Lumkani Technical Question

The task is to write a system/program that computes a *Days from Suspension Report* and some other statistics related to payment types and agents.

The problem

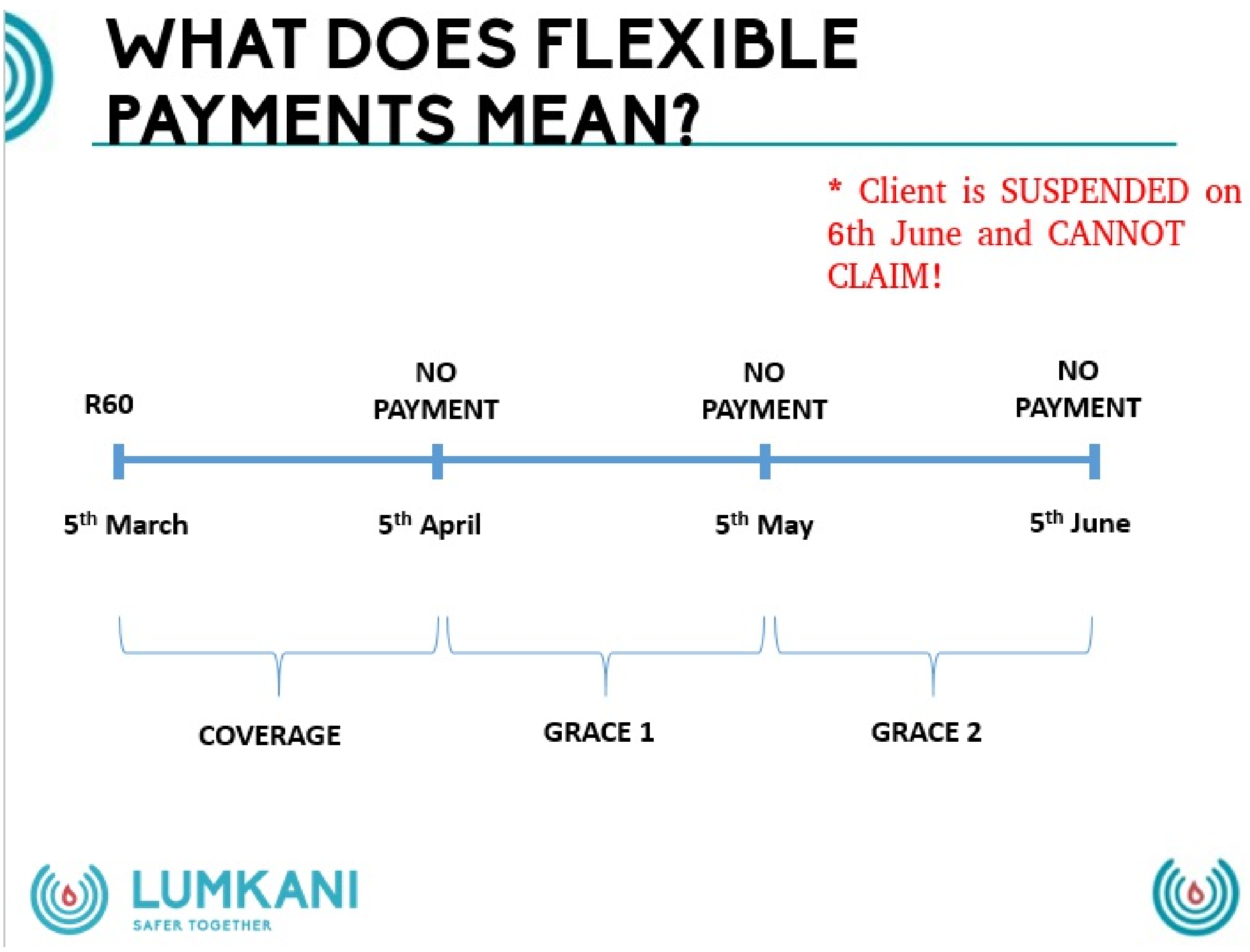
Payments.csv shows a list of payments based on device_id. We would like to use the payment ‘history’ to calculate the time that a client’s policy is valid for. This can be looked at as the total number of days until the client is suspended (Days from Suspension).

Once a payment has been made a client is covered for 30 days (this is the time during which any claims will be paid out). We know that clients will not always have the money on their exact payment date, and income is volatile, so after the 30 days of coverage period, the client enters into the grace period for 30 days. If a client misses their monthly payment they will go into the grace period. They can then make a payment to catch up. Furthermore, after the end of the grace period a client will still have 30 days of their suspension period. A client will only be suspended after 91 days of their first payment. Another way of looking at it, if a client is up to date on payments, a client is 90 days from suspension.

We have numerous products with different premiums, but the same logic applies to all of them.

We want to give our agents as much information possible to manage their clients. We want to be able to give agents a “filtered” list that shows their most pressing clients first. Furthermore, from the operational team perspective, we want two more reports:

1. Collection per agent, per day, and per payment type
2. Total collection per payment type



Main Concepts and Terminology

Valid Policy: It depends on the number of days until the client is suspended (Days from Suspension)

Payment: Once a payment has been made a client is covered for 30 days.

Coverage Period: Period which any claims will be paid out.

Grace 1 Period: We know that our clients will not always have the money on their exact payment date, and income is volatile, so after the 30 days of coverage period, the client enters into the grace period for 30 days.

Catch up payment: If a client misses their monthly payment they will go into the grace period. They can then make a payment to catch up.

Grace 2 Period: After the end of the grace 1 period a client will still have 30 days of their suspension period called Grace 2 Period.

Suspended Client: A client will only be suspended after 91 days of their first payment. Another way of looking at it, if a client is up to date on payments, a client is 90 days from suspension.

Agents: Lumkani agents are people from the community that work closely with our clients in field. They collect payments, install devices, replace devices, and other actions.

How to run the program

The program must accept two parameters: `<input_file> <output_folder>`. The first parameter is a csv file with the name YYYY_MM_DD_payments.csv, and the second is a folder which must not yet exist.

YYYY_MM_DD_payments.csv file content

This CSV file contains the list of all payments logs created in our system until the date YYYY_MM_DD in the file name. File format is:

```
id,payment_type,payment_amount,payment_signature_image,payment_photo,created,st
atus,agent_user_id,device_id
```

Where:

- **id**: Payment transaction id
- **payment_type**: Payment type used by the client or operational team i.e. CASH, CLIENT_REFERRAL, etc.
- **payment_amount**: Payment amount.
- **payment_signature_image**: Client's signature image url.
- **payment_photo**: Proof of Payment image url.
- **created**: Payment creation time
- **status**: Payment status.
- **notes**: Payment notes, added by operational team.
- **agent_user_id**: Lumkani agent identifier, agent responsible for the payment in question
- **device_id**: Client/Device identifier, payer.

For example:

```
7529,CLIENT_REFERRAL,150,,,2018-12-08 18:01:42,SUCCESSFUL,Created on watchtower
by Ines Niragira,47,17945
7528,CASH,60,payments/signatures/7528_signature.jpg,payments/proof/7528_pop.jpg,
2018-12-08 17:38:49,SUCCESSFUL,,13,6891
```

Output folder content

This folder must contain the result files of the execution of the program:

- `days_from_suspension_report.csv`, containing the list of clients and their days from suspension
- `agent_collection_report.csv`, containing collections per agent, per day, and per payment type
- `payment_type_report.csv`, containing total collections per payment type

days_from_suspension_report.csv file

A CSV file with that contains the list of clients and their days from suspension
using the following format:

```
device_id,days_from_suspension
```

Where:

- **device_id**: Client/device identifier.
- **days_from_suspension**: Calculated day from suspension.

For example:

```
18773,123
2345,85
321,30
```

The file should be ordered by `days_from_suspension` in desc order.
The days until suspension should be relative to today's date.
e.g. If a client paid only once 91 days ago, their days from suspension on the report generation date is 0.

agent_collection_report.csv

A CSV file containing the collections per agent, per day, and per payment type

The format for this file is:

```
agent_user_id,date,payment_type,total_amount
```

Where:

- **agent_user_id**: Lumkani agent identifier, agent responsible for the payment in question.
- **date**: Payment date
- **payment_type**: Payment type used by the client or operational team i.e. CASH, CLIENT_REFERRAL, etc.
- **total_amount**: Total amount paid on the day with the payment type in question.

File must be ordered by `agent_user_id` and `date`

For example `agent_collection_report.csv`:

```
1,2018-12-10,CASH,1800
1,2018-12-11,CLIENT_REFERRAL,60
2,2018-12-02,CARD,150
2,2018-12-11,BANK_DEPOSIT,360
```

payment_type_report.csv

A CSV file containing total collections per payment type

The format for this file is:

```
payment_type,total_amount
```

Where:

- **payment_type**: Payment type used by the client or operational team i.e. CASH, CLIENTE_REFERRAL, etc.
- **total_amount**: Total amount paid with the payment type in question.

File must be ordered by `payment_type` in lexicographic order.

For example:

```
CARD,43200
CASH,20000
EASY_PAY,84000
```

Program requirements

Input data limits

There are 100 rows in the sample input file.
You can assume that in the private input test file, there are no more than 100 000 payments.

Challenge submissions requirements

- Coding challenge should be submitted as *ZIP* folder
- There should be full source code for the program. It's better to remove all compiled binary artifacts if applicable.
- Solution should be tested and unit-tests should be included.
- Submission should have `README.md` file describing:
    - How to compile and run the program
    - How to run tests
    - Some description of the algorithms and decisions taken during implementation

Considerations

You can solve this problem in any language. Although Python is preferred, the language you chose will not count against you.

Other important things to consider:

- Think carefully about the Domain Model of this problem
- Project structure and organization matters
- Readability counts
- Consider scalability, and take note of the input data limits

Concepts that we value

- SOLID principles
- Composition over inheritance
- Clean Code and Clean Architecture
- Python Data model (https://docs.python.org/3/reference/datamodel.html)
- Zen of Python (https://www.python.org/dev/peps/pep-0020/)
- Static code analyses and linting (pylint, flake8, black, mypy)
- Conventions and PEPs
- Tests as documentation