Project Report: Available Time Finder

Baangkok Vanijyananda 630610746

What I did

Introduction

In short, I wrote a Scala program: "Available Time Finder" to read the data of free time & day of week of each person and compute what **time** in each **day of week** will **everyone** be **free together**.

In detail:

1. I studied Scala
2. I created a google spreadsheet (with instructions) for my friends to fill in the data
3. The data is exported into CSV format (see ***Data*** section)
4. I run the Scala program (see ***Program*** section) on that data
5. Viola~ here is the time in each day of week that everyone is free together!

Data

The table have the following columns

Active	Name	Day of week	Free time start	Free time end
1 or 0	alphaNumeric	M, T, W, Th, F, S, Su	0-2400	0-2400

Definition for each column

Column	Definition
Active	States whether this row is active or not. So you don't need to delete/rewrite entire row.
Name	Name of the person
Day of week	The day at which they have some available time
Free time start	The start of their free time on that day
Free time end	The end of their free time on that day

Notes

- Days at which there are no data for that person will count as that person being NOT available the entire day
- Example use case for `Active` : I would likely to be available the same time for every Monday, but this Monday I might be busy, but if I am not busy, I will be available at this time)
- Multiple time ranges in the same day can be filled in on different rows
    - in each day of week, only time range(s) where everyone is free will be the output

Program

The program structures are as follows

- FileReader (object)
- SimpleCSVParser (object)
- Main (object)
    - TimeData (class)
    - TimeEvent (class)
    - TimeRange (class)
    - TimeRanges (class)

FileReader

A simple object with one method `readFile(filePath: String)` . `readFile` reads a file from the `filePath` and turns it into a long String.

SimpleCSVParser

A simple object with one method `parse( input: String, pattern: Regex, omitFirstLine: Boolean ): List[List[String]]` . `parse` takes in a long String as `input` , an optional regex `pattern` for parsing, and an optional `omitFirstLine` to omit first line of CSV and outputs a `List[List[String]]` which the inner `List[String]` contains the parsed columns in a row.

TimeData

A class which encapsulates the immediate output from the SimpleCSVParser with a more readable interface. `TimeData(name, dow, timeStart, timeEnd)` .

TimeRange

A class which describe a range [start, end] `TimeRange(val start: Int = 0, val end: Int = 2400)` .

- `intersect(other: TimeRange): TimeRange` takes another TimeRange and outputs a resulting intersection of that and itself.
- `def in(other: TimeRange): Boolean` takes another TimeRange and outputs whether itself is between that or not.
- `def isValidTime: Boolean` returns whether start, end is in range [0,2400] and start <= end.

TimeRanges (with an s)

A class which describe a List[TimeRange] `TimeRanges(val timeRanges: List[TimeRange])` .

- `def get: TimeRanges` outputs the List[TimeRange] in a sorted by start, end ascending order.
- `def add(tr: TimeRange): TimeRanges` takes a TimeRange and add it to the list.

TimeEvent

A class which describe a time at which someone starts to be free or stop to be free `TimeEvent(val time: Int, val event: Int)` .

Mainly used in the algorithm for multiple time ranges in a single day.

Main

An object which is an entry point of the program.

- `main()` the entry point of the program.
- `findAvailableTimeAndPeopleV2( timeData: List[TimeData] ): List[DowTimePeopleV2]` main method for calculating our main output, supports multiple time ranges in a single day
- `findTimeRanges( tes: List[TimeEvent], max: Int, count: Int = 0, tr: TimeRange, trs: TimeRanges) ): TimeRanges` a method for calculating free time ranges for each day.
- `findAvailablePeopleV2( trs: TimeRanges, timeData: List[TimeData] ): List[String]` a method for calculating who is free in the given time ranges for each day.
- `printDowTimePeopleList` self explanatory.
- more output manipulation functions for printing
- more deprecated functions

Algorithm for `findAvailableTimeAndPeopleV2` , `findTimeRanges`

I call it "Cumulative Sum Sweep" or "Cum Sum Sweep".

```
// for each day: <--- here is findAvailableTimeAndPeopleV2
//    turn TimeData into TimeEvents
//    sort TimeEvents by time
//    iterate TimeEvents to find TimeRanges: <--- here is findTimeRanges
//      if available = max number of people
//        record start
//      if avail no longer at max
//        record end
//        create a TimeRange, add it to TimeRanges
//        reset TimeRange
```

What I learned

- In Scala

    - Objects vs Classes
    - type
    - val vs var (immutable vs mutable)
    - function definitions
    - reading a file
    - parsing
        - Regex (same as Java Regex)
    - map, filter, reduce
    - tail recursion
    - Option (maybe in haskell)
    - Either
- Usage of functional programming mindset in a real project with real data