Chapter 11

Input / Output and Exception Handling

Reading and Writing Textfiles

- Data is often stored in files such as a text file
- We need to read that data into our program
- Simplest mechanism
  - Scanner class
  - First construct a FileReader object with the name of the input file.
  - Then use the FileReader to construct the Scanner

Input Files

- FileReader reader = new FileReader(“input.txt”);
- Scanner in = new Scanner (reader);
- Now use standard Scanner objects to read

Output Data

- Create an output file using PrintWriter
  - PrintWriter out = new PrintWriter(“output.txt”);
  - If the output files exits, it is emptied before output
  - If it doesn’t exist, it will be created
- Now use print and println methods to output
  - out.println(29.95);
  - out.println(new Rectanble(5,10,15,25);
  - out.println(“Hello World”);
  - Converts numbers to decimal string representations
  - Uses toString to convert objects to strings

Finished

- Close input
  - in.close()
- Close output
  - out.close()
  - Exist program without close may loose data

File Doesn't Exist

- Get a FileNotFoundException
- We need the following code

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

Example

```
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.Scanner;

public class LineNumberer
{
		public static void mian(String[] args)
				throws FileNotFoundException
		{
				Scanner console = new Scanner (System.in);

				System.out.println("Input file: ");
				String inputFileName = console.next();

				System.out.println("Output file: ");
				String outputFileName = console.next();
```

Example (cont)

```
        FileReader reader = new FileReader(inputFileName);
                Scanner in = new Scanner(reader);

                PrintWriter out = new PrintWriter(outputFileName);
                int lineNumber = 1;

                while (in.hasNextLine())
                {
                        String line = in.nextLine();
                        out.println("/* " + lineNumber + "*/ " + line);
                        lineNumber ++;
                }

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

File Name Contains Backslashes

- Windows file name
- C:\homework\input.dat

- Must use double backslashes

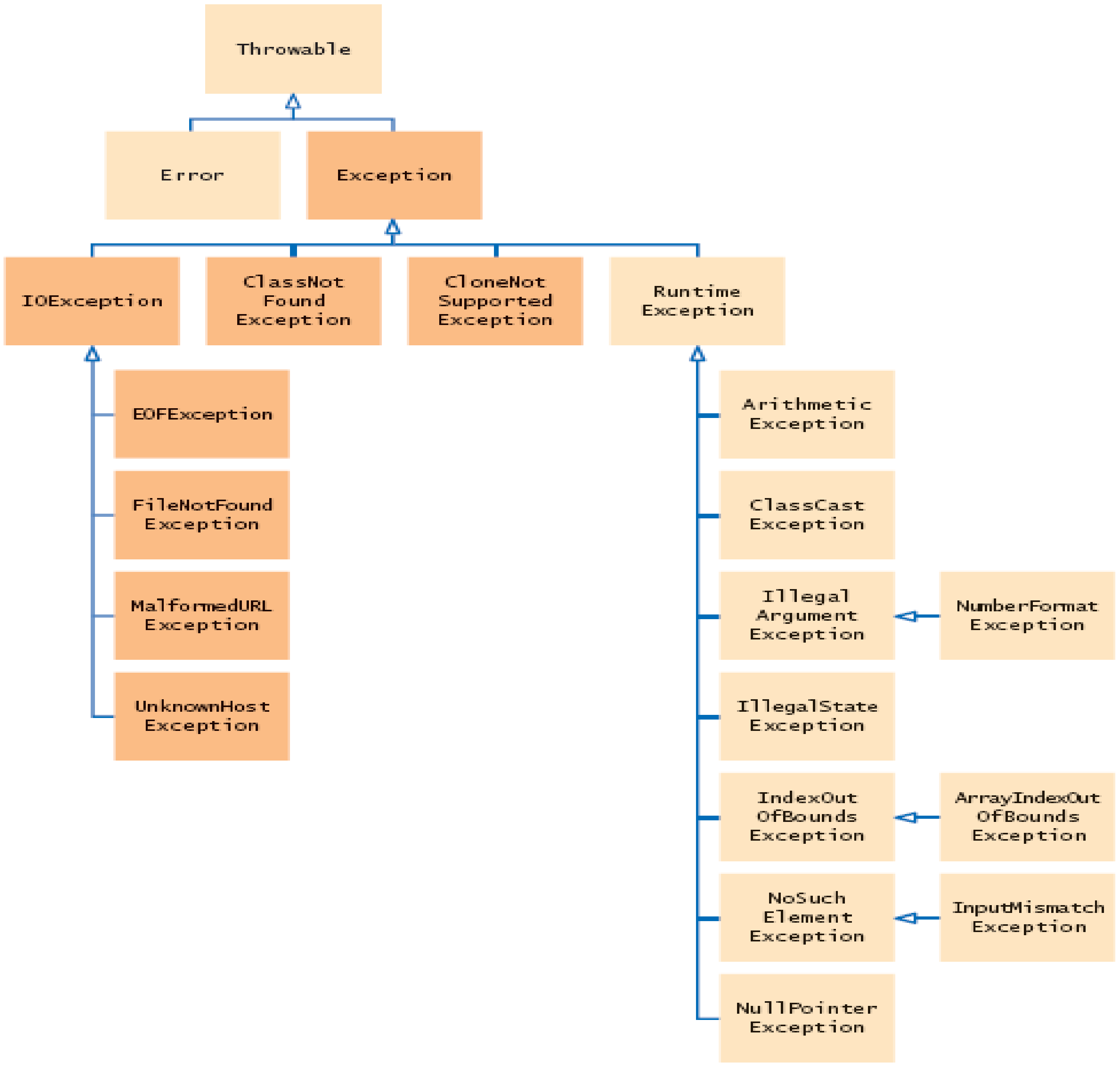
```
in = new FileReader(“c: \\homework\\input.data”);
```

Throwing Exceptions

- Two main aspects to exception handling
  - Reporting
  - Recovery
- The point of reporting is often far apart from the point of recovery
  - What do we do if we find a problem?

Exception Handling

- Flexible mechanism for passing control from the point of error reporting to a competent recovery handler.
- When you encounter an error condition you just throw an appropriate exception.
- Then what
  - Look for an appropriate exception class
  - Java provides many classes


Throwable
Error
Exception
IOException
ClassNot Found Exception
CloneNot Supported Exception
Runtime Exception
EOFException
FileNotFound Exception
MalformedURL Exception
UnknownHost Exception
Arithmetic Exception
ClassCast Exception
Illegal Argument Exception
NumberFormat Exception
IllegalState Exception
IndexOut OfBounds Exception
ArrayIndexOut OfBounds Exception
NoSuch Element Exception
InputMismatch Exception
NullPointer Exception

Example

```
public class BankAccount
{
    public void withdraw(double amount)
    {
            if (amount > balance)
            {
                IllegalArgumentException exception = new
                    IllegalArgumentException("Amount exceeds balance");
                throw exception;
            }
            balance = balance = amount:
            …………

            }
}
```

Other Options

- Instead of

```
IllegalArgumentException exception = new
        IllegalArgumentException("Amount exceeds balance");
throw exception;
```

- Can use

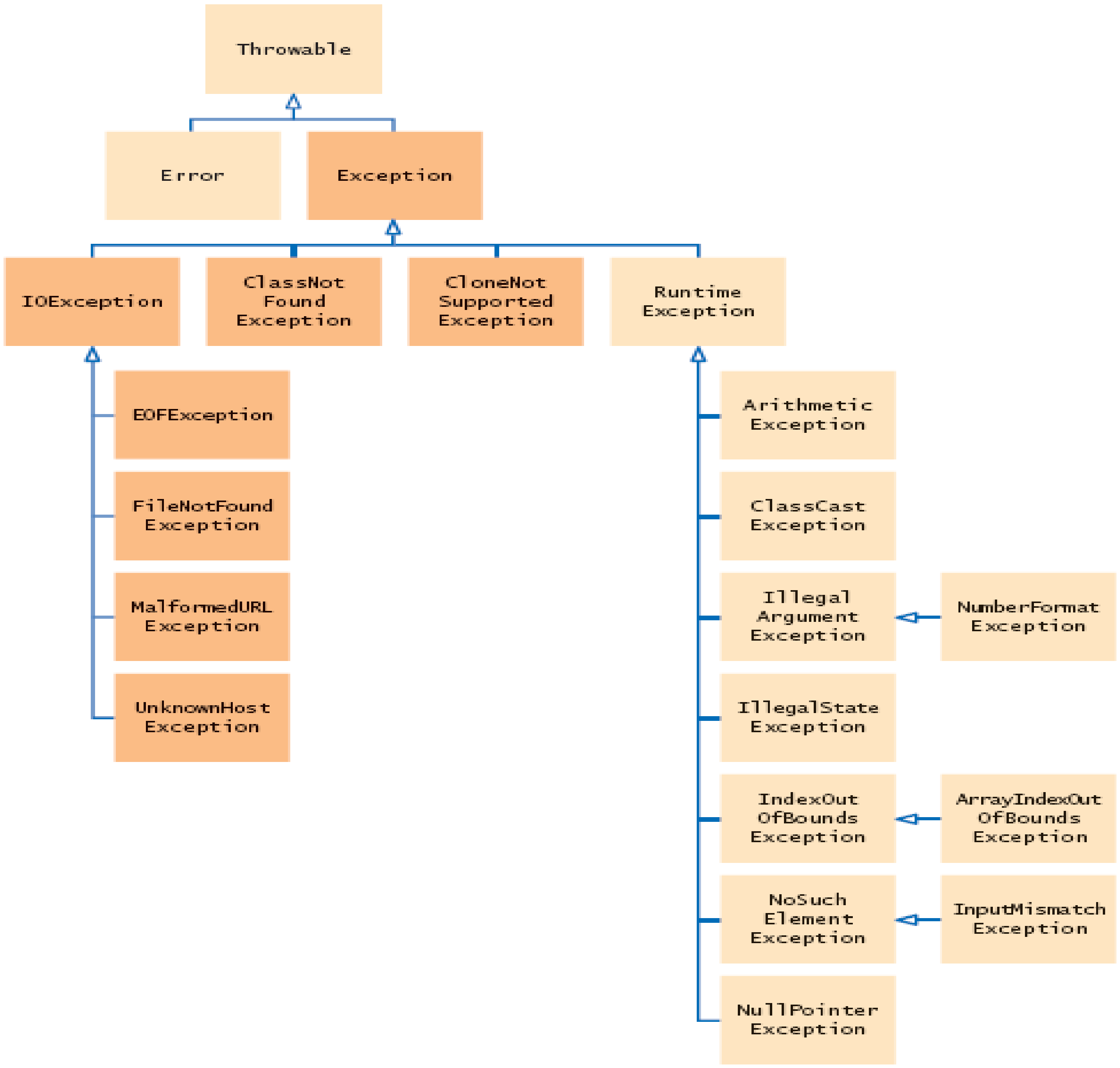
```
throw new IllegalArgumentException
                (“Amount exceeds balance”);
```

Checked and Unchecked Exceptions

- Checked exceptions
  - When you call a method that throws a checked exception, compiler checks that you don't ignore it.
  - You must tell the compiler what to do
  - Likely to occur at times – no matter how careful you are
- Unchecked Exceptions
  - Not required to handle
  - Considered your fault

Throws Clause

- Signals the caller that your method may encounter an exception.
- Your method may throw multiple exceptions
  - Separate by commas
- Be aware of the hierarchy of the exceptions.


Throwable
Error
Exception
IOException
ClassNot Found Exception
CloneNot Supported Exception
Runtime Exception
EOFException
FileNotFound Exception
MalformedURL Exception
UnknownHost Exception
Arithmetic Exception
ClassCast Exception
Illegal Argument Exception
NumberFormat Exception
IllegalState Exception
IndexOut OfBounds Exception
ArrayIndexOut OfBounds Exception
NoSuch Element Exception
InputMismatch Exception
NullPointer Exception

Try and Catch Block

- Try Block
  - One or more statements that may cause an exception.
  - Put statements that may cause an exception inside the try block.

```
try
   {
     String filename = ...;
     FileReader reader = new FileReader(filename);
     Scanner in = new scanner(reader);
     String input = in.next();
     int value = Integer.pareseInt(input);
     ......
   }
```

Catch

- Put the handler (what you want done) inside the catch.

```
catch(IOExceptions exception)
   {
        exception.printStackTrace();
   }
catch (NumberFromatException exception)
{
   System.out.println(“Input was not a number”)
}
```

Finally Clause

- You need to take some action whether or not an exception is thrown.
- For example close your files.
- These go in a finally block

```
finally
{
   out.close();
}
```

Finally Clause

- Once a try block is entered, the statements in a finally clause are guaranteed to be executed, whether or not an exception is thrown.

Designing Your Own Exceptions

- You have a condition that is not handled by the standard java exceptions.
- For example, amount > balance

Throw new
InsufficitentFundsException(“withdrawal of “ + amount + “ exceeds balance of “ + balance);

You need to define the InsufficientFundsException class

Designing Your Own Exceptions

- Checked or Unchecked
  - Fault of external event – checked
  - Fault of internal event - unchecked

Exception Class

```
public class InsufficientFundsException
                                        extends RuntimeException
{
            public InsufficientFundsExcetpion()
            {
            }

            public InsufficientFundsException(String message)
            {
                        super(message)
            }
}
```