How could interoperability be implemented (Eolang-Java)?

From the findings in (*Exploring the Eo-Java Interoperability*), one of the major mechanisms that enable these languages to interoperate with Java is to share some similarities with Java language and support some Java syntax. The languages discussed, in a sense, are like supersets of Java. Thus, they support Java features and extend more features. Their syntaxes are not very different from that of Java, so it makes it easy for them to support Java and interoperate. Xtend compiles to Java but the syntax is rather very close to that of Java. Eolang, currently, hardly supports any of Java features or syntax.

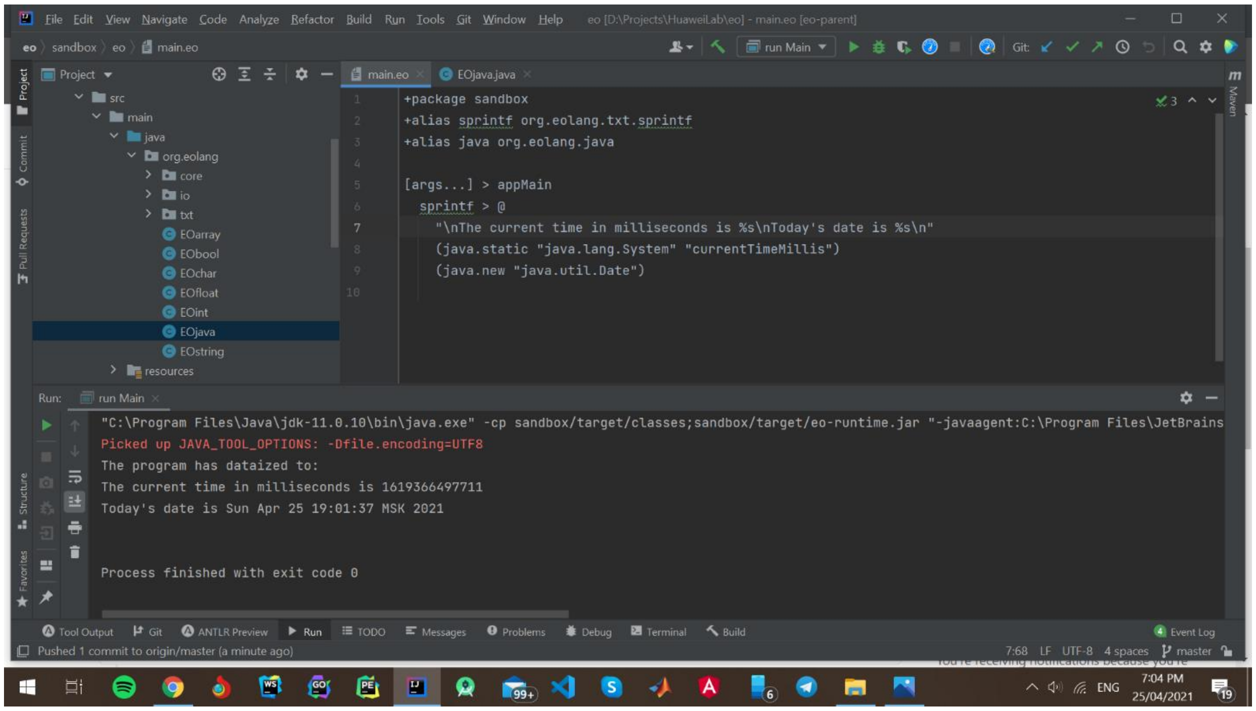
With the current implementation of Eolang, perhaps some core objects could be developed to interface java objects so that programmers can simply create and make copies of Java objects in Eolang. This could be one of the ways Eolang might support Java. The objects will seem like a wrapper library for Java APIs to make them interoperable. This can be a good start with progressive developmental updates to, perhaps, establish greater support of Java and interoperability of Eolang with Java.

In theory, Eolang could increase the number of atoms in the language core library. In Eolang atoms (`if`, `sprintf`, `add`, `stdout`, `and` `length`) have to be implemented in Java, not in Eolang. These objects are datarized at runtime [21]. Although this may provide access to use some Java objects on Eolang, it may not establish interoperability with Java fully. But this can be a start, with further improvements later. Perhaps regular programmers could be given the chance to be able to make custom atom by writing their own java classes and somehow making them atoms in Eolang. This might be rather a lot of work because many Java object may have to be developed as atoms in Eolang, for use in Eolang. Thus, the limitation is that more than a few wrappers would be required to be able to use the different varieties of java libraries and APIs. Listing 1 shows a concept of how interoperability could look like in Eolang. Fig. 1 shows a proof of this concept.

Listing 1. A Concept of EO-Java Interoperability

```
+alias org.eolang.java

[] > main
  stdout
    sprintf
      "Today is %s"
      (java.new "java.util.Date").toString
```



```
+package sandbox
+alias sprintf org.eolang.txt.sprintf
+alias java org.eolang.java

[args...] > appMain
  sprintf > @
    "\nThe current time in milliseconds is %s\nToday's date is %s\n"
    (java.static "java.lang.System" "currentTimeMillis")
    (java.new "java.util.Date")
```

```
"C:\Program Files\Java\jdk-11.0.10\bin\java.exe" -cp sandbox/target/classes;sandbox/target/eo-runtime.jar "-javaagent:C:\Program Files\JetBrains
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF8
The program has dataized to:
The current time in milliseconds is 1619366497711
Today's date is Sun Apr 25 19:01:37 MSK 2021


Process finished with exit code 0
```

Fig. 1. Proof of concept. The static java method, "currentTimeMillis" is executed using the java wrapper "java.static" which turns the "currentTimeMillis" into an Eolang object. The "java.new" wrapper is used to instantiate the "java.util.Date" class.

Summary

The Eo-java wrapper can:

- Create an instance of a class (e.g., java.new "java.util.Date")
- Execute static methods of a class (e.g., java.static "java.lang.System" "currentTimeMillis")
- Execute methods of an instance of a class (e.g., java.instanceMethod "java.util.date" "getTime")

Limitation

Type casting and type conversion are a challenge. The data types are different in both languages so it is a challenge to anticipate what data types may be passed as parameters and what types of data (parameter data type) the various java methods may expect while applying the reflection API in the wrapper.

Reference

Exploring the Eo-Java Interoperability, page 13-15