Finager

(Version 1.0)

Apr. 11th, 2017

Group 07
Suri Jia, Mingnan Su, Jiuwei Wang, Wang Yang

CS 2XB3 L01
Department of Computing and Software, McMaster University

Report revision history

Apr.11th - All team members construct design specifications document

Team members

Team members	Student number	Roles and responsibilities
Suri Jia	400013552	Programmer, Tester
Mingnan Su	400016478	Programmer, Web developer, Leader, Scrum manager
Jiuwei Wang	400061882	Programmer, Documentation
Yang Wang	400072652	Programmer, Tester

Contribution page

Name	Role(s)	Contributions	Comments
Suri Jia	Programmer, Tester	Data input & storage processing & categorization; algorithm design; integration testing; ReadData module programming & testing; domain, partial MIS documentation;	
Mingnan Su	Programmer, Web developer, Leader, Scrum manager	module decomposition; module design; eclipse project to gitlab repo; project management; other modules programming & debugging & testing; web developing; compatibility error fixing; module design, functional requirements documentation;	
Jiuwei Wang	Programmer, Documentation	Dataset classification; Javadoc documentation; ratio, smallcatg module programming & testing; perfer module testing; elucidate uses relationships; partial MIS documentation;	
Yang Wang	Programmer, Tester	Data modification; output usability correction; linear regression design; entry, partition module programming & testing; web app testing; project abstract, partial MIS documentation;	

Executive Summary

This project, namely Finager, is focusing on giving budget suggestion to common households in Canada. Firstly, it acquires history household expenditure of different categories based on provinces around Canada. Then, these acquired data are analyzed and processed at the back-end in java, to predict any future year's cost in terms of categories. Finally, there will be an output of budget form through a web application. The budget form will be based on the user selection of provinces and input of specific year. At the same time, the user's income, saving goals, and preference of categories will be adjustable factors of the budget form. According to another province the user selected, the difference of main categories' expenditure between the two provinces will be compared and presented. In the end, the average household expenditure of provinces around all Canada will be sorted, and attached to the output form to show a comparison of provincial consumption level.

Uses Relationships & UML diagram

Entry Module uses LinearRegression Module
(Read from the expenditure data and then use LinearRegression algorithm predicts the expenditure for a given year)

Partition Module uses nothing
(Get the information of predicted expenditure of all category from Entry Module and then partition the data set, get the predicted average expenditure of selected provinces.)

ReadData Module uses Digraph Module
(Separate data from the output of Partition Module and built a digraph)

Ratio Module uses ReadData Module
(Get the overall predicted average expenditure value from ReadData Module and then calculate the ratio K.)

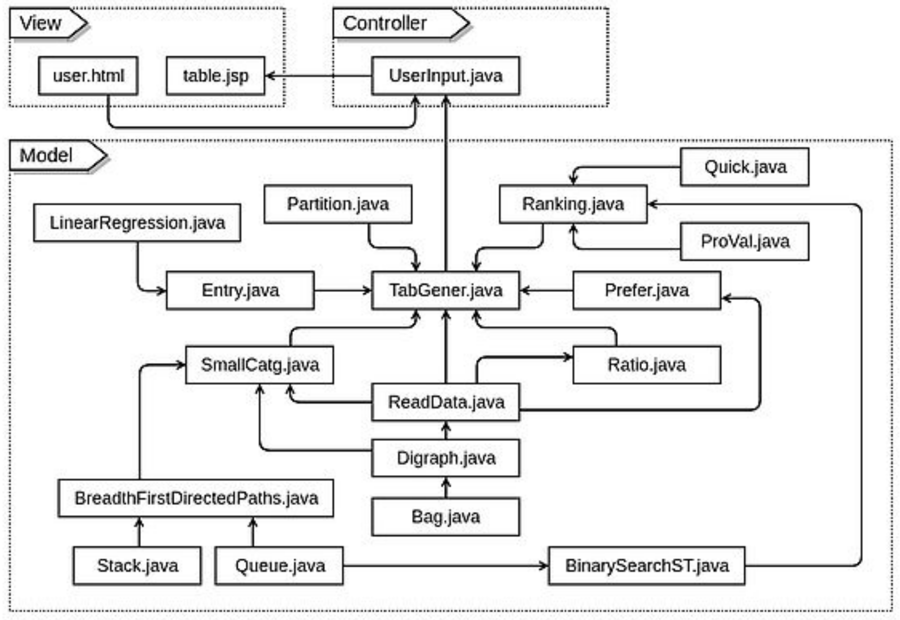
SmallCatg Module uses ReadData, Digraph, BreadthFirstDirectedPath Module
(Get all small category value of given big category index by using bfs algorithm)

Prefer Module uses ReadData Module
(Adjust expenditure value uses user preferences and get from ReadData Module.)

ProVal Module does not use any Module
(A ADT used by ranking module)

Ranking Module uses ProVal, Quick, BinarySearchST Module
(Uses Quick Module ranking the expenditure values, and use BinarySearchST module search for specific provinces data)

TabGener Module uses Entry, Partition, ReadData, Ranking, Ratio, SmallCatg, Prefer Module
(Transfer user input to all other java file and apply them.)



Explanation of why decomposed into these modules

In this project, we want to build an web application on tomcat Apache server by using MVC design pattern. So inside the box represents the view, we have user.html which takes the input from user, and table.jsp which return the output. Inside the controller design, we have one class UserInput.java which connects to the “view” box, and use model classes to compute the result. There is one concept in module decomposition which is that “Fan-in is better than fan-out”. Based on this concept, we create a class called TabGener.java which use all computation module together. Hence, when controller need model to perform the calculation based on user input, controller only need to call one class (which is TabGener.java). TabGener.java use seven classes to finished the computation, where each class in the seven classes do a part of the calculation. Entry and Partition module is used to simplify the dataset. ReadData module is used to generate a graph and store all the value in the dataset. SmallCatg, Ratio and Prefer module are used to adjust the data based on user input. Ranking module is created to sort and search through the dataset in order to provide more features of our web application.

Module Specifications Interface

Bag Module
Template Module
from http://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/Bag.java.html

Digraph Module
Template Module
from http://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/Digraph.java.html

BinarySearchST Module
Template Module
from http://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/BinarySearchST.java.html

BreathFirstDirectedPaths Module
Template Module
from
http://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/BreadthFirstDirectedPaths.java.html

LinearRegression Module
Template Module
from http://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/LinearRegression.java.html

Queue Module
Template Module
from http://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/Queue.java.html

Quick Module
Template Module
from http://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/Quick.java.html

Stack Module
Template Module
from http://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/Stack.java.html

Entry Module

Abstract
The Entry module will generate select year's expenditure data by a vector of string from original file. Linear regression module is applied on all history data and generate a formula to predict the future year's expenditure. All the data process are finished in the constructor.

Trace back to requirements
Performance - This class simplify the dataset by 35 times firstly.

Uses
LinearRegression.java

Syntax
Exported Access Programs

Routine name	In	Out	Exceptions
Entry	Vector<String>, Double		
getoutput		Vector<String>	

Semantics

State Variables
selectyear: double
data: Vector<String>
output: Vector<String>

State Invariant
none

Assumptions
The entry constructor takes a vector of string as input and output data can only be accessed through the getoutput method.

Information Hiding Reasoning
All local variables are created as private variables to enhance the idea of information hiding. Other modules will be unable to perform make changes to the values of the current module unless there exists a method that allows other modules to change the value externally or a function that alters the value of the local variables internally during the process of calculation. This module is a modification process of the original data and getoutput method is the only accessible method to the output data.

Access Routine Semantics
Entry (A, Selectyear):

- transition: data, selectyear = A, Selectyear; Data marked as 2007 price and data outside Canada are omitted. Each category data from start year to end year are analyze by linear regression and get a formula related with year. Substitute selectyear inside the formula and the result will be stored in the output.
- output: none
- exception: none

getoutput ():

- output: out = output
- exception: none

Partition Module

Abstract
The partition module will generate selected province expenditure from the entry module's output method of selected year's expenditure input.

Trace back to requirements
Maintainability - we omit unnecessary data from our calculation, make the program easy to follow

Uses
None

Syntax
Exported Access Programs

Routine name	In	Out	Exceptions
Partition	Vector<String>, String		none
partitionoutput		Vector<String>	none

Semantics

State Variables
Province: String
Partitioninput: Vector<String>
Partitionoutput: Vector<String>

State Invariant
none

Assumptions
The partition constructor takes a vector of string, which should be the output of entry as input and another string input for province selection.

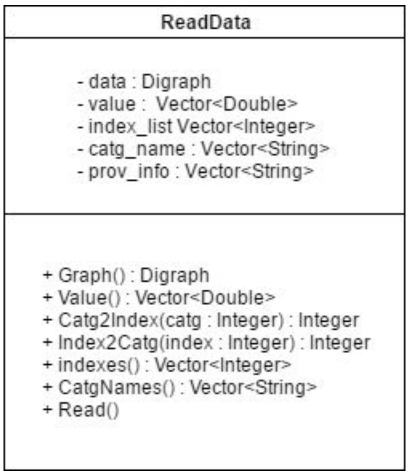
Information Hiding Reasoning
All local variables are created as private variables to enhance the idea of information hiding. Other modules will be unable to perform make changes to the values of the current module unless there exists a method that allows other modules to change the value externally or a function that alters the value of the local variables internally during the process of calculation. This module is a partitioning process of the entry output data and partitionoutput method is the only accessible method to the output data.

Access Routine Semantics
Partition (partitioninput, province):
- transition: Partitioninput, Province = partitioninput, province; Based on the input province, the related data from Entry module output are extracted and store in the output of this module.
- output: none

- exception: none

partitionoutput ():
- output: out = Partitionoutput
- exception: none

ReadData Module


ReadData
- data : Digraph
- value : Vector<Double>
- index_list Vector<Integer>
- catg_name : Vector<String>
- prov_info : Vector<String>
+ Graph() : Digraph
+ Value() : Vector<Double>
+ Catg2Index(catg : Integer) : Integer
+ Index2Catg(index : Integer) : Integer
+ indexes() : Vector<Integer>
+ CatgNames() : Vector<String>
+ Read()

Abstract
The ReadData module will generate a category graph by separating data from Partition module.

Trace back to requirements
Verifiability - The total value of small categories is the value of big category, which could be taken as a fact in our dataset, are used in to a mathematical formula in this module.

Uses
Digraph

Syntax
Exported Access Programs

Routine name	In	Out	Exceptions
ReadData	Vector<String>		none
Graph		Digraph	none
Value		Vector<Double>	none
Catg2Index	int	int	none
Index2Catg	int	int	none
indexes		Vector<Integer>	none
CatgNames		Vector<String>	none
Read			none

Semantics

State Variables
data: Digraph
value: Vector<Double>
index_list: Vector<Integer>
catg_name: Vector<String>
prov_infro: Vector<String>

State Invariant
none

Assumptions
The constructor takes a vector of string as input and call local read method to set the graph.

Information Hiding Reasoning
All local variables are created as private variables to enhance the idea of information hiding. Other modules will be unable to perform make changes to the values of the current module unless there exists a method that allows other modules to change the value externally or a function that alters the value of the local variables internally during the process of calculation. In this module, no value will be changed after initiation.

Access Routine Semantics
ReadData (file):
    transition: prov_info = file; Read()
    output: none
    exception: none

Graph():
    output: out = data
    exception: none

Value():
    output: out = value
    exception: none

Catg2Index(int catg):
    output: out = position of given category index in the list
    exception: none

Index2Catg(int index):
    output: out = category index of given position in the list
    exception: none

indexes():
    output: out = list of category index
    exception: none
CatgNames():
    output: out = all categories of selected provinces
    exception: none

Local function:
Read(): separate information from file, build a digraph with edges from big categories to small categories.

Ratio Module

Abstract
Given the income and saving from user, this module generates a value k, which is used to adjust the value in dataset to affordable expenditure value to user.

Trace back to requirements
Understandability - The dataset takes a default average expenditure value among households, which might not be the expenditure value that user could afford. We use this module to adjust the value to a relatively reasonable value. Hence, the expenditure table that user get in the end will make more sense since the consumption level is already matching.

Uses
ReadData.java

Syntax
Access Routines Programs

Routine name	In	Out	Exceptions
Ratio	Double, Double, ReadData		
overall		Double	
k		Double	

Semantics

State Variables
user: Double
rd: ReadData

State Invariant
none

Assumptions
The constructor Ratio is called for each abstract object before any other access routine is called for that object. The constructor cannot be called on an existing object.

Information Hiding Reasoning
Both state variables are private and prevent being changed. Other modules will be unable to perform make changes to the values of the current module unless there exists a method that allows other modules to change the value externally or a function that alters the value of the local variables internally during the process of calculation. In this module, no value will be changed after initiation.

Access Routine Semantics
Ratio (income, saving, rd):
- transition: user := income - saving, rd := rd
- output: out := self
- exception: none

overall ():
- output: out := rd.value.get(0)

- exception: none

k ():
- output: out := user/overall()
- exception: none

SmallCatg Module

Abstract
This module consists of one main function: take the index of big category, return all small categories' value.

Trace back to requirements
Maintainability - This module separate one complex math calculation, put into a single module. Even if the module consists of one method, the only method is complex enough to be separated.

Uses
ReadData.java, Digraph.java, BreadthFirstDirectedPaths.java

Syntax
Access Routine Programs

Routine name	In	Out	Exceptions
SmallCatg	ReadData	~	~
smaller	real	Vector<Double>	~

Semantics

State Variables
rd: ReadData

State Invariant
none

Assumptions
The SmallCatg constructor is called for each abstract object before any other access routine is called for that object. The constructor can only be called once.

Information Hiding Reasoning
The state variable is private to enhance the idea of information hiding. Other modules will be unable to perform make changes to the values of the current module unless there exists a method that allows other modules to change the value externally or a function that alters the value of the local variables internally during the process of calculation. In this module, no value will be changed after initiation.

Access Routine Semantics
SmallCatg (rd):
- transition: rd := rd
- output: out := self
- exception: none

smaller (v):
- output: result := all small category expenditure of given big category

Prefer Module

Abstract
This module adjust the value of big categories by user preference.

Trace back to requirements
Reliability - this module does not modify the value stored in ReadData module. Instead, this module have its own function to adjust the value from dataset, and modify it based on the preferences of the user.

Uses
ReadData.java

Syntax
Access Routine Programs

Routine name	In	Out	Exceptions
Prefer	Double[], ReadData		
Value	real	Double	
vals		Double[]	
Replace			

Semantics

State Variables
pref: Double[ ]
rd: ReadData

State Invariant
none

Assumptions
The constructor Prefer is called for each abstract object before any other access routine is called for that object. The constructor cannot be called on an existing object.

Information Hiding Reasoning
The state variables are private to enhance the idea of information hiding. Other modules will be unable to perform make changes to the values of the current module unless there exists a method that allows other modules to change the value externally or a function that alters the value of the local variables internally during the process of calculation. In this module, no value will be changed after initiation.

Access Routine Semantics
Prefer (user, read):
- transition: pref,rd = user,read; replace()
- exception: none

value (catg):
- output: out := pref [catg]
- exception: none

vals ():
- output: out := pref
- exception: none

<u>Local Functions:</u>
Replace ():
- transition: Calculate the big category expenditure adjust by user preferences.
- exception: none

ProVal Module

Abstract
This is an Abstract Data Type which helps to combine the related provincial name and expenditure information into many attributes of a single object, used for simplification in storing and retrieving data in other modules.

Trace back to requirements
Reliability - This module provides an ADT which could be used to store the string of the province and the expenditure value of the province. This make the programmer easy to program and have its own function to fulfill the purpose.

Uses
N/A

Syntax
Exported Access Programs

Routine name	In	Out	Exceptions
ProVal()	String, Double		
getVal()		Double	
getProv()		String	
compareTo()	ProVal	Integer	
toString()		String	

Semantics

State Variables
prov: String
val: Double

State Invariant
none

Assumptions
The constructor ProVal is called for each abstract object before any other access routine is called for that object. The constructor cannot be called on an existing object.

Information Hiding Reasoning
All local variables are created as private variables to enhance the idea of information hiding. Other modules will be unable to perform make changes to the values of the current module unless there exists a method that allows other modules to change the value externally or a function that alters the value of the local variables internally during the process of calculation. In this module, no value will be changed after initiation.

Access Routine Semantics
ProVal (province, value):
- transition: prov, val = province, value
- output: none
- exception: none

getVal ():

- output: out = val
- exception: none

getProv ():

output: out = prov

exception: none

compareTo (other):

output: out = whether the passed in object is greater than(+1), less than(-1) or equal to(0) the current object

exception: none

toString ():

output: out = object with object all object attributes

exception: none

Ranking Module

Abstract
This module is used to generate a ranking of the expenditure levels across Canada, and methods will be called before the program generates and output for the user to display the data in descending order.

Trace back to requirements
Maintainability - This module is used to sort and search through the datset to provide more feature to the user. The separation of module make programmer easy to maintain the program.

Uses
ProVal

Syntax
Exported Access Programs

Routine name	In	Out	Exceptions
Ranking()	Vector<String>, String		
Find()	String	Double	
getRank()		ProVal	

Semantics

State Variables
data: Vector<String>
ranked: ProVal[]

State Invariant
none

Assumptions
The Ranking constructor is called for each abstract object before any other access routine
is called for that object. The constructor can only be called once.

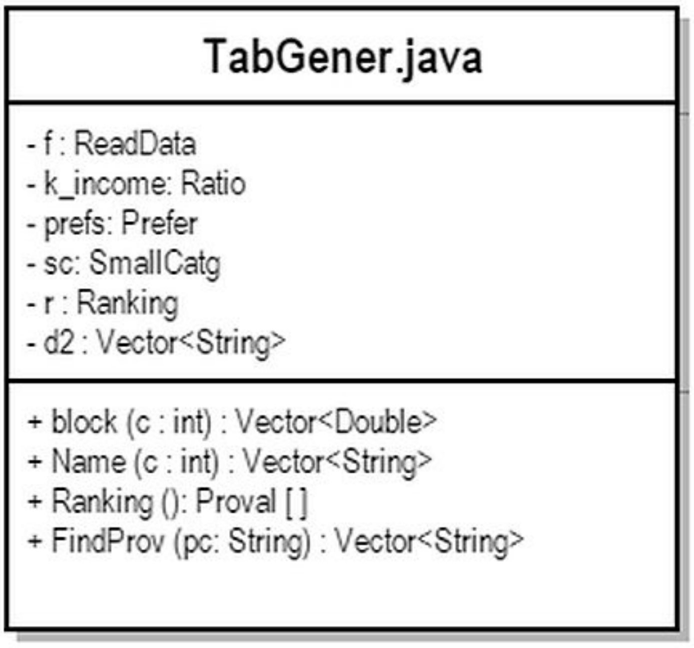
Information Hiding Reasoning
All local variables are created as private variables to enhance the idea of information hiding. Other modules will be unable to perform make changes to the values of the current module unless there exists a method that allows other modules to change the value externally or a function that alters the value of the local variables internally during the process of calculation. In this module, no value will be changed after initiation.

Access Routine Semantics
ProVal (allData, Categories):
    transition: data = allData, ranked initializes length to the length of *Categories*
    output: none
    exception: none

find (province):
    output: out = expenditure value at index of matching value to the inputted province
    exception: none

```
getRank ():
        output: out = ranked
        exception: none
```

TabGener Module



<u>Abstract</u>
This module combine all the computation components together and present the final result in a block. Each block's first element is big category value, follow by smaller categories' value.

<u>Trace back to requirements</u>
Verifiability - This module combine all the math computational modules together and connect them by math. In this way, the algorithm and math way is clear to see and maintain. Also, the reasoning behind the math is clear to show.

<u>Uses</u>
Entry, Partition, ReadData, Ratio, Prefer, Ranking

<u>Syntax</u>
Exported Access Programs

Routine name	In	Out	Exceptions
new TabGener()	Vector<String>, String, Double, Double, Double, Double[ ]		
Block()	int	Vector<Double>	
Name()	Int	Vector<String>	
Ranking		ProVal[ ]	
FindProv()	String	Vector<String>	

<u>Semantics</u>

<u>State Variables</u>
f: ReadData
k_income: Ratio

prefs: Prefer
sc: SmallCatg
r: Ranking

State Invariant
none

Assumptions
The constructor is called for each abstract object before any other access routine
is called for that object. The constructor can only be called once.

Information Hiding Reasoning
All local variables are created as private variables to enhance the idea of information hiding. Other modules will be unable to perform make changes to the values of the current module unless there exists a method that allows other modules to change the value externally or a function that alters the value of the local variables internally during the process of calculation. In this module, no value will be changed after initiation.

Access Routine Semantics
ProVal (data, prov, year, income, saving, pref):
    transition: pass the data into Entry and Partition module, get simplified data d3;
        f = ReadData(d3);
        k_income, prefs, sc, r := Ratio(income,saving,f), Prefer(pref,f), SmallCatg(f)
    output: none
    exception: none

block (c):
    output: out = big + small category expenditure value for given big category index c
    exception: none

Name (c):
    output: out = big + small category name for given big category index c
    exception: none

Ranking():
    output: out = pass the data into Entry first, search for final consumption expenditure, return the sorted array with sorted value
    exception: none

FindProv(pc):
    output: out = search through the dataset, find the all the big category value of given provinces
    exception: none

UserInput

Abstract

Controller class which takes data from user, and then use model classes to compute the result, finally forward the output to user.

Table.jsp

Abstract

A web page to show the output to user.

User.html

Abstract

A web page to get the input from user.

Internal evaluation of design

We use MVC design pattern in the design of this project. There are three components in our project: view, controller and model. View and controller part in this project is pretty simple. We only consider one interaction between user and computer: user fill in all the information that program needs, hit “submit” button, and then the program computes the result and forward to user, done. There is no further interaction between user and computer, which could be considered as too “simple” for an web application. In this project, our main focus is on model part. We use a directed graph, where nodes represents the categories, and the edge represents the relationship between categories (for example, a points to b -> b is a component of a, which means b is the small category of a). After that, we use breadth first search to check which small category is a part of bigger category. We also add two more features for user: Consumption level ranking and provinces consumption level comparison. In model classes, we implement these two features by using Quick Sort and Binary Search. In all, our design does not focus on the the user interface of the app, instead we put much more effort on the algorithm design of the model classes.