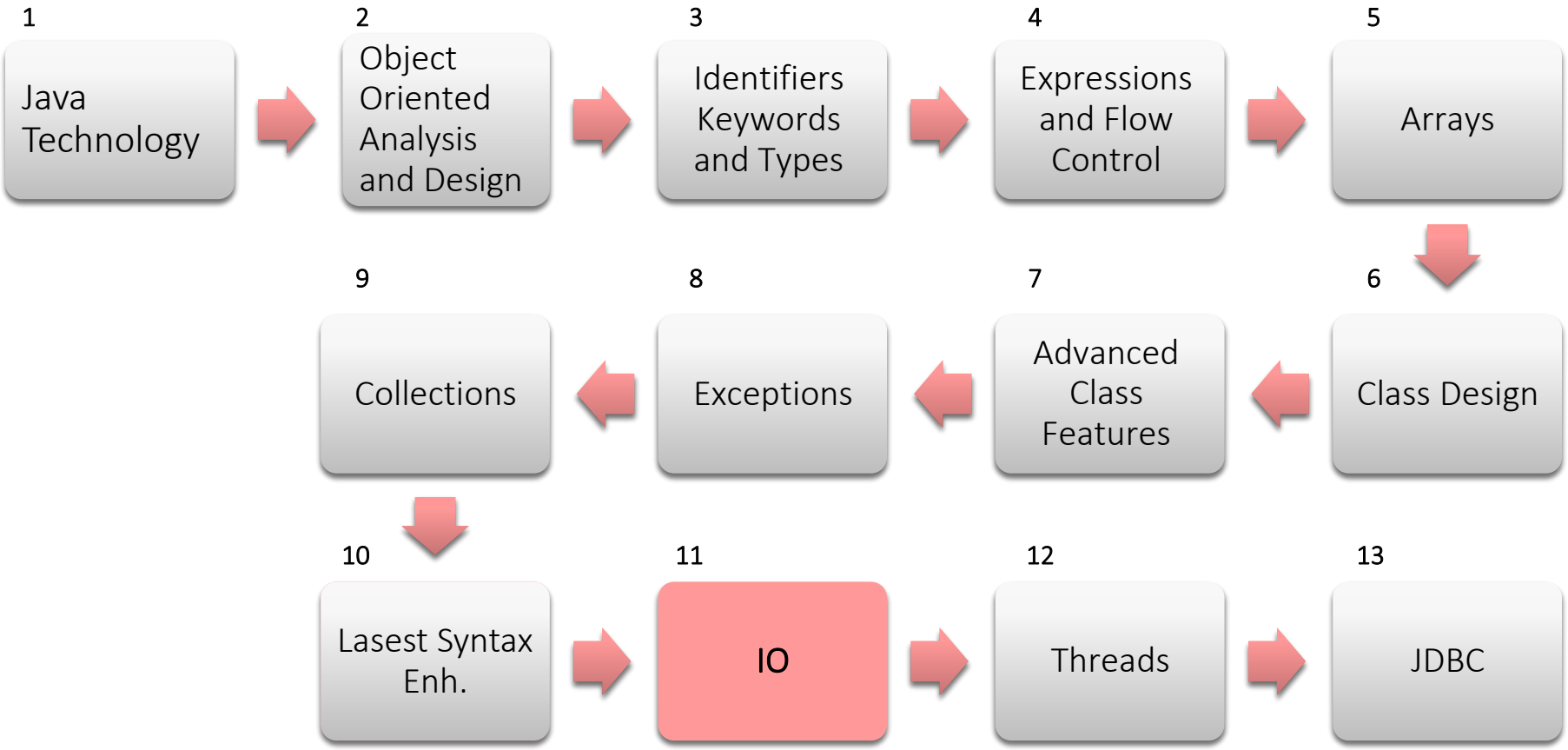
I/O

Character and Binary Streams

JOHN BRYCE
Leading in IT Education
matrix company

Objectives



Objectives

JOHN BRYCE
Leading in IT Education
matrix company

By the end of this session

- You'll get familiar with the different types of I/O

Types of I/O

- Streaming data
  - Blocking: java.io package
  - Non-blocking: java.nio package appeared in JDK 1.3

- Random access
  java.io.RandomAccessFile

Streams

JOHN BRYCE
Leading in IT Education
matrix company

- A stream can be thought of as a flow of data from a source to a sink
- A source stream initiates the flow of data, also called an input stream
- A sink stream terminates the flow of data, also called an output stream
- Sources and sinks are both node streams
- Types of node streams are files, memory, and pipes between threads or processes

Streams

JOHN BRYCE
Leading in IT Education
matrix company

byte streams:

- InputStream
- OutputStream

character streams:

- Reader
- Writer

Streams

JOHN BRYCE
Leading in IT Education
matrix company

- Main IO endpoints:

  File System:

  - File (FileReader / FileWriter / FileInputStream / FileOutputStream)

  Networking:

  - Socket (InputStream / OutputStream)
  - ServerSocket (InputStream / OutputStream)

  Threads communication:

  - Pipes (InputStream / OutputStream)

- You can get streams from other objects (network sockets, JDBC BLOB types, etc.)

Streams

Each program can access three streams by default:

- System.in - InputStream
- System.out - PrintStream (OutputStream)
- System.err - PrintStream

Streams

JOHN BRYCE
Leading in IT Education
matrix company

Echo program:

```
import java.io.IOException;

public class Echo {
  public static void main(String[] args) {
    try {
      int c;
      while ((c = System.in.read()) != -1) {
        System.out.write(c);
      }
    } catch(IOException ex) {
      System.err.println("An error has occured.");
                ex.printStackTrace();
    }
  }
}
```

InputStream Methods

- `int available()`
- `int read()`
- `int read(byte[] b)`
- `int read(byte[] b, int off, int len)`
- `long skip(long n)`
- `void close()`

- `void mark(int readlimit)`
- `boolean markSupported()`
- `void reset()`

Exceptions

- Most methods throw `java.io.IOException`
- that must be handled
- this is true for OutputStreams, Readers and Writers too

OutputStream Methods

- `void write(byte[] b)`
- `void write(byte[] b, int off, int len)`
- `void write(int b)`
- `void flush()`
- `void close()`

Reader Methods

JOHN BRYCE
Leading in IT Education
matrix company

- `int read()`
- `int read(char[] cbuf)`
- `int read(char[] cbuf, int off, int len)`
- `long skip(long n)`
- `boolean ready()`
- `void close()`

- `void mark(int readAheadLimit)`
- `boolean markSupported()`
- `void reset()`

Writer Methods

- `void write(char[] cbuf)`
- `void write(char[] cbuf, int off, int len)`
- `void write(int c)`
- `void write(String str)`
- `void write(String str, int off, int len)`

- `void flush()`
- `void close()`

Byte Stream or Character Stream?

JOHN BRYCE
Leading in IT Education
matrix company

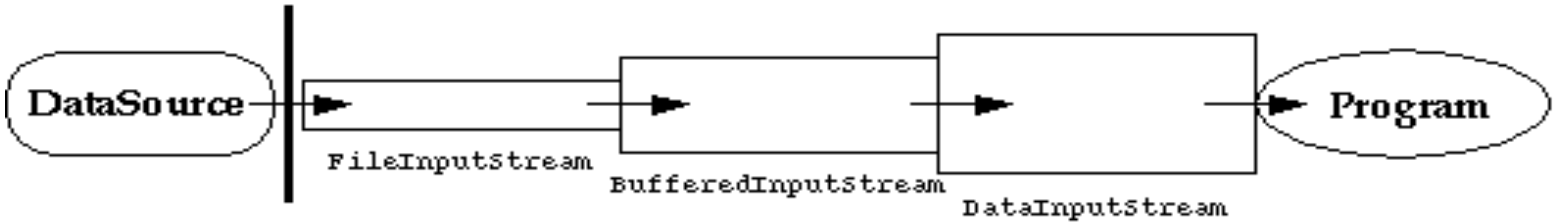
- Use byte streams for binary data
- Use character streams for unicode data
- character conversion is needed to create a Reader over an InputStream
- character conversion is needed to create an OutputStream over a Writer

Filter Streams – Decorators

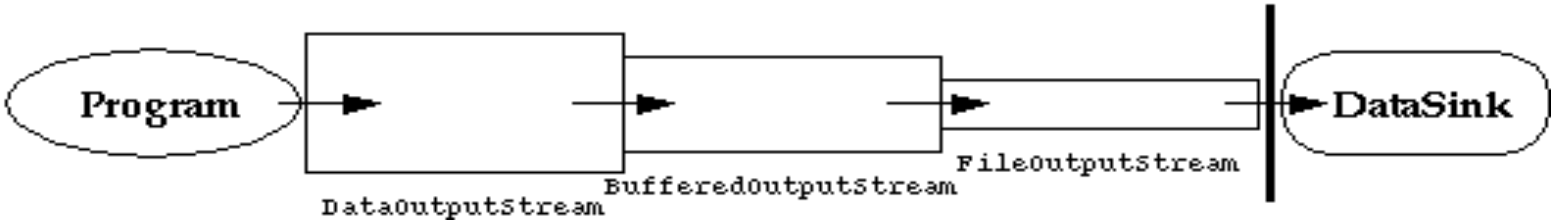
- A stream created over another stream is called a filter stream.
- Subclasses of
  - `FilterInputStream`
  - `FilterOutputStream`
  - `FilterReader`
  - `FilterWriter`
- Filter streams are used frequently to decorate IO into:
  - Texts
  - Primitives
  - Objects

I/O Stream Chaining - Decoration

Input Stream Chain



Output Stream Chain



Text Decorators

- Text decoration: BufferedReader / BufferedWriter
  - Provides Line based reading and writing:
  - Reading: BufferedReader.readLine() returns String or null on end of file
  - Writing: BufferedWriter.write(String line) & BufferedWriter.newline()

Text Decorators

JOHN BRYCE
Leading in IT Education
matrix company

```
public class EchoLines {
  public static void main(String[] args) {
    try {
      BufferedReader in = new BufferredReader(new FileReader("c:/temp/myFile.txt"));
     BufferedWriter out = new BufferredWriter(new FileWriter("c:/temp/copy.txt")));
      String s;
      while ((s = in.readLine()) != null) {
        out.write(s);
        out.newline();
      }
    } catch(java.io.IOException ex) {
      System.err.println("An error has occured.");
      ex.printStackTrace();
    }
  }
}
```

Primitives Decorators

JOHN BRYCE
Leading in IT Education
matrix company

- Primitives decoration: DataInputStream / DataOutputStream
  - Provides primitive based reading and writing:
  - Reading:
    - readUTF() : String
    - readInt() : int
    - readBoolean() : Boolean ….
  - Writing:
    - writeUTF(String)
    - writeInt(int)
    - writeBoolean(boolean)

Objects Decorators

JOHN BRYCE
Leading in IT Education
matrix company

- Object decoration: ObjectInputStream / ObjectOutputStream
  - Provides object based reading and writing:
  - Reading:
    - readObject() : Object
  - Writing:
    - writeObject (Object)
  - All objects must implement Serializable
  - More regarding serialization – later

Switching from binary to text IO

JOHN BRYCE
Leading in IT Education
matrix company

Assume

- You have a binary stream source (InputStream)
- You know that the stream holds textual data
- You'd like read line by line – (decorate with BufferedReader)
- BUT – how can reader decorate an input stream ???
- Simply by using : InputStreamReader
  - Takes an input stream to decorate
  - Acts like a reader

InputStreamReader

JOHN BRYCE
Leading in IT Education
matrix company

Constructors:

- `InputStreamReader(InputStream in)`
- `InputStreamReader(InputStream in, Charset cs)`
- `InputStreamReader(InputStream in, CharsetDecoder dec)`
- `InputStreamReader(InputStream in, String charsetName)`

- Converts the bytes read from the underlying InputStream to characters according to the specified (or default) character coding

InputStreamReader

```
public class EchoLines {
  public static void main(String[] args) {
    try {
      BufferedReader in;
      in = new BufferedReader(new InputStreamReader(System.in));
      String s;
      while ((s = in.readLine()) != null) {
        System.out.println(s);
      }
    } catch(java.io.IOException ex) {
      System.err.println("An error has occured.");
      ex.printStackTrace();
    }
  }
}
```

Switching from binary to text IO

JOHN BRYCE
Leading in IT Education
matrix company

Assume

- You have a textual array (String / char[])
- You know that target expects to get it as a binary stream
- You'd like write text data as output stream
- BUT – how can writer decorate to binary stream output???
- Simply by using : OutputStreamWriter
  - Takes an output stream to decorate and write to
  - Acts like a writer

OutputStreamWriter

JOHN BRYCE
Leading in IT Education
matrix company

Constructors:

- `OutputStreamWriter(OutputStream out)`
- `OutputStreamWriter(OutputStream out, Charset cs)`
- `OutputStreamWriter(OutputStream out, CharsetEncoder enc)`
- `OutputStreamWriter(OutputStream out, String charsetName)`

- Converts characters to bytes according to the character coding before writing

Reading Text Files

- Normally a `FileReader` can be used
- If the file has a different character encoding, use FileInputStream and `InputStreamReader`
- Constructors:
  - `FileReader(String filename)`
  - `FileReader(File file)`
  - `FileInputStream(String filename)`
  - `FileInputStream(File file)`

Writing Text Files

JOHN BRYCE
Leading in IT Education
matrix company

- `FileWriter`
- `FileInputStream` and `OutputStreamWriter`
- Files can be appended:
  - FileWriter(File file)
  - FileWriter(File file, boolean append)
  - FileWriter(String fileName)
  - FileWriter(String fileName, boolean append)
- Same applies to FileOutputStream

Serialization

- Only the object's data are serialized
- Data marked with the transient keyword are not serialized

```
1    public class MyClass implements Serializable {
2        public transient Thread myThread;
3        private String customerID;
4        private int total;
5    }
```

```
1    public class MyClass implements Serializable {
2        public transient Thread myThread;
3        private transient String customerID;
4        private int total;
5    }
```

- Serialization stores the state of an object to a file; storing the state of an object is called persistence

Object Streams

JOHN BRYCE
Leading in IT Education
matrix company

- Serializable objects can be stored and loaded from object streams:
  - `ObjectInputStream`
  - `ObjectOutputStream`

- Serialization is easy in Java
- Implement `java.io.Serializable` interface
- No methods in the interface

Writing an Object to a File Stream

```
1   import java.io.*;
2   import java.util.Date;
3
4   public class SerializeDate {
5
6   SerializeDate() {
7   Date d = new Date ();
8
9   try {
10    FileOutputStream f =
11    new FileOutputStream          ("date.ser");
12     ObjectOutputStream s =
13    new ObjectOutputStream (f);
14        s.writeObject (d);
15        s.close ();
16     } catch (IOException e) {
17   e.printStackTrace ();
18     }
19   }
20
21     public static void main       (String args[]) {
22          new SerializeDate();
23     }
24   }
```

Writing an Object to a File Stream

```
1   import java.io.*;
2   import java.util.Date;
3
4   public class SerializeDate {
5
6   SerializeDate() {
7   Date d = new Date ();
8
9    try {
10    FileOutputStream f =
11    new FileOutputStream      ("date.ser");
12    ObjectOutputStream s =
13    new ObjectOutputStream (f);
14       s.writeObject (d);
15       s.close ();
16     } catch (IOException e) {
17   e.printStackTrace ();
18     }
19   }
20
21     public static void main (String args[]) {
22       new SerializeDate();
23     }
24   }
```

Reading Object from File Stream

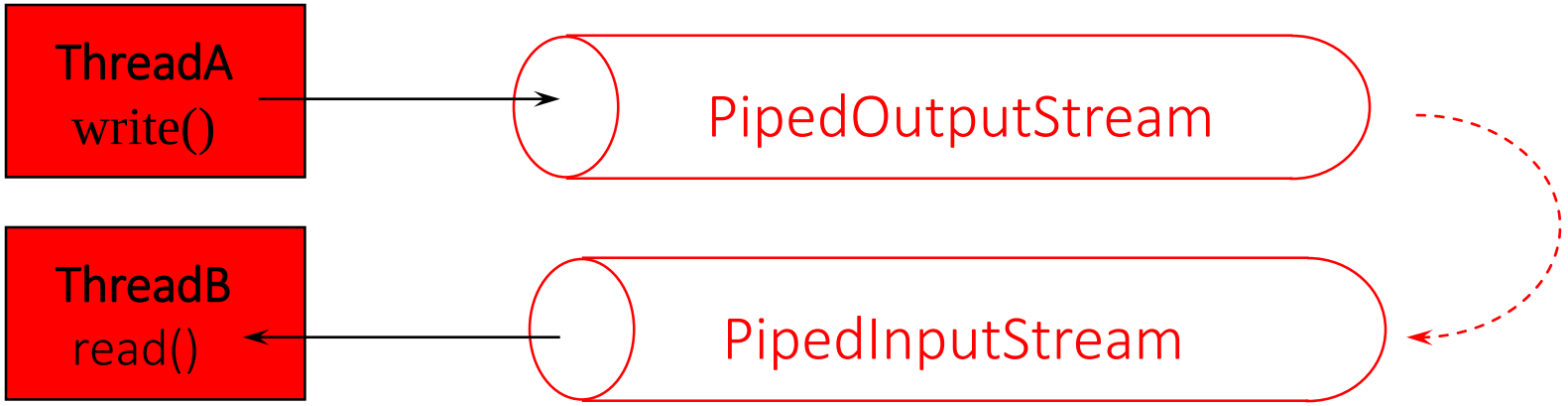
JOHN BRYCE
Leading in IT Education
matrix company

```
1   import java.io.*;
2   import java.util.Date;
3
4   public class UnSerializeDate {
5
6   UnSerializeDate () {
7   Date d = null;
8
9   try {
10    FileInputStream f =
11    new FileInputStream      ("date.ser");
12    ObjectInputStream s =
13    new ObjectInputStream (f);
14    d = (Date) s.readObject ();
15    s.close ();
16    } catch (Exception e) {
17    e.printStackTrace ();
18    }
19
20    System.out.println(
21    "Unserialized Date object from      date.ser");
22    System.out.println("Date: "+d);
23    }
24
25    public static void main (String      args[]) {
26    new UnSerializeDate();
27    }
28    }
```

Pipes

- PipedWriter
- PipedReader

- PipedOutputStream
- PipedInputStream



Using pipes

JOHN BRYCE
Leading in IT Education
matrix company

```
PipedWriter pout = new PipedWriter();
PipedReader pin  = new PipedReader(pout);
```

or

```
PipedReader pin  = new PipedReader();
PipedWriter pout = new PipedWriter(pin);
```

or

```
PipedReader pin  = new PipedReader();
PipedWriter pout = new PipedWriter();
pin.connect(pout); // or pout.connect(pin);
```

RandomAccessFile

- Constructors:
  - `RandomAccessFile(File file, String mode)`
  - `RandomAccessFile(String name, String mode)`
- mode is usually ”r” or ”rw” .
- file pointer can be read and positioned:
  - `int skipBytes(int)`
  - `void seek(long)`
  - `long getFilePointer()`

More Classes in `java.io` Package

- File
- FileFilter
- FilenameFilter

- StreamTokenizer

File Class

- Represents an operating file location
  - `exists()`
  - `canRead(), canWrite()`
  - `delete()`
  - `renameTo(File)`
  - `list(FilenameFilter), list(FileFilter)`
  - `etc.`
- Static utility methods:
  - `createTempFile(...)`
  - `listRoots()`

File Class

- File myFile;
- myFile = new File("myfile.txt");
- myFile = new File("MyDocs", "myfile.txt");

Directories are treated just like files in Java; the File class supports methods for retrieving an array of files in the directory

- File myDir = new File("MyDocs");

  myFile = new File(myDir, "myfile.txt");

StreamTokenizer

- Works much like `StringTokenizer`
- more soffisticated:
  - knows comments
  - knows quotes
  - words and numbers can be different type of tokens
  - whitespaces can be defined

Working with URLs

JOHN BRYCE
Leading in IT Education
matrix company

- In many 'Internet oriented' IO we would like to download HTML page in order to read an record it content
- It is easy to do via URL object by obtaining its InputStream and process it as text stream:

```
URL url=new URL("https://google.com");
BufferedReader in=new BufferedReader(
                        new InputStreamReader(
                        url.openStream()));
String line=in.readLine();
while(line!=null){
      System.out.println(line);
      line=in.readLine();
}
```

io-objects

io-primitives

io-text

References

- http://java.sun.com 

- SUN Educational Services SL-275