Advent of Code in Java

vJHUG December 2020

Anastasopoulos Spyros

@anastasop

https://adventofcode.com/

- One programming puzzle every day from 1/12 to 25/12
    - No time constraints. You can solve it whenever you like, even after publication date.
    - For maximum fun, it is recommended to try and solve it before the next one is published.
- Only basic programming skills are needed for most of the puzzles
    - Language does not make a difference. All are good: c++, python, java, go, js, ruby, scala
    - For 2 or 3 puzzles you may need to consult your CS notes
- Each puzzle has 2 parts, A and B. First you solve A and then you get B.
    - A is simpler. B adds a new question and/or challenges your solution of A.
    - Each successful solution gives you one ⭐ The goal is to collect all 50 of them
    - All players get the same puzzle but different input
- Just for fun and for the joy of programming. No prizes or red carpets.
    - An excellent opportunity to experiment with a new language
    - People discuss and share solutions on github, twitter, reddit

Anatomy of a puzzle

- The descriptions are very sleek, like fairy tales with elves, santa, airplanes, computers, monsters, ghosts, goblins, clouds etc
- Eventually they are reduced to plain english: find the number of input records that satisfy the following properties and compute the sum of the third column.
- Here be dragons
  - A general program that finds a solution for any input may be very difficult
  - Stay on your input. There is a solution in it, find the records that justify it
- Simple algorithms on modest hardware solve all puzzles in seconds
- Brute force approaches however do not work even on sophisticated hardware
- Behind the scenes https://www.youtube.com/watch?v=CFWuwNDOnIo

Me and Advent of Code

2018 12 ⭐ C++ to experiment with latest features

2019 23 ⭐ Go to focus on the puzzles

2020 47 ⭐ Java to dust off my skills https://github.com/anastasop/aoc2020

2021 ?? ⭐ any recommendations?

Java has full buzzword support

- memory safety
- garbage collection
- concurrency
- static typing
- good readability/writability ratio
- object oriented
- rich ecosystem
- distributed
- functional programming support
- good performance

Yes, but specifically for Advent of Code?

- ~~memory safety~~
- ~~garbage collection~~
- ~~concurrency~~
- ~~static typing~~
- good readability/writability ratio
- ~~object oriented~~
- ~~rich ecosystem~~
- ~~distributed~~
- functional programming support
- ~~good performance~~

Idiomatic Java: create and compose abstractions

- Abstraction layers add costs
  - More things to keep on head
  - Additional code for information hiding
  - Bootstrapping the code
  - But <u>no performance costs</u>
- Breaking the abstractions can result in smaller or faster programs
  - Not uncommon to see C++ or python programs that solve a puzzle only with `main()`
- Where is the red line?
  - Should i wrap a `String` to make an `Instruction`?
  - Should i wrap a `List<List<Integer>>` to make a `Rule`?
  - Should i add a method `getCustomerCount()` that just delegates to `List.size()`
- For production code apply established engineering guidelines
  - For Advent of Code experiment and play with the rules

Expectations

```
Policy.matches(String)
Passport.isValid()
Group.addAnswers(String)
Rule.add(Bag)
Machine.run(List<Instructions>)
Floor.occupiedCount()
Ship.move(Waypoint)
```

```
Mask.apply(Long)
Number.seenBefore()
Cube.setActive()
Cursor.advance()
Deck.score()
Cup.find()
Waypoint.rotate(LEFT)
```

So, how did it go with Java?

- Boxing, unboxing of primitive types is annoying
  - `lines().mapToInt() -> Stream<Integer>`
  - No `Arrays.stream(boolean[] array)`
- Integer arithmetic
  - No unsigned types.
  - `var i = 5` is an integer. Use `5L` if you need a long.
  - Bit fiddling
    - Correct test `if (i & pow(2) > 0)`
    - `1 << N vs 1L << N if N >32`
    - `N >>> 1 vs N >> 1`
    - Negative numbers have high bit set to 1 (2s complement)

So, how did it go with Java?

- Data structures are cheap and handy
  - Use new, filter, copy instead of mutating. Especially with recursive methods.
    - Nobody expects `java.util.ConcurrentModificationException`
  - Special attention when you implement `hashCode`
  - Don’t forget `equals` if you implement `hashCode` and vice versa.
- Some times `arrays + pos` are preferable. Collections lack the concept of a pointer to position in the data structure. (C++ iterators or Go slices)
- First time i used `Outer.new Inner(a, b)` to create an inner class
- Interfaces work
  - Code reads well and provides insights on problem and solution
  - to solve part B the API remains the same and the implementation changes

Streams and Lambdas

```
var board = new BufferedReader(new InputStreamReader(System.in)).lines()
    .map(line -> line.strip().toCharArray()).toArray(char[][]::new);


var input = new BufferedReader(new InputStreamReader(System.in)).lines()
    .filter(line -> !line.isEmpty()).count();


var max = new BufferedReader(new InputStreamReader(System.in)).lines()
    .mapToInt(A::position).max().getAsInt();

final var counter = new AtomicLong();
new BufferedReader(new InputStreamReader(System.in)).lines()
    .forEach(l -> counter.addAndGet(1))
```

A selection of my favorite puzzles

Day 4

Given a collection of objects `{p1: v1, p2: v2, ...}` and a list of constraints for the values, count the valid objects.

```
ecl:brn pid:760753108 byr:1931 hcl:#ae17e1
```

```
byr (Birth Year) - four digits; at least 1920 and at most 2002.
```

```
hcl (Hair Color) - a # followed by exactly six characters 0-9 or a-f.
```

The infamous business rules validation problem.

Day 4

There are two approaches:

1. `isValid()` method: `foo = new Foo(); foo.setA(); foo.isValid();`
2. Builder pattern: `b = new Builder(); b.setA(); b.build(); // may throw`

I am a fan of builders. They simplify the code and favor immutability.

Day 4

Lambdas make the code cleaner. A better alternative than anonymous inner classes or classes named just for the case.

```
class Criterion {
    final Pattern pattern;
    final Predicate<String> pred;
    public boolean matches(String s) { return pattern.matcher(s).matches() && pred.test(s); }
}


Criterion[] criteria = new Criterion[]{
    new Criterion("^#[a-z0-9]{6}$", s -> { return true; }),
    new Criterion("^\\d{4}$", s -> { var year = Integer.valueOf(s); return 2020 <= year && year <= 2030; }),
    new Criterion("^\\d{9}$", s -> { return true; })
}

Arrays.stream(criteria).filter(crit -> crit.matches(s)).count();
```

Day 18

Evaluate arithmetic expressions with `integers ( ) + *` but + has higher precedence than *

```
1 * 2 + 3 * 4 = 20
```

```
2 * 3 + (4 * 5) = 46
```

```
5 + (8 * 3 + 9 + 3 * 4 * 3) = 1445
```

```
5 * 9 * (7 * 3 * 3 + 9 * 3 + (8 + 6 * 4)) = 669060
```

A typical exercise in a compilers course. There are many solutions.

Day 18

This is usually done with a yacc like tool: ANTLR or JavaCC

For old times' sake i wrote a recursive descent parser.

Day 18

Start with the grammar, left factor it and then one procedure for each non-terminal.

With some more work it can be written as a predictive parser (no recursion)

```
expr -> expr * term | term
term -> term + factor | factor
factor -> ( expr ) | NUMBER
```

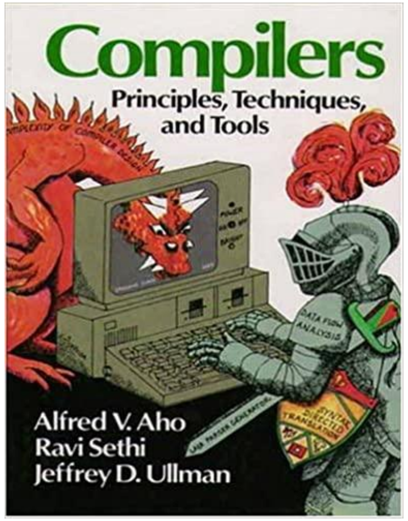
```
expr -> term expr1

expr1 -> * term expr1 | ε

term -> factor term1

term1 -> + factor term1 | ε

factor -> ( expr ) | NUMBER
```



Day 18

```
public static Optional<Long> evalTerm1(Cursor cur) {
       if (cur.follows("+")) {
           cur.take("+");
           var a = evalTerm(cur);
           var b = evalTerm1(cur);
           return b.isPresent() ? b.map(c -> c + a) : Optional.of(a);
       }
       return Optional.empty();
}
public static Optional<Long> evalExpr1(Cursor cur) {
       if (cur.follows("*")) {
           cur.take("*");
           var a = evalTerm(cur);
           var b = evalExpr1(cur);
           return b.isPresent() ? b.map(c -> c * a) : Optional.of(a);
       }
       return Optional.empty();
}
```

Day 18

Dynamic languages provide `eval` and operator overloading. This makes it easy.

```
class N
 attr_accessor :val
 def initialize(v); @val = v; end
 def -(other); self.class.new(val * other.val); end
 def *(other); self.class.new(val + other.val); end
end

def parse(expr)
 eval(expr.gsub(/\*/, "-").gsub(/\+/, "*").gsub(/(\d+)/, 'N.new(\1)')).val
end
```

Day 19 - my favorite

Find the input strings that match a set of rules

```
42: 9 14 | 10 1
9: 14 27 | 1 26
10: 23 14 | 28 1
1: "a"
11: 42 31
14: "b"

bbabbbbaabaabba
babbbbaabbbbbabbbbbbaabaaabaaa
aaabbbbbbaaaabaababaabababbabaaabbababababaaa
```

Similar to Day 18, a recursive descent parser will do (famous last words)

Day 19

Not that simple. Converting the grammar to yacc:

```
4 rules never reduced

conflicts: 108 shift/reduce, 29 reduce/reduce
```

Which means we cannot write a general parser for the grammar, just one that handles the input. Special care should be taken to avoid infinite loops.

Good hints are in the implementation of regular expression matchers

1. The practice of programming book had an implementation using recursion
2. A series of excellent blog posts on the topic.

Day 19

```
Set<Integer> advanceRule(final String s, final Set<Integer> positions, final int num)

Set<Integer> advanceAltRule(final String s, final Set<Integer> positions, final List<Integer> rule)

Set<Integer> advanceCharRule(final String s, final Set<Integer> positions, int c)
```

- Create and copy immutable data structures. Clean code, helps debugging
- Need to deal with sets because the grammar is ambiguous. For this problem we don’t care which rules matched, just that some rules matched.
- Need a sentinel to figure out then we are done. The empty set will not do, we cannot tell if there was a match or not. I decided to keep positions in the sets and we are done if the set includes the end of the string

Day 19

Notable solutions that worth study

- An implementation in python of the recursive parser. Very clean
    - https://github.com/fogleman/AdventOfCode2020/blob/main/19.py
- An implementation in Rust using a virtual machine. Instead of maintaining sets, each matcher is a thread of instructions and forks on every alternative rule. Finishes successfully if it finds a match.
    - https://github.com/felixge/advent-2020/blob/main/day19-2/src/main.rs
- An implementation in Clojure that build regular expressions on the fly and uses them to match the strings. The OTF translator handles recursion
    - https://github.com/leahneukirchen/adventofcode2020/blob/master/day19.clj

Day 23

A game. You start with a circular list of integers 653427918 (`next(8) = 6 && prev(6) = 8`) and a current position `pos`. Move the three numbers next to the current position to a new position `f(pos)`, move the current position to next element and repeat 1000000 times. What is the final list?

Clearly linked lists should be preferred. Solutions with arrays will have much overhead because there are operations in the middle of the list.

This looks like of job for dancing links "... a technique for reverting the operation of deleting a node from a circular doubly linked list."

Day 23

Java does not provide circular lists. You can either use `LinkedList<T>` and take care of `prev`, `next` at the edges or implement a new linked data structure.

```
static class Cup {
    Cup prev;
    Cup next;
    int value;
}

// code may look like this
curr.next.prev = dest;
curr.next.next.next.next = dest.next;
dest.next.prev = curr.next.next.next;
dest.next = curr.next;
```

Collection frameworks provide all lot but not everything. Sometimes we may have to implement custom data structures. Like bicycle, you never forget it.

Was it fun?

Thank you