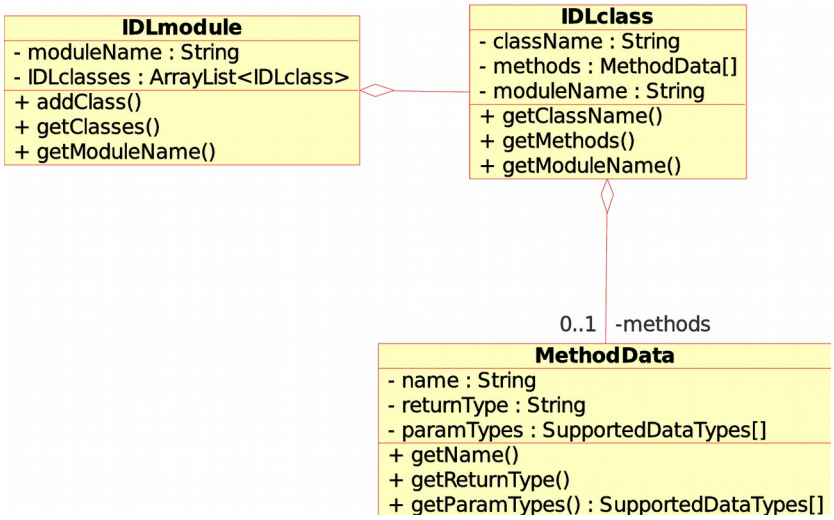
Beispiel für einen einfachen IDL-Parser

(Anmerkung: Die Benutzung dieses Beispielcodes oder Teilen davon ist **optional**!)

Für den hier vorgestellten Parser gelten die Einschränkungen aus der Aufgabenstellung. (1 Modul pro Datei, etc.)
Der vorliegende Parser liefert für eine gegebene IDL-Datei eine Datenstruktur der nachfolgenden Form:



In der `main()`-Methode (Klasse `Parser`) braucht es nur zwei Aufrufe – Konstruieren des Readers für die IDL-Datei und Anstossen des Parsens:

```
IDLfileReader in = new IDLfileReader(new FileReader(IDLfileName));
IDLmodule module = parseModule(in);  // Parse IDL file
```

Um den hieraus resultierenden Baum von der Modul-Ebenen ausgehend abzuscannen, gehe man wie in der Methode `printModule(IDLmodule module)` der Klasse `Parser` vor:

```
System.out.println("module: "+ module.getModuleName());

// classes
IDLclass[] classes = module.getClasses();
for (int i=0; i<classes.length; i++) {
     System.out.println(" class: " + classes[i].getClassName());

     // methods
     MethodData[] methods = classes[i].getMethods();
     for (int k=0; k<methods.length; k++) {
            System.out.print("  method: " + IDLCompiler.getSupportedIDLDataTypeName(
                   methods[k].getReturnType()) + " " + methods[k].getName() + " ");

            // parameters
            SupportedDataTypes[] paramTypes = methods[k].getParamTypes();
            for (int m=0; m<paramTypes.length; m++) {
                   System.out.print(IDLCompiler.getSupportedIDLDataTypeName(paramTypes[m])
                                                                              + " ");
            }
            System.out.println();
     }
}
```

Für die beiliegende Beispiel-IDL-Datei werden nachfolgende Ausgaben bzw. Daten geliefert:

```
~/parseer> java idl_compiler.Parser
module: math_ops
 class: Calculator
  method: double add double double
  method: string getStr double
```

```
module math_ops {
  class Calculator {
      double add(double a, double b);
      string getStr(double a);
  };
};
```

Der vom **Compiler** zu generierende Java-Code sieht dann so aus:

Zu generierender Java-Code

```
package math_ops;


public abstract class _CalculatorImplBase ... {

  public abstract double add(double a, double b);

  public abstract String getStr(double a);

  public static _CalculatorImplBase narrowCast(
              Object rawObjectRef) { ... }

     ...
}


<ggf. weitere hier benötigte Klassen/Interfaces>
```