

Section 1: Program

Included in the directory.

Section 2: Teamwork

My partner, Jander, provided three rather thorough tests that covered parts of the Catscript grammar that my other tests did not cover. He was also very quick to respond to my suggestions to make updates and was available to hop on a video conference to go over his documentation when requested (I had him add a few more examples).

Further, his documentation included quite a few examples with details about how they work. He did include quite a few details that I did not include which inspired me to also include the sections that he did.

Finally, his tests are included as the file `jander_tests.txt`

Section 3: Design pattern

I used memoization in CatscriptType.java on line 36. I used memoization because we return a new ListType everytime getListType is called. This can make the software run slow and thus provide a subpar user experience.

I used memoization to cache the ListTypes so we are not creating a new time everytime it is called. Depending on the application it is parsing this can save a lot of time and remove one of the biggest bottlenecks (recursive descent parsers are slower than parser generators in general).

Below is the memoization that I included in my code.

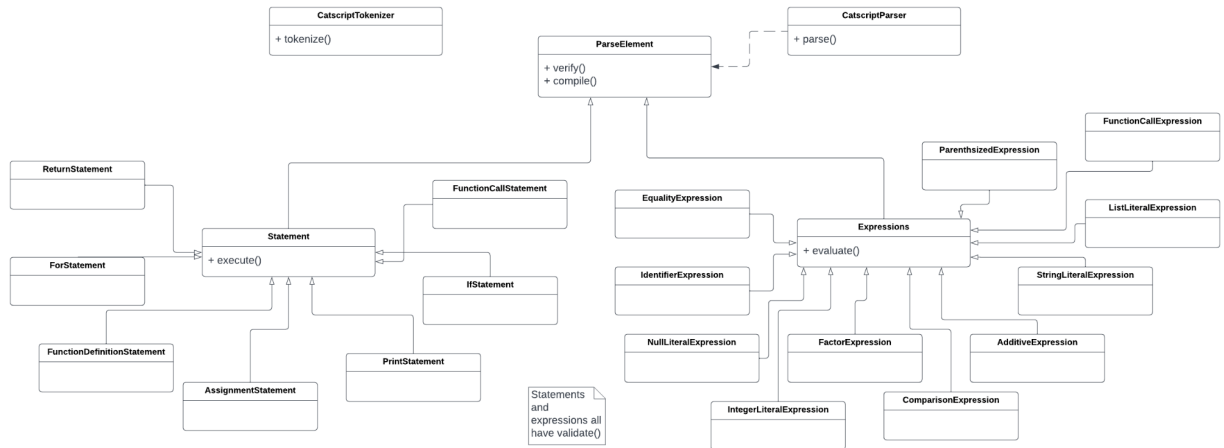
```
static ConcurrentHashMap<CatscriptType, ListType> cache = new
ConcurrentHashMap<>();
public static CatscriptType getListType(CatscriptType type) {
    ListType returnType = cache.computeIfAbsent(type,
catscriptType -> new ListType(type));
    return returnType;
}
```

Section 4: Technical Writing

Included as file technical_writing.md

Section 5: UML

https://lucid.app/lucidchart/8d3919aa-e19f-496d-b94c-e70508931443/edit?invitationId=inv_f640348f-2bf7-4040-8d4a-ed7b7143e4a7



Section 6: Design trade-offs

We used a recursive descent parser compared to a parser generator. Although this decision was made for me by Carson, I am glad he made this decision. Having taken this class previously with a different parser design that was much more difficult to learn, the recursive descent parser we made this semester not only helped in my understanding of parsers and got me excited to develop.

Some of the benefits of recursive descent are that the code is typically easier to understand and debug. The error messages are easier to make descriptive and to add to the parser.

Section 7: Software development life cycle model

We used Test Driven Development and it has been my favorite way to work on a project and I am slightly disappointed I had to wait till my senior year to experience it in a class.

Not only was I able to debug my program starting with the test to see exactly what the test was expecting, but it made it clear what each function was supposed to do. It also made it very clear what my grade would be when I turned in the project because I could run the tests myself to see if I was happy with the grade (the amount of tests passing).