Java API Elements (Object,i18n, regex, I/O)

Angel Gruev
Dreamix ltd.

I18N

Terms

- Internationalization
  - process of creating an application so it can be adapted to different languages and countries without coding changes
  - also known as “i18n” (since there are 18 characters between the *i* and *n* in Unicode)

Terms (cont)

- Characteristics of an i18n application
  - By adding localized text, the app can display information in any language/country
  - Textual elements are not hard-coded; they are stored externally and retrieved at run-time

Terms (cont)

- Characteristics of an i18n application (cont)
  - Support for new languages/countries does not require re-compilation
  - Culturally dependent data (dates, currencies, etc.) conforms to user's language/country
  - Can be *localized* quickly

Terms (cont)

- Localization
  - process of adapting software *for a specific language/country* by adding locale-specific components and text translations
  - sometimes abbreviated as l10n (10 letters between L and N in Unicode)

Terms (cont)

- Localizing an application
  - text translation is the most time-consuming phase of l10n (the human element)
  - sounds and images may need localized if they are culturally sensitive
    - red indicates “danger” in the US, but in some countries it means “purity” … likewise the icon of a mailbox isn’t familiar outside the U.S., and instead a mail envelope icon should be used
  - formatting of numbers, dates, and currencies may impact the UI layout strategy

Terms (cont)

- Object central to l10n is the user's *locale*
- Locale
  - political, cultural, and region-specific elements (in Java, expressed as a language code and country code)

Terms

- Locale (cont)
  - has the form xx_YY
  - xx is two-character language code (ISO-639)
  - YY is two-character country code (ISO-3166)
  - Examples:
    - en_US - United States English
    - en_GB - Great Britain English
    - es_MX - Mexico Spanish (Espanol)

java.util.Locale

```
Locale enUSLocale = new Locale(“en”, “US”);
Locale frCALocale = new Locale(“fr”, “CA”);
Locale locale = Locale.US;

Locale.getDefault().toString()// “en_US”
locale.getLanguage()          // “en”
locale.getCountry()           // “US”
locale.getDisplayName() // “English (United States)”
locale.getDisplayLanguage()   // “English”

Locale.setDefault( Locale.FRANCE );
locale.getDisplayName() // “anglais (Etats-Unis)”
```

* Display names are shown according to the default locale

What is a “locale”?

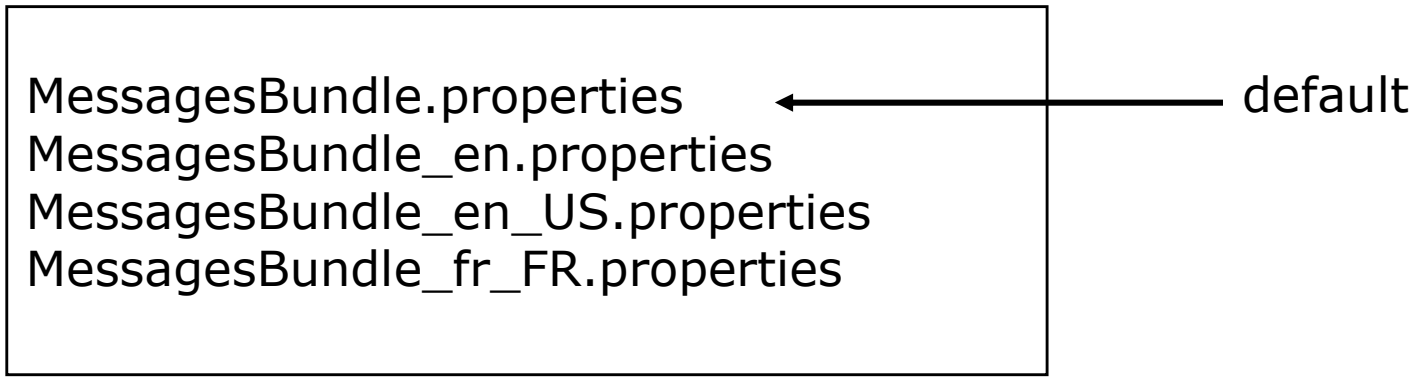
- Locale objects are only identifiers.
- After defining a Locale, you pass it to other objects that perform useful tasks, such as formatting dates and numbers.
- These objects are called locale-sensitive, because their behavior varies according to Locale.
- A ResourceBundle is an example of a locale-sensitive object.

Internationalization in J2SE

- Create properties files
  - externalized locale-specific UI messages
- Create the locale
- Create a resource bundle (using the locale)
- Retrieve UI messages from the resource bundle

J2SE

- Creating properties files
  - Plain text file
  - Will reside in classpath
  - One file for each locale (filename convention)
  - When paired with a locale, the closest matching file will be selected



J2SE

- Creating properties files (cont)

MessagesBundle_en_US.properties

```
greetings = Hello.
farewell = Goodbye.
inquiry = How are you?
```

MessagesBundle_fr_FR.properties

```
greetings = Bonjour.
farewell = Au revoir.
inquiry = Comment allez-vous?
```

J2SE

- Create the locale and resource bundle

```
Locale currentLocale = new Locale("fr", “FR",
"UNIX");

ResourceBundle messages =
ResourceBundle.getBundle(“MessagesBundle",
currentLocale);
```

- Retrieve UI messages from the resource bundle

```
System.out.println(messages.getString(“greetings”));
```

Eclipse is *Almost* Helpful

- The “Externalize Strings” option gets you part-way to i18n
- But…
  - Vague key names
  - Only works on Java files (not JSPs, which we’ll get to shortly)

J2SE Side Notes

- Properties files and resource bundle keys should follow human-readable naming conventions (not “key1”, “key2”, etc.)
- Some UI generation engines for thick clients (e.g. SwiXML) have support for i18n; something to consider if you’re creating a Swing app

Internationalization in J2EE

- Locale stored in HTTP session
- Resource bundles stored in properties files; told to load in web.xml
- Locale-specific messages accessed in the Web tier (generally) using tag libraries

Numbers and Currencies!

- What's wrong with my numbers?
  - Americans say: `345,987.246`
  - Germans say: `345.987,246`
  - French say: `345 987,246`

Numbers...

- Supported through NumberFormat!

```
Locale[] locales = NumberFormat.getAvailableLocales();
```

  - Shows what locales are available. Note, you can also create custom formats if needed.

```
Double amount = new Double(345987.246);
NumberFormat numberFormatter;
String amountOut;

numberFormatter = NumberFormat.getNumberInstance(currentLocale);
amountOut = numberFormatter.format(amount);
System.out.println(amountOut + "    " + currentLocale.toString());
```

```
345 987,246    fr_FR
345.987,246    de_DE
345,987.246    en_US
```

Money!

- Supported with: NumberFormat.getCurrencyInstance()

```
Double currency = new Double(9876543.21);
NumberFormat currencyFormatter;
String currencyOut;

currencyFormatter = NumberFormat.getCurrencyInstance(currentLocale);
currencyOut = currencyFormatter.format(currency);
System.out.println(currencyOut + "    " + currentLocale.toString());
```

```
9 876 543,21 F     fr_FR
9.876.543,21 DM    de_DE
$9,876,543.21      en_US
```

Percents?

- Supported with: NumberFormat.getPercentInstance()!

```
Double percent = new Double(0.75);
NumberFormat percentFormatter;
String percentOut;

percentFormatter = NumberFormat.getPercentInstance(currentLocale);
percentOut = percentFormatter.format(percent);
```

“A Date and Time…

- Supported with:
  - DateFormat.getDateInstance

```
DateFormat dateFormatter =
            DateFormat.getDateInstance(DateFormat.DEFAULT, currentLocale);
```

  - DateFormat.getTimeInstance

```
DateFormat timeFormatter =
            DateFormat.getTimeInstance(DateFormat.DEFAULT, currentLocale);
```

  - DateFormat.getDateTimeInstance

```
DateFormat dateTimeFormatter = DateFormat.getDateTimeInstance(
            DateFormat.LONG, DateFormat.LONG, currentLocale);
```

Date example...

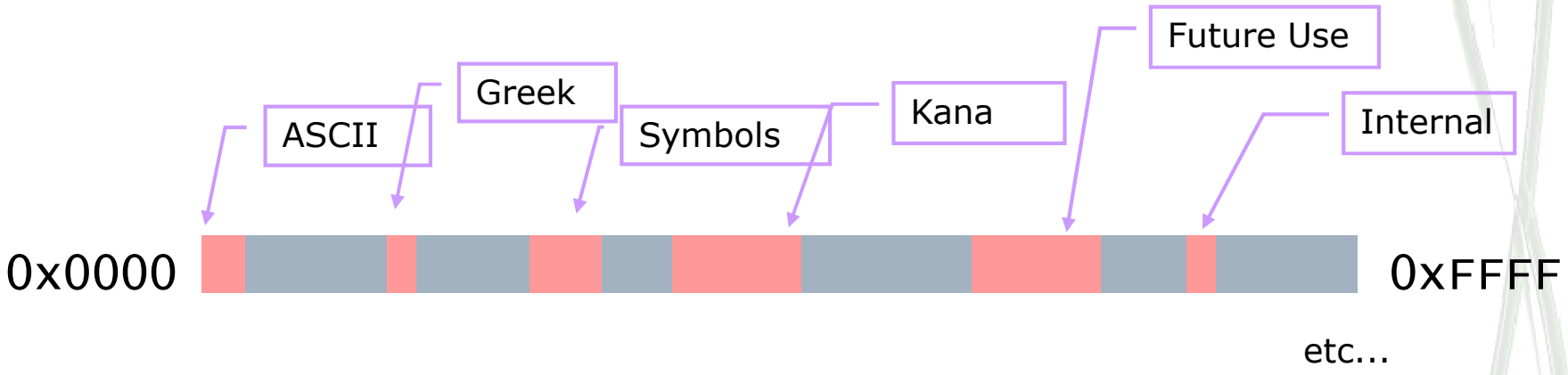
- Supported with: DateFormat.getDateInstance!

```
Date today;
String dateOut;
DateFormat dateFormatter;

dateFormatter = DateFormat.getDateInstance(DateFormat.DEFAULT, currentLocale);
today = new Date();
dateOut = dateFormatter.format(today);

System.out.println(dateOut + "    " + currentLocale.toString());
```

```
9 avr 98    fr_FR
9.4.1998    de_DE
09-Apr-98    en_US
```

Characters...

- 16 bit!
- 65,536 characters
- Encodes all major languages
- In Java Char is a Unicode character
- See unicode.org/



Java support for the Unicode Char...

- Character API:
  - isDigit
  - isLetter
  - isLetterOrDigit
  - isLowerCase
  - isUpperCase
  - isSpaceChar
  - isDefined
- Unicode Char values accessed with:

```
String eWithCircumflex = new String("\u00EA");
```

Java support for the Unicode Char...

- Example of some repair...
  - **BAD!**

```
if ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z'))
   // ch is a letter
```

  - **GOOD!**

```
if (Character.isLetter(ch))
 // ch is a letter
```

Regular Expressions in Java

Readings

- SUN regexps tutorial

http://java.sun.com/docs/books/tutorial/extra/regex/index.html

- Java.util.regex API

http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/package-summary.html

Regular Expressions

- Regular expressions (regex's) are sets of symbols and syntactic elements used to match patterns of text.

Regular Expressions

- A regular expression is a kind of pattern that can be applied to text (Strings, in Java)
- A regular expression either matches the text (or part of the text), or it fails to match
  - If a regular expression matches a part of the text, then you can easily find out which part
  - If a regular expression is complex, then you can easily find out which parts of the regular expression match which parts of the text
  - With this information, you can readily extract parts of the text, or do substitutions in the text
- Regular expressions are an extremely useful tool for manipulating text
  - Regular expressions are heavily used in the automatic generation of Web pages

Some simple patterns

abc	exactly this sequence of three letters
[abc]	any *one* of the letters a, b, or c
[^abc]	any character *except* one of the letters a, b, or c (immediately within an open bracket, ^ means "not,"but anywhere else it just means the character ^)
[a-z]	any *one* character from a through z, inclusive
[a-zA-Z0-9]	any *one* letter or digit

Basic Syntax

Char	Usage	Example
.	Matches any single character	.at = cat, bat, rat, 1at…
*****	Matches zero or more occurrences of the single preceding character	.*at = everything that ends with at 0*123 = 123, 0123, 00123…
[…] **[^…]**	Matches any single character of the ones contained Matches any single character except for the ones contained	[cbr]at = cat, bat, rat. [^bc]at = rat, sat…, *but not* bat, cat. <[^>]*> = <…anything…>
^	Beginning of line	^a = line starts with a
$	End of line	^$ = blank line (starts with the end of line)
****	Escapes following special character: . \ / & [ ] * + -> \. \\ \/ \& \[\] * \+	[cbr]at\. = matches cat., bat. and rat. only
…	…	

Sequences and alternatives

- If one pattern is followed by another, the two patterns must match consecutively
  - For example, [A-Za-z]+[0-9] will match one or more letters immediately followed by one digit
- The vertical bar, |, is used to separate alternatives
  - For example, the pattern abc|xyz will match either abc or xyz

Matches

- Input string consumed from left to right
- Match ranges: inclusive of the beginning index and exclusive of the end index
- Example:

Current REGEX is: foo

Current INPUT is: foofoofoo

I found the text "foo" starting at index 0 and ending at index 3.

I found the text "foo" starting at index 3 and ending at index 6.

I found the text "foo" starting at index 6 and ending at index 9.

Predefined Character Classes

.	Any character (may or may not match line terminators)
\d	A digit: [0-9]
\D	A non-digit: [^0-9]
\s	A whitespace character: [\t\n\x0B\f\r]
\S	A non-whitespace character: [^\s]
\w	A word character: [a-zA-Z_0-9]
\W	A non-word character: [^\w]

Quantifiers

Greedy	Reluctant	Posse-ssive	Meaning
X?	X??	X?+	X, once or not at all
X*	X*?	X*+	X, zero or more times
X+	X+?	X++	X, one or more times
X{n}	X{n}?	X{n}+	X, exactly n times
X{n,}	X{n,}?	X{n,}+	X, at least n times
X{n,m}	X{n,m}?	X{n,m}+	X, at least n but not more than m times

Quantifier Types

- **Greedy**: first, the quantified portion of the expression eats the whole input string and tries for a match. If it fails, the matcher backs off the input string by one character and tries again, until a match is found.
- **Reluctant**: starts to match at the beginning of the input string. Then, iteratively eats another character until the whole input string is eaten.
- **Possessive**: try to match only once on the whole input stream.

Example

- **Greedy**:

  Current REGEX is: .*foo

  Current INPUT is: xfooxxxxxxfoo

  I found the text "xfooxxxxxxfoo" starting at index 0 and ending at index 13.

- **Reluctant**:

  Current REGEX is: .*?foo

  Current INPUT is: xfooxxxxxxfoo

  I found the text "xfoo" starting at index 0 and ending at index 4.

  I found the text "xxxxxxfoo" starting at index 4 and ending at index 13.

- **Possessive**

  Current REGEX is: .*+foo

  Current INPUT is: xfooxxxxxxfoo

  No match found.

Capturing groups

- In regular expressions, parentheses are used for grouping, but they also *capture* (keep for later use) anything matched by that part of the pattern
  - Example: ([a-zA-Z]*)([0-9]*) matches any number of letters followed by any number of digits
  - If the match succeeds, \1 holds the matched letters and \2 holds the matched digits
  - In addition, \0 holds everything matched by the entire pattern
- Capturing groups are numbered by counting their *opening parentheses* from left to right:
  - ((A) (B (C)))
    1 2 3 4
    \0 = \1 = ((A)(B(C))), \2 = (A), \3 = (B(C)), \4 = (C)
- Example: ([a-zA-Z])\1 will match a double letter, such as letter

Capturing groups in Java

- If m is a matcher that has just performed a successful match, then
  - m.group(*n*) returns the String matched by capturing group *n*
    - This could be an empty string
    - This will be null if the pattern as a whole matched but this particular group didn't match anything
  - m.group() returns the String matched by the entire pattern (same as m.group(0))
    - This could be an empty string
- If m didn't match (or wasn't tried), then these methods will throw an IllegalStateException

RegExps in Java

- Two important classes:
  - **java.util.regex.Pattern** -- a compiled representation of a regular expression
  - **java.util.regex.Matcher** -- an engine that performs match operations by interpreting a Pattern
- Example

```
Pattern p = Pattern.compile("a*b");
Matcher m = p.matcher("aaaaab");
boolean b = m.matches();
```

- ! To produce a slash in a Java String: “//”

Doing it in Java, I

- First, you must *compile* the pattern

```
import java.util.regex.*;
Pattern p = Pattern.compile("[a-z]+");
```

- Next, you must create a *matcher* for a specific piece of text by sending a message to your pattern

```
Matcher m = p.matcher("Now is the time");
```

- Points to notice:
  - Neither Pattern nor Matcher has a public constructor; you create these by using methods in the Pattern class
  - The matcher contains information about *both* the pattern to use *and* the text to which it will be applied

Doing it in Java, II

- Now that we have a matcher m,
  - m.matches() returns true if the pattern matches the entire text string, and false otherwise
  - m.lookingAt() returns true if the pattern matches at the beginning of the text string, and false otherwise
  - m.find() returns true if the pattern matches any part of the text string, and false otherwise
    - If called again, m.find() will start searching from where the last match was found
    - m.find() will return true for as many matches as there are in the string; after that, it will return false
    - When m.find() returns false, matcher m will be *reset* to the beginning of the text string (and may be used again)

A complete example

```
import java.util.regex.*;

public class RegexTest {
    public static void main(String args[]) {
        String pattern = "[a-z]+";
        String text = "Now is the time";
        Pattern p = Pattern.compile(pattern);
        Matcher m = p.matcher(text);
        while (m.find()) {
            System.out.print(text.substring(m.start(),
                                            m.end()) + "*");
        }
    }
}
```

Output: ow*is*the*time*

Example

```
import java.util.regex.*;
public class RegEx{
    public static void main( String args[] ){
        String amounts = "$1.57 $316.15 $19.30 $0.30 $0.00
    $41.10 $5.1 $.5";
        Pattern strMatch = Pattern.compile(
    "\\$(\\d+)\\.(\\d\\d)" );
            Matcher m = strMatch.matcher( amounts );
            while ( m.find() ){
                System.out.println( "$" + ( Integer.parseInt(
    m.group(1) ) + 5 )
                    + "." + m.group(2) );
            }
    }
}
```

=> Adds 5$ to every amount except the last two

Spaces

- There is only one thing to be said about spaces (blanks) in regular expressions, but it's important:
  - *Spaces are significant!*
- A space stands for a *space*--when you put a space in a pattern, that means to match a space in the text string
- It's a *really bad idea* to put spaces in a regular expression just to make it look better

Thinking in regular expressions

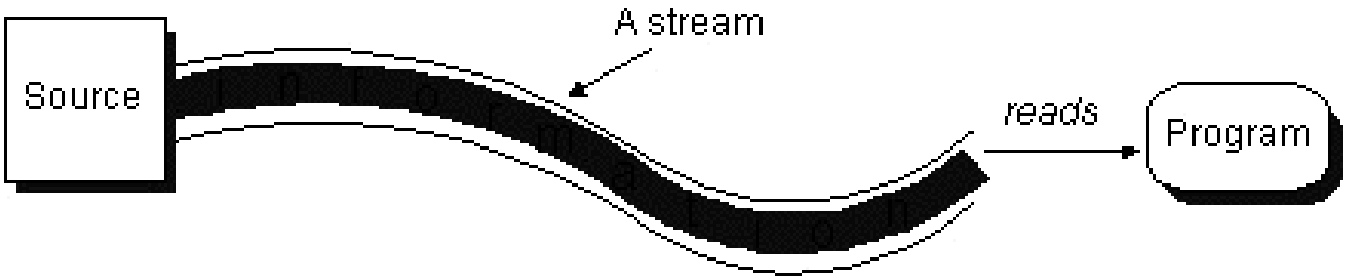
- Regular expressions are *not* easy to use at first
  - It's a bunch of punctuation, not words
  - The individual pieces are not hard, but it takes practice to learn to put them together correctly
  - Regular expressions form a miniature programming language
    - It's a different kind of programming language than Java, and requires you to learn new thought patterns
  - In Java you can't just *use* a regular expression; you have to first create Patterns and Matchers
  - Java's syntax for String constants doesn't help, either
- Despite all this, regular expressions bring so much power and convenience to String manipulation that they are well worth the effort of learning

I/O in JAVA

Reading & Writing Data

- Data can come from many Sources & go to many Destinations
  - Memory
  - Disk
  - Network
- Whatever the Source or Destination, a Stream has to be opened to Read/Write Data

Reading & Writing Data



Reading

Open a Stream

While more Information

Read

Close the Stream

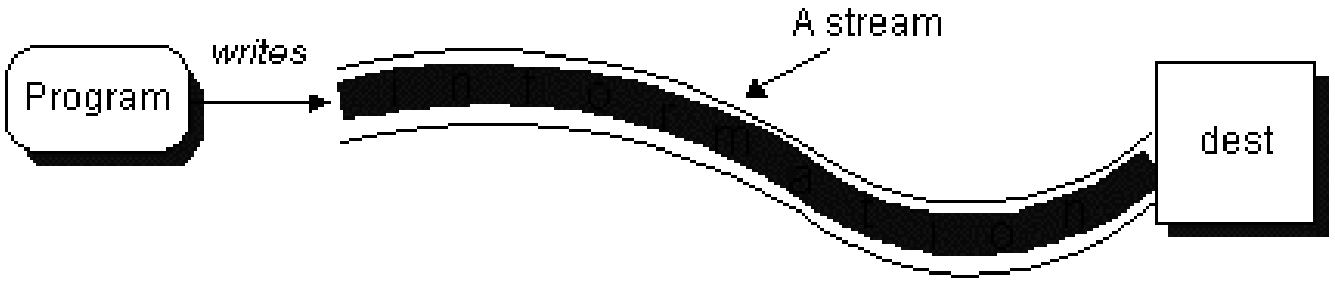
Writing

Open a Stream

While more Information

Write

Close the Stream

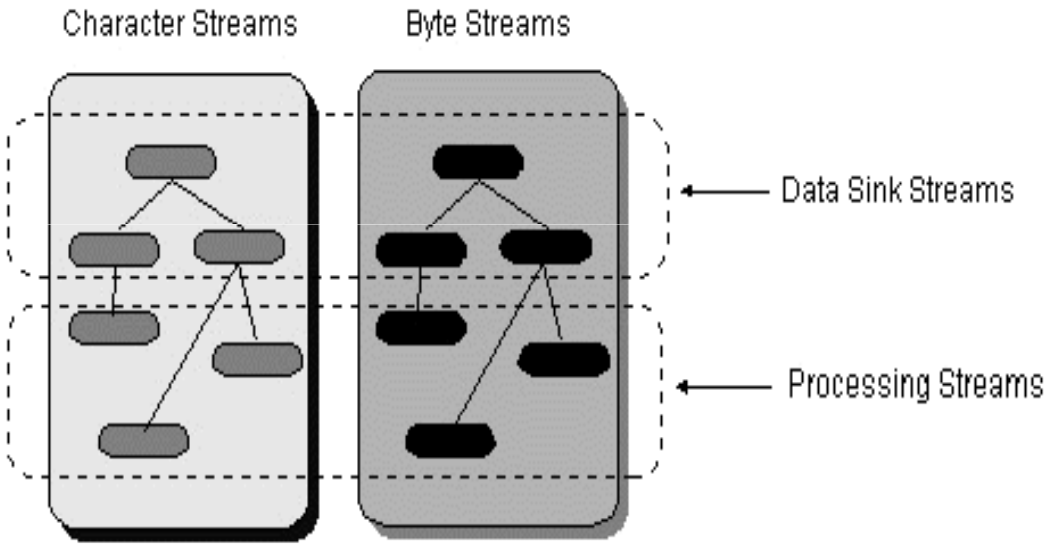


Reading & Writing Data

- java.io Package includes these Stream Classes
  - *Character Streams* are used for 16-bit Characters – Uses *Reader* & *Writer* Classes
  - *Byte Streams* are used for 8-bit Bytes – Uses *InputStream* & *OutputStream* Classes Used for Image, Sound Data etc.

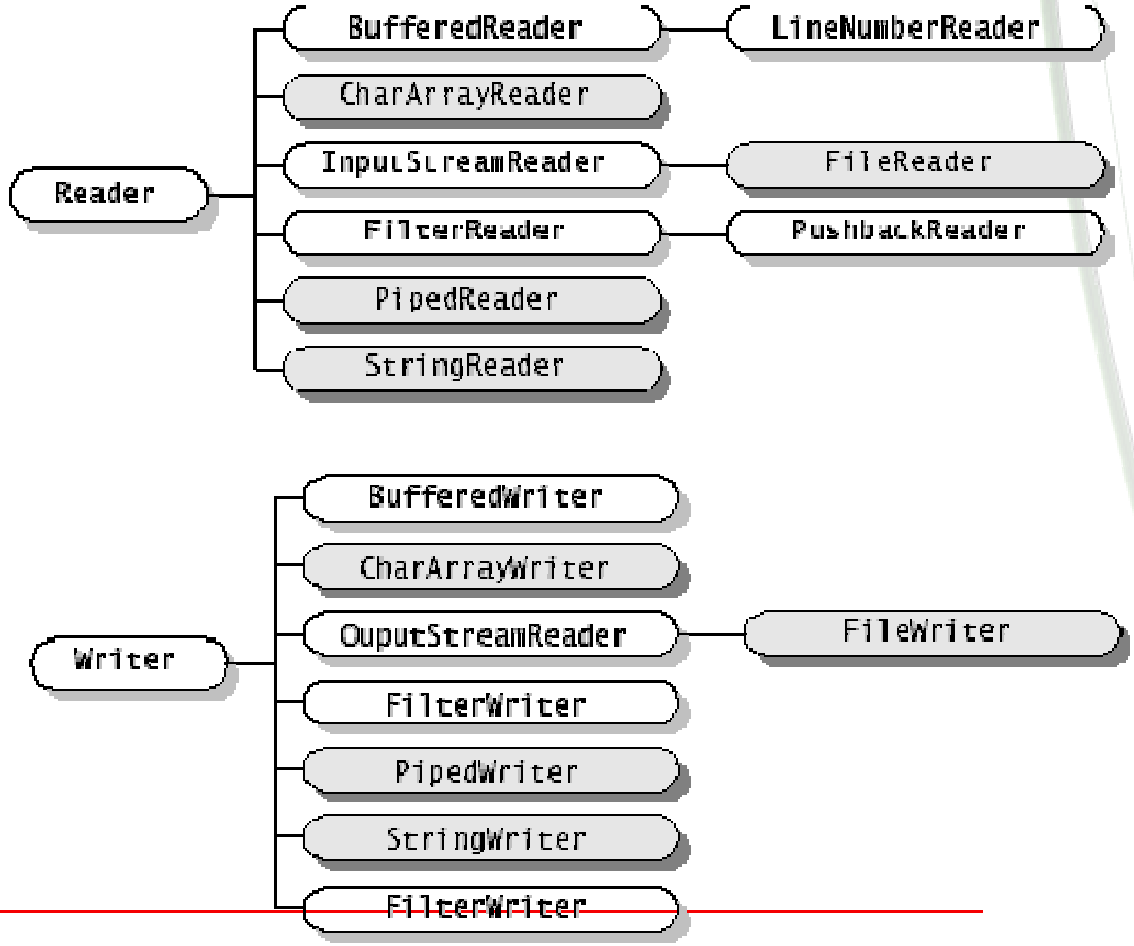
Reading & Writing Data

- Data Sinks
  - Files
  - Memory
  - Pipes
- Processing
  - Buffering
  - Filtering



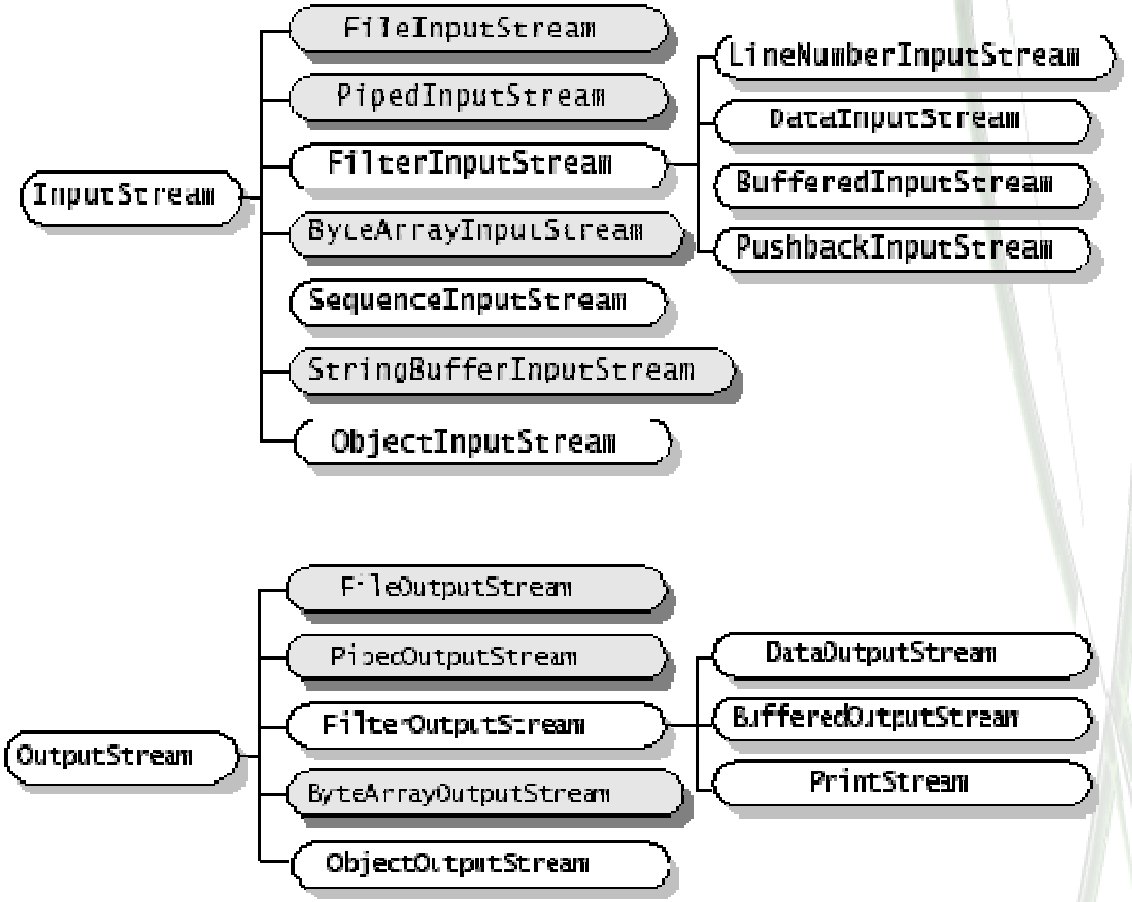
Character Streams

- Reader and Writer are abstract super classes for character streams (16-bit data)
- Sub classes provide specialized behavior



Byte Streams

- InputStream and OutoutStream are abstract super classes for byte streams (8-bit data)
- Sub classes provide specialized behavior



I/O Super Classes

- Reader and InputStream define similar APIs but for different data types

```
int read()
int read(char cbuf[])
int read(char cbuf[], int offset, int length)
```

Reader

```
int read()
int read(byte cbuf[])
```

InputStream

```
int read(byte cbuf[], int offset, int length)
```

I/O Super Classes

- Writer and OutputStream define similar APIs but for different data types

int write()
int write(char cbuf[])
int write(char cbuf[], int offset, int length)

Writer

int write()
int write(byte cbuf[])
int write(byte cbuf[], int offset, int length)

OutputStream

Type of I/O	Streams	Description
Memory	CharArrayReader CharArrayWriter ByteArrayInputStream ByteArrayOutputStream	Use these streams to read from and write to memory. You create these streams on an existing array and then use the read and write methods to read from or write to the array.
	StringReader StringWriter StringBufferInputStream	Use StringReader to read characters from a String in memory. Use StringWriter to write to a String. StringWriter collects the characters written to it in a StringBuffer, which can then be converted to a String. StringBufferInputStream is similar to StringReader, except that it reads bytes from a StringBuffer.
Pipe	PipedReader PipedWriter PipedInputStream PipedOutputStream	Implement the input and output components of a pipe. Pipes are used to channel the output from one thread into the input of another.
File	FileReader FileWriter FileInputStream FileOutputStream	Collectively called file streams, these streams are used to read from or write to a file on the native file system.
Object Serializati-on	*N/A* ObjectInputStream ObjectOutputStream	Used to serialize objects.

Stream wrapping

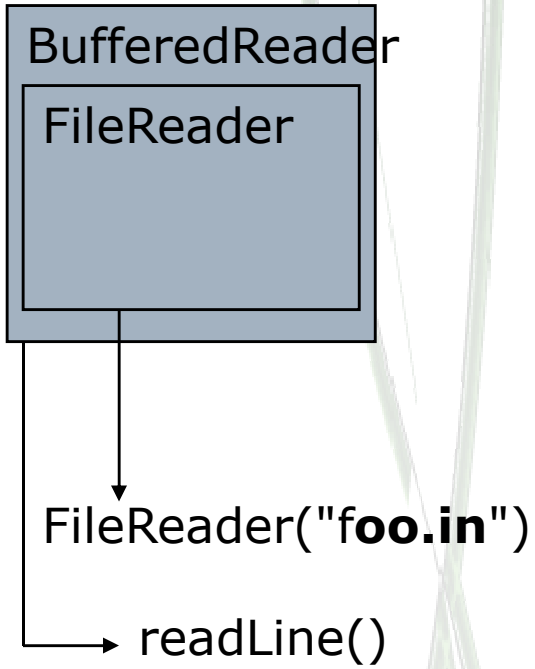
- BufferedReader class can be used for efficient reading of characters, arrays and lines

```
BufferedReader in = new BufferedReader(new
      FileReader("foo.in"));
```

- BufferedWriter and PrintWriter classes can be used for efficient writing of characters, arrays and lines and other data types

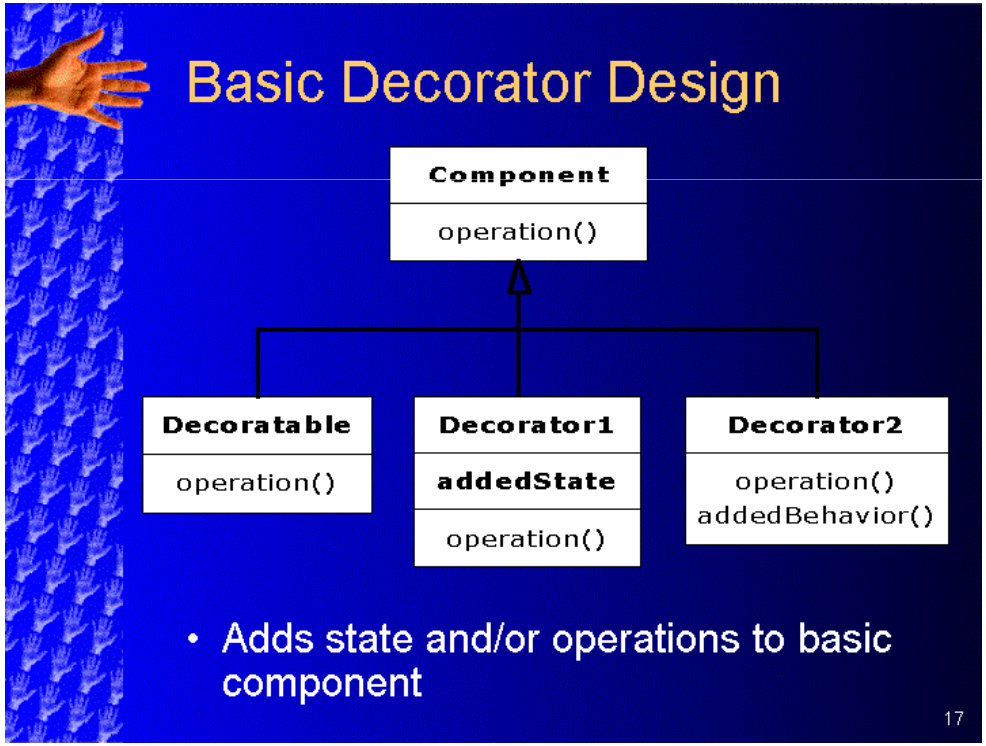
```
BufferedWriter out = new
      BufferedWriter(newFileWriter("foo.out"));
```

```
PrintWriter out= new PrintWriter(new BufferedWriter(new
      FileWriter("foo.out")));
```



Decorator Pattern

- Capabilities are added using a design called the **Decorator Pattern**.



Purpose of Decorator

- Best way to think of this is as follows:
  - There are two important issues when constructing an i/o library
    - Where the i/o is going (file, etc).
    - How the data is represented (String, native type, etc.)
  - Rather than create a class for each combination, Decorator classes allow you to mix and match, augment functionality of base classes.
  - This is a bit confusing but is very flexible.
  - Decorators can also add other capabilities, such as peek ahead, push back, write line number, etc.

File Handling

- To Read from & Write to Files
  - FileReader / FileInputStream
  - FileWriter / FileOutputStream

FileReader - FileInputStream… What is the difference anyway?

- The Streams are Opened when they are Created
- They can be Closed by using the close() Method

File Handling - Character Streams

```
import java.io.*;
public class CopyCharacters {
public static void main(String[] args) throws IOException {
      File inputFile = new File("InputFile.txt");
      File outputFile = new File("OutputFile.txt");
      FileReader in = new FileReader(inputFile);
      FileWriter out = new FileWriter(outputFile);
      int c;

      while ((c = in.read() ) != -1)   // Read from Stream
            out.write(c);              // Write to Stream
      in.close();
      out.close();
      }
}
```

Create File Objects

Create File Streams

Close the Streams

File Handling

- The *File* Object represents the File that is being read or written to

- FileStreams can even be created without the File Object
  - **FileReader**(String fileName)

Getting User Input in Command Line

- Read as reading from the standard input device which is treated as an input stream represented by System.in

```
BufferedReader input= new
      BufferedReader(newInputStreamReader(System.in
));
System.out.println("Enter the name :"  );
String name =input.readLine();
```

- Throws java.io.IOException

Object Serialization

- To allow to *Read & Write Objects*
- The State of the Object is represented in a *Serialized* form sufficient to reconstruct it later
- Streams to be used
  - *ObjectInputStream*
  - *ObjectOutputStream*

Object Serialization

- Object Serialization is used in
  - Remote Method Invocation (RMI) : communication between objects via sockets
  - Lightweight persistence : the archival of an object for use in a later invocation of the same program

Object Serialization

- An Object of any Class that implements the *Serializable Interface* can be serialized
  - public class MyClass implements Serializable {

    ...

    }
- Serializable is an Empty Interface, no methods have to be implemented

Object Serialization

- Writing to an ObjectOutputStream

```
FileOutputStream fos = new FileOutputStream("t.tmp");
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeInt(12345);
oos.writeObject("Today");
oos.writeObject(new Date());
oos.close();
```

- ObjectOutputStream must be constructed on another Stream

Object Serialization

- Reading from an ObjectInputStream

```
FileInputStream in = new
FileInputStream("Time");
ObjectInputStream s = new
ObjectInputStream(in);
String today = (String)s.readObject();
Date date = (Date)s.readObject();
```

- The objects must be read from the stream in the same order in which they were written

Object Serialization

- Specialized behavior can be provided in serilazation and deserialization by implementing the following methods

```
private void writeObject(java.io.ObjectOutputStream out)
    throws IOException
```

```
private void readObject(java.io.ObjectInputStream in) throws
    IOException, ClassNotFoundException;
```

How Do I…

- Determine If a File or Directory Exists
- Create a Temporary File
- Traverse the Files and Directories Under a Directory
- load a zipped file
- Copy text file

How Do I determine If a File or Directory Exists

```
boolean exists = (new
    File("filename")).exists();
if (exists) {
    // File or directory exists
}
else {
    // File or directory does not exist
}
```

How Do I Create a Temporary File

```
try {
  // Create temp file.
  File temp = File.createTempFile("pattern",
    ".suffix");
  // Delete temp file when program exits.
  temp.deleteOnExit();
  // Write to temp file
  BufferedWriter out = new BufferedWriter(new
    FileWriter(temp));
  out.write("aString");
  out.close();
}
catch (IOException e) { … }
```

How Do I traverse the Files and Directories Under a Directory

```
// Process all files and directories under dir
public void visitAllDirsAndFiles(File dir) {
    //do something useful with the file object
    process(dir);
    if (dir.isDirectory()) {
       String[] children = dir.list();
       for (int i=0; i<children.length; i++) {
              visitAllDirsAndFiles(
                     new File(dir ,children[i]));
       }
     }
}
```

How Do I load a zipped file

```
public void loadZipFile(String name, String zipName) {
  Try{
       ZipInputStream zin = new ZipInputStream(new FileInputStream(zipName));
        ZipEntry entry;
        String text=“”;
       //find entry with matching name in archive
        while ((entry = zin.getNextEntry()) != null){
              if (entry.getName().equals(name))   {
              // read entry into text area
              BufferedReader in = new BufferedReader(new
       InputStreamReader(zin));                                    String line;
            while ((line = in.readLine()) != null) {
                text.concat(line);
                text. concat("\n");  } }
                zin.closeEntry();
}
    zin.close();}
 catch (IOException e) {e.printStackTrace();}}
```

Copying a Text File

```
void copyFile(FileReader inputFile,
              FileWriter outputFile)
                                throws IOException {
    final int bufferSize = 1024;
    char[] buffer = new char[bufferSize];
    // Read the first chunk of characters.
    int numberRead = inputFile.read(buffer);
    while(numberRead > 0){
        // Write out what was read.
        outputFile.write(buffer,0,numberRead);
        numberRead = inputFile.read(buffer);
    }
    outputFile.flush();
}
```

Reading from URLs

```
import java.net.*;
import java.io.*;
public class URLReader {
    public static void main(String[] args) throws Exception {
            URL yahoo = new URL("http://www.yahoo.com/");
            BufferedReader in = new BufferedReader(
                                    new InputStreamReader(
                                    yahoo.openStream()));
            String inputLine;
            while ((inputLine = in.readLine()) != null)
                System.out.println(inputLine);
            in.close();
    }
}
```