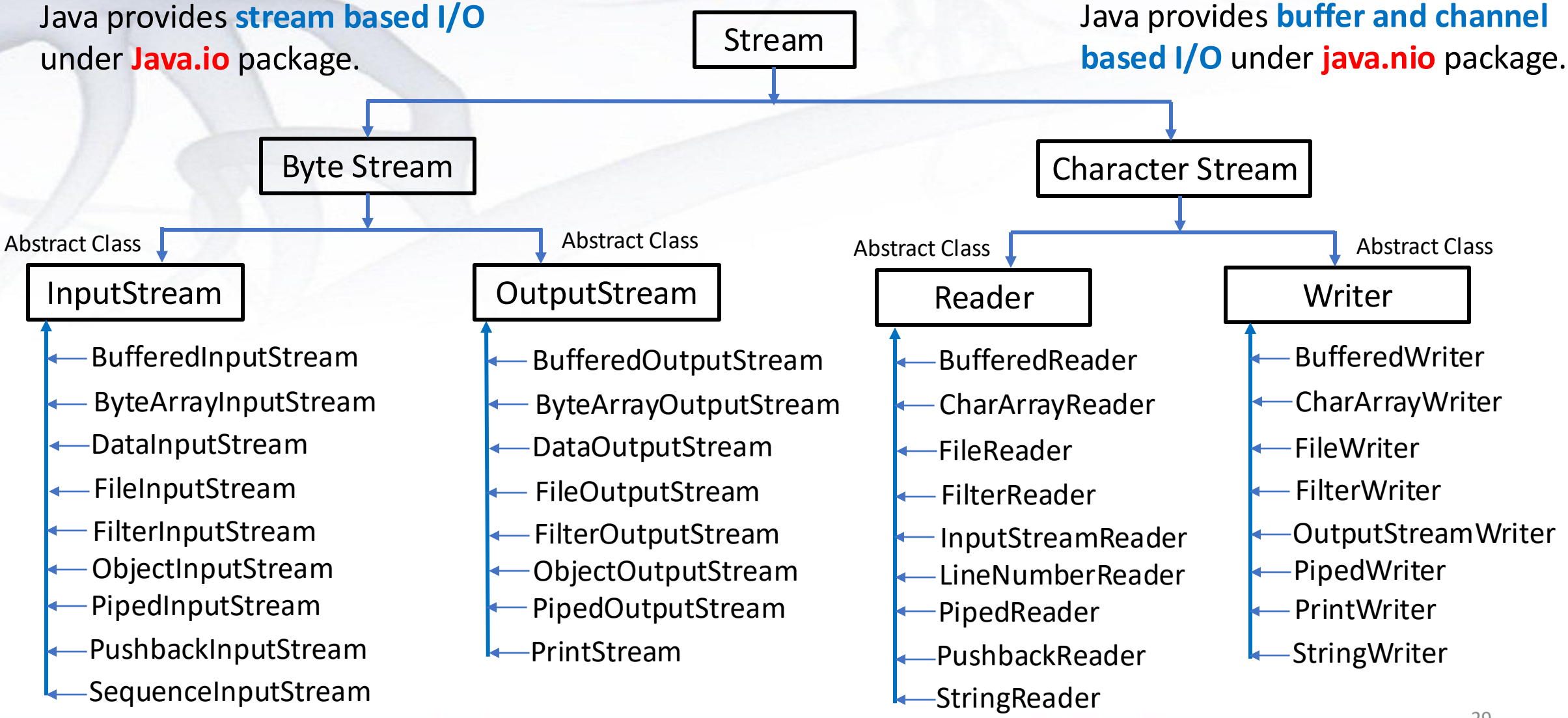
```
tmp exists
FileName: tmp
Path: /tmp
Absolute Path: /tmp
Parent: /
Size: 128
Last Modified: 1740722214482
Is Directory: true
Is File: false
FreeSpace: 396161658880
Usable Space: 396161658880
```

```
home
usr
bin
sbin
.file
etc
var
Library
System
.VolumeIcon.icns
private
.vol
Users
Applications
opt
dev
Volumes
tmp
cores
```

Stream based File Operation in Java

Java provides **stream based I/O** under **Java.io** package.

Java provides **buffer and channel based I/O** under **java.nio** package.



Stream based File Operation in Java

Some Methods of InputStream

Method	Description
int read()	Reads a single byte and return as an integer value. Return -1 at the end of the stream.
int read(byte[] buf)	Reads up to *buffer.length* bytes into *buffer* and returns the actual number of bytes that were successfully read.
int read(byte[] buf, int offset, int numBytes)	Reads up to *numBytes* bytes into *buffer* starting at *buffer*[*offset*], returning the number of bytes successfully read.
int available()	Returns number of bytes available for reading.
void mark(int n)	Set mark which is valid for n bytes read.
void reset()	Reset the input pointer to set mark.
long skip(long n)	Skips n bytes and returns number actually skipped.

Some Methods of OutputStream

Method	Description
void write(int b)	Write a single byte to output stream.
void write(byte[] buf)	Write a complete array of bytes to an output stream.
void write(byte[] buf, int offset, int numBytes)	Writes a subrange of *numBytes* bytes from the array *buffer*, beginning at *buffer*[*offset*].
void flush()	Finalizes the output state so that any buffers are cleared.
void close()	Closes the output stream. Further write attempts will generate an **IOException**.
static OutputStream nullOutputStream( )	Returns an open, but null output stream, which is a stream to which no output is actually written.

FileInputStream & FileOutputStream

- Constructors and Opening of FileInputStream:

```
FileInputStream(String filePath / File fileObj) throws FileNotFoundException;
```

Example:

```
1. FileInputStream f0 = new FileInputStream(”test1.cpp");
2. File f = new File(”test2.cpp");
   FileInputStream f1 = new FileInputStream(f);
```

- Constructors and Opening of FileOutputStream:

FileOutputStream(String filePath / File fileObj) throws FileNotFoundException;

FileOutputStream(String filePath / File fileObj, Boolean append) throws FileNotFoundException;

Example:

```
1. FileOutputStream f2 = null; f2 = new FileOutputStream(”file1.txt");
2. FileOutputStream f3 = new FileOutputStream(“file2.txt”, true);
```

- `close()` method closes files.

```
void close() throws IOException;
```

Java Program 1 with Closeable

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class CopyFile1 {
    public static void main(String[] args) {
        int num;
        FileInputStream fin = null;
        FileOutputStream fout = null;

        if (args.length != 2){
            System.out.println("Usage: CopyFile from to--");
            return;
        }

        try{
            fin = new FileInputStream(args[0]);
            fout = new FileOutputStream(args[1]);
            do{
                num = fin.read();
                if ( num != -1 ) fout.write(num);
            } while (num != -1);
        } catch (IOException e){
            System.out.println("I/O Error: " + e);
        } finally {
            try{
                if (fin != null) fin.close();
            } catch (IOException e2) {
                System.out.println("Error in closing input file.");
            }
            try{
                if (fout != null) fout.close();
            } catch (IOException e2){
                System.out.println("Error in closing output file.");
            }
        }
    }
}
```

Java Program with AutoCloseable

- ***Automatic resource management*** (ARM) is performed based on expanded version of try statement, i.e., ***try-with-resources*** statement.
- The scope of the file remains within the try-block.

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
```

```
public class CopyFile2 {
    public static void main(String[] args) {
        int num;

        if (args.length != 2){
            System.out.println("Usage: CopyFile from to--");
            return;
        }

        try (FileInputStream fin = new FileInputStream(args[0]);
              FileOutputStream fout = new FileOutputStream(args[1])){

            do{
                num = fin.read();
                if ( num != -1 ) fout.write(num);
            } while ( num != -1 );

        } catch (IOException e) {
            System.out.println("I/O Error: " + e);
        }
    }
}
```