JDecaf Companion

The JDecaf Language

JDecaf is a Java-like language like Groovy. It is closer to Java than Groovy is in terms of syntax and it is not as loose and forgiving as Groovy. This implies that JDecaf may not be as flexible as Groovy in development but for the very same reason it can provide a better service as a learning tool in a computer programming course.

The main advantage of JDecaf is consistency and compatibility in relation to Java. All the things that work in Java are guaranteed to work in exactly the same way in JDecaf. There are no discrepancies in terms of behaviour between the two languages. Language constructs of Java can be seamlessly ported into JDecaf and compile without problems but not vice versa. This means that learning JDecaf is essentially the same as learning Java, but simpler.

One feature that really makes JDecaf a very handy tool is its overly simplified I/O (input/output... at the moment limited to reading from the keyboard and writing on the screen). It supports the following methods:

- print(String)
- println(String)
- readInt()
- readDouble()
- readLine()

Every statement in this language must be terminated by a semicolon. Furthermore, arguments in method calls must be parenthesised. Parentheses must be present even if no arguments are needed in the method call. The following simple example depicts these rules:

```
int x=readInt();

int y=readInt();

int sum=x+y;

println(“sum: “+sum);
```

JDecaf programs can be developed with any text editor and they have to be saved with a .jdc extension, otherwise the JDecaf compiler will not recognise them.

The JDecaf Editor

The recommended editor for learning JDecaf is its own editor. JDecaf Editor is just a simple text editor written in Java. The purpose in this case is to provide some functionality in terms of editing source code without all the complexity of Integrated Development Environments (e.g. Eclipse), that we will learn to use further down the line. It can recognise JDecaf and Java language constructs and visually distinguish them from other elements in the text. Functionality is kept to a minimum so that students

can learn good coding practices by implementing them themselves. The editor can be obtained from the following github repository:

https://github.com/skarkalas/JDecafEditor.git

Please consult the readme.txt carefully before installation. You may experience a small delay on start-up and as you write the first letters of your code. This is normal.

The JDecaf Compiler

The JDecaf compiler can be used for both .jdc and normal .java files. In the latter case it compiles the java files as a normal java compiler. The user can also mix files in the same compilation instruction. The software can be cloned from the following repository:

https://github.com/skarkalas/JDecafCompiler.git

Please note that the compiler requires a separate installation.

More Advanced Development with JDecaf (for later)

As said at the beginning of this text, any statement expressed in pure Java, is considered valid JDecaf code and thus processed by the compiler. Therefore, more advanced features like class definitions can also be part of a JDecaf program and seamlessly interoperate with the main section of it. The rule is simple: The main section has to be written first in the .jdc file (at the top) and after this section is completed any number of class definitions may follow (at the bottom). These classes cannot be set as public and individual statements are not allowed in between and after them. The following code fragment depicts this rule:

```
int x = readInt();
int y = readInt();
int sum = SimpleMath.sum(x,y);
SimpleMessage smessage = new SimpleMessage();
String message = smessage.getMessage(sum);
println(message);

class SimpleMath {
      public static int sum(int x,int y) {
            return x+y;
      }
}

String fname="";        // Incorrect! no code after defining a class

class SimpleMessage {
      String message="The sum is";

      public String getMessage(int sum) {
            return message+":"+sum;
      }
}

String sname="";        // Incorrect! No code after defining a class
```