Javascripty Redux

Scenario

If you're in this class, then most likely you have completed Principles of Programming Languages and are well skilled in the art of building interpreters for a target programming language. Alas, this skill is a blessing and a curse, as you are now in charge of creating a family of tools for parsing and interpreting various different languages. Among your tasks is to build a parser, interpreter and type checker for both the Javascript and Lua languages. Given your experience in this class, you understand that change occurs quite often, and you expect languages to be added in the future. What you would like would be to have a single program which loads a file and, based on the extension of the filename, create an appropriate Parser, Type Checker and Interpreter, and execute the file. Ideally, you should be able to easily add different target languages (e.g., LolCode) without majorly extending or rewriting your program

Problem

Your task is to determine an appropriate way to create appropriate *Parser*, *TypeChecker* and *Interpreter* objects for whichever language the file is written in. Since some languages are very similar in syntax or semantics, it may not always be ideal to simply create a large class hierarchy of *Parsers*, *TypeCheckers* and *Interpreters*, as some classes may be used for multiple languages (for example, by the time a Lisp program and Haskell program are parsed, they may be able to use the same *Interpreter* class). Copying the same class twice for two different languages would be wasteful! Conversely, languages evolve, so you also don't want to bind one set of tools to a particular language.

From the perspective of the driver program, it's ideal to only have to change one line of code...

An interface for *Parser*, *TypeChecker* and *Interpreter* is provided. **You do not need to build an actual Parser or Interpreter*,*** these will just need to output something like "Interpreting a Javascript program!" to the console. In addition, an *AST* class is provided as a placeholder for an actual abstract syntax tree.

Deliverables

1. Identify the design pattern you used to solve this problem, and the participants (i.e., the roles each class takes).
2. An implementation in a language of your choice.
3. A class diagram of your solution (including existing classes), so future developers can easily see how to work with your solution.