McMaster University
Software Engineering Department
SFWR ENG 2XB3

Budget Facilitator

April 11, 2015
Group 03

Members
Shandelle Murray
Mitchell Coovert
Athidya Raveenthranehru

Table Of Contents

Revisions Page

Team Members	Student Numbers	Roles & Responsibilities
Mitchell Coovert	1306701	Programmer: sorting and searching algorithms, GUI
Shandelle Murray	1303109	Programmer: ADT, Log Admin, Tester
Athidya Raveenthranehru	1316204	Project Leader, Researcher, Designer

By virtue of submitting this document we electronically sign and date that the work being submitted by all the individuals in the group is their exclusive work as a group and we consent to make available the application developed through [CS] or [SE]-2XB3 project, the reports, presentations, and assignments (not including my name and student number) for future teaching purposes.

Contributions Page

Name	Roles	Contributions
Mitchell Coovert	Programmer: sorting, and searching algorithms, GUI	Completed the majority of the implementation, tested, wrote requirements trace back and internal review
Shandelle Murray	Programmer: ADT, Log Admin, Tester	Completed parts of requirements and design specifications including diagrams, updated log, helped test
Athidya Raveenthranehru	Project Leader, Researcher, Designer	Delegated tasks, completed large parts of requirements and design specifications

Executive Summary:

In the fast-paced, consumer-based society that we live in today, people are often concerned about their desires for more time and more financial freedom. Although implementing a budget to govern spending is usually useful and worthwhile, people will often not make the effort to control, monitor, or record their spending habits. They will often lose receipts and not take the time each day to analyze areas where they may unnecessarily be overspending. Budget Facilitator is a simple solution to this issue. This application will organize users information in a simple convenient manner. It takes the users individual purchases as input and their budget and organizes and compares these values in an efficient manner to the user. The application will take in purchase information such as the item, date of purchase, price and category of the item such as transportation, entertainment, food, utilities or one of many others. It will sort the purchases by date and display the information in lists or graphs against the budget for each category. The user can then use this information to adjust their spending habits as necessary in order to achieve a balanced budget with minimal effort.

Requirements

1. *Purchases Handling*

1.1 allow user to input their purchases (input name, category, price of item and date)
1.2 allow user to edit or delete purchases
1.3 let user search for individual purchase
1.4 store all individual purchases
1.5 organize purchases by category and date
1.6 visually display purchases in their ordered manner in a list format and graph formal (line graph, bar graph, pie graph)
1.7 delete purchases that are dated over 2 years previous to the current date

2. *Budget Facilitating*

2.1 allow user to input budget for each category
2.2 show budget comparison to their purchases in the graph and list formats
2.3 give notification if approaching or exceeding budget in a category

Requirements Traceback

Requirements	**Model**	**View Controller**
1.1	DataStorage(), ConcatPur(),updateSpent(),addPurchasesTo()	addScene()
1.2	remove()	removeScene()
1.3	DataStorage(), BinarySearch()	SearchDateScene(), searchPurchaseScene()
1.4	DataStorage()	
1.5	ReadDate(), readCategory(), mergeSort()	ByDateScene(), byCategoryScene(), byPurchaseScene()
1.6	DataStorage(), ReadDate(),readCategory()	PieScene(),lineScene()
1.7		
2.1	DataStorage()	
2.2	ReadCategory(), readDate(), calcCat(), calcSpending()	PieScene(), lineScene()
2.3	budgetCompare()	

Description of Classes

Binary Search
Description of Implementation: implemented to allow user to search through their purchases for viewing or removing purchases

Module Interface Specification

Methods

Access Program Name	**Parameters Used**	**Description**	**References**
BinarySearch	int capacity	Initiates two arrays of size input parameter capacity, one of Purchases and one of Dates	None
BinarySearch	None	Initiates two arrays of size 2, one of Purchases and one of Dates	None
put	Purchase pur Date date	Inserts a purchase and date to the arrays in the correct order	delete rank compareTo resize check
get	Purchase pur	Returns date of Purhase pur	compareTo rank
delete	Purchase pur	Deletes a purchase from the array along with its corresponding date	isEmpty rank compareTo resize check
contains	Purchase pur	Checks if Purchase pur is in Purchase array purs	get
isEmpty	None	Checks if Purchase array purs is empty	size
size	None	Returns size of Purchase array purs	None
min	None	Returns earliest (by date) purchase	isEmpty

max	None	Returns most recent (by date) purchase	isEmpty
floor	Purchase pur	Return Purchase pur or the closest purchase to the date of input purchase that is before the date of pur	rank compareTo
ceiling	Purchase pur	Returns Purchase pur or the closest purchase to the date of input purchase that is after the date of pur	rank
rank	Purchase pur	Returns index at which Purchase pur would be ordered in the purs array of Purchases	compareTo
select	int k	Return value at index k of Purchases array purs	None
deleteMin	None	Deletes oldest purchase	delete min
deleteMax	None	Deletes most recent purchase	delete max
size	Purchase lo, Purchase hi	Returns the length of array where lo is the oldest purchase and hi is the most recent purchase	compareTo contains rank
purchases	Purchase lo, Purchase hi	Returns iterable Purchases of array purs	rank contains
Purchases	None	Calls purchases with parameters min() and max()	min max

Module Internal Design

Constants

Name	**Value**	**Type**
initCapacity	2	static final int

Variables

Name	**Type**
dates	Date[]

purs	Purchase[]
N	int

Methods

Access Program Names	**Parameters**	**Description**	**References**
resize	int newSize	Resizes purs and dates arrays to the size of the input newSize	None
check	None	Checks if array purs is sorted and if the ranks are in order	isSorted rankCheck
isSorted	None	Checks if purs array is sorted	compareTo size
rankCheck	None	Checks if purs is in proper rank order	rank select compareTo size

Implementation of Methods

1) @SuppressWarnings(“unchecked”)
public BinarySearch(int capacity)
make new array of Purchases called purs of size capacity
make new array of Dates called dates of size capacity

2) public Binary Search()
initializes arrays purs and dates to be size 2

3) public void put (Purchase pur, Date date)
if(date is null) call delete(pur) and end of method
i is value of calling rank(pur)
if(key is already in table) dates at array index i = input date and end of method
if(N equals length of purs) resize purs using resize method by calling resize(2*purs.length)
for (starting at j = N decremened by one as long as j is greater than i)
value of purs[j-1] assigned to purs[j]
value of dates[j-1] assigned to dates[j]
input pur assigned to purs at index i
input date assigned to dates at index i
make sure check() method returns true

4) public Date get (Purchase pur)
if(isEmpty() returns true) return null

int i is value of method rank(pur) (parameter pur)
if (i is less than N &&(and) purs at index i is equal to input pur) return dates at index i
else return null

5) public void delete(Purchase pur)
if (isEmpty() method returns true) end method
i is value of method rank(pur) with parameter pur
if (i is equal to N or purs at index i is equal to input parameter pur) then end method
for(j initially equals i ; incremented while j is still less than N -1)
value of purs[j+1] assigned to purs[j]
value of dates[j+1] assigned to dates[j]
decrement N
make purs at index N equal null
make dates at index N equal null
if(purs is 1/4 full or less) then call resize(purs.length/2) to make purs half the length
make sure check() method returns true

6) public boolean contains(Purchase pur)
return true if get(pur) value is not null; else return false

7) public boolean isEmpty()
return true is size() is 0; else return false

8) public int size()
return value of N

9) public Purchase min()
if (method isEmpty() returns true) return null
else return purs[0]

10) public Purchase max()
if (method isEmpty() returns true) return null
else return purs[N-1]

11) public Purchase floor(Purchase pur)
integer i = value of rank(pur)
if(i is less than N &&(and) input pur is equal to purs at index i) return purs at index i
if(i is 0) return null
else return purs at index i -1

12) public Purchase ceiling (Purchase pur)
integer i = value of rank(pur)
if(i equals N) return null
else return purs[i]

13) public int rank(Purchase pur)
lo is set to 0; hi is set to N-1

while (lo is less than or equal to hi)
integer m = lo + (hi – lo) /2
integer cmp is -1 if pur is less than purs[m], 0 is pur is equal to purs[m], and 1 if pur is greater than purs[m]
if (cmp less than 0) hi equals m -1
else if (cmp is greater than 0) lo = m + 1
else return m
return lo if lo is greater than hi

14) public Purchase select(int k)
if (k is less than 0) or if (k is less than or equal to N) return null
else return purs[k]

15) public void deleteMin()
calls delete method on oldest purchase in purs (at index 0)

16) public void deleteMax()
calls delete method on the most recent purchace in purs (at index N)
17) public int size(Purchase lo, Purchase hi)
if (lo is less than hi) return 0
if(contains(hi) is true) return rank(hi) – rank(lo) + 1
else return rank(hi) – rank(lo)

18) public Iterable<Purchase> purchases (Purchase lo, Purchase hi)
initialize queue of purchases called queue
if(lo is null &&(and) hi is null) return queue
if(lo is null) throw a null pointer exception with message “lo is null in keys()”
if(hi is null) throw a null pointer exception with message “hi is null in keys()”
if(lo is greater than hi) return queue;
for(integer i = value of rank(lo) while i is less than value of rank(hi) and incrementing i)
add purs at index i to the queue
if(contains(hi) is true) add purs at index (rank of hi) to the queue
return the queue

19) public Iterable<Purchases> purchases()
return the value of purchases(min(), max())

20)@SuppressWarnings(“unchecked”)
private void resize (int newSize)
check if newSize is greater than or equal to N else exit method
make temporary Date and Purchase arrays of the size newSize
for(integer i equals 0, while i is less than N and increment i each time)
temporary Purchase array at index i = purs at index i
temporary Dates array at index i = dates at index i
make temporary Purchase array purs (set temp array to purs)
make temporary Dates array dates (set temp array to dates)

21)private boolean check()
return true if (isSorted() is true &&(and) if rankCheck() is true)
else return false

22) private boolean isSorted()
for(integer i is 1, while i is less than the size of purs array; increment i each time)
if(purs[i] is less than purs[i-1] return false
else return true

23) private rankCheck()
for(integer i = 0, while i is less than size of purs, increment i each time)
if(i is not equal to the value of(rank(select(i))
return false
else return true

Budget
Description of Implementation: Implemented to handle the users budget. There are set and get methods where the user will be able to input their budget for each category of spending and see the value that they have set in other representations when graphing purchases and comparing to budget.

Module Interface Specification

Types

Name	**Origin**
Budget	Public

Methods

Access Program Name	**Parameters Used**	**Description**
getTransportation	None	returns value of transportation budget
getEntertainment	None	returns value of entertainment budget
getFood	None	return value of food budget
getUtilities	None	return value of utilities budget
getSchool	None	return value of school budget
getClothes	None	return value of clothes budget
getEssentials	None	return value of essentials budget
getOther	None	return value of budget of items that do not fit into a category
getTotal	None	return value of total budget
setTransportaion	Double trans	Set the transportation budget to

		the value of trans
setEntertainment	Dounle ent	Set the entertainment budget to the value of ent
setFood	Double food	Set the food budget to the value of food
setUtilities	Double util	Set the utilities budget to the value of util
setSchool	Double school	Set the school budget to the value of school
setClothes	Double clothes	Set the clothes budget to the value of clothes
setEssentials	Double essen	Set the essentials budget to the value of essen
SetOther	Double other	Set the others budget to the value of other
CompareTo	Budget comp	compares total budget and comp's budget

Module Internal Design

Variables

Name	**Type**
trans	double
ent	double
food	double
util	double
school	double
clothes	double
essen	double
other	double

Methods

Access Program Name	**Parameters Used**	**Description**
updateTot	None	updates the value of the total budget

Implementation of Methods

1) private void updateTot()

- adds all the category budget variables together to update the total budget

2) public double getTransportation()
   - returns value of transportation budget
3) public double getEntertainment()
   - returns value of entertainment budget
4) public double getFood()
   - returns value of food budget
5) public double getUtilities()
   - returns value of utilities budget
6) public double getSchool()
   - returns value of school budget
7) public double getClothes()
   - returns value of clothes budget
8) public double getEssentials()
   - returns value of clothes budget
9) public double getOther()
   - returns value of other budget
10) public double getTotal()
   - returns value of all budget categories added together
11) public void setTransportation(double trans)
   - set transportation budget as input parameter
12) public void setEntertainment(double ent)
   - set entertainment budget as input parameter
13) public void setFood(double food)
   -set food budget as input paramset schooleter
14) public void setUtilities(double util)
   -set utilities budget as input parameter
15) public void setSchool(double school)
   -set school budget as input parameter
16) public void setClothes(double clothes)
   -set clothes budget as input parameter
17) public void setEssentials(double essen)
   -set essentials budget as input parameter
18) public void setOther(double other)
   -set other budget as input parameter
19) public int compareTo(Budget comp)
   if(the budget is equal to the input budget comp) then return 0
   else (if (budget is less than input budget comp) then return -1
      else return 1)

Category

Description of Implementation: Enumerator of the categories of purchases.

Module Interface Specification

Types

Name	Origin
TRANSPORTATION	enum
ENTERTAINMENT	enum
FOOD	enum
UTILITIES	enum
SCHOOL	enum
CLOTHES	enum
ESSENTIALS	enum
OTHER	enum

Data Storage
Description of Implementation: class used to manipulate csv file where the data of all the purchases is stored for this application

Module Interface Specification

Methods

Access Program Name	Parameters Used	Description	References
dataStorage	Budget goal Budget current	Writes budget information of input parameter goal to csv file, calls init current with parameter as current	getTransportation getEntertainment getFood getUtilities getSchool getClothes getEssentials getOther getTotal
updateSpent	None	updates spending by adding purchase information	getTransportation getEntertainment getFood getUtilities getSchool getClothes getEssentials getOther getTotal
concatPur	Purchase pur[]	Adds a purchase to the csv file	date.toString
readDate	Date beg Date end	Given two dates it will output an array of	date.compareTo

		Purchases of items purchased within the date range	
readCategory	Category cat	Given a category will output an array of Purchases of items purchased that are from this category	
remove	Purchase pur	Removes a specific purchase from the csv file	
clearHistory	None	Removes all stored information in the csv file	
budgetCompare	None	Compares your set budget to current spending	

Module Internal Design

Variables

Name	**Type**
set	Budget
current	Budget

Types

Name	**Origin**
Budget	private

Methods

Access Program Name	**Parameters Used**	**Description**	**References**
initCurrent	Budget cur	Writes current budget information to csv file	getTransportation getEntertainment getFood getUtilities getSchool getClothes getEssentials getOther getTotal
addPurchasesTo	None		

splitter	String[] array	Helps format input in a specific manner to assist other methods	

Implementation of Methods

1) public DataStorage(Budget goal, Budget current) throws IOException
    set up to write onto the csv
    writes in budget for each category that user has set for themselves
    closes write file
    calls initCurrent(current)

2) private static void initCurrent(Budget cur)
    set up to write onto the csv
    writes in current spending for each category based on users purchases
    closes the csv file

3) private static void updateSpent() throws IOException, FileNotFoundException
    reads csv file with purchases in it and closes read file
    adds purchases using addPurchasesTo()
    opens write file and write in current spending and closes write file

4) public static void concatPur (Purchase[] pur) throws IOException
    set up to write onto csv file
    set data to be an array of information of the purchase (price, item, category, date)
    write data to file
    close writer
    calls updateSpent() to update spendings

5) public static Purchase[] readDate (Date beg, Date end) throws FileNotFoundException, IOException
    opens csv file to read purchases and
    saves array of Purchases called output of purchases that have dates that fall between the two input dates
    closes read file
    returns array of Purchases output

6) public static Purchase[] readCategory (Category cat) throws IOException, FileNotFoundException
    opens csv file reader
    saves array of Purchases called output of purchases that are in the category of the input category
    closes read file
    returns array of Purchases output

7) public static void remove (Purchase pur) throws IOException, FileNotFoundException
    open read file and open a write file to a temporary stoarage
    read from reader file and write to temporary storage file

skip writing Purchase pur into writer file and continue writing
make the temporary storage file 2 and storage fle 1
rename file2 to file1 (temp storage becomes storage)
close reader and writer file

8) public static void clearHistory() throws IOException, FileNotFoundException
open csv file reader and writer
write empty lines to file in place of information (remove all information stored in the storage csv file)
close writer and reader

9) public static Budget addPurchasesTo() throws FileNotFoundException, IOException
opens csv reader file
adds all categories spendings
adds total of each categories spendings together to get total spending
close reader

10) public static int budgetCompare()
compares budget to spending

11) private static String[] splitter(String[] array)
String array temp saves split of array[0] at [“]
string array output of length temp length + 1 divided by 2
j equals 0
for(integer i equals 0; while i is less than temp length plus 1 divided by 2; increment i)
output[i] equals value of temp at index j
j incremented by 2
return output

Date
Description of Implementation: Date abstract data type used in other classes to easily access information of the date of the purchase

Module Interface Specification

Types

Name	**Origin**
Date	Public

Methods

Access Program Name	**Parameters Used**	**Description**
Date	int day int month int year	constructor method to construct abstract data type Date

toString	None	returns the date in the format DD/MM/YYYY
setDay	int day	sets the day to be the the value of the input parameter
setMonth	int month	sets the month to be the value of the input parameter
setYear	int year	sets the year to be the value of the input parameter
getDay	None	returns the day
getMonth	None	returns the month
getYear	None	returns the year
compareTo	Date date	Compares two Date type objects

Module Internal Design

Variables

Name	**Type**
day	int
month	int
year	int

Implementation of Methods

1) public Date(int day, int month, int year)
    initialize day to input day
    initialize month to input month
    initialize year to input year

2)public String toString()
    return string in format DD/MM/YYYY

3) public void setDay(int day)
    initialize day to input day

4) public void setMonth(int month)
    initialize month to input month

5) public void setYear(int year)
    initialize year to input year
6) public int getDay()
    return value of day

7) public int getMonth()
    return value of month

8) public int getYear()
    return value of year

9) public int compareTo(Date date)
    if (current year is greater than the input dates year) return 1
    if (current year is less than input dates year) return -1
    if (current month is greater than the input dates month) return 1
    if (current month is less than input dates month) return -1
    if (current day is greater than input dates day) return 1
    if (current day is less than input dates day) return -1
    else return 0

GUI

Description of Implementation: ViewController of the application, allows user interaction and is face of the application to clients

Module Interface Specification

Types

Name	**Origin**
Date	Public

Methods

Access Program Name	**Parameters**	**Description**
start	Stage primaryStage	Home screen
homeScene	None	Sets parametes for welcome scene
pieScene	None	Pie chart displaying categories of spending
lineScene	None	Line chart displaying spending versus budget
byDateScene	None	Line chart displaying spending versus budget between a specific interval of days
dateTableScene	Purchase [] purs	Table display of a list of purchases divided into 5 columns (price, name, category and date), each purchase in own row
byPurchaseScene	None	Table display of all purchases divided into 5 columns (price, name, category and date), each purchase in own row

ByCategoryScene	None	Screen that allows user to choose what category of spendings they would like to see, has buttons for each category and a submit button
categoryTableScene	Purchase[] purs	Table display of a list of purchases divided into 5 columns (price, name, category and date), each purchase in own row
addScene	None	Window to allow user to input a purchase that is not in the saved data
removeScene	None	Window to allow user to input a purchase that is in the saved data that they want to remove
searchPurchasesScene	None	Window to allow user to input a purchase that they want to search for in the saved data

Module Internal Design

Variables

Name	**Type**
stage	Stage
storage	DataStorage

Methods

Access Program Name	**Parameters**	**Description**
addHBox	None	Creates an Hbox with two buttons for the top reigon
addVBox	None	Creates a Vbox with a list of lnks for the left reigon
addGridPane	None	Creates buttons allowing user to do different things such as searching or clearing history used for homescreen
calcCat	None	Calculates total spending per category returns double array with each categories total spending per array index
calcSpending	None	Used by line graph to calculate total spending

Implementation of Methods

1) public void start(Stage primaryStage)
    stage is primary stage
    scene calls homescene
    sets parameters and title for homescreen

2) public Scene homeScene()
    sets parameters for homescene box
    sets border parameters
    returns scene(border)

3) public Scene pieScene()
    sets up pie chart to display pie chart divided by category
    sets parameters and title
    sets border and grid and titles
    scene equals Scene(border)
    return scene

4) public Scene lineScene()
    define axis
    label axis
    create chart
    define series and set names for series
    add data to series
    set buttons and parameters for buttons
    set border and titles
    set grid
    scene equals scene(border)
    return scene

5) public Scene byDateScene()
    new border
    new box
    add scene titles
    grid and parameters
    add buttons
    set text fields
    label beginnng date and ending date
    fill text fields
    scene equals scene(border)
    return scene

6) public Scene dateTableScene(Purchase[] purs)
    new border
    new box
    set titles
    set parameters
    set grid parameters
    divide columns and name them
    5 columns are price name category and date
    put all information into table
    scene equals scene(border)
    return scene

7) public Scene byPurchaseScene()
    new border
    new box
    set titles
    set parameters
    set grid parametes
    set home button
    divide columns and name them
    5 columns are price name category and date
    read storage by category
    add purchases to an array of purchases
    mergesort purchase array by date
    put all purchase information to table of price, name, categories, and dates
    scene equals Scene(border)
    return border

8) public Scene byCategoryScene()
    new border
    new box
    set titles
    set parameters
    set grid parametes
    set home button
    set category buttons for user to choose list of purchases for that category
    set submit button to choose category
    return border

9) public Scene categoryTableScene(Purchase[] purs)
    new border
    new box
    set titles
    set parameters
    set grid parametes
    set home button
    divide columns and name them
    5 columns are price name category and date
    put all information into table
    scene equals scene(border)
    return scene

10) public Scene addScene()
    new border
    new box
    set titles
    set parameters
    set grid parametes

set home button
set textfields for user to input price, item, day, month, year of purchase they want to add
set buttons for choosing category of purchase
set submit button
scene equals scene(border)
return scene

11)public Scene removeScene()
new border
new box
set titles
set parameters
set grid parametes
set home button
set textfields for user to input price, item, day, month, year of purchase they want to remove
set buttons for choosing category of purchase
set submit button
scene equals scene(border)
return scene

12) public Scene searchPurchaseScene()
new border
new box
set titles
set parameters
set grid parametes
set home button
set textfields for user to input price, item, day, month, year of purchase they want to search for
set buttons for choosing category of purchase
set submit button
scene equals scene(border)
return scene

13) public Scene searchedScene(Purchase pur)
new border
new box
set titles
set parameters
set grid parametes
set home button
set titles of price, name , category and date
show information of searched purchase w/ the purchase info under the corresponding titles
scene equals scene(border)
return scene

14) public Scene searchDateScene()
new border

new box
set titles
set parameters
set grid parametes
set home button
set textfields for user to input date
set submit button
scene equals scene(border)
return scene

15) private Hbox addHBox()
creates hbox and sets parameters for hbox
retusn hbox

16) private Vbox addVBox()
creates vbox and sets parameters
returns vbox

17) private GridPane addGridPane()
new gridpane and set parameters
set text to prompt user to click a button
sets up buttons that home screen will use
return grid

18) private Double[] clacCat()
read date file for purchases for each category
save totals to an array of doubles called output
return output

19) private Double[] calcSpending[]
read data file for purchases
calculates spending
returns double array called dbl

Item
Description of Implementation: Item is an ADT used in other classes to easily access the price and name of a purchase

Module Interface Specification

Types

Name	**Origin**
Item	Public

Methods

Access Program Name	Parameters	Description
Item	String name double price	Constuctor for adt Item
Item	None	Default constructor for adt Item sets values to 0 and null
getName	None	Returns name of item
getPrice	None	Returns price of item
setName	String name	Sets name of item to be input name
setPrice	double price	Sets price of item to input price
equals	Item item	Checks if current item and input item are equal or not
compareTo	Item good	Compares current item to input item

Module Internal Design

Variables

Name	Type
name	String
price	double

Implementation of Methods

1) public Item(String name, double price)
    set current name to input name
    set current price to input price

2) public Item ()
    set current name to null
    set current price to 0

3) public String getName()
    return current name

4) public double getPrice()
    return current price

5) public void setName(String name)
    set current name to input name

6) public void setPrice(double price)

set current price to input price

7) public boolen equals(Item item)
    if(current item is the same as input item) return true
    else return false

8) public int compareTo(Item good)
    if(price of current item is greater than price of item good) return 1
    else (if (price of item good is greater than price of current item) return -1
        else return 0)

Merge

Description of Implementation: An implementation of merge sort that is used to sort the purchases stored in the application.

Module Interface Specification

Methods

Access Program Name	**Parameters Used**	**Description**
mergeSort	Comparable[ ] a	Copies array of Comparable items to auxiliary array and calls private mergeSort to sort the original array

Module Internal Design

Methods

Access Program Name	**Parameters Used**	**Description**
mergeSort	Comparable [ ] a, Comparable [ ] aux, int left, int right	Recursive method called by the public method mergeSort to sort an array of Comparable items
merge	Comparable[ ] a, Comparable[ ] aux, int left, int right, int rightEnd	Used by mergeSort to merge sorted arrays together
less	Comparable v, Comparable w	Helper method for sort
exch	Comparable[ ] a, int i, int j	Helper method for sort

Implementation of Methods

1. public static void mergeSort(Comparable [ ] a){
    -copy entire array a to auxiliary array

-call to private method mergeSort with a, the auxiliary array, 0, and the length of a - 1
}

2. private static void mergeSort(Comparable [ ] a, Comparable [ ] aux, int left, int right){
-if left is smaller than right
-find center of array which is (left + right)/2
-recursive call to mergeSort to sort from left to center
-recursive call to mergeSort to sort from center +1 to right
-call to merge with the array a, the auxiliary array aux, left, center+1, and right
}

3. private static void merge(Comparable[ ] a, Comparable[ ] aux, int left, int right, int rightEnd){
-initialize variable leftEnd to the array element before right (right-1)
-initialize k to left
-initialize num to rightEnd- left + 1
-while left is smaller than or equal to leftEnd and right is smaller than or equal to rightEnd
-if the value in the array a at index left is smaller than the value at index right
-the value of aux at index k becomes the value of a at index left
-increment k and increment left by one
-else
-the value of aux at index k becomes the value of a at index right
-increment k and increment right by one
-while left is smaller than or equal to leftEnd
-copy rest of first half of array by setting the value of aux at k to the value of a at left and incrementing k and left

-while right is smaller than or equal to rightEnd
-copy rest of right half of array by setting the value of aux at k to the value of a at right and incrementing k and right
-for (i = 0 to i < num, incrementing i by 1 and decrementing rightEnd by 1)
-copy the auxiliary array back by setting the value of a at rightEnd to the value of aux at rightEnd
}

4. private static boolean less(Comparable v, Comparable w){
-return true if v is smaller than w, otherwise return false
}

5. private static boolean exch(Comparable[] a, int i, int j){
-initialize temporary Comparable t to the value of a at i
-set the value of a at index i to the value of a at index j
-set the value of a at index j to the value of t
}

Purchase

Description of Implementation: Abstract data type to store the Item, category, and date of a purchase. Includes get and set methods to access and modify the fields as well as methods that provide functionality such as returning a string representation of the purchase and comparing two purchases. The class implements the Comparable<Purchase> interface.

Module Interface Specification

Methods

Access Program Name	**Parameters Used**	**Description**
Purchase	Item item, Category cat, Date date	Constructor to create Purchase object with item, category, and date set
Purchase	None	Constructor to create Purchase item with 0 or null values for the fields
toString	None	Returns string representation of purchase
getItem	None	Get method to allow access item
getCategory	None	Get method to allow access to category of purchase
getDate	None	Get method to allow access to date of purchase
setItem	Item item	Set method to set item purchased
setCategory	Category cat	Set method to set category of purchase
setDate	Date date	Set method to set date of purchase
equals	Purchase pur	Returns whether same item or not
compareTo	Purchase pur	Compares dates and categories of two purchases

Module Internal Design

Variables

Name	**Type**
item	Item
category	Category
date	Date

Types

Name	***Origin***
Item	Implemented within application
Category	Implemented within application
Date	Implemented within application

Implementation of Methods

1. public Purchase(Item item, Category cat, Date date){
   -initialize instance variable item to parameter item
   -initialize instance variable category to parameter cat
   -initialize instance variable date to parameter date
   }

2. public Purchase(){
   - initialize instance variable item to new Item
   - initialize instance variable category to null
   - initialize instance variable date to null
   }

3. public String toString(){
   -return string "You spent " + instance variable price + " buying " + instance variable item + " on " + instance variable date
   }

4. public Item getItem(){
   -return the instance variable item
   }

5. public Category getCategory(){
   -return the instance variable category
   }

6. public Date getDate(){
   -return the instance variable date
   }

7. public void setItem(Item item){
   -set instance variable item to parameter item
   }

8. public void setCategory(Category cat){

-set instance variable category to parameter cat
}

9. public void setDate(Date date){
   -set instance variable date to parameter date
   }

10. public boolean equals(Purchase pur){
    -if (the instance variable item equals pur's item and the categories and dates are also equal)
        -return true
    -else
        -return false
    }

11. public int compareTo(Purchase pur){
    -if the instance variable item is greater than pur's item or the instance variable date is greater than pur's date
        -return 1
    -else
        - if the instance variable item is smaller than pur's item or the instance variable date is smaller than pur's date
            -return -1
        -else
            -return 0 since they are equal in value
    }

Quick
Description of Implementation: An implementation of quick sort to sort through purchases stored in the application.

Module Interface Specification

Methods

Access Program Name	**Parameters Used**	**Description**
quick	Comparable[ ] dates	Shuffles the contents of the Comparable array and calls for the array to be sorted

Module Internal Design

Methods

Access Program Name	**Parameters Used**	**Description**

quick	Comparable[ ] dates, int low, int high	Sorts an array of Comparable items
less	Comparable v, Comparable w	Helper method for quick sort to compare two items
exch	Comparable[ ] dates, int i, int j	Helper method for quick sort to exchange two items
shuffle	Comparable[] array	Shuffle the contents of the array so that it is randomly sorted

Implementation of Methods

1. public static void quick(Comparable[] dates) {
   -call shuffle to randomly shuffle the dates array
   -call private method quick to sort the dates array
   {

2. private static void quick(Comparable[] dates, int low, int high) {
   -if high is smaller than or equal to low
   -return to caller
   -initialize int lt to low, int i to low +1 and int gt to high
   -set Comparable v to the value of dates at index low
   -while i is smaller than or equal to gt
   -set int cmp to the value -1 if the value of dates at index i is smaller than the value of v; set cmp to the value of 0 if the values of dates at index i is equal to the value of v; set cmp to the value of 1 if the value of dates at index i is greater than the value of v
   -if cmp is smaller than 0
   -call exch to exchange the values of dates at indices lt and i
   -increment lt and i by 1
   -else
   -if cmp is greater than 0
   -call exch to exchange the values of dates at indices i and gt
   -decrement gt by 1
   -else
   -the values are equal so increment i by 1

   -recursively call quick to sort from low to lt-1
   -recursively call quick to sort from gt+1 to high

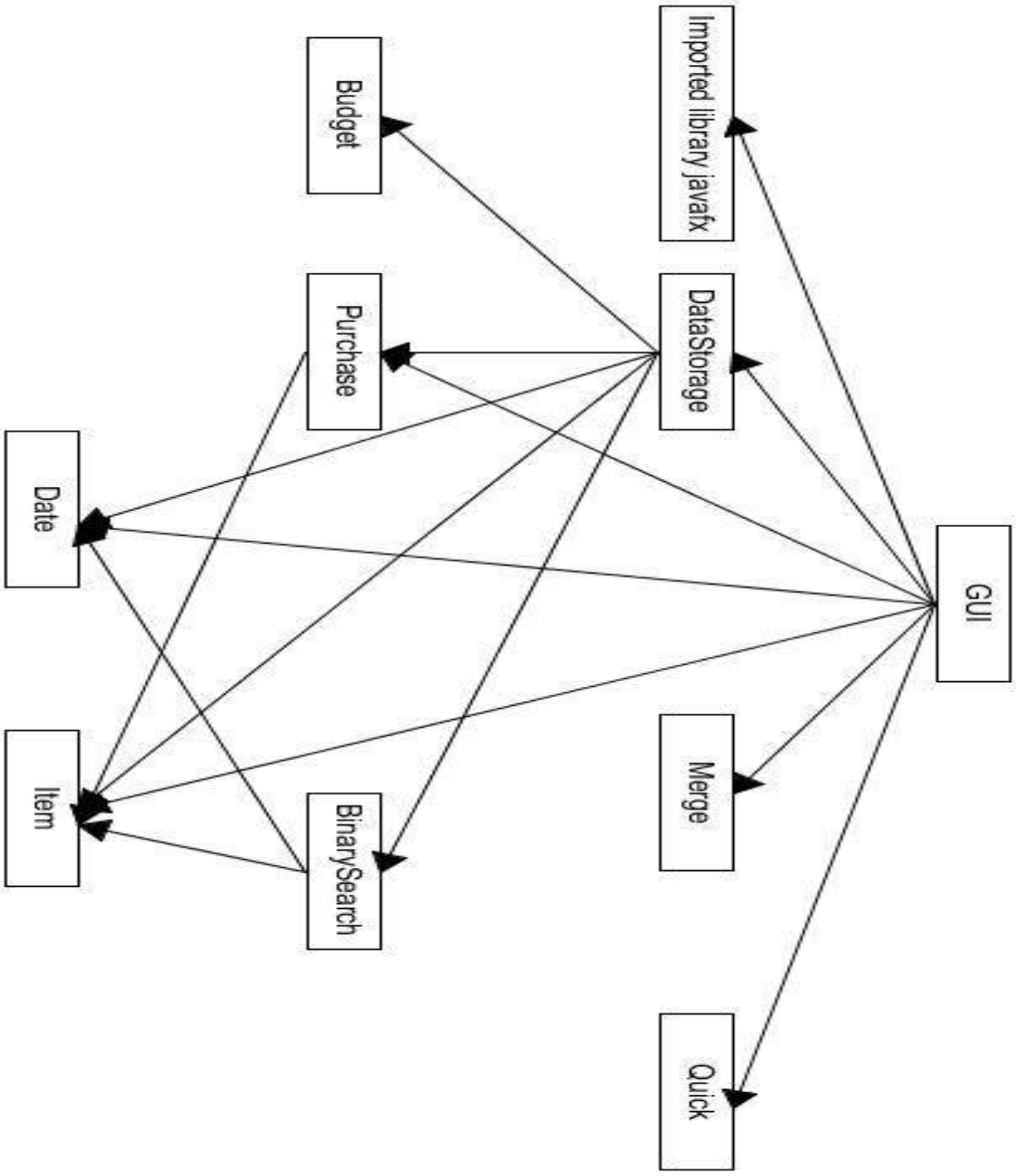
   }

3. private static boolean less(Comparable v, Comparable w) {
   -using the compareTo method, return true if v is smaller than w, otherwise return false
   }

4. private static void exch(Comparable[] dates, int i, int j) {
    -initialize temporary Comparable t to the value of dates at i
    -set the value of dates at index i to the value of dates at index j
    -set the value of dates at index j to the value of t
   }

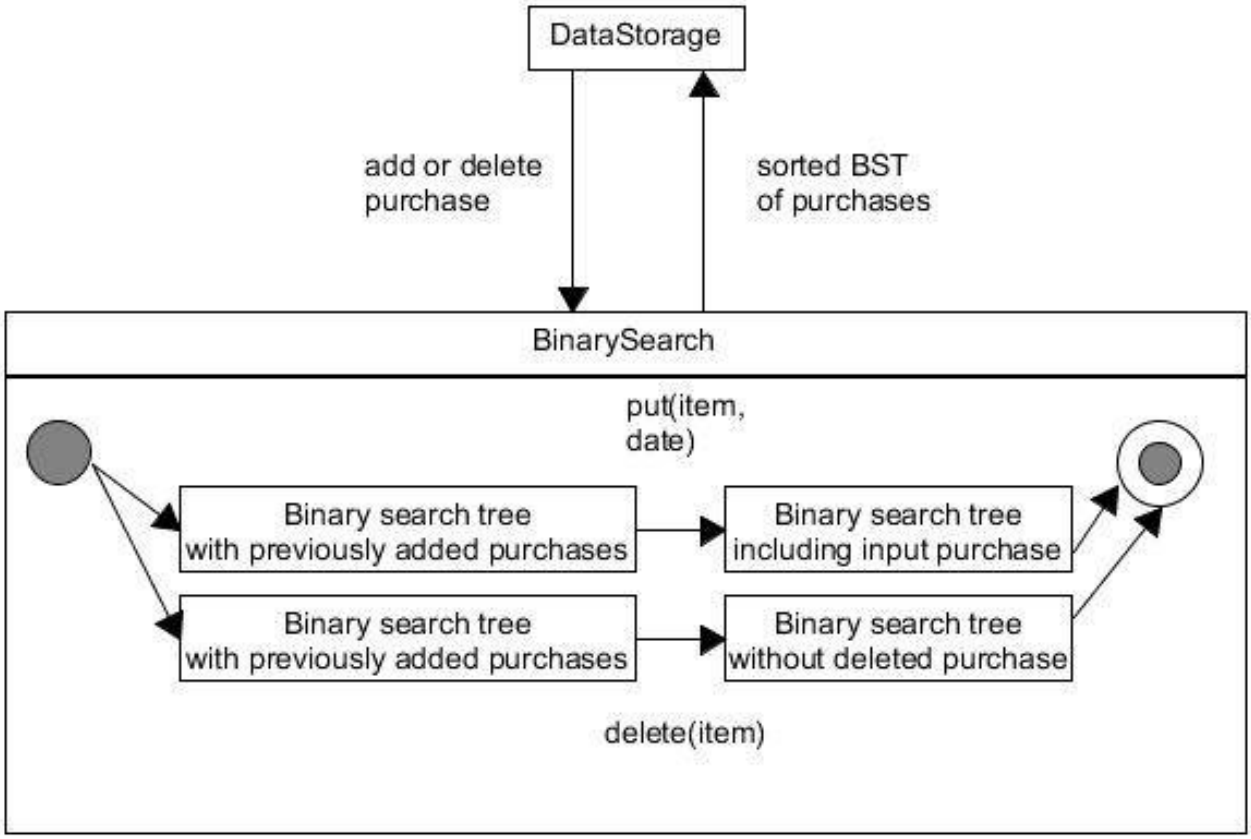
5. private static void shuffle(Comparable[] array) {
    -set int n to the length of array
    -for (i = 0 to i < length of array, increment i by 1 each time)
        -set int random to a random integer from i to a range of n-i
        -set Comparable randomElement to the value of array at index random
        -set the value of array at index random to the value of array at index i
        -set the value of array at index i to the value of randomElement
   }

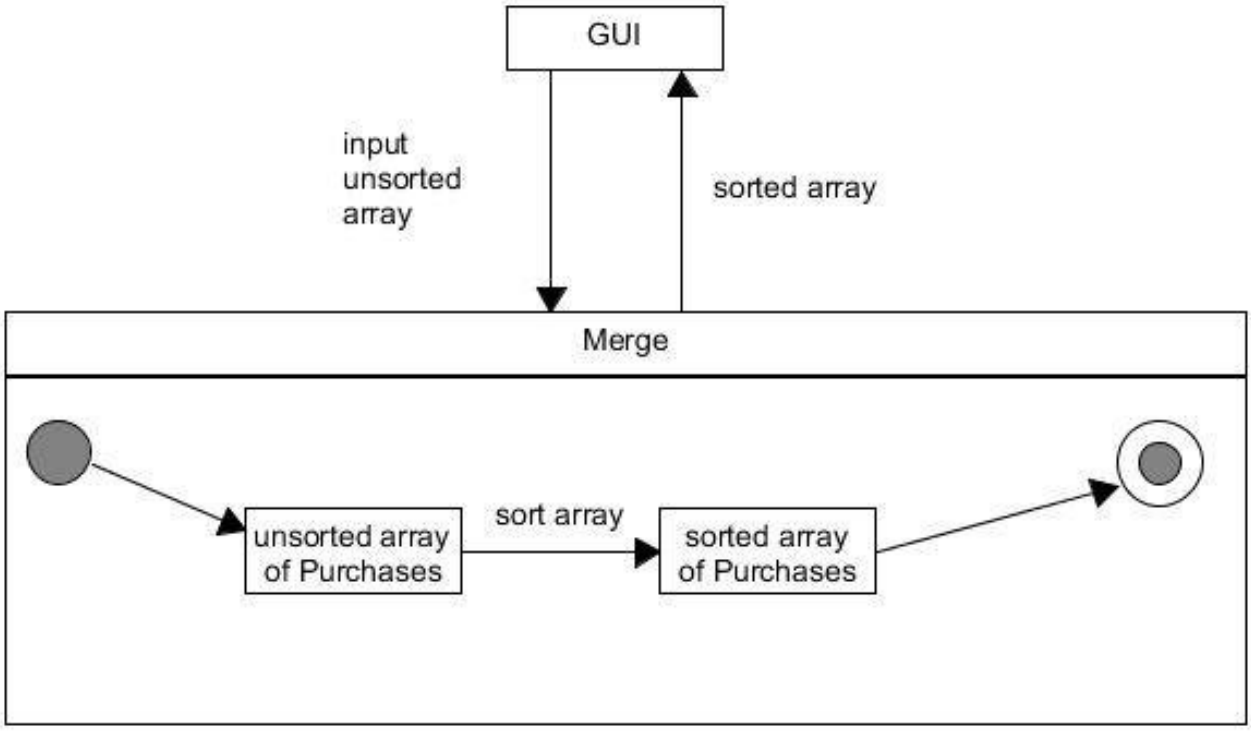
Uses Relationship



UML Class Diagram

UML State Machine Diagrams





Internal Review/Evaluation of Design

The implementation of the budget facilitator used CSV files to store the budget and purchase information because CSV's have enough space to hold the large amount of purchases that a person makes and it is easily manipulated. It also stored in the hard drive so as to keep the information for longer. Binary Search was also used for it's simplicity. The GUI uses the JavaFX library because of its large number of tools such as the graphs that compare the budgets. It also makes the GUI much more aesthetically appealing. MergeSort is our primary sorting algorithm because it works quickly and efficiently for large amounts of data. The code suffers because it could have been more modularized. The error checking is also lacking because the user can easily break the program with invalid inputs.