

Design Pattern

One key design pattern that we used was Memoization, or the flywheel pattern. This pattern can be used to speed up programs by storing the results of a repeated function call in a cache. This prevents duplicate, expensive function calls and can thus speed up execution. Essentially, the program can ‘memorize’ the results of certain computations allowing them to be computed again in constant time (via lookup) at the cost of some space. Note that memorize \neq memoize, but the difference is largely semantic where memoize is the specific computing related term.

Below is an example of us using memoization in the CatscriptType.java file.

```
1 // Use Memoization
2 private static Map<CatscriptType, ListType> cache = new HashMap<>();
3 public static CatscriptType getListType(CatscriptType type) {
4     ListType l = cache.get(type);
5     if (l != null)
6         return l; // if its already in the cache, return it
7
8     l = new ListType(type);
9     cache.put(type, l); // save it in the cache for next time
10    return l;
11 }
```

Essentially the goal is to reduce the number of the small list type objects that are created. The cache (line 2) creates a map object which is where we save the list types. Then we check if we already have an object saved in the cache (line 6) and if so we return it. Otherwise (line 8) we create a new object and save it in the cache before returning.