CS61B Lecture #8

- Midterm tentatively scheduled for the evening of 19 October (Tuesday). Format: 2 hour, open-book. We will accommodate students with conflicts as needed; please arrange an alternative time with us the week before the exam (11–15 Oct.).
- Project #1 (coming soon) will be due on 13 Oct. (Wednesday night).
- Some advice: The Blue Reader contains information on the tools you are using (debugger, Emacs, make, javac compiler, etc.). *Be curious* about the software you're using and the software we give you. For example, what is our test framework doing and how? (How could you find this out?)

Example: Comparable

- Java library provides an interface to describe Objects that have a *natural order* on them, such as String Comparable, BigInteger and BigDecimal:

```
public interface Comparable { // Java 1.4 version for now
  /** 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);
}
```

- 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.

Example: Readers

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

```
public interface Reader {  // Real java.io.Reader is abstract class
  /** 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);

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

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

- Can't write `new Reader()`; it's abstract. So what good is it?

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);

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

  public int read () { return (read (buf1) == -1) ? -1 : buf1[0]; }

  private char[] buf1 = new char[1];
}
```

Implementation of Reader: StringReader

The class `StringReader` reads characters from a String:

```
public class StringReader extends AbstractReader {
  private String str;
  private 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;
  }
}
```

Using Reader

Consider this method, which counts words:

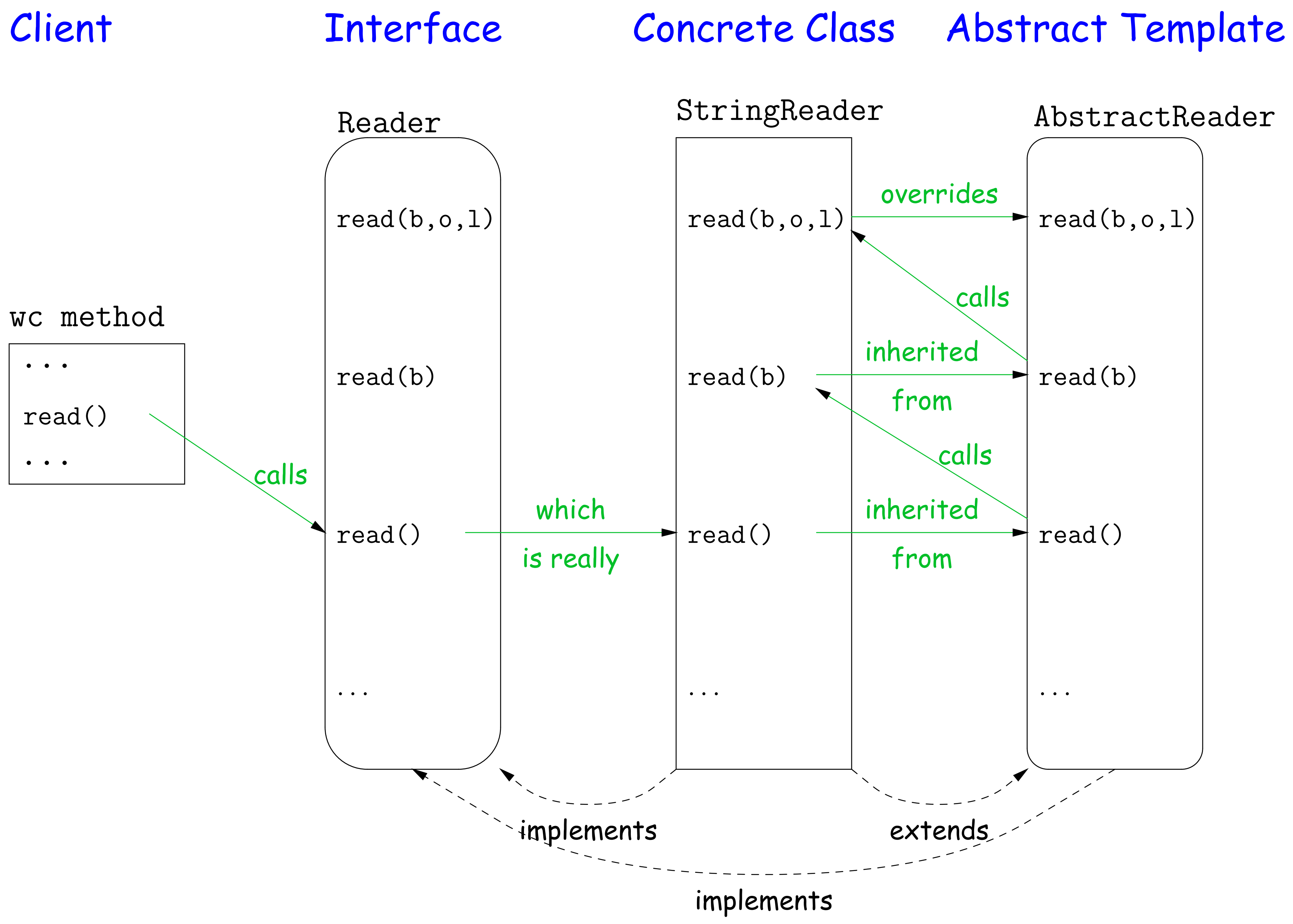
```
/** The total number of words in R, where a "word" is
 *  a maximal sequence of non-whitespace characters. */
int wc (Reader r) {
  int c0, count;
  c0 = ' '; cnt = 0;
  while (true) {
    int c = r.read ();
    if (c == -1) return count;
    if (Character.isWhitespace ((char) c0) && ! Character.isWhitespace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

This method works for *any* `Reader`:

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

$\left.\begin{array}{l}\\ \\ \\ \end{array}\right\}$ other implementations of Reader

How It Fits Together



Lessons

- The `Reader` interface class served as a *specification* for a whole set of readers.
- Ideally, most client methods that deal with `Readers`, like `wc`, will specify type `Reader` for the formal parameters, not a specific kind of `Reader`, thus assuming as little as possible.
- And only when a client creates a new `Reader` will it get specific about what subtype of `Reader` it needs.
- 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").