

Exceptions and File I/O

CS111 Computer Programming



Department of Computer Science
Wellesley College

Exceptions in Java

- o For **reliability**: a means of describing and recovering from run time errors.
- o When unusual condition arises, program **throws** an exception. Code that responds to the situation **catches** the exception.
- o Exceptions are objects whose classes inherit from the `Throwable` class.
- o **Unchecked exceptions** (subclasses of `Error` or `RuntimeException`)
- o **Checked exceptions** must be acknowledged.

List Exceptions

```
class ListException extends RuntimeException
{
    public ListException (String message)
    {
        super(message);
    }
}

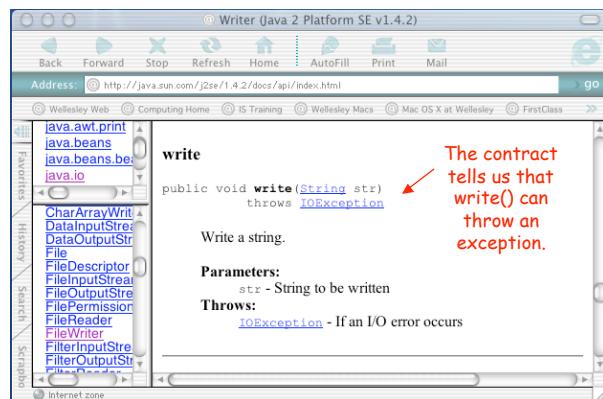
class EmptyIntList extends IntList
{
    ...
    public int head ()
    {
        throw new ListException("Cannot take head of empty list");
    }

    public IntList tail ()
    {
        throw new ListException("Cannot take tail of empty list");
    }
    ...
}
```

Exceptions and File I/O 18-3

When good programs go bad

Methods in java use **exceptions** to tell the calling code,
"Something bad happened. I failed."



Exceptions and File I/O 18-4

Risky business

The screenshot shows the DrJava IDE interface. In the code editor, there is a Java file named 'WriteAFile3.java' with the following content:

```
// page 425 of Head First Java
import java.io.*;
class WriteAFile3 {
    public static void main (String[] args) {
        FileWriter writer = new FileWriter ("votes.txt");
        writer.write("George W. Bush\n");
        writer.write("John F. Kerry\n");
        writer.close();
    }
}
```

In the 'Compiler Output' tab, it shows:

4 errors found:
File: /Users/rshull/Desktop/io/WriteAFile3.java [line: 8]
Error: unreported exception java.io.IOException; must be caught
or declared to be thrown
File: /Users/rshull/Desktop/io/WriteAFile3.java [line: 10]
Error: unreported exception java.io.IOException; must be caught

At the bottom right of the IDE window, it says 'Exceptions and File I/O 18-5'.

Humoring Java

- o The compiler needs to know that YOU know you're calling a risky method.
- o You can pass the problem along to your caller
- o Or you can wrap the risky code in something called a **try/catch** to make the compiler relax.

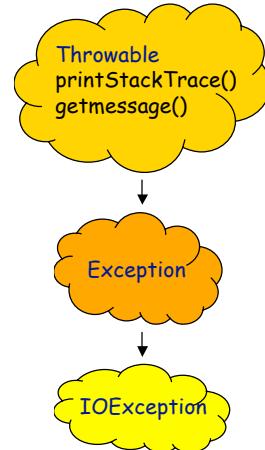


Exceptions and File I/O 18-6

An exception is an object*



```
try {  
  
    // do risky thing  
  
} catch (Exception ex) {  
  
    // try to recover  
    // block runs only  
    // if Exception is  
    // thrown  
  
}
```



*Of type `Exception`.

Exceptions and File I/O 18-7

Example: Catching an exception

```
public static isMember(int n, IntList lst)  
{  
    try {  
        return (n == head(lst)) || isMember(tail(lst));  
    } catch (ListException e) {  
        return false;  
    }  
}
```

*This is generally considered poor programming practice.

Exceptions and File I/O 18-8

File I/O: Election 2008

- o Electronic voting is very much in the news.
- o After the citizens vote, the results must be written to a file for safe keeping.
- o How do we do this?



Exceptions and File I/O 18-9

Writing a String to a text file

```
import java.io.*; ← FileWriter  
                      lives in  
                      java.io package  
  
class WriteAFile  
{  
    public static void main (String[] args)  
    {  
        FileWriter writer = new FileWriter ("votes.txt");  
  
        writer.write( "George W. Bush\n" );  
        writer.write( "John F. Kerry\n" );  
  
        writer.close(); ← The write() method  
                      takes a String  
    }  
} ← Tidy up when done
```

If the file "votes.txt" does not exist, FileWriter will create it

Exceptions and File I/O 18-10

Risky business

The screenshot shows the DrJava IDE interface. The code editor window contains a Java file named WriteAFile3 with the following content:

```
// page 425 of Head First Java
import java.io.*;
class WriteAFile3 {
    public static void main (String[] args) {
        FileWriter writer = new FileWriter ("votes.txt");
        writer.write("George W. Bush\n");
        writer.write("John F. Kerry\n");
        writer.close();
    }
}
```

The code editor has syntax highlighting. Below the code editor, the status bar displays "4 errors found". The error messages are:

- File: /Users/rshull/Desktop/io/WriteAFile3.java [line: 8]
Error: unreported exception java.io.IOException; must be caught or declared to be thrown
- File: /Users/rshull/Desktop/io/WriteAFile3.java [line: 10]
Error: unreported exception java.io.IOException; must be caught

To the right of the code editor is a "Compiler" panel set to "javac" with a checked "Highlight source" option. The status bar at the bottom right of the IDE window also shows "javac".

Exceptions and File I/O 18-11

Guarding against failure

```
import java.io.*;

class WriteAFile
{
    public static void main (String[] args)
    {
        try {
            FileWriter writer = new FileWriter ("votes.txt");

            writer.write("George W. Bush\n");
            writer.write("John F. Kerry\n");

            writer.close();

        } catch (IOException ex) {
            System.out.println("Voter error. Inform Supreme Court.");
            ex.printStackTrace();
        }
    }
}
```

Exceptions and File I/O 18-12

BufferedWriter

```
import java.io.*;  
  
class WriteAFile  
{  
    public static void main (String[] args)  
    {  
        try {  
            BufferedWriter writer =  
                new BufferedWriter(new FileWriter ("votes.txt" ));  
  
            writer.write("George W. Bush\n");  
            writer.write("John F. Kerry\n");  
  
            writer.close();  
  
        } catch (IOException ex) {  
            System.out.println("Voter error. Inform Supreme Court.");  
            ex.printStackTrace();  
        }  
    }  
}
```

We chain a BufferedWriter
to FileWriter for more
efficient I/O.

Exceptions and File I/O 18-13

The beauty of buffers

File I/O without buffers is
like shopping without a cart.



String is put into
a buffer with
other Strings

When the buffer is full,
the Strings are all written to



is written to

is chained to

String

BufferedWriter
(chain stream that
works with characters)

FileWriter
(connection stream
that writes characters
as opposed to bytes)

File

Exceptions and File I/O 18-14

Reading from a text file

```
import java.io.*;
class ReadAFile
{
    public static void main (String[] args)
    {
        try {
            File myFile = new File( "votes.txt" );
            FileReader fileReader = new FileReader(myFile);
            BufferedReader reader = new BufferedReader(fileReader);

            String line = null; // holds each input line
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
            reader.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

FileReader is a connection stream for characters connecting to text file

Wrap in try/catch

Chain fileReader to BufferedReader for more efficient reading

Exceptions and File I/O 18-15

A magic eight ball

- o We write an "eight ball" program and answers yes/no questions.
- o The eight ball keeps responses like

Without a doubt.
It seems unlikely.
My sources say no.
It is certain.
Concentrate and ask again.

in a file named
answers.txt



Exceptions and File I/O 18-16

Magic8Ball

```
import java.awt.*;
import java.applet.*;
public class Magic8Ball
{
    public static void main (String[] args)
    {
        System.out.println(StringChooser.chooseLine("answers.txt"));
    }
}
```

↗
StringChoose.chooseLine()
reads responses from
"answers.txt", then
randomly returns one

Exceptions and File I/O 18-17

First we load our answers

```
import java.io.*;
import java.util.ArrayList;

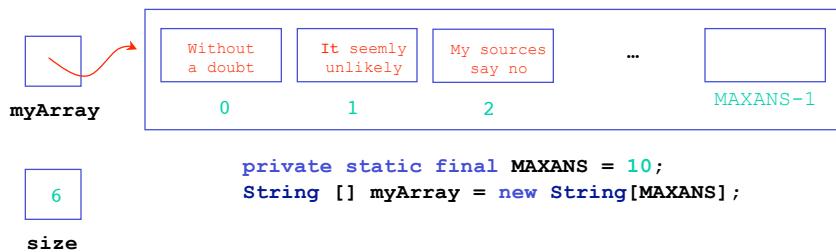
public class StringChooser
{
    public static String chooseLine (String inFile)
    {
        try {
            File myFile = new File(inFile);
            BufferedReader reader =
                new BufferedReader(new FileReader (myFile));

            // But where do we put them all?

            reader.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return "Ask me again tomorrow";      // Got to return something
    }
}
```

Exceptions and File I/O 18-18

An array might be nice*



*Why?

Exceptions and File I/O 18-19

Stuffing the answers into an array

```
public class StringChooser
{
    private static final MAXANS = 10;           // Max number of answers allowed
    public static String chooseLine (String inFile)
    {
        String [] myArray = new String[MAXANS]; // Stuff into a String array
        try {
            File myFile = new File(inFile);
            BufferedReader reader =
                new BufferedReader(new FileReader (myFile));

            String line = null;
            int size = 0;
            while ((line = reader.readLine()) != null) {
                myArray[size] = line;
                size++;
            }

            reader.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return "Ask me again tomorrow"; }
}
```

Exceptions and File I/O 18-20

Focus please!

```
...  
private static final MAXANS = 10; // Max number answers allowed  
...  
String [] myArray = new String[MAXANS];  
...  
String line = null;  
int size = 0;  
while ((line = reader.readLine()) != null) {  
    myArray[size] = line;  
    size++;  
}  
  
myArray [ ]  
Without a doubt    It seemly unlikely    My sources say no ...  
0                1                2                MAXANS-1  
size [ ]  
6
```

Exceptions and File I/O 18-21

Avoiding array out of bounds

```
...  
private static final MAXANS = 10; // Max number answers allowed  
...  
String [] myArray = new String[MAXANS];  
...  
String line = null;  
int size = 0;  
while (((line = reader.readLine()) != null) && (size < MAXANS)){  
    myArray[size] = line;  
    size++;  
}  
  
myArray [ ]  
Without a doubt    It seemly unlikely    My sources say no ...  
0                1                2                MAXANS-1  
size [ ]  
6
```

Exceptions and File I/O 18-22

Rolling the dice

```
public class StringChooser
{
    // Stuff from previous slides
    private static Randomizer rzer =
        new Randomizer(System.currentTimeMillis()); // Our dice

    public static String chooseLine (String inFile)
    {
        try {
            // Stuff in the try block to fill up the array

            return (String) (myArray[rzer.nextInt(0,size-1)]);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        // Rest of program
    }
}
```

Annotations:

- Red arrow pointing to `System.currentTimeMillis()`: Returns time in milliseconds for a random seed
- Red arrow pointing to `rzer.nextInt(0,size-1)`: Returns a random integer between and size-1

Exceptions and File I/O 18-23

Pick a number

```
import java.util.*;
public class Randomizer
{
    // Instance Variable:
    private Random rand;

    // Constructor Method:
    public Randomizer (long seed)
    {
        rand = new Random(seed); // Set seed for repeatability.
    }
    public int intBetween (int lo, int hi)
    {
        int result = lo + (Math.abs(rand.nextInt()) % (hi + 1 - lo));
        return result;
    }
}
```

Exceptions and File I/O 18-24

The dice are tossed

```
private static Randomizer rzer =
    new Randomizer(System.currentTimeMillis());

private static chooseLine (String filename)
{
    // lots of stuff in between
    return (String) (myArray[rzer.nextInt(0, size-1)]);
}

// meanwhile in the Randomizer class
public int intBetween (int lo, int hi)
{
    int result = lo + (Math.abs(rand.nextInt()) % (hi + 1 - lo));
    return result;
}
```

myArray
size

Without a doubt It seemly unlikely My sources say no ... MAXANS-1

0 1 2

Exceptions and File I/O 18-25

Putting it all together

```
public class StringChooser {
    private static final MAXANS = 10; // Max number of answers allowed
    private static Randomizer rzer =
        new Randomizer(System.currentTimeMillis()); // Our dice
    public static String chooseLine (String inFile)
    {
        String [] myArray = new String[MAXANS];
        try {
            File myFile = new File(inFile);
            BufferedReader reader =
                new BufferedReader(new FileReader (myFile));
            String line;
            int size = 0;
            while (((line = reader.readLine()) != null) && (size < MAXANS)) {
                myArray[size] = line;
                size++;
            }
            reader.close();
            return (String) (myArray[rzer.nextInt(0,size-1)]);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return "Ask me again tomorrow";
    }
}
```

Exceptions and File I/O 18-26