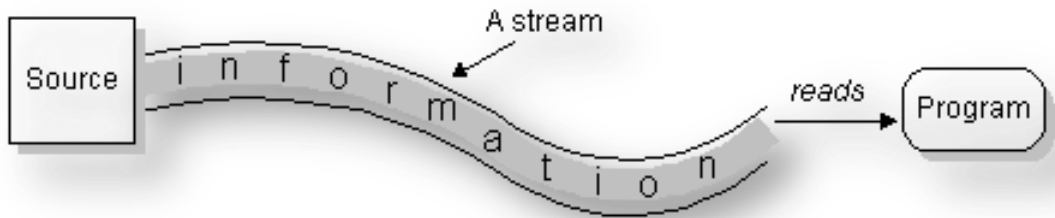


FILE I/O

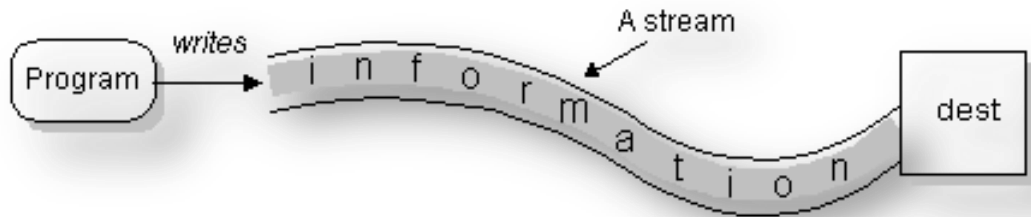
Arka Prokash Mazumdar

Stream

- INPUT



- OUTPUT



Stream

- Unix
 - I/O is stream of bytes
- Java
 - Whole number of bytes
- Here: byte ->character
- ***Stream***: an object that either
 - delivers data to its destination (screen, file, etc.) or
 - that takes data from a source (keyboard, file, etc.)
- Acts as a buffer between the data source and destination

Stream

- ***Input stream***: a stream that provides input to a program
 - **System.in** is an input stream
- ***Output stream***: a stream that accepts output from a program
 - **System.out** is an output stream
- A stream connects a program to an I/O object
 - **System.out** connects a program to the screen
 - **System.in** connects a program to the keyboard

Character

- In Languages prior to Java
 - Character = 8bits
 - ASCII character set
- In Java
 - UNICODE
 - 16 bit

I/O overview

- In this context it is input to and output from programs
- Input can be from keyboard or a file
- Output can be to display (screen) or a file
- Advantages of file I/O
 - permanent copy
 - output from one program can be input to another

Files

- All data and programs are ultimately just zeros and ones
 - Each digit can have one of two values, hence binary
 - Bit is one binary digit
 - Byte is a group of eight bits

Files (2)

- **Text files:** The bits represent printable characters
 - One byte per character for ASCII, the most common code
 - For example, Java source files are text files
 - So is any file created with a "text editor"

Files (3)

- **Binary files:** The bits represent other types of encoded information
 - For example, executable instructions or numeric data
- These files are easily read by the computer but not humans
- They are not "printable" files
 - Actually, you can print them, but they will be illegible
 - ***"Printable"*** means *"easily readable by humans when printed"*

Text Files

- Clean Text (**.txt**) files are the simplest kind of files
 - Text files can be used by many different programs
- Formatted text files (e.g. **.doc** files) also contain binary formatting information
 - Programs that know how to read these codes can use such files
- Compilers, in general, work only with text

What we already know

- Reading data from keyboard using scanner
 - `Scanner kybd = new Scanner(System.in);`
 - `width = kybd.nextInt();`

Advantage of Using Files

- Stored Data can be read from file
- Data can be stored permanently in file
- Data can be shared by multiple programs

File Class

- Part of **java.io**
- Does not operate on streams
 - Unlike other classes in java.io
- Describes the properties of a “file”
- Obtain / Manipulate information like
 - Permissions, date and time, directory path etc.
- Directory is treated as a File
 - List of filenames (additional property)

File Class

- Usages
 - File(String *directoryPath*)
 - File(String *directoryPath*, String *fileName*)
 - File(File *dirObj*, String *fileName*)
 - File(URI *uriObj*) (since Java 2, ver 1.4)
- Example
 - File D1 = new File("/");
 - File D2 = new File("/", "testFile.txt");
 - File D2 = new File(D1, "testFile.txt");

Reading from File

- Must import java.io.*
- Create a Scanner object to read from a file
 - Use the name of the file to read in place of System.in
 - File must be in same folder as Java program
 - Else, **checked exception** will occur
 - checked exception must be handled
 - can have any number of Scanner objects
 - one to read from each file
 - one to read from keyboard
- When processing is done: all files should be closed

- Scanner object methods are used to read from file in same way as they read from keyboard:
 - nextInt()
 - nextDouble()
 - next()
 - nextLine()
- hasNext() method:
 - returns true if there is more data to read
 - returns false when end of file is reached
 - can use hasNext() method in a while loop to process all data in a file

Example

```
//input ages from file and calculate average
import java.util.*;
import java.io.*;
public class AverageAge
{
    public static void main (String [] args)
    {
        double sum = 0;
        int count = 0;
        try
        {
            //create Scanner linked to input file
            Scanner in = new Scanner(new File("in.txt"));
            String name;
            double age;
```

```

//read until end of file reached
while (in.hasNext())
{
    name = in.next();
    age = in.nextDouble();
    sum = sum + age;
    count++;
}

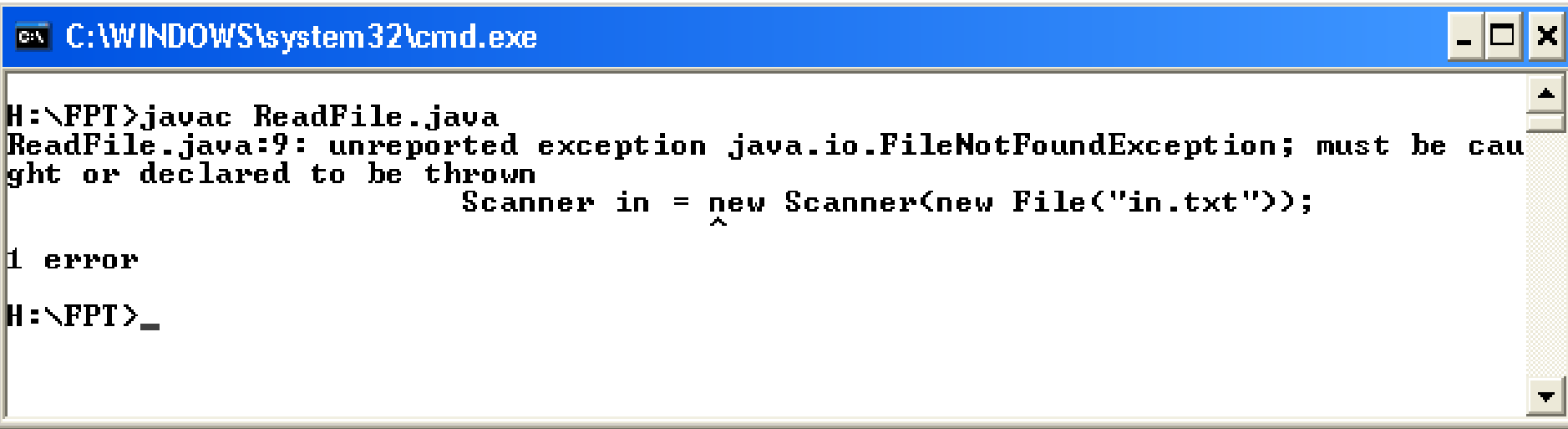
//close file
in.close();

double average = sum / count;
System.out.println("The average age is: " + average);
}

//handle any errors
catch(ArithmeticException e)
{
    System.out.println("Cannot calculate average - no data");
}
catch(IOException e)
{
    System.out.println("Error reading from file");
}
}
}

```

Exception



A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe" along with standard window control buttons (minimize, maximize, close). The command prompt shows the following text:

```
H:\FPT>javac ReadFile.java
ReadFile.java:9: unreported exception java.io.FileNotFoundException; must be caught or declared to be thrown
        Scanner in = new Scanner(new File("in.txt"));
                        ^
1 error
H:\FPT>_
```

The error message indicates that the `FileNotFoundException` is not a checked exception and therefore does not need to be caught or declared in the method signature. The caret (^) points to the `new File` constructor call in the `Scanner` instantiation.

Buffering of I/O

- Not Buffered
 - Each byte is read from or written to disk
 - Some delay for each byte
 - High overhead
- Buffered
 - Chunk of bytes are accessed (read/write) in the disk
 - Disk operation: per buffer-size
 - Low overhead

Open a Stream

- Creating connection to external data
- After connecting, no need to remember the actual location or the file name
- A **FileReader** is used to connect to a file that will be used for input:

```
FileReader fileReader =  
    new FileReader(fileName);
```
- fileName refers to the name and the external location of the file.

- Example
- `FileReader file = new FileReader("C:\\testing.txt")`
- Above is Windows style declaration
- As `'\'` is the escape character, `"\\"` must be written to represent single `\`
- Linux style:
- `FileReader file = new FileReader("bin/ testing.txt ");`

Use the Stream

- Read OR Write
- May need to manipulate the data

```
int charAsInt;  
charAsInt = fileReader.read( );
```

- The `fileReader.read()` method reads one character at a time and
 - returns it as an `integer`,
 - or `-1` if there are no more characters to read

- Meaning of the data depends on the file encoding technique used in that file
 - ASCII , UNICODE, ...
 - E.g While reading an ASCII file, **97** will mean **“a”**
- The input can be typecast to **char**
char ch = (char)fileReader.read();

- Instead of “Not Buffered” method, it is preferred to use “Buffered” methods to read files
 - And a way to read files as a stream of characters
- A **BufferedReader** will read chunk of data at a time
 - convert integers to characters too

Reading Lines

- The constructor for **BufferedReader** takes a **FileReader** parameter:

```
BufferedReader bufferedReader =  
    new BufferedReader (fileReader) ;
```

Reading a Line:

```
String s;  
s = bufferedReader.readLine ( ) ;
```

- Return **null** if there is nothing more to read

Close the Stream

- Should close the file after work
- Use the close() method of BufferedReader class to close the stream

bufferedReader.close();

- If the program ends normally, it will close all the streams automatically

Then Why Bother Closing??

- There is a limit on the number of streams that you can have open at one time
- You must close a stream before you can open it again
- It could get damaged if left open and the program terminates abnormally

Writing to File

- must import `java.io.*`
- create a `PrintWriter` object to write to a file
 - need the name of the file to write to
 - file will be in same folder as Java program
 - if file already exists contents will be overwritten
 - if file doesn't exist, new one is created
 - checked exception must be handled
- can have any number of `PrintWriter` objects

Methods

- PrintWriter object methods are used to write data to file:
 - print()
 - println()
 - printf()

```
//input numbers and output odd ones to odd.txt, even ones to even.txt
import java.util.*;
import java.io.*;
public class OddsEvens
{
    public static void main (String [] args)
    {
        try
        {
            //create PrintWriters linked to output files
            Scanner kybd = new Scanner(System.in);
            PrintWriter even = new PrintWriter("even.txt");
            PrintWriter odd = new PrintWriter("odd.txt");
            int num = kybd.nextInt();

            //read until -1 entered
            while (num != -1)
            {
```



```
        if (num % 2 != 0)
            odd.println(num);
        else
            even.println(num);

        num = kybd.nextInt();
    }

    //close files
    odd.close();
    even.close();
}

catch(IOException e)
{
    System.out.println("Error writing to file"); }
}
}
```