UBC OKANAGAN

COSC 121
Computer Programming II

Text IO

Dr. Mostafa Mohamed

Acknowledgement: The slides mainly rely on the textbook of Y. D. Liang titled “Introduction to Java Programming, 10th Ed.”, Pearson Edu. Inc.

Outline

Previously:

- Java Exceptions

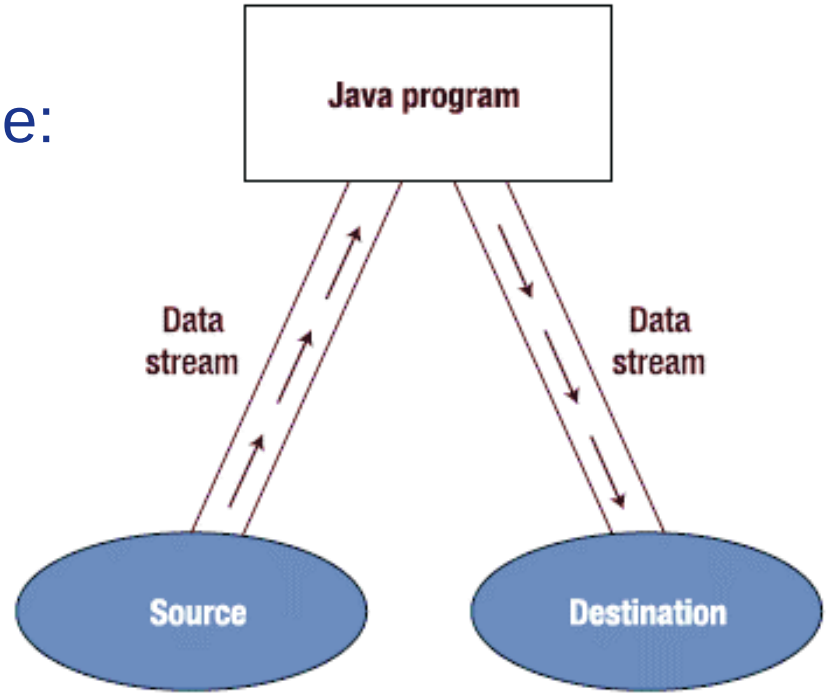
Today:

- Intro to Java I/O Streams
- Text I/O classes
  - `Scanner, PrintWriter`
  - `BufferedReader, BufferedWriter`
- The `File` Class
- Try-with-resources
- Reading from the Web

Input and output streams

A stream is a **sequence of bytes**, representing a **flow of data** from a **source** to a **destination**.

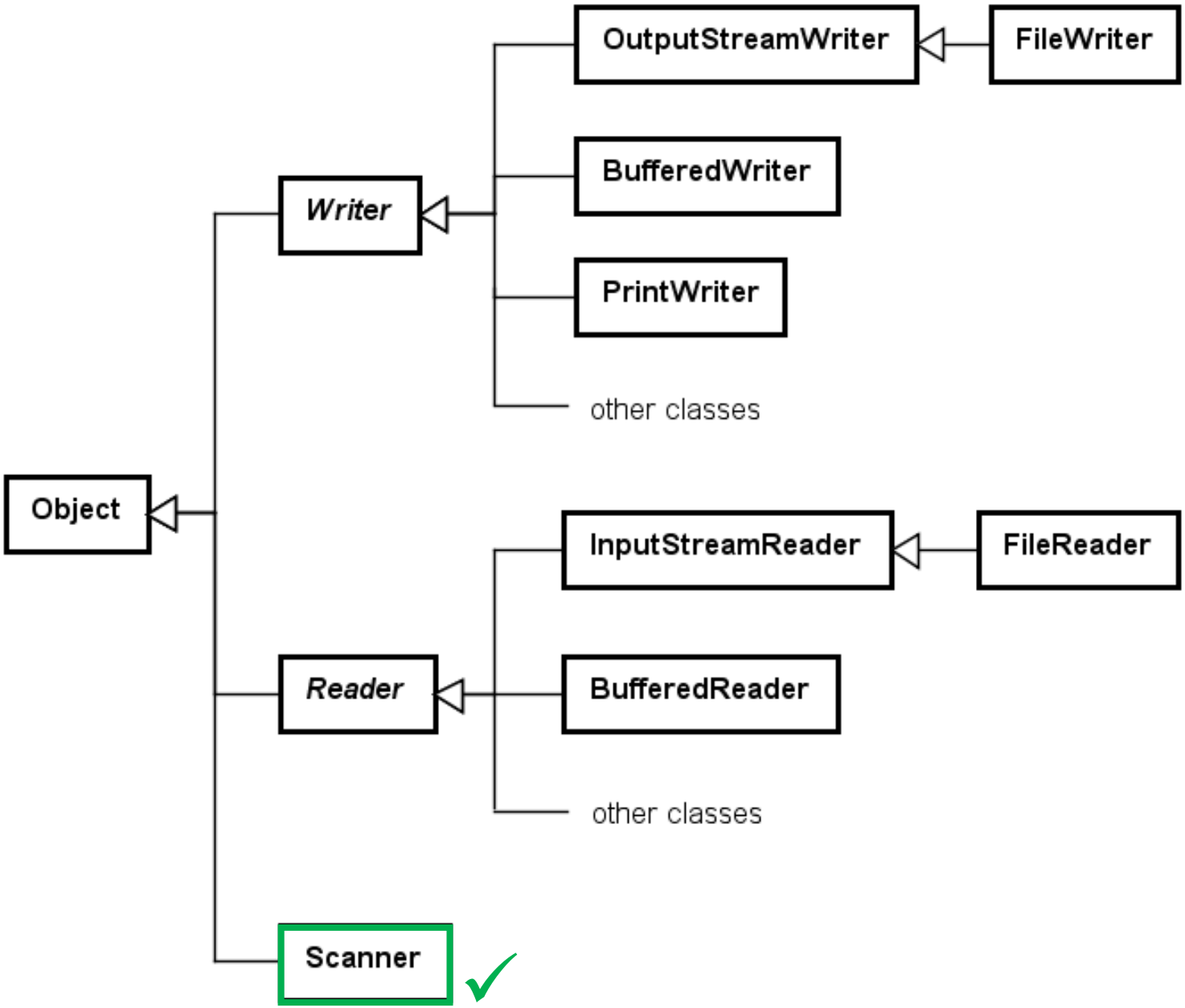
- Sources and destinations include:
  - keyboard, screen, data files, and networked computers.



Any read or write is performed in three simple steps:

1. Define an object of a class used to read or write from a stream
2. Read or write data using the methods of the above object.
3. Close the stream.

Text I/O Classes



Scanner

Reading text from the keyboard using `Scanner` class

Import
- java.util.* library that includes the Scanner class

Step1: create a Scanner object, use keyboard (System.in) as source

Step2: use the Scanner object to get data

Step3: close the Scanner when finished using it!

```
import java.util.*;
...
Scanner sc = new Scanner(System.in);
String name = sc.next();
int age = sc.nextInt();
System.out.println("you wrote:" + name+ "\n" +age);
sc.close();
```

Scanner, cont.

Reading text from a file using Scanner class

Import :
- java.util.* to use Scanner class
- java.io.* to use File class

Step1a: create a File object that points to your text file.

Step1b: create a Scanner object, use the file object as the source

Step2: use the Scanner object to get data

Step3: close the Scanner when finished using it!

```
import java.util.*;
import java.io.*;
...
File myFile = new File("C:/testing.txt");
Scanner sc = new Scanner(myFile);
String s = sc.nextLine();
System.out.println(s);
sc.close();
```

Note that for this code to work properly, we need to write some **error handling code** to handle IO exceptions.

Scanner, cont.

Common Scanner methods

input methods:

next() Returns next "token"

nextLine() Returns an entire input line as a String.

nextInt() Returns next integer value.

nextDouble() Returns next double value.

nextXYZ() Returns value of type XYZ (primitive value if possible), where XYZ is one of BigDecimal, BigInteger, Boolean, Byte, Float, or Short.

useDelimiter(pattern: String): Sets this scanner's delimiting pattern and returns this scanner.

when reading a token using next(), the token is identified by the scanner delimiter.

the default delimiter is space " ".

Scanner, cont.

Common Scanner methods

test methods (used for error checking and loops):

hasNext() True if another token is available to be read.

hasNextLine() True if another line is available to be read.

hasNextInt() True if another int is available to be read.

hasNextDouble() True if another double is available to be read.

hasNext*XYZ*() XYZ stands for an input type.

Source: http://leepoint.net/notes-java/summaries/summary-scanner.htm

Scanner, cont.

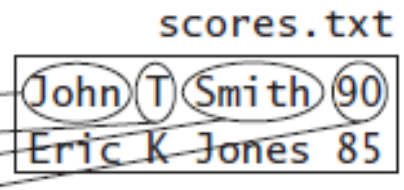
```
import java.util.Scanner;

public class ReadData {
  public static void main(String[] args) throws Exception {
    // Create a File instance
    java.io.File file = new java.io.File("scores.txt");

    // Create a Scanner for the file
    Scanner input = new Scanner(file);

    // Read data from a file
    while (input.hasNext()) {
      String firstName = input.next();
      String mi = input.next();
      String lastName = input.next();
      int score = input.nextInt();
      System.out.println(
        firstName + " " + mi + " " + lastName + " " + score);
    }

    // Close the file
    input.close();
  }
}
```



Caution!

Not having the right format in your text file may lead to errors. For example, both programs below generate exceptions.

- Can you explain why??

```
File source = new File("test1.txt");
Scanner in = new Scanner(source);
while(in.hasNext()) {
    System.out.println(in.next());
    System.out.println(in.nextInt());// Exception
    System.out.println(in.nextInt());
}
```

test1.txt

```
John 23 10
Lili 26 15
Mark 12 1
Bill
```

```
File source = new File("test1.txt");
Scanner in = new Scanner(source);
while(in.hasNextLine()) {
    System.out.println(in.next()); // Exception
    System.out.println(in.nextInt());
    System.out.println(in.nextInt());
}
```

test2.txt

```
John 23 10
Lili 26 15
Mark 12 1

```

there is a newline here!

Solution

Solution 1: make sure you have the right format

- Not good enough. You need to protect your code from crashing!

Solution 2: use defensive programming or exception handling.

Defensive Programming

```
File source = new File("test1.txt");
Scanner in = new Scanner(source);
while(in.hasNextLine()) {
      if(in.hasNext())
            System.out.println(in.next());
      if(in.hasNextInt())
            System.out.println(in.nextInt());
      if(in.hasNextInt())
            System.out.println(in.nextInt());
}
```

Exception Handling

```
File source = new File("test1.txt");
Scanner in = new Scanner(source);
try{
   while(in.hasNextLine()) {
        System.out.println(in.next());
        System.out.println(in.nextInt());
        System.out.println(in.nextInt());
   }
} catch(NoSuchElementException e) {…}
```

PrintWriter

Writing text to a file using `PrintWriter` class

Import
- java.io.* to use the PrintWriter class

Step1: create a File object that points to your text file

Step1b: create PrintWriter object, use the file object as destination

Step2: use the PrintWriter object to write text

Step3: close the PrintWriter when finished using it!

```
import java.io.*;
. . .
File myFile = new File("C:/aaa.txt");
PrintWriter pr = new PrintWriter(myFile);
pr.println("aaaaaa");
pr.println("bbbbbb");
pr.close();
```

- You can use print(), println(), and printf() methods
  - data in aaa.txt will be overwritten.
- You need to write some **error handling code** to handle the situation when the file cannot be created.

PrintWriter, cont'd

Print into a file:

```
PrintWriter out = new PrintWriter(new File("test.txt"));
out.println("ABC");
out.close();
```

Print into a file (same as above):

```
PrintWriter out = new PrintWriter("test.txt");
out.println("ABC");
out.close();
```

Print to screen (standard output)

```
PrintWriter out = new PrintWriter(System.out);
out.println("ABC");
out.close();
```

BufferedReader

using BufferedReader

The **BufferedReader** wraps **another reader class** and improves performance.

- (1) **InputStreamReader** to get input from keyboard as bytes and converts it to characters
- (2) **FileReader** to get input from character files
  - FileReader is a subclass of InputStreamReader

The **BufferedReader** as a wrapper

- provides a **readLine** method to read a line of text.
- **Improves efficiency (faster than Scanner)**

Java uses similar “writer” classes for writing data.

- OutputStreamWriter, FileWriter, BufferedWriter

BufferedReader with InputStreamReader.

Ex1: reading input from keyboard using **InputStreamReader**

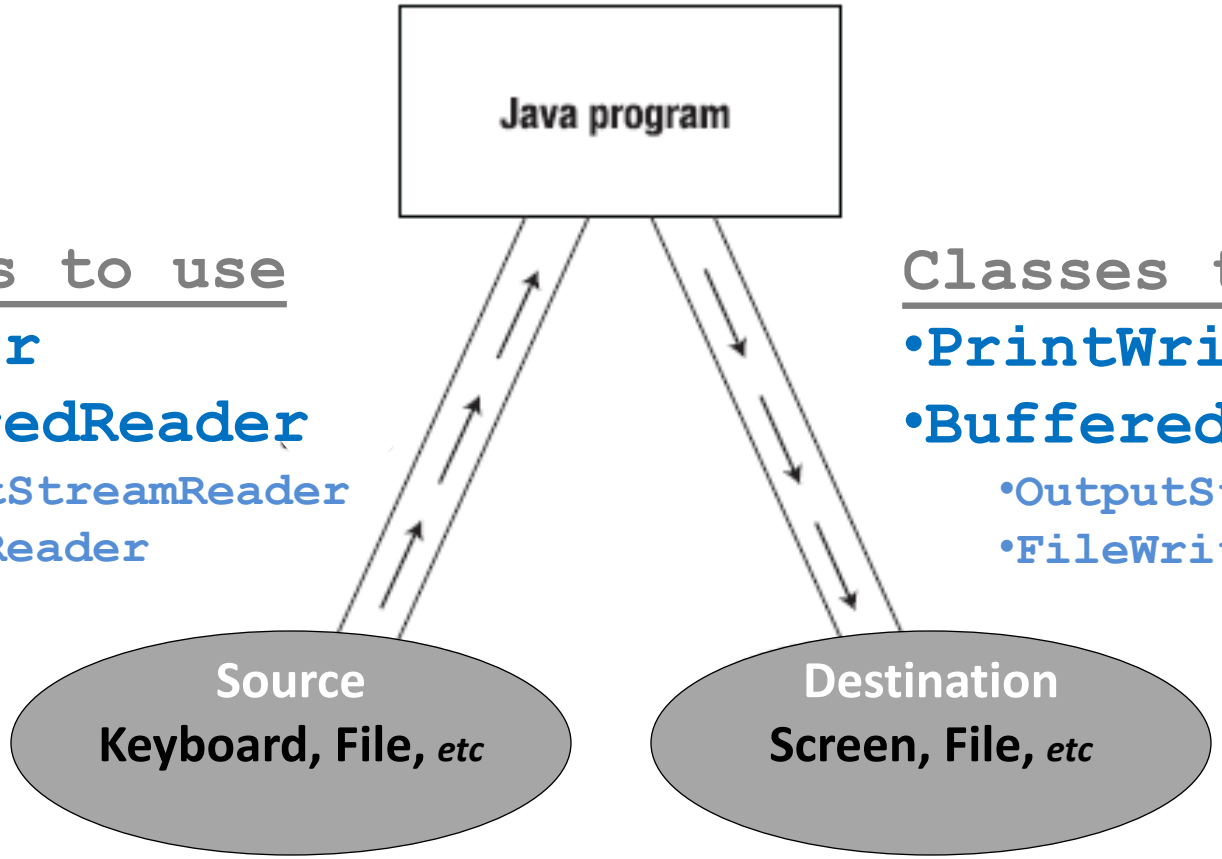
```
import java.io.*;
public class ex2 {
    public static void main(String[] args) throws IOException{
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader in = new BufferedReader(isr);
        System.out.print("What is your name? ");
        String name = in.readLine();
        System.out.println("Welcome " + name);
        in.close();
    }
}
```

BufferedReader with FileReader.

Example2: reading input from a File using FileReader

```
import java.io.*;
public class ex2 {
    public static void main(String[] args) throws IOException{
        FileReader fr = new FileReader("c://abc.txt");
        BufferedReader in = new BufferedReader(fr);
        String s;
        while((s = in.readLine()) != null)
            System.out.println(s);
        in.close();
    }
}
```

Text I/O



Classes to use

- Scanner
- BufferedReader
  - InputStreamReader
  - FileReader

Classes to use

- PrintWriter
- BufferedWriter
  - OutputStreamWriter
  - FileWriter

Text I/O, cont.

READING	From Keyboard	From File

`Scanner`	`Scanner in = new Scanner(X);`
	`myString = in.next(); myInt = in.nextInt(); //etc`
	`in.close();`	

Replace X with:	`System.in`	`new File("C:/filename")`

`BufferedReader`	`BufferedReader in = new BufferedReader(X);`
	`myString = in.readLine();`
	`in.close();`	

Replace X with:	`new InputStreamReader(System.in)`	`new FileReader("C:/filename")`

Text I/O, cont.

WRITING	To Screen	To File

PrintWriter	`PrintWriter out = new PrintWriter(X);`
	`out.println("some text here");`
	`out.close();`	

Replace X with:	`System.out`	`new File("C:/filename")`

BufferedWriter	`BufferedWriter out = new BufferedWriter(X);`
	`out.write("some text here");`
	`out.close();`	

Replace X with:	`new outputStreamWriter(System.out)`	`new FileWriter("C:/filename")`
	OR `System.out.println("text");`	

The File Class

The File class provides an abstraction that deals with most of the machine-dependent complexities of files and path names in a machine-independent fashion. The filename is a string. The File class is a wrapper class for the file name and its directory path.

Some useful File methods:

- exists(): boolean
  - Returns true if the file or the directory exists.
- canRead(): boolean
  - Returns true if the file exists and can be read.
- isDirectory(): boolean
  - Returns true if the File object represents a directory.
- isFile(): boolean
  - Returns true if the File object represents a file

See other methods in the textbook.

Try-with-resources

Programmers often forget to close the file. JDK 7 provides the followings new try-with-resources syntax that automatically closes the files.

```
try (declare and create resources) {
  Use the resource to process the file;
}
```

Try-with-resources

```
import java.io.*;
public class WriteDataWithAutoClose {
    public static void main(String[] args) throws Exception {
        File file = new File("scores.txt");
        if (file.exists()) {
            System.out.println("File already exists");
            System.exit(0);
        }
        try (PrintWriter output = new PrintWriter(file);) {
            output.print("John T Smith ");
            output.println(90);
            output.print("Eric K Jones ");
            output.println(85);
        }
    }
}
```

Reading from the Web

After a URL object is created, you can use the openStream() method defined in the URL class to open an input stream and use this stream to create a Scanner object as follows:

```
URL url = new URL("http://www.google.ca/html");
Scanner input = new Scanner(url.openStream());
  //you can also use BufferedReader with InputStreamReader
```

```
import java.net.URL;
import java.util.Scanner;
public class readweb {
    public static void main(String[] args) throws Exception{
        URL url = new URL("http://www.google.ca/index.html");
        Scanner input = new Scanner(url.openStream());
        while(input.hasNextLine())
            System.out.println(input.nextLine());
        input.close();
    }
}
```

Idea to Check the Validity of Input TYPE

```
boolean done = false;// assume input is invalid
while(!done){              // keep repeating as long as input is INVALID
    try{                     // try to read the input
        /*read input*/     // will throw InputMismatchException if input is invalid
        done = true;       // reached only if previous statement doesn't through
                             an exception. Once valid is true, the loop will stop!
    }catch(InputMismatchException e){
        /* display an error message */
        /* use Scanner readLine to discard remaining parts of the input)
    }
}
/* Use your input */              //we are sure we have the right input here.
```

Note: InputMismatchException is thrown by Java when the input does not match the type. It can also be explicitly thrown by you whenever the input doesn't match your own criteria, e.g.,

```
if(age <= 0) throw new InputMismatchException();
```

Example 1

Use a mix of defensive programming (for dealing with –ve values) and exception handling (for dealing with non-integers)

```
Scanner in = new Scanner(System.in);
System.out.println("How old are you?");
boolean done = false;
while(!done) {
   try {
      int age = in.nextInt();
      while(age<0) {
         System.out.println("Age cannot be negative. try again!");
         age = in.nextInt();
      }
      System.out.println("You are " + age);
      done = true;
   }catch(InputMismatchException e) {
      System.out.println("Age must be an integer. Try again");
      in.nextLine();
   }
}
in.close();
```

Example 2

The following code assumes the user enters valid integer values. Modify the code so that the code ensures that the denominator is not equal to zero.

```
import java.util.Scanner;

public class Q1 {
    public static void main(String[] args) {
        int a, b;
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the numerator: ");
        a = input.nextInt();
        System.out.print("Enter the denominator: ");
        b = input.nextInt();
        System.out.printf("Integer division of %d / %d is %d\n", a, b, a/b);
        input.close();
    }
}
```

Defensive Programming Solution

This code ensure **valid range**, but **does not check the type** (e.g. what if the user enters a string, not an integer?)

```
import java.util.Scanner;

public class Q1DefensiveProg {
    public static void main(String[] args) {
        int a, b;
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the numerator: ");
        a = input.nextInt();
        System.out.print("Enter the denominator: ");
        b = input.nextInt();
        while(b == 0){
            System.out.print("Denominator can't be zero. Try again: ");
            b = input.nextInt();
        }
        System.out.printf("Integer division of %d / %d is %d\n", a, b, a/b);
        input.close();
    }
}
```

Exception Handling Solution

Exception handling is used to ensure a/b is valid; **the type is still not checked**

```
Scanner input = new Scanner(System.in);
System.out.print("Enter numerator: ");
int a = input.nextInt();
// get denominator and make sure it is not zero
boolean done = false;
while (!done) {
    try {
        System.out.print("Enter denominator: ");
        int b = input.nextInt();
        System.out.printf("%d/%d=%d", a, b, a / b);
        done = true;
    } catch (ArithmeticException e) {
        System.out.println("Denominator can't be zero. Try again!");
    }
}
input.close();
```

Exception Handling Solution, *cont'd*

Exception handling can also be used to ensure input is valid (**range and type**)

```
Scanner input = new Scanner(System.in);
boolean done = false;
while (!done) {
    try {
        System.out.print("Enter numerator: ");
        int a = input.nextInt();
        System.out.print("Enter denominator: ");
        int b = input.nextInt();
        System.out.printf("%d/%d=%d", a, b, a / b);
        done = true;
    } catch (ArithmeticException e) {
        System.out.println("Denominator can't be zero. Try again!");
    } catch (InputMismatchException e) {
        System.out.println("You must enter an integer. Try again!");
        input.nextLine();
    }
}
input.close();
```