

CS61B Lecture #8

Administrative:

- Discussion 117 (11-12 Thurs) moves to 412 MLK Student Union *next week*.
- Homework 2 due tonight, lab 2 due Tuesday.
- Test #1: Thursday evening, 11 October. See me the week before if you need an alternative time.

Today: Object-Oriented Examples

Readings Today: *On To Java*, chapters 14-16, 19, 20

To Read: *On To Java*, chapters 17, 18, 27, 35, 36. *Programming Into Java*, §4.4, §4.5, §4.7, §4.8, §4.10, §5.6.3, §5.8.

Last modified: Mon Sep 17 10:40:06 2001

CS61B: Lecture #8 1

Example: Comparable

- Java library provides an interface to describe Objects that have a *natural order* on them.

```
public interface Comparable {
    /** Returns value <0, == 0, or > 0 depending on whether
     * THIS is <, ==, or > OBJ. Throws ClassCastException
     * if OBJ is not comparable to THIS. */
    int compareTo (Object obj);
}
```

- Type String is Comparable, as are BigInteger and BigDecimal in java.math, and others.
- Might use in a general-purpose max function:

```
/** The largest value in array A, or null if A empty. */
public static Object max (Comparable[] A) {
    if (A.length == 0) return null;
    Comparable result = A[0];
    for (int i = 1; i < A.length; i += 1)
        if (result.compareTo (A[i]) < 0)
            result = A[i];
    return result;
}
```

- Now max(S) will return maximum value in S if S is an array of Strings, or any other kind of Object that implements Comparable.

Last modified: Mon Sep 17 10:40:06 2001

CS61B: Lecture #8 2

Example: Readers

- Java class java.io.Reader abstracts *sources of characters*.
- Here, we present a revisionist version (not the real thing):

```
public interface Reader {
    /** Release this stream: further reads are illegal */
    void close ();

    /** Read as many characters as possible, up to LEN,
     * into BUF[OFF], BUF[OFF+1],..., and return the
     * number read, or -1 if at end-of-stream. */
    int read (char[] buf, int off, int len);

    /** Read and return single character, or -1 if
     * end-of-stream reached. */
    int read ();

    /** Short for read (BUF, 0, BUF.length). */
    int read (char[] buf);
}
```

- Is an interface, so all methods abstract, public.
- Can't write new Reader(); it's incomplete (abstract).
- So what good is it?

Last modified: Mon Sep 17 10:40:06 2001

CS61B: Lecture #8 3

Generic Partial Implementation

- According to their specifications, some of Reader's methods are related.
- Can express this with a partial implementation, which leaves key methods unimplemented and provides default bodies for others.
- Result still abstract: can't use new on it.

```
/** A partial implementation of Reader. Complete
 * implementations must override close and read(,,).
 * They MAY override the other read methods for speed. */
public abstract class AbstractReader implements Reader {
    public abstract void close ();
    public abstract int read (char[] buf, int off, int len);

    /** Read and return single character, or -1 if
     * end-of-stream reached. Default implementation;
     * may be overridden for speed. */
    public int read ()
    { return (read (buf1) == -1) ? -1 : buf1[0]; }

    public int read (char[] buf)
    { return read(buf,0,buf.length); }

    /** Temporary buffer for read(), created once for speed.
     private char[] buf1 = new char[1];
}
```

Last modified: Mon Sep 17 10:40:06 2001

CS61B: Lecture #8 4

Implementation of Reader: InputStreamReader

The class InputStreamReader reads characters from an InputStream (such as System.in). Here's a simplified version:

```
public class InputStreamReader extends AbstractReader {
    private InputStream str;

    /** A Reader whose source is bytes from STR */
    public InputStreamReader (InputStream str)
    { this.str = str; }

    public void close () { str.close (); }

    public int read (char[] buf, int off, int len) {
        // InputStreams read bytes, so...
        byte[] bytes = new byte[len];
        int cnt = str.read (bytes, 0, len);
        for (int i = 0; i < cnt; i += 1)
            buf[i+off] = (char) bytes[i];
        return cnt;
    }
}
```

- Only had to define the two abstract routines, and the rest are simply inherited.

Last modified: Mon Sep 17 10:40:06 2001

CS61B: Lecture #8 5

Implementation of Reader: FileReader

The class FileReader reads characters from a file. Here's a simplified version:

```
public class FileReader extends InputStreamReader {

    /** A Reader whose source is bytes from the file named
     * FILENAME */
    public FileReader (String fileName)
        throws FileNotFoundException
    {
        super (new FileInputStream (fileName));
    }
}
```

- A FileInputStream is a kind of InputStream whose bytes come from a file.
- Everything but the constructor is inherited.
- The super(...) syntax means "call the constructor of my superclass with these arguments." Must be first statement in constructor.
- The throws syntax announces that the constructor for FileReader might throw the given exceptions. This would happen if the constructor for FileInputStream couldn't find the specified file.

Last modified: Mon Sep 17 10:40:06 2001

CS61B: Lecture #8 6

Implementation of Reader: StringReader

The class StringReader reads characters from a String:

```
public class StringReader extends AbstractReader {
    private String str;
    int k;
    /** A Reader delivering the characters in STR. */
    public StringReader (String str)
        { this.str = str; k = 0; }

    public void close () { str = null; }

    public int read (char[] buf, int off, int len) {
        if (k == str.length ())
            return -1;
        len = Math.min (len, str.length () - k);
        str.getChars (k, k+len, buf, off);
        k += len;
        return len;
    }
}
```

Last modified: Mon Sep 17 10:40:06 2001

CS61B: Lecture #8 7

Using Reader

Consider this method, which counts words:

```
/** The total number of words in R, where a "word" is
 * a maximal sequence of non-whitespace characters. */
int wc (Reader r) {
    int c0, cnt;
    c0 = ' '; cnt = 0;
    while (true) {
        int c = r.read ();
        if (c == -1)
            return cnt;
        if (Character.isWhitespace ((char) c0)
            && ! Character.isWhitespace ((char) c))
            cnt += 1;
        c0 = c;
    }
}
```

- This method works for any Reader:

```
// Number of words in standard input.
wc (new InputStreamReader (System.in))
// Number of words in file named fileName:
wc (new FileReader (fileName))
// Number of words in the String someText:
wc (new StringReader (someText))
```

Last modified: Mon Sep 17 10:40:06 2001

CS61B: Lecture #8 8

Lessons

- The Reader interface class served as a *specification* for a whole set of readers.
- Ideally, most client methods that deal with Readers will specify type Reader for the formal parameters, rather than any specific kinds of Reader,
- And only when a client creates a new Reader will it get specific.
- That way, client's methods are as *widely applicable* as possible.
- Finally, AbstractReader is a tool for implementors of non-abstract Reader classes, and not used by clients.
- Alas, Java library is not pure. E.g., AbstractReader is really just called Reader and there is no interface. In this example, we saw what they *should* have done!
- The Comparable interface allows definition of functions that depend only on a limited subset of the properties (methods) of their arguments (such as "must have a compareTo method").

Last modified: Mon Sep 17 10:40:06 2001

CS61B: Lecture #8 9