Fast I/O in Java (for Programming Contest Websites)

Standard mechanisms for input and output in Java take a great deal of time. In particular, Scanner methods check many details to sanitize the input as best as possible. If the input format is known, many of these checks are unnecessary.

An alternate class, called BufferedReader, allows for faster input, especially if you are reading more than 10^5 tokens to solve a problem. The class is very simple and only allows reading of one full input line at a time. Since more than one token may be on a line, BufferedReader must be used in conjunction with a StringTokenizer object, whenever a line has more than one token.

To set up a BufferedReader, do this:

```
BufferedReader stdin =
              new BufferedReader(new InputStreamReader(System.in);
```

This class also requires us to make two additions to our code.

(a) We must import java.io.*:

```
import java.io.*;
```

(b) We must report that our main method may throw an IOException:

```
public static void main(String[] args) throws IOException {
// code for class in here.
}
```

The only method we'll use from BufferedReader is readLine():

```
String line = stdin.readLine();
```

This line of code reads a full line from standard input, strips the newline character and returns the full line as a string.

If we know that the String has several tokens, then we can use a StringTokenizer object to separate out the tokens:

```
StringTokenizer tok = new StringTokenizer(line);
```

If we happen to know that the delimiters between tokens are different than a space on the line, we can pass those characters into the StringTokenizer constructor. For example, let's say that our delimiters are either a space, colon or comma, we would create the StringTokenizer object as follows:

```
StringTokenizer tok = new StringTokenizer(line, " ,:");
```

Now, let's take a look at the important methods in the StringTokenizer class:

Modifier and Type	Method and Description
int	**countTokens**() Calculates the number of times that this tokenizer's nextToken method can be called before it generates an exception.
boolean	**hasMoreTokens**() **Tests if there are more tokens available from this tokenizer's string.**
String	**nextToken**() **Returns the next token from this string tokenizer.**

We can think of a StringTokenizer as starting with a bookmark at the front of it. The countTokens() method will return the number of separate tokens AFTER the position of the bookmark whenever you

call it. The hasMoreTokens() method returns true if there's at least one token after the position of the bookmark. The nextToken() method advances the bookmark from its current position to the position of the next delimiter (or end of the Tokenizer) making sure it's advanced past at least one non-delimiter character, then it returns the full string of non-delimiter characters that it advanced through.

Consider this string:

```
"Jelena  20:02:13,Finish :, Last"
```

If the delimiters are the space, colon and comma, then a StringTokenizer constructed with this string will have 6 total tokens:

```
Jelena
20
02
13
Finish
Last
```

As you can see, it doesn't matter if there is 1 or several delimiters between tokens, all get skipped. So, the first time nextToken gets called, "Jelena" gets returned. The next time "20" (as a String) gets returned. And so forth.

To convert a String to an int, long or double, respectively we use the following methods:

```
int x = Integer.parseInt(s);
long x = Long.parseLong(s);
double x = Double.parseDouble(s);
```

We can encapsulate all of this in a new class called FastScanner, so that we can call all the Scanner methods, but in the back end, a combination of the tools above are actually being used so that the input is read in faster, BUT, when coding the programmer acts as if they are using Scanner. Here is a version of a FastScanner class:

```
class FastScanner {

    BufferedReader br;
    StringTokenizer st;

    public FastScanner(InputStream i) {
        br = new BufferedReader(new InputStreamReader(i));
        st = new StringTokenizer("");
    }

    public String next() throws IOException {
        if(st.hasMoreTokens())
            return st.nextToken();
        else
            st = new StringTokenizer(br.readLine());
        return next();
    }

    public int nextInt() throws IOException {
        return Integer.parseInt(next());
    }

    public long nextLong() throws IOException {
        return Long.parseLong(next());
    }

    public double nextDouble() throws IOException {
        return Double.parseDouble(next());
    }

}
```

After adding this class to your code, when you need to read in items from main, just create a FastScanner object instead of a Scanner object:

```
FastScanner stdin = new FastScanner(System.in);
```

Fast Output in Java

A single System.out.println() statement is relatively slow. Thus, if we do more than 10^5 of these, that can really slow down a program. A better strategy is to build the output in a single object, and then print out that whole object.

Unfortunately, Strings are immutable in Java, so they can't be added to quickly. If we do the statement:

```
s = s + t;
```

in Java, where s and t are strings n times, where t is a single character, the run time will be $O(n^2)$ because each time the string concatenation occurs, all the characters in s, followed by all the characters in t are copied into a new String object, which s then refers to.

Instead, we would like to make use of the principles in CS1. If we double the length of our buffer each time we fill it as we add new items, we can get an amortized run time of O(1) per adding each character. One class that allows for this in Java is StringBuffer.

Here is how to use it.

(1) Create an empty StringBuffer object:

```
StringBuffer sb = new StringBuffer();
```

(2) Whenever you need to, append a string to the object as follows:

```
sb.append(s);
```

Note that you don't get any spaces or newlines for "free" so you have to explicitly put those in. Let's say that you want to output n integers, one per line. Here is a sketch of how that might look:

```
StringBuffer sb = new StringBuffer();
for (int i=0; i<n; i++) {
    int res = solve();
    sb.append(res+"\n");
}
```