Worksheet 5: Language Integration

Updated: 19th September, 2019

1. Java Native Interface (JNI)

Use the JNI to implement your own console IO class, as follows:

```
import java.util.List;

public class ConsoleIO
{
    ... // <-- load native code library.

    public native static double read(double defaultValue);
    public native static void printStr(String text);
    public native static void printList(List<String> list);
}
```

The `read()` method should attempt to read a real number from the console and return it, or otherwise return `defaultValue` if the user fails to enter a valid number.

`read()` should not require any JNI "`(*env)->...`" calls, but `printStr()` and `printList()` will. You will need to consult the JNI reference documentation, and in particular the functions for obtaining C strings from Java's UTF strings.

For `printList()`, the objective is to simply print out every value in the list, separated by commas. So, if the list contained the strings "abc", "def" and "ghi", then the method should print "`abc, def, ghi`".

You should be able to use Gradle to build the project as shown in the lecture notes. You should also be able to incorporate a simple Java test harness to see whether your IO class works.

Note: This exercise is purely for educational purposes, of course. Native code is best used to interact with C-based libraries not otherwise available in Java, or to perform C-based optimisations of complex algorithms. But in an educational setting, both of these require a lot of additional overhead on top of understanding how JNI itself works.

However, if you have some spare motivation, you could (for instance) see if you can use the C-based OpenGL/freeglut libraries to work in a Java application.

End of Worksheet