COP5555 Assignment 6

Due Friday, April 24 at 5pm.

Create a class CodeletBuilder that can be called from a java program in order to use our little language as an embedded language in a Java program.

Here is an example:

```
package cop5555sp15;

import java.util.List;

public class InitializeList {

	public static void main(String[] args) throws Exception{
		String source = "class ListInit{\n"
				+ "def l1: @[int];\n"
				+ "def l2: @[string];\n"
				+ "def i1: int;\n"
				+ "l1 = @[300,400,500];\n"
				+ "l2 = @[\"go\", \"gators\"];\n"
				+ "i1 = 42;\n"
				+ "}";
		Codelet codelet = CodeletBuilder.newInstance(source);
		codelet.execute();

		@SuppressWarnings("rawtypes")
		List l1 = CodeletBuilder.getList(codelet, "l1");
		System.out.println(l1.size());
		System.out.println(l1.get(0));
		int i1 = CodeletBuilder.getInt(codelet,  "i1");
		System.out.println(i1);
	}

}
```

The CodeletBuilder class has a static method newInstance that takes a source String and returns a Codelet object. You can execute the codelet by calling its execute method. The variables in the codelet class can be obtained with getList, getInt, getBoolean, and getString methods. These take the codelet and the name of the variable. CodeletBuilder should also provide a method to get the source from a File object and methods to set int, boolean, and string variables in our language as in the skeleton class on the next page:

```
package cop5555sp15;

import java.io.File;
import java.io.FileReader;
import java.lang.reflect.Field;
import java.util.List;
import cop5555sp15.ast.CodeGenVisitor;
import cop5555sp15.ast.Program;
import cop5555sp15.ast.TypeCheckVisitor;
import cop5555sp15.symbolTable.SymbolTable;

public class CodeletBuilder {

	public static Codelet newInstance(String source) throws Exception{
		//TODO
	}
	public static Codelet newInstance(File file) throws Exception {
		//TODO
	}
	@SuppressWarnings("rawtypes")
	public static List getList(Codelet codelet, String name) throws Exception{
		//TODO
	}
	public static int getInt(Codelet codelet, String name) throws Exception{
		//TODO
	}
	public static void setInt(Codelet codelet, String name, int value) throws
Exception{
		//TODO
	}
	public static String getString(Codelet codelet, String name) throws Exception{
		//TODO
	}
	public static void setString(Codelet codelet, String name, String value)
throws Exception{
		//TODO
	}
	public static boolean getBoolean(Codelet codelet, String name) throws
Exception{
		//TODO
	}
	public static void setBoolean(Codelet codelet, String name, boolean value)
throws Exception{
		//TODO
	}
}
```

To implement the newInstance methods, look at the setup in the Junit tests from previous assignments.

To implement the get methods, we will use reflection. The same approach works for all of the get methods: use the getClass method to get a class object. Then get a

Field object using the class object's getDeclaredField method. Get the value of the field, cast it to the required type, and return it. Here is the complete implementation for getInt. The others are similar.

```
	public static int getInt(Codelet codelet, String name) throws Exception{
		Class<? extends Codelet> codeletClass = codelet.getClass();
		Field l1Field = codeletClass.getDeclaredField(name);
		int i =  (int) l1Field.get(codelet);
		return i;
	}
```

For the set methods, get the Class object, get the Field, and then invoke the set method on the field. Here is the implementation for setInt

```
	public static void setInt(Codelet codelet, String name, int value) throws
Exception{
		Class<? extends Codelet> codeletClass = codelet.getClass();
		Field l1Field = codeletClass.getDeclaredField(name);
		l1Field.set(codelet, value);
	}
```

You don't need a setList method. We assume that you don't want to change the List object, just its contents, which you can do with the List methods. (I have not carefully defined the semantics of lists and when they are instantiated in our language. If you want, you might think about that problem as you are working on this.) For our purposes, you may assume that none of the get methods are used until after the execute methods has been called at least once.

You do not need to worry about handling syntax or any other errors in a graceful way. You just want to ensure that it will work for correct source programs similar to the test cases we have previously done. In addition, the following code. which invokes execute twice, should work and produce output:

```
first time
0
2
second time
```

```java
package cop5555sp15;

import java.util.List;

/** This class illustrates calling execute twice after modifying the int variable
 * in the codelet.  The expected output is

     first time
     0
     2
     second time

*/
public class CallExecuteTwice {

	public static void main(String[] args) throws Exception{
		String source = "class CallExecuteTwice{\n"
				+ "def i1: int;\n"
				+ "if (i1 == 0){print \"first time\";}\n"
				+ "else {print \"second time\";};\n"
				+ "}";
		Codelet codelet = CodeletBuilder.newInstance(source);
		codelet.execute();
		int i1 = CodeletBuilder.getInt(codelet,  "i1");
		System.out.println(i1);
		CodeletBuilder.setInt(codelet, "i1", i1+2);
		System.out.println(CodeletBuilder.getInt(codelet,  "i1"));
		codelet.execute();
	}

}
```

You should also submit two programs of your own that are different from any of the test cases previously given. Use CallExecuteTwice above as an example. Call these Example1 and Example2 and include a brief comment on what they do and what the expected output is.

Turn in:

A jar file containing source code for CodeBuilder.java, Example1.java, and Example2.java along with all of the rest of the code from your project that is needed to execute them.