Credit Card Reward Points System Report

§ Introduction

This is an `Android/Kotlin` project that provides an android application to maximize the reward points based on customer's monthly credit card transactions.

Key points for this project: `Kotlin`, `Android Application Development`, `MVVM architecture`, `interactive user-interface`, `dynamic programming`

§ Assumptions

In this project, we assume that the user input (i.e. customer's monthly credit card transaction record) will be in format of Json text.

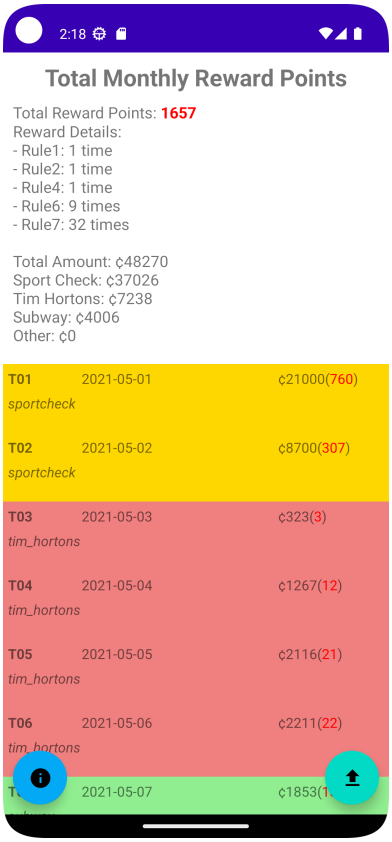
Example:

```
{
  "T01": {"date": "2021-05-01", "merchant_code" : "sportcheck", "amount_cents": 21000},
  "T02": {"date": "2021-05-02", "merchant_code" : "sportcheck", "amount_cents": 8700},
  "T03": {"date": "2021-05-03", "merchant_code" : "tim_hortons", "amount_cents": 323},
  "T04": {"date": "2021-05-04", "merchant_code" : "tim_hortons", "amount_cents": 1267},
  "T05": {"date": "2021-05-05", "merchant_code" : "tim_hortons", "amount_cents": 2116},
  "T06": {"date": "2021-05-06", "merchant_code" : "tim_hortons", "amount_cents": 2211},
  "T07": {"date": "2021-05-07", "merchant_code" : "subway", "amount_cents": 1853},
  "T08": {"date": "2021-05-08", "merchant_code" : "subway", "amount_cents": 2153},
  "T09": {"date": "2021-05-09", "merchant_code" : "sportcheck", "amount_cents": 7326},
  "T10": {"date": "2021-05-10", "merchant_code" : "tim_hortons", "amount_cents": 1321}
}
```

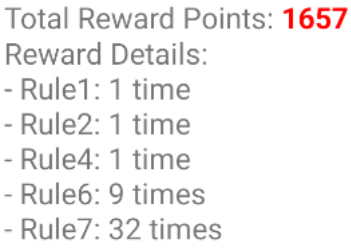
To emphasis the functionalities of this project, we further assume that the user input will only consist of one month's data and will always be **valid** (i.e. correct format and legitimate data).

§ Demo

When open up the project, it usually takes some time to download some android dependencies and plugins. After auto-configuration process, we run the application by clicking on `run`. An emulator of this project shows up as follows:

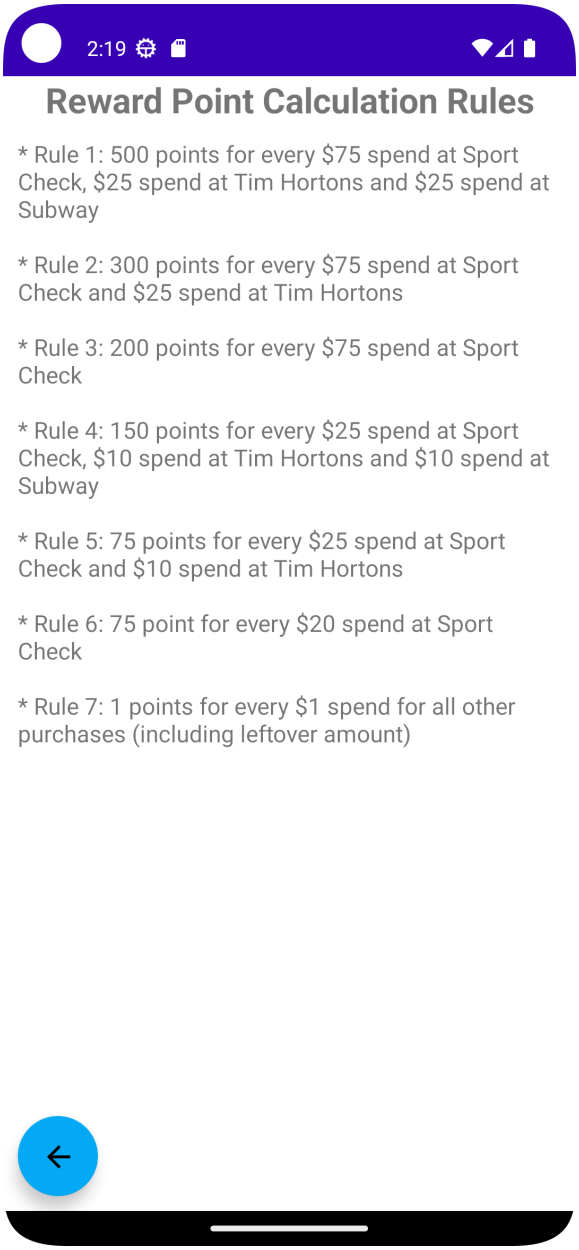


By default, it opens up in the home page and uses the `sample transactions`. Monthly total reward points will be calculated at the top of the phone labelled in **red**. Other relevant calculations are also provided as references. Specifically, we provide data about the rules that are applied to the transaction and the number of times one rule has been used. (Note that we don't display unused rules).



The maximum reward points applied for each transaction is indicated in each transaction entry in **red** besides the amount of money in cents for the transaction.

On the buttom-left conner of the home page, we have an information button that could direct to a description page about all the rules the system support.



At the same location on page `Reward Point Calculation Rules`, user can navigate back to the home page.

On the buttom-right conner of the home page, we have an uploading button that could direct to a json file uploading page, user can paste in their json data in the provided space.

At the same location on user input page, user can navigate back to the home page. Note that user does not need to submit and confirm the input, navigating back to the home page indicates updating the transaction information.

§ Features

- Correctly calculated max total monthly reward points earned based on a customer’s credit card purchase
- Correctly calculated max reward points earned for each transaction
- Implemented a user-friendly interactive interface
- Provided more calculation details on the transactions
- Provided a visual display of scrollable transaction list
- Provided interface to upload different transaction information in Json formatted text

§ Analysis

Problem Analysis

We are given a list of one customer's transactions for a particular month and a list rules applied for reward points. We want to calculate and maximize reward points based on the transactions.

Observe the rules applied to the transactions:

- Rule 1: 500 points for every $75 spend at Sport Check, $25 spend at Tim Hortons and $25 spend at Subway
- Rule 2: 300 points for every $75 spend at Sport Check and $25 spend at Tim Hortons
- Rule 3: 200 points for every $75 spend at Sport Check
- Rule 4: 150 points for every $25 spend at Sport Check, $10 spend at Tim Hortons and $10 spend at Subway
- Rule 5: 75 points for every $25 spend at Sport Check and $10 spend at Tim Hortons
- Rule 6: 75 point for every $20 spend at Sport Check
- Rule 7: 1 points for every $1 spend for all other purchases (including leftover amount)

Note that if we have enough money for Rule 3, say $75 for 200 points, then we can choose to apply 3 times Rule 6 alternatively with $15 dollars remaining that can be applied by Rule 7, which accumulates to $3 \times 75 + 15 = 240$ points. Hence Rule 3 gains less value than Rule 6, and we should always choose Rule 6 over Rule 3.

In addition, note that if we have enough money for Rule 5, say $25 on Sport Check and $10 on Tim Hortons for 75 points, then we can choose to apply Rule 6 alternatively with $15 dollars remaining that can be applied by Rule 7, which accumulates to $75 + 15 = 90$ points. Hence Rule 5 gains less value than Rule 6, and we should always choose Rule 6 over Rule 3.

Now it remains to us with 5 rules:

- Rule 1: 500 points for every $75 spend at Sport Check, $25 spend at Tim Hortons and $25 spend at Subway
- Rule 2: 300 points for every $75 spend at Sport Check and $25 spend at Tim Hortons
- Rule 4: 150 points for every $25 spend at Sport Check, $10 spend at Tim Hortons and $10 spend at Subway
- Rule 6: 75 point for every $20 spend at Sport Check
- Rule 7: 1 points for every $1 spend for all other purchases (including leftover amount)

Algorithm Analysis

The idea is that we use dynamic programming to find max reward point using Rule 1, Rule 2, Rule 4, Rule 6 as they use marked merchant code of `Sport Check`, `Tim Hortons` and `Subway`. And we apply Rule 7 on any remaining dollers.

For all rules (except for Rule 7) applied, we normalize the data into 5 dollers per unit to save space and improve time complexity, that is

- Rule 1: 500 points for every $15 spend at Sport Check, $5 spend at Tim Hortons and $5 spend at Subway
- Rule 2: 300 points for every $15 spend at Sport Check and $5 spend at Tim Hortons
- Rule 4: 150 points for every $5 spend at Sport Check, $2 spend at Tim Hortons and $2 spend at Subway
- Rule 6: 75 point for every $4 spend at Sport Check

Define `opt[sc][th][su]` to be a list of `[# of rule1 applied, # of rule2 applied, # of rule4 applied, # of rule6 applied, max reward points]` when `sc` amount of 5 dollars spent at Sport Check, `th` amount of 5 dollars spent at Tim Hortons and `su` amount of 5 dollars spent at Subway. Specifically, `opt[sc][th][su][4]` stores the max reward points at the current state.

For each `rule` from Rule 1, Rule 2, Rule 4, Rule 6 with `sc`, `th`, `su`, if the `rule` is applicable, then

$$opt[sc][th][su][4] = \max\{opt[sc][th][su][4], opt[sc - rule.sc][th - rule.th][su - rule.su][4] + rule.rewardPoint\}$$

where `rule.sc`, `rule.th`, `rule.su` and `rule.rewardPoint` is the required spent at Sport Check, the required spent at Tim Hortons, the required spent at Subway and the reward point for the `rule` respectively.

If we update `opt[sc][th][su][4]`, then we also increment the number of current `rule` being applied by one.

If we encounter a reward point that is larger than any reward point we found, we update `maxPoint` and record the indices to find such max reward point.

We finish the dynamic programming and calculate unused money by subtracting used money from total amount money. Then we apply Rule 7 on the dollar part.

A `Kotlin` version of code is also provided as follows, you can also find it under `credit_card_reward_points_system\app\src\main\java\com\example\credit_card_reward_points_system\MainActivityViewModel.kt`.

```
private fun calculateRewardPoints() {
    totalPoint = 0
    ruleAppliedNums = mutableListOf<Int>(0, 0, 0, 0, 0)
    rule7AppliedNum = 0

    val sportCheckNormDollar = sportCheckAmount / 500
    val timHortonsNormDollar = timHortonsAmount / 500
    val subwayNormDollar = subwayAmount / 500

    /*
    * opt[sc][th][su] defines a list of [# of rule1 applied, # of rule2 applied, # of rule4 applied, # of rule6 applied, max reward
points]
    * Note that we only have 4 rules, opt[sc][th][su][4] used to store max reward points
    * */
    val opt = List(sportCheckNormDollar + 1) {
        List(timHortonsNormDollar + 1) {
            List(subwayNormDollar + 1) {
                MutableList(rewardRuleList.size + 1) { 0 }
            }
        }
    }

    // we track the max reward points with optimal argument
    var maxPoints = 0
    var sportCheckMaxPoints = 0
    var timHortonsMaxPoints = 0
    var subwayMaxPoints = 0

    for ((ruleIndex, rule) in rewardRuleList.withIndex()) {
        for (sc in 0..sportCheckNormDollar) {
            for (th in 0..timHortonsNormDollar) {
                for (su in 0..subwayNormDollar) {
                    // Only try to apply the rule when the requirement satisfies and we achieve higher reward points
                    if (sc >= rule.sportCheckNorm &&
                        th >= rule.timHortonsNorm &&
                        su >= rule.subwayNorm &&
                        opt[sc][th][su][4] < opt[sc - rule.sportCheckNorm][th - rule.timHortonsNorm][su - rule.subwayNorm][4] + rul
e.rewardPoint
                    ) {
                        opt[sc][th][su][4] =
                            opt[sc - rule.sportCheckNorm][th - rule.timHortonsNorm][su - rule.subwayNorm][4] + rule.rewardPoint
                        opt[sc][th][su][ruleIndex] =
                            opt[sc - rule.sportCheckNorm][th - rule.timHortonsNorm][su - rule.subwayNorm][ruleIndex] + 1
                    }

                    if (opt[sc][th][su][4] > maxPoints) {
                        maxPoints = opt[sc][th][su][4]
                        sportCheckMaxPoints = sc
                        timHortonsMaxPoints = th
                        subwayMaxPoints = su
                    }
                }
            }
        }
    }


    val optimalSol = opt[sportCheckMaxPoints][timHortonsMaxPoints][subwayMaxPoints]
    ruleAppliedNums = mutableListOf(optimalSol[0], optimalSol[1], optimalSol[2], optimalSol[3])
    val usedAmount = mutableListOf(sportCheckMaxPoints, timHortonsMaxPoints, subwayMaxPoints)

    // All remaining dollars will be applied by rule 7
    rule7AppliedNum = (totalAmount - usedAmount.sum() * 500) / 100

    totalPoint = optimalSol[4] + rule7AppliedNum

}
```

The idea of calculating maximum reaward point for each transaction is simply going through all rules. However, observe that one transaction cannot be used for more than one merchant, hence only Rule 6 and Rule 7 are applicable. In this case, we try to apply as many time as Rule 6, and use the leftover money for Rule 7. If the merchant is not Sport Check, then we can only apply Rule 7.

§ Note

The input of this project is purposefully chosen to be a Json formatted as it is indicated in the question description file, this way, if this project is a back-end service, then our system can easily handle API request and process the queries data from API easily. This can be done by simply changing the retriving text from user input field to an API service requet.