Spidr

{ Matt Meisinger, Akshata Ramesh,
Alex Dong, Kate Haas }

Motivation

- Easy HTML manipulation: for extracting images, links, and other content from specific portions of the page.
- Traversing the hierarchical structure of an HTML page in a not-too-complicated manner.
- For example, might want to retrieve all links that a given url is linked to.
- If using Java, need to add extra Java-specific code, libraries, etc.
- Make something that has familiar Java/C++ type syntax, yet does not contain a lot of pre-processing.

Initial Idea

- Have an object oriented structure for the language: have Url, Page, and others as objects.
- Implement helper functions to make frequent processes easier to re-compute.
- Keep syntax close to Java/C++ which we are all familiar with.
- Sounded too much like Java! So, decided to discard the object-oriented construct and some of the syntax.
- Decouple HTML parsing. Could be handled by something else: like a library in another language.
- Started working with Java's JSoup HTML parsing library.
- Chose JSoup because it seemed intuitive, and uses jQuery-like selectors to pick out parts of the HTML.
- Finally came up with something simpler than our initial idea, yet rigorous.

Tutorial I

```
function void main(){
    string s = "Hello World"
    println(s)
}
```

```
function int foo(int x){
    int a = x
    double b = 2.5
    boolean = true /*or false*/
    if(a) {return 2} else {return 3}
}
```

- 3 different loop structures: typical for/while, and a special 'loop' which is very much like a for-each:

```
int[] list = [1,2,3]
loop(list i)
    print(i) /*prints 123*/
```

- List initializers, unlike Java, may be passed as arguments of functions:

```
print([1,2,3])
```

Tutorial II : The Return

- Special types: urls and selectors.
- `url`

    -> Created from a string using the colon operator:

```
string columbia = "http://www.columbia.edu"
url c_link = :columbia
```

- `selector`

    -> Used to pick out specific parts of html

    -> Syntax:

```
<<element_name@attribute>>
<<element_name.class_name@attribute>>
```

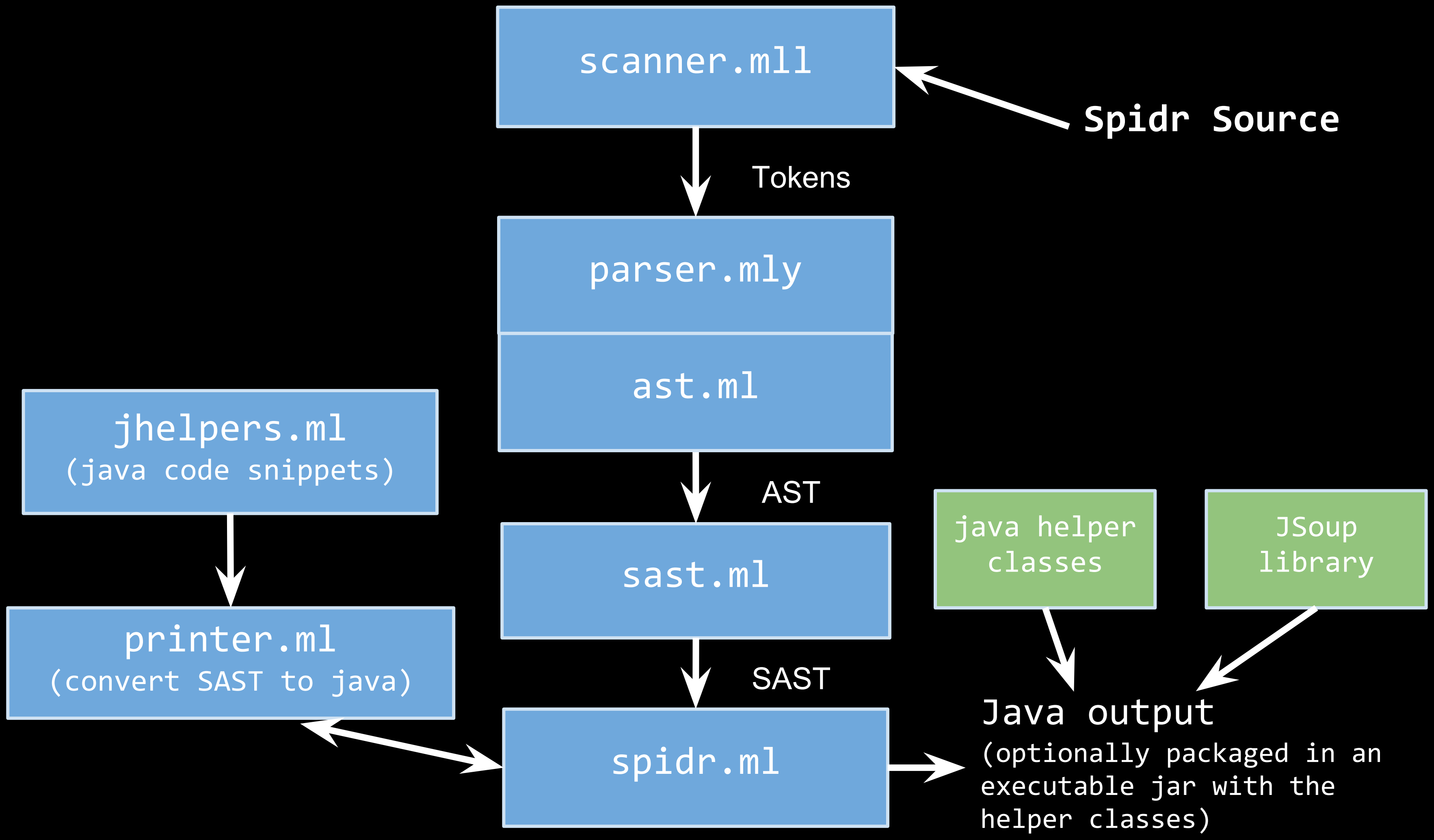
    -> For example:

```
selector s = <<a@href>>
string[] links = c_link * s
```

Example

```
/*The following demo crawls site specified in startUrl, and returns all
active links the page, and all active links on those pages. */
function void main() {
    url startUrl = :"http://www.cs.columbia.edu/~sedwards/software.html"
    url[] children = getChildUrls(startUrl, 2)
    println("Completed!")
}
function url[] getChildUrls(url u, int depth) {
    if (depth == 0)
        return [u]
    else {
        string[] links = u * <<a@href>>
        url[] activeChildren = []
        loop (links l) {
            if (live(:l))
                activeChildren = activeChildren + getChildUrls(:l, depth-1)
        }
        return [u] + activeChildren
    }
}
```

Implementation



Implementation

- Compilation
  - -> Simple .java file using -s option
  - -> Includes standard helpers in the output Java file
  - -> Java file requires references to SUrl, SSelector and SAttSelector classes, and JSoup Java library
  - -> -e option
    - -> compiles the input into .java
    - -> jars it with the helper classes and JSoup library
    - -> runs the resulting executable .jar file
- Tests
  - -> Test java output and actual output when executed
  - -> Over the course of the project built up suite of 74 tests
  - -> Execute test suite with 'make test' or 'make testexe'

Lessons Learned

- Splitting up the work: harder because of the interdependencies of different portions of the compiler. Could potentially lead to a lot of bugs, confusion, and delays.
- Unforeseen ambiguities in syntax and semantics: took up more time than we had planned for.
- A better intuition on such issues before we started could have left us with more time to enhance certain parts of the language.
- Coding in Ocaml: @&*$^#!   ->   Steep learning curve.

  However, the idea of parsing, scanning, and abstract syntax trees made much more sense mostly due to the nature of the functional style of programming in Ocaml.
- Debugging: Learnt the most because bugs can exist at the most obscure levels.

Who Did What

- parser.mly/scanner.mll: Matt, Alex
- ast.ml: Matt, Alex, Akshata
- sast.ml: Matt, Alex, Kate, Akshata
- Makefile: Matt, Alex
- spidr.ml: Matt, Alex
- jhelpers.ml: Matt, Alex
- printer.ml: Matt, Alex, Akshata
- Testing: All members
- Final Report: Kate (in collaboration with Matt, Alex, & Akshata)