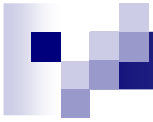




# Basics of programming 3

Java input/output



# *Wrapper classes*



# Wrapper classes

- For each primitive type a class is defined
  - Integer, Byte, Boolean, etc.
  - sometimes an object is needed
- Constructors
  - `Integer(int i)`, `Integer(String s)`
- Constants
  - `MAX_VALUE`, `MIN_VALUE`, `SIZE`
  - `NaN`, `NEGATIVE_INFINITY`, etc.



# Wrapper classes and boxing

- Transformations
  - rotateLeft, reverse, signum, etc.
- Conversion methods
  - intValue(), longValue(), parseInt(String s), valueOf(...)
- Wrapper ↔ primitive conversion is automatic

```
int a = 2;
Integer b = 3;
a = a + b;
Integer d = 3+3; // int -> Integer
System.out.println(d*3); // Integer -> int
```

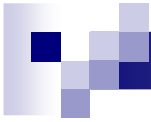


# Wrapper classes and boxing 2

- Conversion is not efficient

```
public static void main(String args[]) {
    Integer i1 = Integer.parseInt(args[0]); // boxing
    Integer i2 = Integer.parseInt(args[0]); // boxing

    i1.equals(i2);           // true, no boxing
    i1 == i2;               // only if -128<i1≤127
    i1 <= i2;               // true, boxing
    i1++;                   // incr, boxing
    i1 > 120;               // boxing
}
```



# *Exception handling*



# Exceptions in C++

- Classic exception handling

```
try {  
    ...  
} catch (E e) { // E type exception  
} catch (...) { // everzthing else  
    ...  
    throw;  
}
```

- No declared exception type

- STL introduced class *exception*



# Exceptions in Java

- based on C++
- common exception superclass
  - `Throwable`
- all possible exceptions must be declared
  - `throws`
- superclasses for different error types
  - `Exception`, `RuntimeException`, `Error`
- closing block: `finally`
- see also: *try with resources* (Java7)



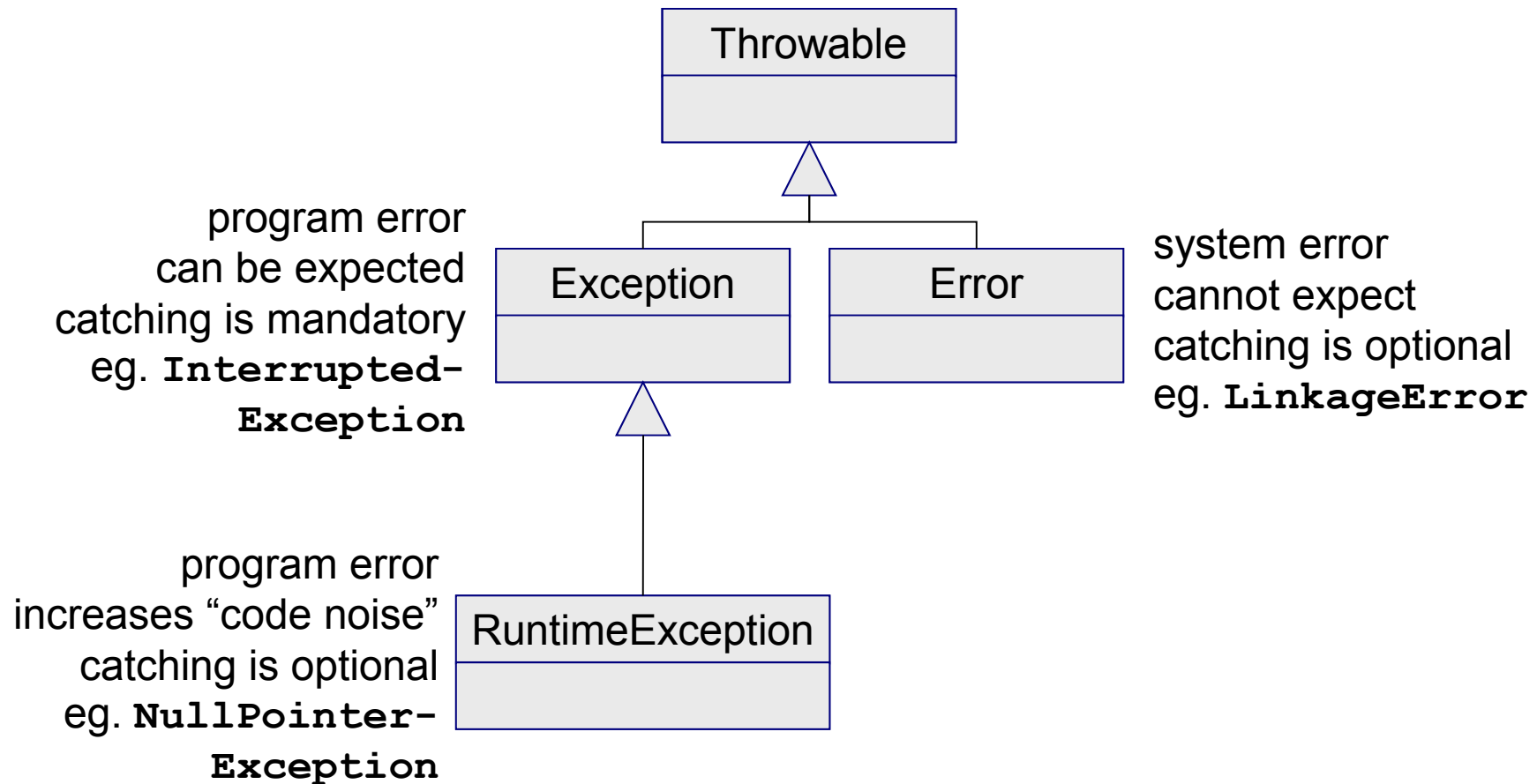


# Exceptions in Java

```
void foo(String s) throws IOException {  
    ...  
}
```

```
FileInputStream fis = new FileInputStream("a");  
try {  
    ...  
    foo(s);  
    ...  
} catch (IOException e) {  
    ...  
    throw e;  
} finally {  
    fis.close();  
}
```

# Exception hierarchy





# Cascade exceptions

```
public AST parseFile(File f) throws ParseException {
    AST ast = new AST();
    try {
        ...
    } catch (IOException e) {
        ParseException pe = new ParseException();
        pe.initCause(e);
        throw pe;
    }
    ...
    return ast;
}
```

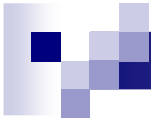
```
try { ast = parseFile(f); }
catch (ParseException pe) { pe.printStackTrace(); }
```



# Cascade exceptions 3

```
ParseException
    at Test2.parseFile(Test2.java:19)
    ...
Caused by: IOException
    at Test2.nextString(Test2.java:12)
    ...
    at Test2.parseFile(Test2.java:17)
```

- in **Throwable** *cause* can be set since Java 1.4
- shown in the *stack-trace*

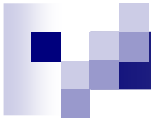


# *Java System Class*



# System class

- `in, out, err`
  - static variables
  - store reference to `stdin, stdout, stderr`
  - have setter methods
- `gc()`
  - starts garbage collection
- `exit(int)`
  - exits JVM
- `getProperty/getProperties, setProperty...`
  - getting system-wide info, like user dir location, etc



# *Java IO Basics*



# Java IO basic principles

- Stream-based communication
  - UNIX legacy
- Two basic types
  - char → unicode, conversion (charset, linefeed)
  - byte → octets, no conversion
- Filtering
  - different functionalities (compression, conversion, etc)
  - IO functionalities can be combined
- package java.io
  - <http://download.oracle.com/javase/6/docs/api/java/io/package-summary.html>





# Basic IO Interfaces

	Input	Output
<b>Char</b>	Reader BufferedReader CharArrayReader FilterReader FileReader ...	Writer BufferedWriter CharArrayWriter FilterWriter FileWriter PrintWriter ...
<b>Byte</b>	InputStream ByteArrayInputStream FileInputStream FilterInputStream PipedInputStream ...	OutputStream ByteArrayOutputStream FileOutputStream FilterOutputStream PipedOutputStream ...



# Reader methods

- `read()`
  - `read(char[] buf, int off, int len)`
    - reads *len* chars
- `mark(int limit), reset(), markSupported()`
  - marking and resetting
- `skip(long n)`
  - skipping *n* chars
- `close()`
  - closes stream
- `ready()`
  - is ready to be read



# Writer methods

- `write(int c)`
  - `write(char[] buf, int off, int len)`
  - `write(String s, int off, int len)`
    - writes *len* chars
- `flush()`
  - flushes the stream
- `close()`
  - closes stream
- `Writer append(...)`
  - similar to *write*, but returns writer
  - enables cascading: `w.append("a").append("b");`



# Special readers

- **BufferedReader**
  - buffers content, can read whole lines
- **CharArrayReader / StringReader**
  - reads from a char array or string
- **FilterReader**
  - abstract class with delegation implemented
- **FileReader**
  - reads from a file



# Reader example

- Reading lines from a file
  - printing to standard output

```
FileReader fr = new FileReader("hello.txt");
BufferedReader br = new BufferedReader(fr);
while (true) {
    String line = br.readLine();
    if (line == null) break;
    System.out.println(line);
}
br.close();
```



# Special writers

- **BufferedWriter**
  - buffered output
- **CharArrayWriter / StringWriter**
  - writes into a charArray or a String
  - toString, toArray, size, etc.
- **FilterWriter**
  - abstract class with delegation implemented
- **FileWriter**
  - writer that writes into a file
- **PrintWriter**
  - prints formatted data: print, println, printf, etc.



# Writer example

- Print the first 10 squares
  - format:  $X * X = Y$

```
Filewriter fw = new Filewriter("squares.txt");
Printwriter pw = new Printwriter(fw);
//Printwriter pw =
// new Printwriter("squares.txt", "ISO-8859-1");
for (int i = 1; i <= 10; i++) {
    pw.println(i+"*" +i+" = "+(i*i));
    //pw.printf("%d*%d = %d\n", i, i, i*i);
}
pw.close()
```



# InputStream methods

- `read()`, `read(byte[] buf, int off, int len)`
  - reads *len* bytes
- `mark(int limit)`, `reset()`, `markSupported()`
  - marking and resetting
- `skip(long n)`
  - skipping *n* bytes
- `close()`
  - closes stream
- `ready()`
  - is ready to be read
- `available()`
  - how many bytes can be read without blocking





# OutputStream methods

- `write(int c)`  
`write(byte[] buf, int off, int len)`  
writes *len* bytes
- `flush()`
  - flushes the stream
- `close()`
  - closes stream



# Special InputStreams

- **ByteArrayInputStream**
  - reads bytes from a byte array
- **FileInputStream**
  - reads bytes from a file
- **FilterInputStream**
  - abstract class with delegation methods



# Special OutputStreams

- **ByteArrayOutputStream**
  - writes bytes into a byte array
- **FileOutputStream**
  - writes bytes to a file
- **FilterOutputStream**
  - abstract class with delegation methods



# Standard IO

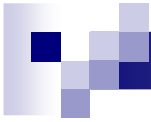
- `java.lang.System` revisited
  - *InputStream in*
    - standard input
    - byte based
  - *PrintStream out, err*
    - standard out, err
    - byte and char based



# Standard IO example

- Reading from stdin, printing to stdout
  - *java.util.Scanner* later

```
InputStreamReader isr =  
    new InputStreamReader(System.in);  
BufferedReader br = new BufferedReader(isr);  
while (true) {  
    String line = br.readLine();  
    if (line == null) break;  
    System.out.println(line);  
}  
br.close();
```



# *Java IO Filters*



# Filter pattern (decorator)

- Basic idea

- same interface
- modified functionality
- calling other object's methods
- concatenated objects: delegation

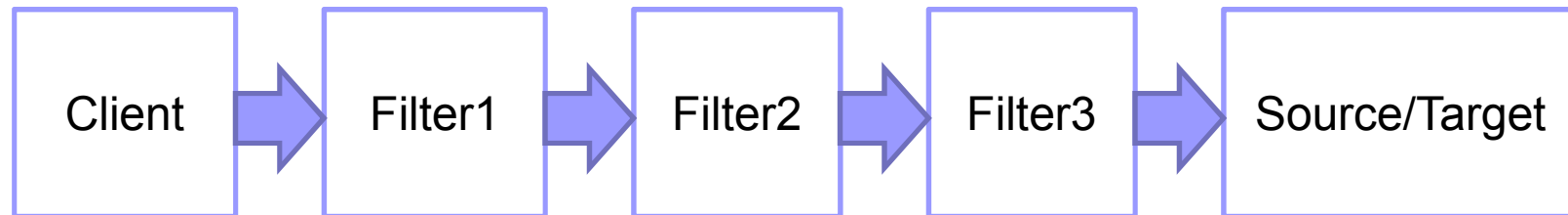
- Sometimes new functionality as well

- e.g. *BufferedReader*
  - `readLine()`

- Usually constructor initialization

- `new BufferedReader(new InputStreamReader(System.in))`

# Filter pattern in use



- Similar to UNIX pipelines
  - `ls -l | grep "^d" | sort -k 9 | tail -n 3`
- Data flow depends on type
  - reader / inputstream: to the left
  - writer / outputstream: to the right
- Filters don't know...
  - who called them
  - how deleguee is implemented





# E.g.: Char and Byte conversion

## ■ Reading

- Source: *InputStream*
- Client needs *Reader*
- Solution: *InputStreamReader*
  - *Reader* interface, but *InputStream* source

## ■ Writing

- *OutputStreamWriter*
  - *Writer* interface, *OutputStream* target

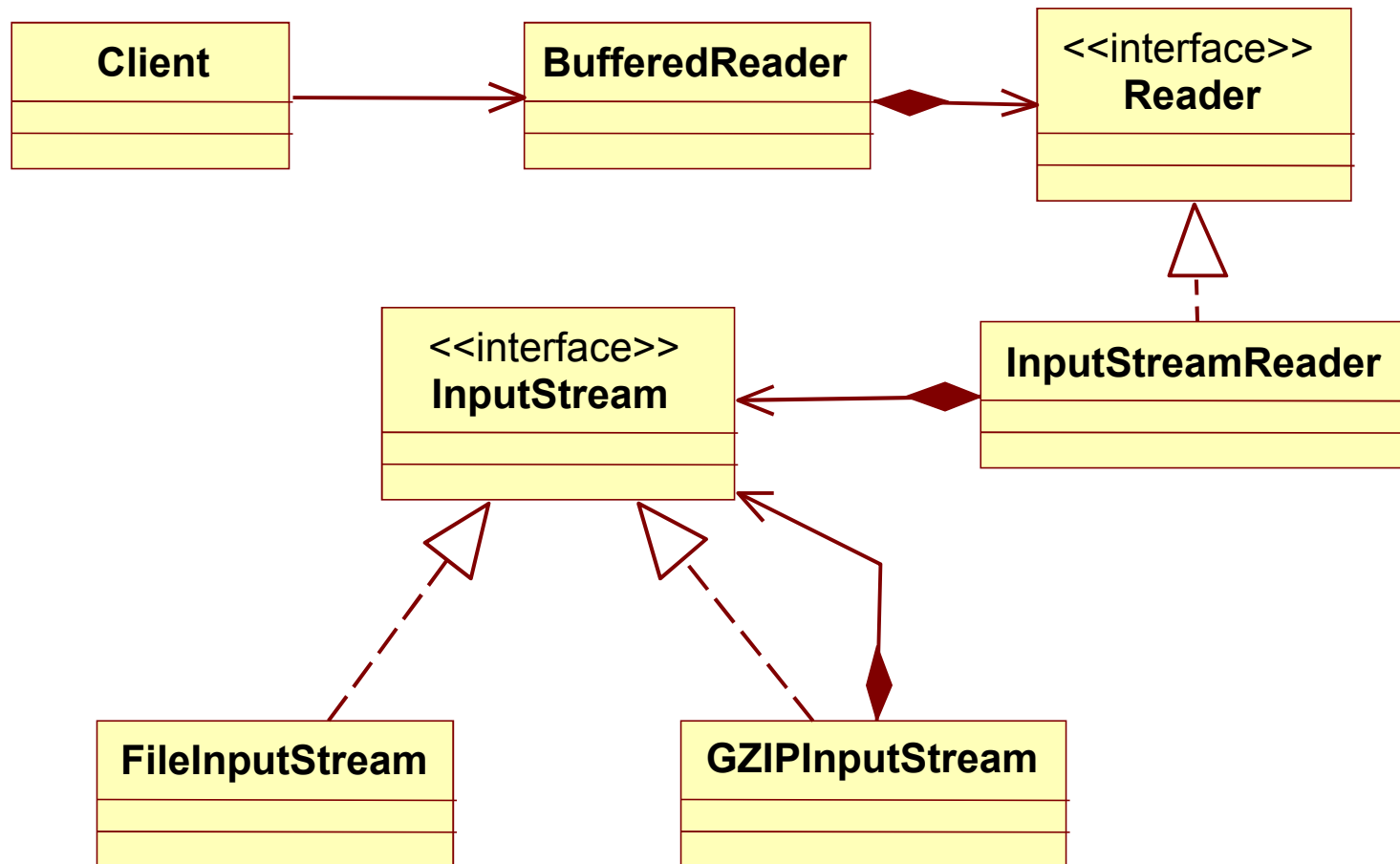


# E.g.: Compression 1

## ■ GZIPInputStream

- provide GZIP decompression
- problem: *print lines from a GZIP compressed file*
  - *like in unix: zcat f.txt.gz*
  
- file content: bytes
- gzip compression: object that reads bytes, returns bytes and decompresses all along
- lines are needed:
  - byte-char conversion
  - chars to lines

# E.g.: Compression 1





# E.g.: Compression 1

```
// reading bytes from file
InputStream fis = new FileInputStream("test.gz");
// un-gzipping bytes
InputStream gis = new GZIPInputStream(fis);
// converting bytes to characters
Reader isr = new InputStreamReader(gis);
// buffering characters into lines
BufferedReader br = new BufferedReader(isr);

while (true) {
    String line = br.readLine();
    if (line != null) System.out.println(line);
    else break;
}
br.close()
```

# E.g.: Compression 2

## ■ GZIPOutputStream

- problem: *read lines and print to compressed file*

```
BufferedReader br = ...;
PrintWriter pw = new PrintWriter(
    new OutputStreamWriter(
        new GZIPOutputStream(
            new FileOutputStream("test.gz"))));
while (true) {
    String line = br.readLine();
    if (line != null) pw.println(line);
    else break;
}
br.close(); pw.close();
```



# Own filter

- `FilterInputStream`, `FilterOutputStream`
- `FilterReader`, `FilterWriter`
  - same interface as superclass
  - default implementation is direct delegation
  - e.g.:

```
public int write(byte[] buf, int off, int len)
throws IOException {
    return out.write(buf, off, len);
}
```

# Own filter 2

- Extend FilterXXX class
  - implement methods as you like
  - use *in* or *out* for delegation
  - don't forget constructor

```
public class MyInFilter
extends java.io.FilterInputStream {
    public MyInFilter(InputStream arg0) {
        super(arg0);
    }
    ...
}
```




# Filter example: CryptoIS/1

```
public class CryptoIS extends FilterInputStream {
    int key;

    public CryptoIS(InputStream arg0, int k) {
        super(arg0);
        key = k;
    }
    private int convert(int c) {
        return c^key; // encrypt-decrypt via xor
    }
    public boolean markSupported() {
        return false;
    }
}
// ...
```





# Filter example: CryptoIS/2

```
//...
public int read() throws IOException {
    int a = in.read();
    return (a<0) ? a : convert(a);
}
public int read(byte[] cbuf, int off, int len)
throws IOException {
    int ret = in.read(cbuf, off, len);
    for (int i = 0; i < ret; i++) {
        cbuf[i+off] = (byte)convert(cbuf[i+off]);
    }
    return ret;
}
}
```



# Filter example: CryptoOS/1

```
public class CryptoOS extends FilterOutputStream {
    int key;
    public CryptoOS(OutputStream arg0, int k) {
        super(arg0);
        key = k;
    }

    private int convert(int c) {
        return c^key;
    }
    // ...
}
```



# Filter example: CryptoOS/2

```
//...
public void write(int b) throws IOException {
    out.write(convert(b));
}
public void write(byte[] cbuf, int off, int len)
throws IOException {
    for (int i = 0; i < len; i++) {
        cbuf[i+off] = (byte)convert(cbuf[i+off]);
    }
    out.write(cbuf, off, len);
}
}
```



# Filter example: usage (to file)

```
// code snippet
try {
    OutputStream os =
        new FileOutputStream("test.txt");

    os = new CryptoOS(os, 13);

    Writer w =
        new OutputStreamWriter(os);
    PrintWriter pw = new PrintWriter(w);

    BufferedReader br = new BufferedReader(
        new InputStreamReader(System.in));
    // ...
}
```



# Filter example: usage (to file)

```
// ...  
  
String line;  
while (true) {  
    line = br.readLine();  
    if (line == null) break;  
    pw.println(line);  
}  
br.close(); // only called on outermost filter  
pw.close(); // only called on outermost filter  
} catch (Exception e) {  
    e.printStackTrace();  
}
```



# Filter example: usage (from file)

```
// code snippet
try {

    InputStream is = new
        FileInputStream("test.txt");

    is = new CryptoIS(is, 13);

    Reader r = new InputStreamReader(is);
    BufferedReader br = new BufferedReader(r);

    // ...
}
```



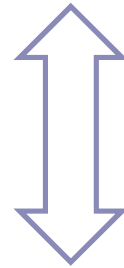
# Filter example: usage (from file)

```
//...
String line;
while (true) {
    line = br.readLine();
    if (line == null) break;
    System.out.println(line);
}
br.close(); // only called on outermost filter

} catch (Exception e) {
    e.printStackTrace();
}
```

# Filter example: result

h e l l o w o r l d ! \r \n



e	h	a	a	b	-	z	b	177	a	i	,	\0	\a
101	104	97	97	98	4	122	98	127	97	105	44	0	7
65	68	61	61	62	2d	7a	62	7f	61	69	2c	00	07