Exceptions

- Input from a file
- Output to a file
- Exceptions
  - Handling - local, external
  - Try, catch, finally

Example - Read from a File

sumfile.java

```
import java.io.*;
import cs111.io.*;
class SumfromFile extends Object{
public static void main(String[] args) throws
   IOException {
//sums a sequence of numbers on a file ended by -1.0
    TokenStream infile = new TokenStream("input");
    double sum = 0.0, d = 0.0;
    for (;d != -1.0;){
            d = infile.readDouble();
            if (d != -1.0) sum += d;
         }
    System.out.println("sum = " + sum);
}}
```

Output to a File

- Can write output to a file instead of to your screen

<output-file-decl> →PrintWriter

<stream-name> = new PrintWriter(

new FileWriter("<filename> "));

```
PrintWriter fileout = new
  PrintWriter(new FileWriter("out1"));
```

- Can use PrintWriter same as System.out; has all the methods: println, print
- Complication of file output: buffering

Buffering

- Computer, for efficiency, is actually writing to a buffer in memory which is then transferred to a file, automatically
- Must make sure all output is transferred to the file eventually; do a "flush" operation
- Buffer is flushed when the file is closed.
  Always close your file.

<close-invoke> →<stream-name>.close( );

```
fileout.close( );
```

Read+Write with Files

```
import java.io.*;
import cs111.io.*;
class SumwithFiles extends Object{
public static void main(String[] args) throws
   IOException {
//sums a sequence of numbers on a file ended by -1.0
   TokenStream infile = new TokenStream("numbers");
   PrintWriter outf = new PrintWriter (new FileWriter
   ("answer"));
   double sum = 0.0, d = 0.0;
   for (;d != -1.0;){
          d = infile.readDouble();
          if (d != -1.0) sum += d;
   }
   outf.println("sum = " + sum);
   outf.close();
}}
```

sumfileio.java

Exceptions

- **What are exceptions?**
  - **Unexpected events during execution**
  - **Java has many predefined exceptions, especially in I/O classes**
    - **File not found, `FileNotFoundException`**
    - **No more data, `EOFException`**
    - **Disk problem, `IOException`**
    - **Problem on network, `InterruptedIOException`**
  - **Exceptions can be defined by user**

Exceptions

- **Why have them?**
  - **To cope with unusual conditions so program can continue execution**
- **How handled?**
  - **Locally - have exception handler code within method that raises the exception**
  - **Externally - pass exception up call chain to caller method for handling**
    - **Results in flow of control between methods**

Exceptions

- **Method must tell Java compiler if it handles an exception**
  - **To pass exception up to caller method, must use throws clause in method declaration**

```
public static void main (String[ ] args)
   throws IOException {
```

  - **To handle exception with a local catch clause leave throws clause out of method declaration**

```
public static void main (String [ ] args) {
```

Syntax of Try, Catch, Finally

<block> → { <statements> }

<try-block> → try <block>

[<catch-list>] [<finally-block>]

<catch-list> → <catch-clause>

<catch-list> → <catch-clause> <catch-list>

<catch-clause> →catch

(<except-type> <identifier>) <block>

<finally-block> → finally <block>

Each catch clause must handle a different exception.

Local Catch: EOFException

sumexcept.java

```
import java.io.*;
import cs111.io.*;
class Sumexcept extends Object{
public static void main(String[] args) throws IOException {
// sums a sequence of numbers entered from the keyboard
  TokenStream inp = new TokenStream();
  double sum = 0.0; double d;
  System.out.println(" Enter numbers to be summed ");
  try{ //user types control-d to signal end of input
       for (;;){//indefinite loop or loop forever
         d = inp.readDouble();//readDouble throws IOException
         sum += d;}   //so main needs throws clause
        }
  catch (EOFException e) {System.out.println("sum = " + sum);}
}}
```

How catch works?

- If no exception generated within *try*
  - *Try* completes
  - Execution continues after last *catch*
- If exception is generated within try
  - Check for matching *catch*, in order, after the try
  - If match, execute that *catch* clause and resume execution after last *catch*
  - If no match, throw exception to caller

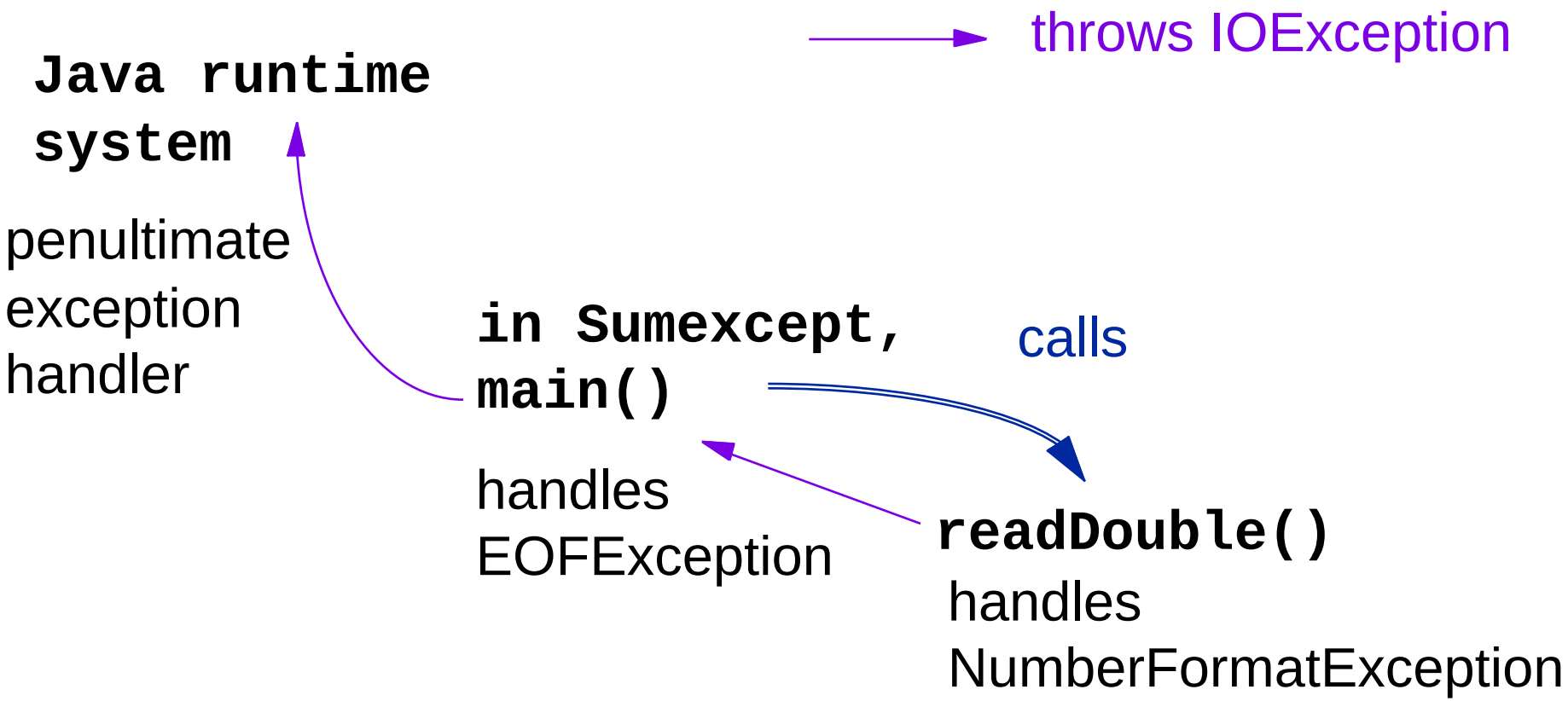
External Catch: IOException

from TokenStream class

```
public double readDouble( ) throws IOException{
...
try{//code to read double numbers off the keyboard
   }
catch (NumberFormatException e){
   System.out.println ("item + " is an invalid double,
   try again.");
   ...
}
...
}
```

```
50 romulus!ryder> java Sumexcept
 Enter numbers to be summed
Input an double: 1 2 3 x
x is an invalid double, try again.
Input an double: 1 2 3 4
Input an double: sum = 16.0
51 romulus!ryder>
```

Dynamic Handling



Summary of Handling

- **Method which detects problem either issues a *throw* or handles exception with a *catch***
- **Method that catches the exception is a dynamic choice, depending on call chain**
- **What happens when exception is raised?**
  - **If handled locally, continue execution normally**
  - **If thrown, abort execution and look for handler in caller; continue up call chain until find handler**
  - **Java runtime system is final handler**

Finally

- ***Finally*** **block is used to do cleanup after all other processing associated with the *try* is executed**
- ***Finally*** **is executed whether or not exception is thrown**
  - **e.g., to close a stream**

Use of Try

- **Textbook example 4.6 uses two constructs**
  - ***Try* inside an indefinite for:** **for (; ;) { try {...} ... }**
    - **Exceptions possible but not necessary *FileNotFoundException***
    - **Need way to break out of for loop: *break, continue***
  - **For inside of a *try*:** **try{ for (; ;) {...} ...}**
    - **Exceptions required *EOFException***
    - **Continue until raise the exception, then end *try* and return result**

Example of Loop Nesting

Overall structure:

- **Outer loop reads a sequence of filenames off another file and creates a new input stream for each filename**
- **Inner loop reads data from each file until encounters -1.0 marker and then computes sum**
- **Try-block ends execution gracefully when no more files are left to read from**

Initialization

DoubleLoopInput.java

```
import java.io.*;
import cs111.io.*;
class SumUsingFiles extends Object{
public static void main(String[] args) throws
   IOException {
// reads in a file name and then sums the integers on
// that file

//inputfiles contains all the filenames as strings
    TokenStream infile = new TokenStream("inputfiles");
//all output to go to file answers
    PrintWriter outf = new PrintWriter (new FileWriter
       ("answers"));
    String file;
```

Try Block and Outer Loop

```
try{ while (true){//declare new file to read numbers
      file = new String(infile.readString());
      System.out.println(" file is " + file);
      TokenStream inp = new TokenStream(file);
      double d = inp.readDouble();
      System.out.println("number =" + d);
      double sum = 0.0;
//********inner loop**********
      System.out.println("sum of numbers on file= "
            + sum);
      //when leave loop have ended the numbers
      outf.println("sum of numbers on file= " +
            sum);
      } }
catch (EOFException e) {
      System.out.println("reached eof");
      outf.close();
      }
```

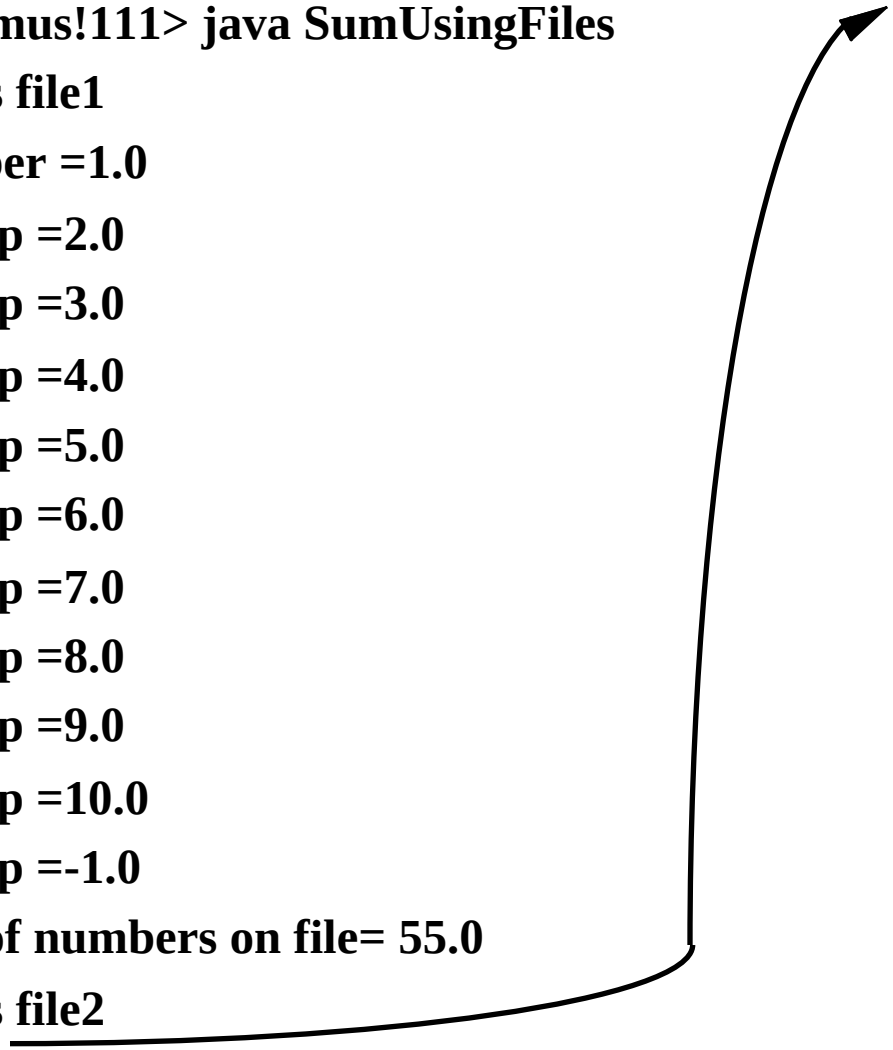
Inner Loop

```
while (d != -1.){
      sum += d;
      d =inp.readDouble();
      System.out.println("in loop =" + d);
 }
```

Contents of file1: 1 2 3 4 5 6 7 8 9 10 -1.
Contents of file2: 10 20 30 40 50 60 -1.
Contents of file3: 100 200 300 -1.
Contents of inputfiles: file1
file2
file3

Output

```
05 remus!111> java SumUsingFiles
 file is file1
number =1.0
in loop =2.0
in loop =3.0
in loop =4.0
in loop =5.0
in loop =6.0
in loop =7.0
in loop =8.0
in loop =9.0
in loop =10.0
in loop =-1.0
sum of numbers on file= 55.0
 file is file2
number =10.0
in loop =20.0
in loop =30.0
in loop =40.0
in loop =50.0
in loop =60.0
in loop =-1.0
sum of numbers on file= 210.0
 file is file3
number =100.0
in loop =200.0
in loop =300.0
in loop =-1.0
sum of numbers on file= 600.0
reached eof
106 remus!111>
```



Alternative Design

- **for inner loop, if we encoded a count of number of inputs on each file, in front of the actual inputs**

```
for (int i = inp.readInt( );  i == 0;  i - -) {
                d = inp.readDouble( );
                sum += d;
                System.out.println("in loop =" + d);
        }
```

This gives structure : while (true) { ... for (...) {} ...}