EzXpns

EzXpns Project Report v0.1

Group W14-2j

Andrew Koh A0097973H

Lee Shu Zhen A0085413J

Yan Ting Zhe A0087091B

Yao Yujian A0099621X

Table of Contents

Project Scope

Problem

There is a lack of expense management system catered to youths in the market.

This is due to two misconceptions.

1. Budgeting are for adults
2. Existing expense management system are suffice for young adults

Budgeting is a lifelong skill. The transition to tertiary education brings about many financial burden. Furthermore, young adults are inexperienced in financial management - they might never be if they don't start managing their expenses. Hence there is a need for a simple and educational expense manager in the market and we want to fill that demand.

Product Description

EzXpns is a standalone desktop expense manager with a simple interface. It is catered for **youths (16-25)** with poor financial planning skills and little motivation to start planning.

We believe less is more. As this product is catered for the general youth population, we have simplified the user interface and focused on features that are most relevant to them. This program has the basic features that allows user to record expense items and do some simple management of their budget and expenses. The user is able to see his/her finance status in one glance.

Many young adults or teenagers are still inexperienced in financial planning. In order to encourage daily use and inculcate good financial planning skills among the youths today, we have introduced gamification to the program and included educational elements to it. It allows them to learn the skills on the way as the software guide them along.

Vision

Our vision is to incorporate 3 values into our product.

Simplicity: Our software will be simple and easy to understand. The best interface is the one you don't notice, where the very first thing you try does exactly what you want it to do, and any accidental actions that prove destructive are easily reversed.

Intelligent: Our software will be intelligent enough to pick up trends on behalf of the user, even before the user himself notices it.

Educational: Our software will teach users on the basics of personal budgeting by giving advice as well as best practice.

EzXpns

Developer's Guide v0.1

1. Introduction

EzXpnz is a simple finance management utility for young adults.

For Version 0.1 we aim to make a working prototype that can demonstrate some basic functionalities, such as create, read, and delete records, basic search, basic report and basic target setting with alerts.

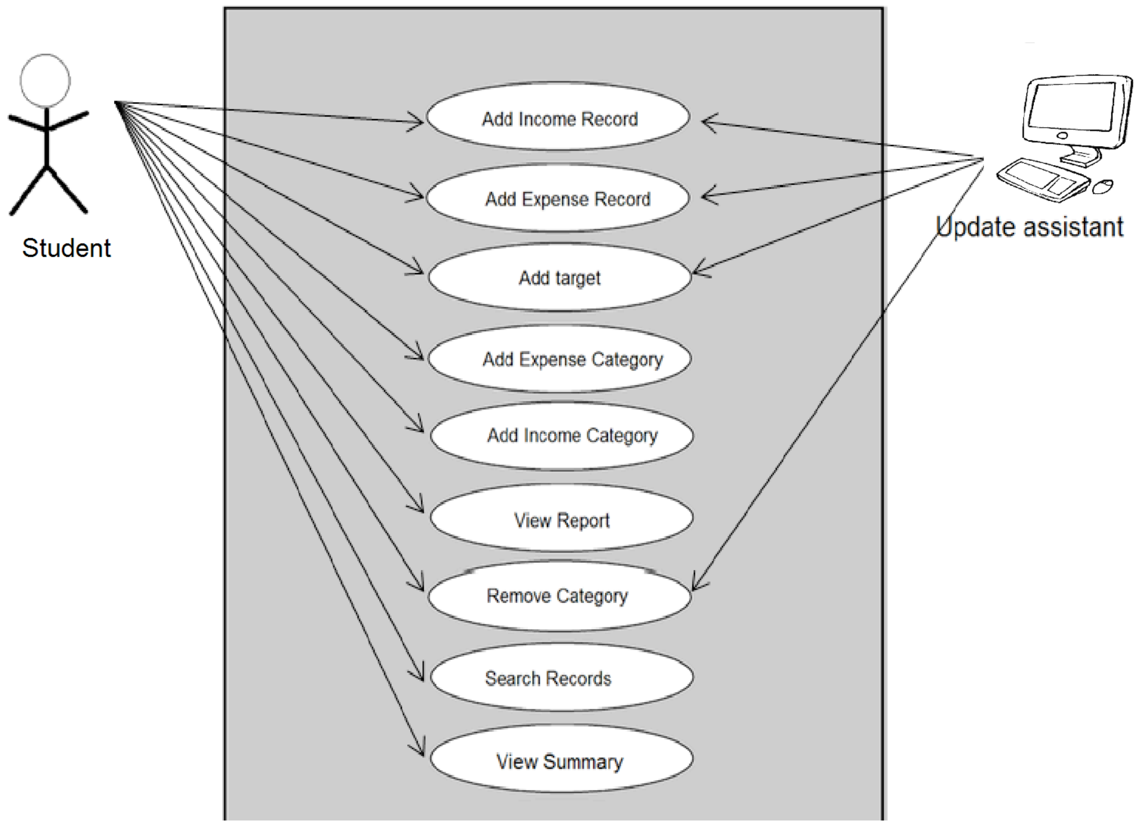
The purpose of this developer guide is to provide an incoming developer with information regarding the design and implementation of the program. This guide will help the developer understand the architecture of the program so that he can start coding as soon as possible

2. Development Infrastructure

The project is written on ***Java Platform Standard Edition (Java SE) 6***, using ***Eclipse*** as the main Integrated Development Environment (IDE). An Eclipse plugin ***WindowBuilder*** is used for the some GUI. The source code is currently hosted on Google Code at http://code.google.com/p/cs2103jan13-w14-2j/. The repository uses Mercurial as the version control system.

3. Use Cases

Use Case Diagram



Use Case: 01 - Add New Expense Record

Actor(s): User
Main Success Scenario (MSS):
1. User requests to add a new expense record
2. System displays new expense record form
3. User fills up the new expense record form
4. System validates the input (the required fields such as amount, date, category)
5. System display success of adding new expense record

Use Case: 02 - Add Income Record

Actor(s): User
MSS:
1. User requests to add a new income record
2. System displays new income record form
3. User fills up the new income record form
4. System validates the input (the required fields such as amount, date, category)
5. System display success of adding new income record

Alternate Flow: Users enters wrongly
4a.1 System displays problems with input
Return to MSS: 3

Use Case: 03 - Add Target

Actor(s): User
MSS:
1. User requests to add a new expense target
2. System display list of expense category
3. User select the category to set a target on
4. System displays form for the target
5. User fills in the required fields
6. System validates the input
7. System stores the target and displays success

End
Alternate Flow: User enters invalid input
4a.1. System displays problems with input
Returns to MSS: 4

Use Case: 04 - Add Expense Category

Actor(s): User
MSS:
1. User requests for Category Manager
2. System creates a window with available options
3. User selects “Add new Category” under expense
4. System creates a field for user to add a new expense category
5. User fills up the required details for adding a new expense category
6. System validates the details (eg. name)
7. System displays success

<u>Alternate Flow: User entered invalid name</u>
5a.1. System displays problems with input
Return to MSS: 4.

Use Case: 05 - Add Income Category

Actor(s): User
MSS:
1. User requests for Category Manager
2. System creates a window with available options
3. User selects “Income” tab
4. System changes to “Income” tab with available options
5. User selects “Add new category” under Income
6. System creates a field for user to add a new income category
7. User fills up the required details for adding a new expense category
8. System validates the details (eg. name)
9. System displays success

<u>Alternate Flow: User entered invalid name</u>
7a.1. System displays problems with input
Return to MSS: 6.

Use Case: 06 - View Report

Actor(s): User
MSS:
1. User requests to view report
2. System displays a form to request for the timeline for the report
3. Users keys in start date and end date
4. System retrieves the records based on the timeframe provided by the User
5. System formats and updates the display of information on the screen

Alternate Flow: User keys in illegal characters (eg. alphabets)
3a1: System clears input and return to MSS at step 2

Use Case: 07 - Remove Category

1. User requests to remove Category
2. System display a list of category
3. User selects a category to be removed
4. System display warns user that all records under this category will be classified under 'un-defined'
5. User accepts condition
6. System displays success

End
Alternate Flow: User rejects condition
5a1: Category is not deleted
Returns to MSS: 2

Use Case: 08- Search Record

Actor(s): User
MSS:
1. User requests to search past records
2. System displays Search window
3. User fills up the relevant fields for the search
4. System retrieves from data storage the matching searches
5. System displays the search results

End

Use Case 09 - View Summary

Actor(s): User
MSS:
1. User requests to view summary
2. System show monthly summary by default and available options
3.User views summary

End
Alternate Flow: User requests to view annual summary
2a1: System show annual summary
Returns to MSS: 3
Alternate Flow: User requests to view daily summary
2b1: System show daily summary
Returns to MSS: 3

4. Architecture

The organisation of each major component is shown in Figure 1.

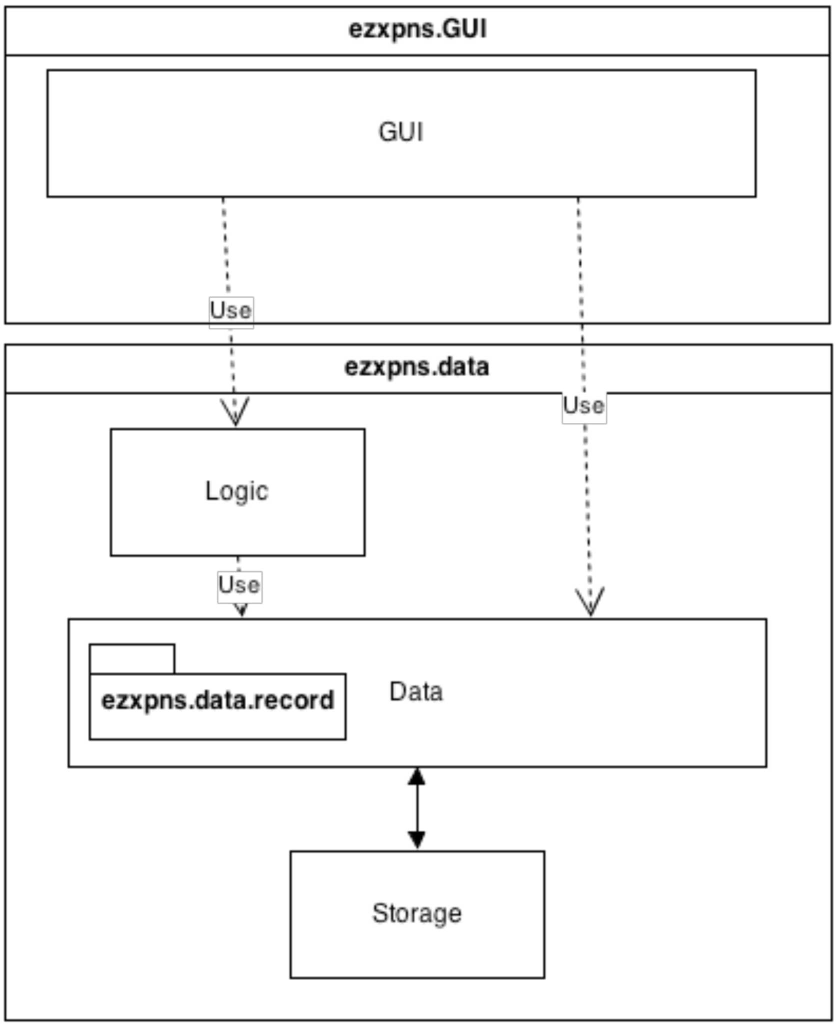
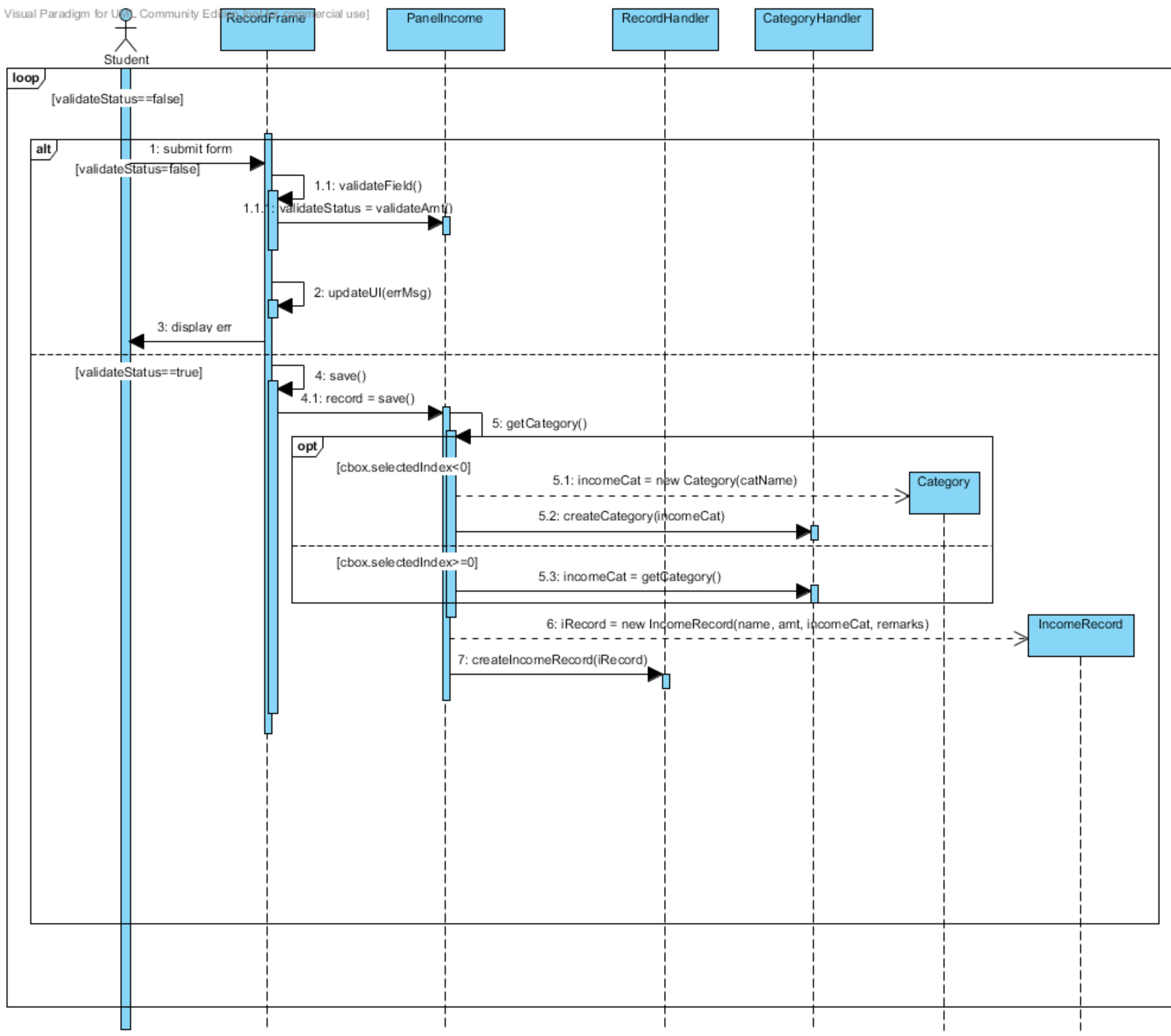


Figure 1. Architecture Diagram

The design of the project is such that each component can function with minimum dependency on others, and that replacing any of them will not have major effect on the rest. As such, components generally communicate with each other via interfaces. The functions of computation and data processing are also strictly separated from user interactions, which allows easier update to the backend without affecting the frontend.

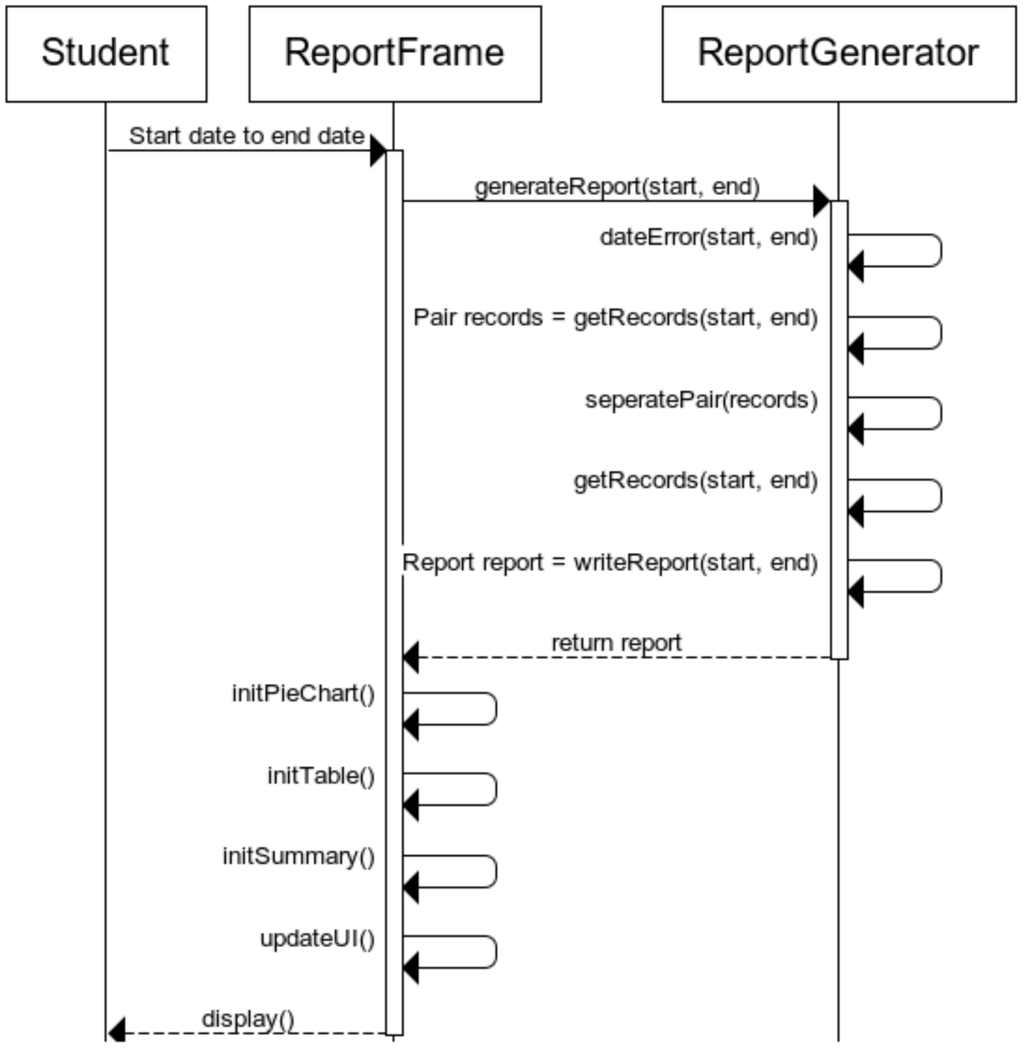
For logic classes that needs some data access, it is recommended to define a `DataProvider interface` nested in the class and require it in the constructor. This interface can then be implemented by either the data manager or its components directly, or by the `Ezxpns` class if it has side effects on other objects. It is also possible to extract methods that implement `DataProvider` out of EzXpns and inject into anonymous classes for better code organization.

Below is the **sequence diagram** of some examples of tasks:

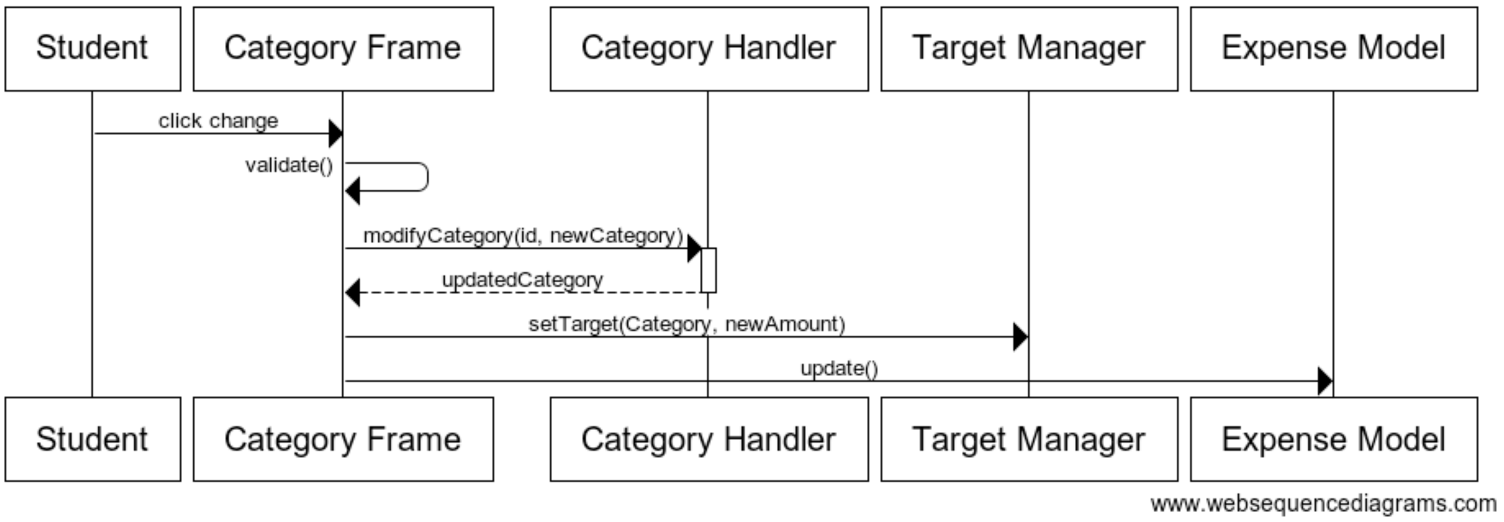


Adding a new record

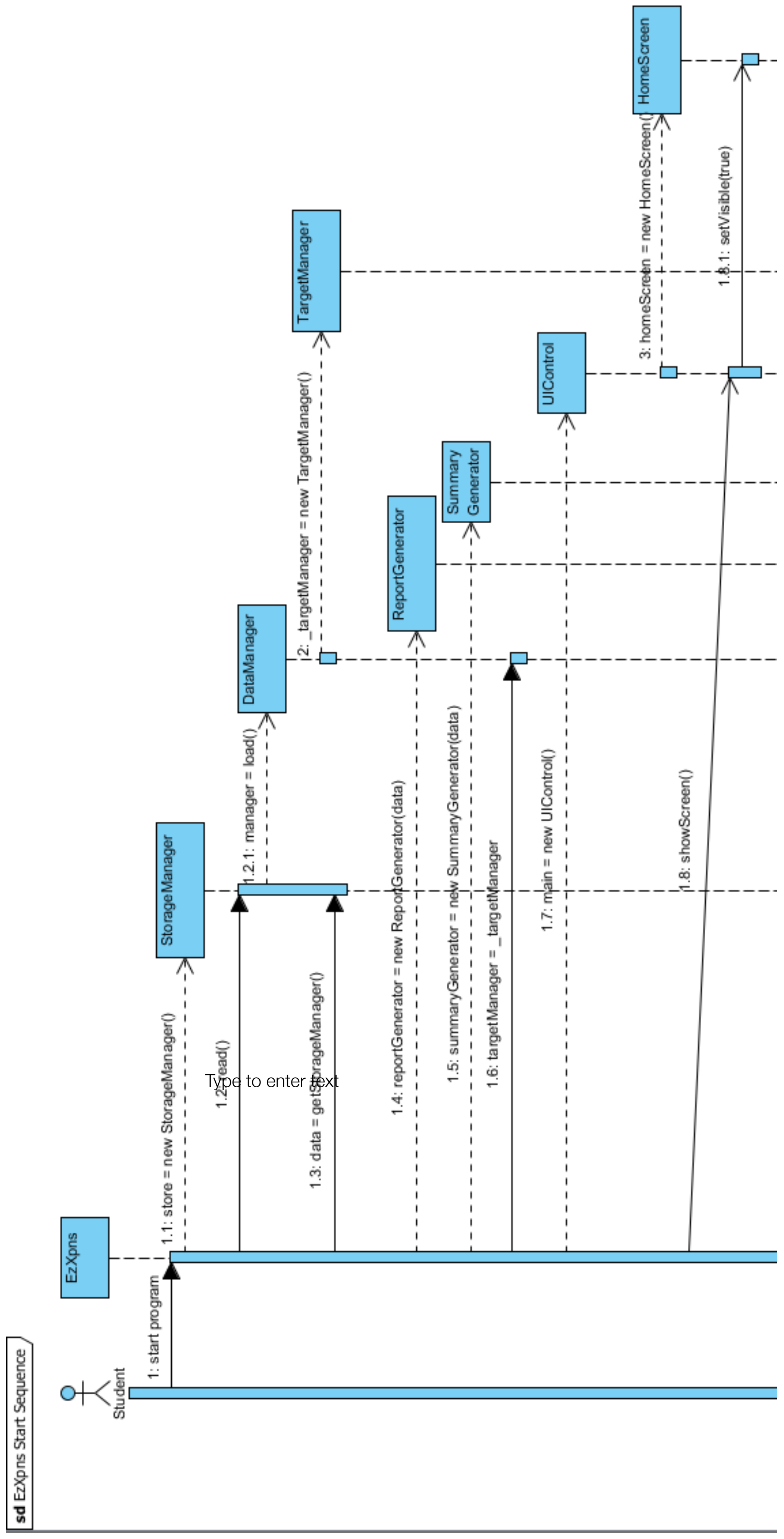
Generating a report



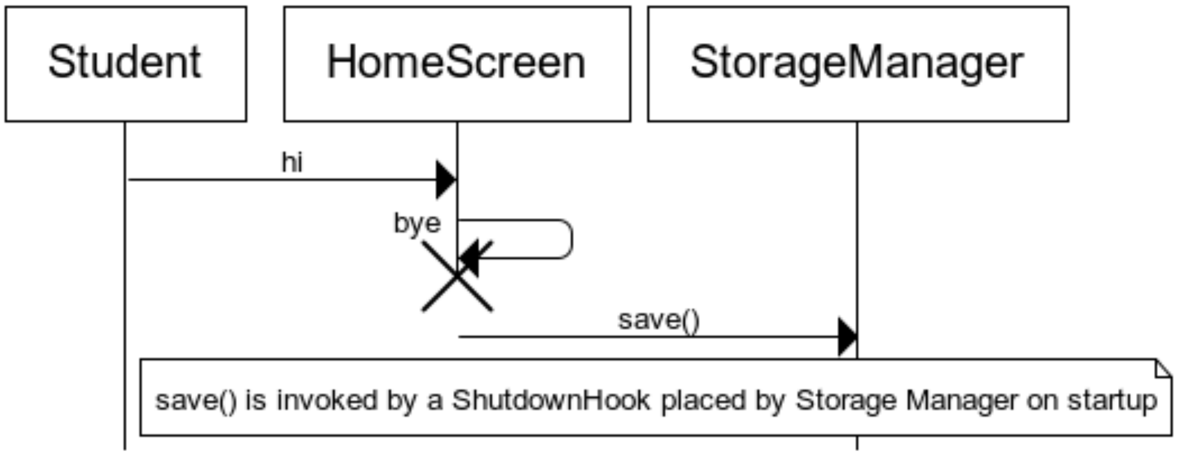
Modify Expense Category



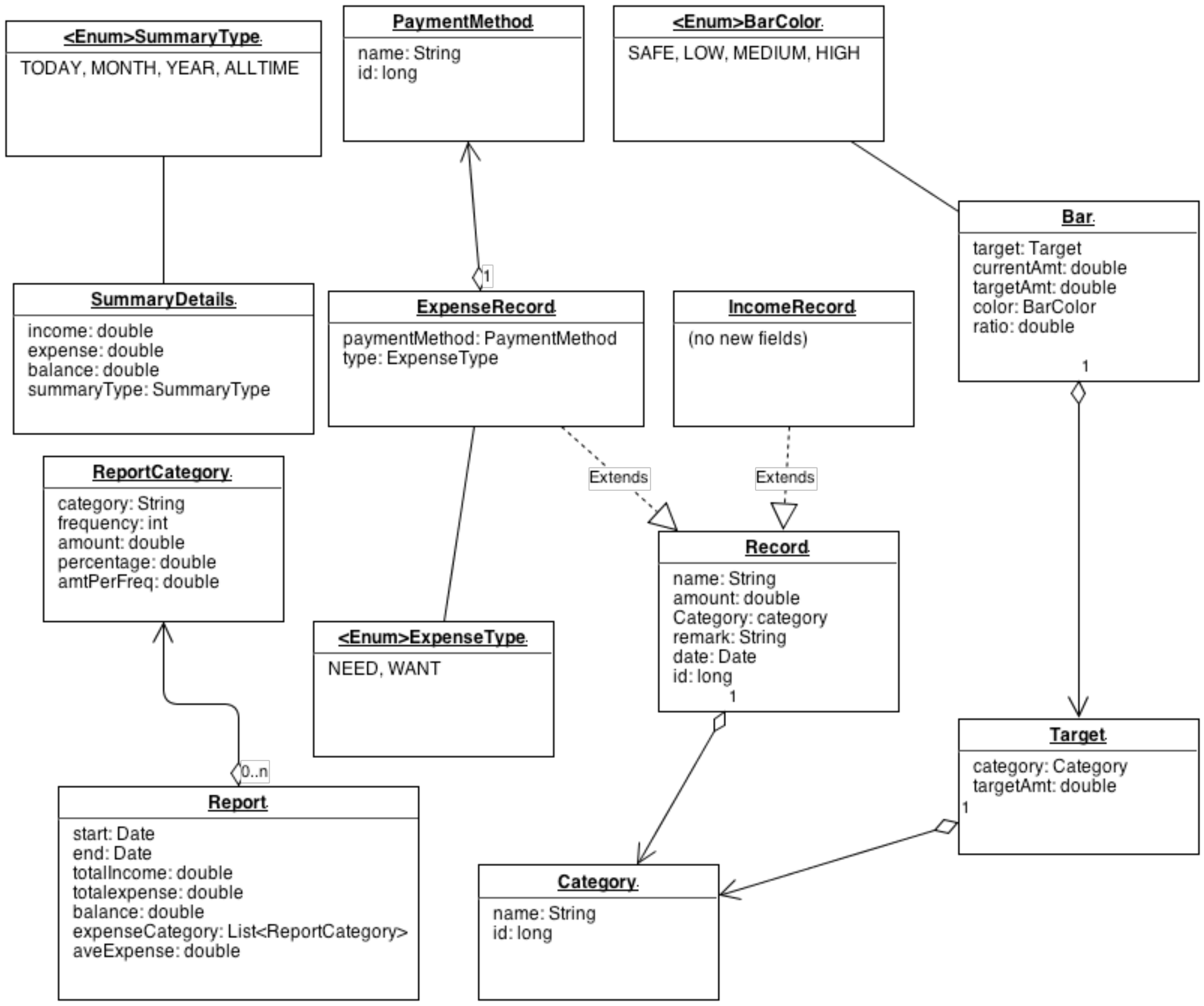
EzXpns Start Sequence


sd EzXpns Start Sequence
Student
EzXpns
StorageManager
DataManager
TargetManager
ReportGenerator
Summary Generator
UIControl
HomeScreen
1: start program
1.1: store = new StorageManager()
1.2: read()
Type to enter text
1.2.1: manager = load()
2: _targetManager = new TargetManager()
1.3: data = getStorageManager()
1.4: reportGenerator = new ReportGenerator(data)
1.5: summaryGenerator = new SummaryGenerator(data)
1.6: targetManager = _targetManager
1.7: main = new UIControl()
1.8: showScreen()
3: homeScreen = new HomeScreen()
1.8.1: setVisible(true)

EzXpns Exit Sequence

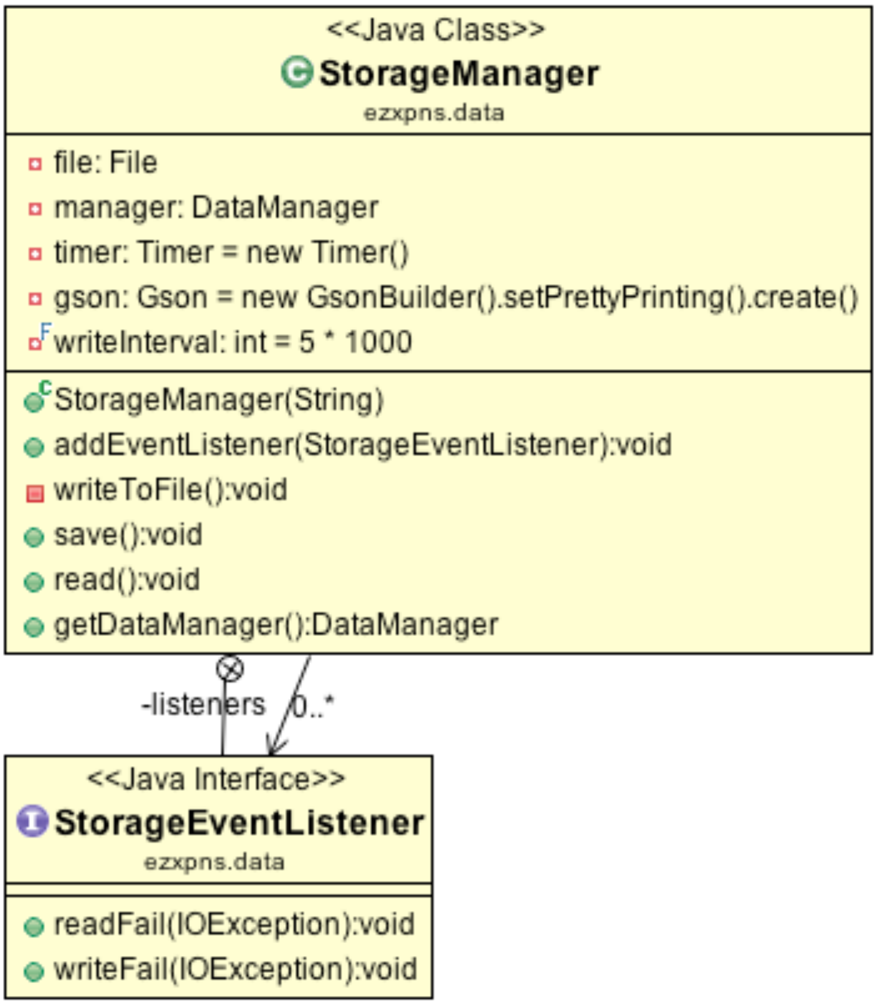

Student
HomeScreen
StorageManager
hi
bye
save()
save() is invoked by a ShutdownHook placed by Storage Manager on startup

The **domain level** is shown below:



Domain level diagram

4.1 Storage


<<Java Class>>
StorageManager
ezxpns.data
file: File
manager: DataManager
timer: Timer = new Timer()
gson: Gson = new GsonBuilder().setPrettyPrinting().create()
writeInterval: int = 5 * 1000
StorageManager(String)
addEventListener(StorageEventListener):void
writeToFile():void
save():void
read():void
getDataManager():DataManager
-listeners
0..*
<<Java Interface>>
StorageEventListener
ezxpns.data
readFail(IOException):void
writeFail(IOException):void

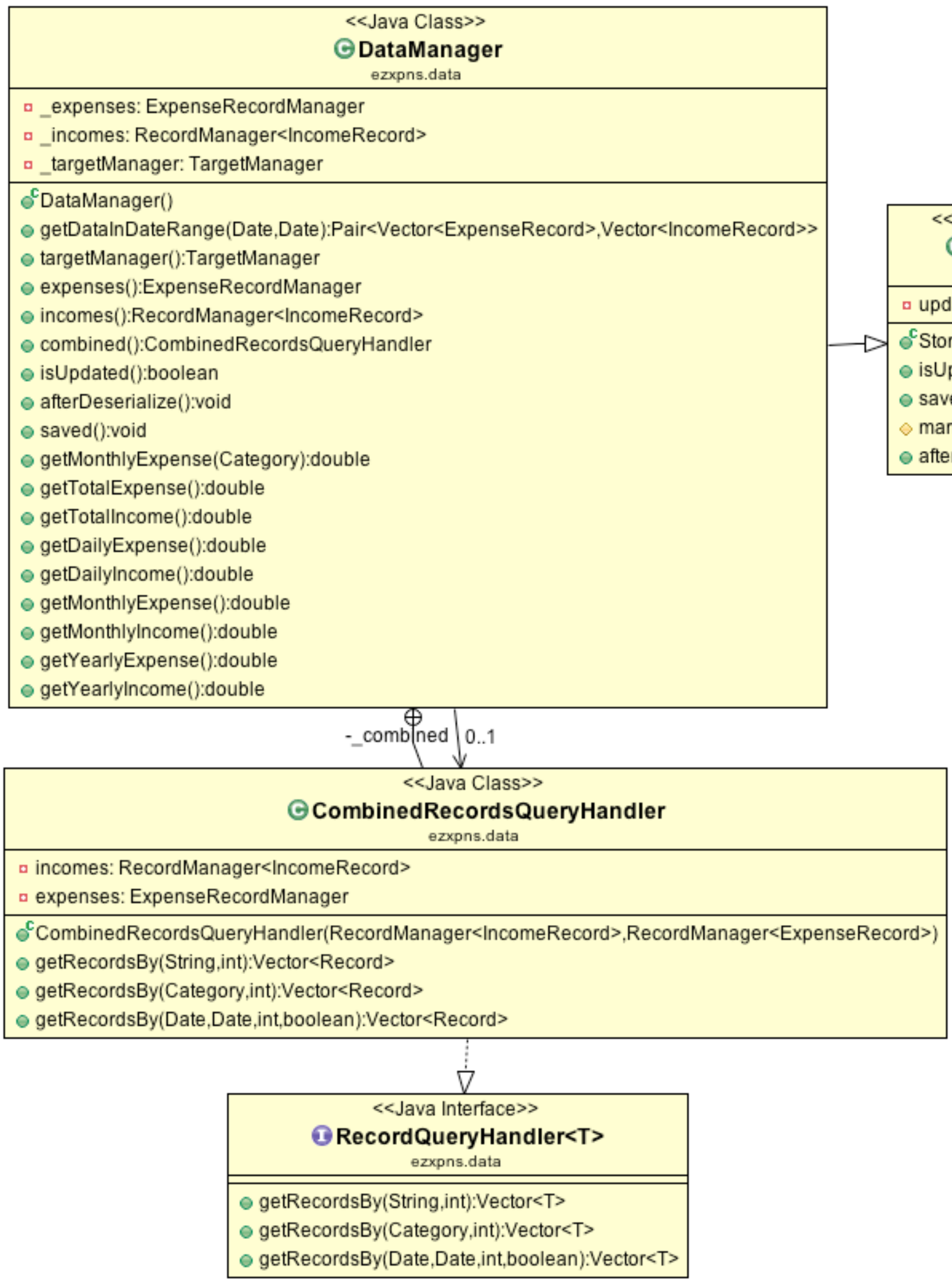
The data is stored in Json format, with gson from Google. All data that are persistent should be in `DataManager`. To work with gson, attributes of members or sub-members of this class that should not be stored and should be marked as `transient`. For example, `private transient CombinedRecordsQueryHandler _combined`. This variable handles query of both expense and income. Since it contains no data of its own, and therefore not persistent, it will be marked `transient`.

Each object that is to be stored should also inherit `Storable`, which allows the `Storage-Manager` to check if data is updated. Whenever a `Storable` changes its internal data, it should mark itself as updated, so that `StorageManager` will save the data.

The storage is also fully automatic. It checks whether the data is updated at each interval (5 seconds currently for testing, likely longer in the future). If there is an update, it serializes the whole internal data, saves it, and marks all data as updated. The same checking is also done when the application quits. So the developer do not need to handle the storage once it is set up.

4.2 Data Management

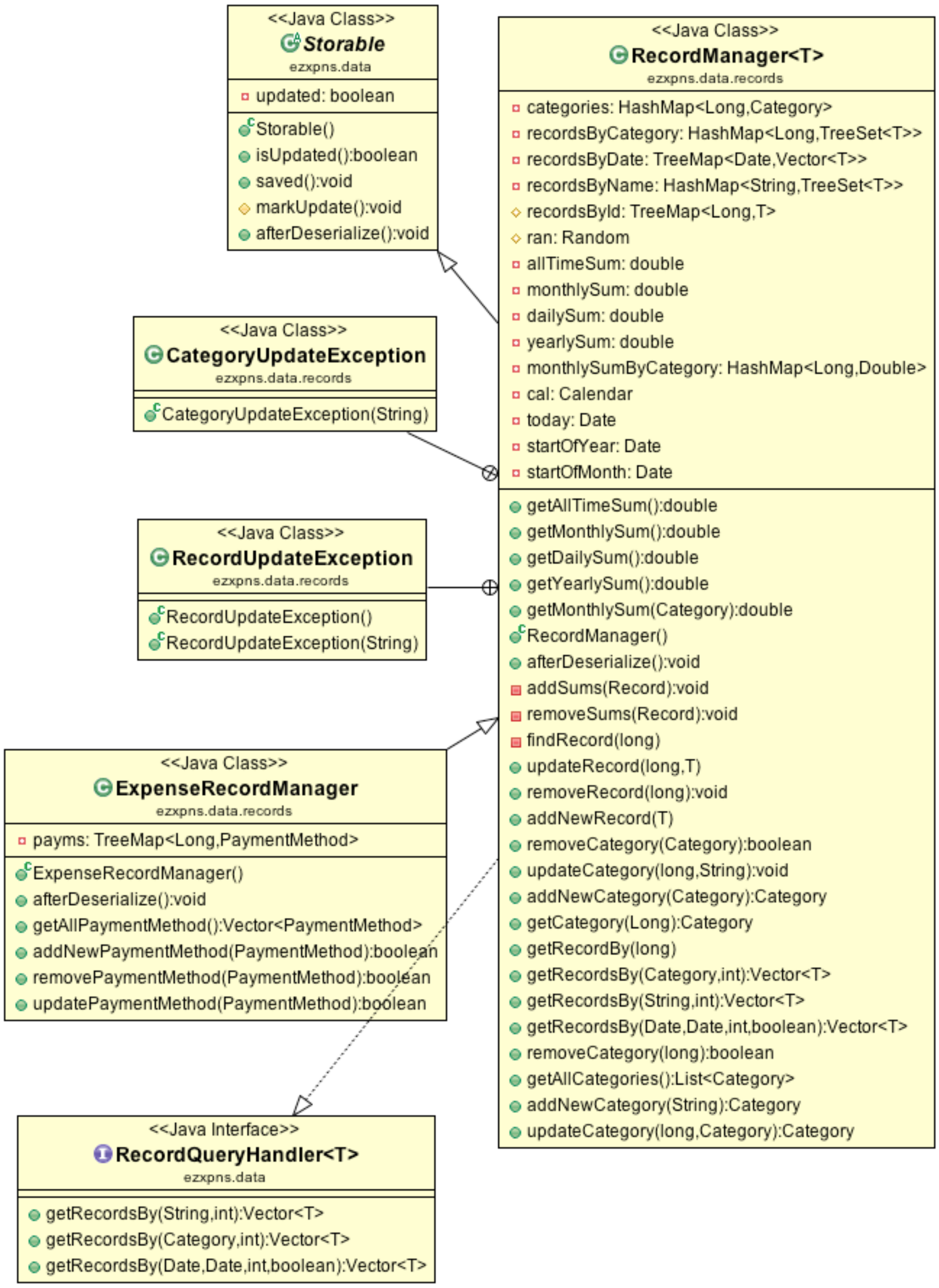
As mentioned, all data will be handled by `DataManager`. Currently, it contains two `RecordManager` (one for income and one for expense) and a `TargetManager`. Below is the `DataManager` class.



<<Java Class>>
DataManager
ezxpns.data
_expenses: ExpenseRecordManager
_incomes: RecordManager<IncomeRecord>
_targetManager: TargetManager
DataManager()
getDataInDateRange(Date,Date):Pair<Vector<ExpenseRecord>,Vector<IncomeRecord>>
targetManager():TargetManager
expenses():ExpenseRecordManager
incomes():RecordManager<IncomeRecord>
combined():CombinedRecordsQueryHandler
isUpdated():boolean
afterDeserialize():void
saved():void
getMonthlyExpense(Category):double
getTotalExpense():double
getTotalIncome():double
getDailyExpense():double
getDailyIncome():double
getMonthlyExpense():double
getMonthlyIncome():double
getYearlyExpense():double
getYearlyIncome():double
<<Java Class>>
Storable
ezxpns.data
updated: boolean
Storable()
isUpdated():boolean
saved():void
markUpdate():void
afterDeserialize():void
-_combined
0..1
<<Java Class>>
CombinedRecordsQueryHandler
ezxpns.data
incomes: RecordManager<IncomeRecord>
expenses: ExpenseRecordManager
CombinedRecordsQueryHandler(RecordManager<IncomeRecord>,RecordManager<ExpenseRecord>)
getRecordsBy(String,int):Vector<Record>
getRecordsBy(Category,int):Vector<Record>
getRecordsBy(Date,Date,int,boolean):Vector<Record>
<<Java Interface>>
RecordQueryHandler<T>
ezxpns.data
getRecordsBy(String,int):Vector<T>
getRecordsBy(Category,int):Vector<T>
getRecordsBy(Date,Date,int,boolean):Vector<T>

DataManager Class Diagram

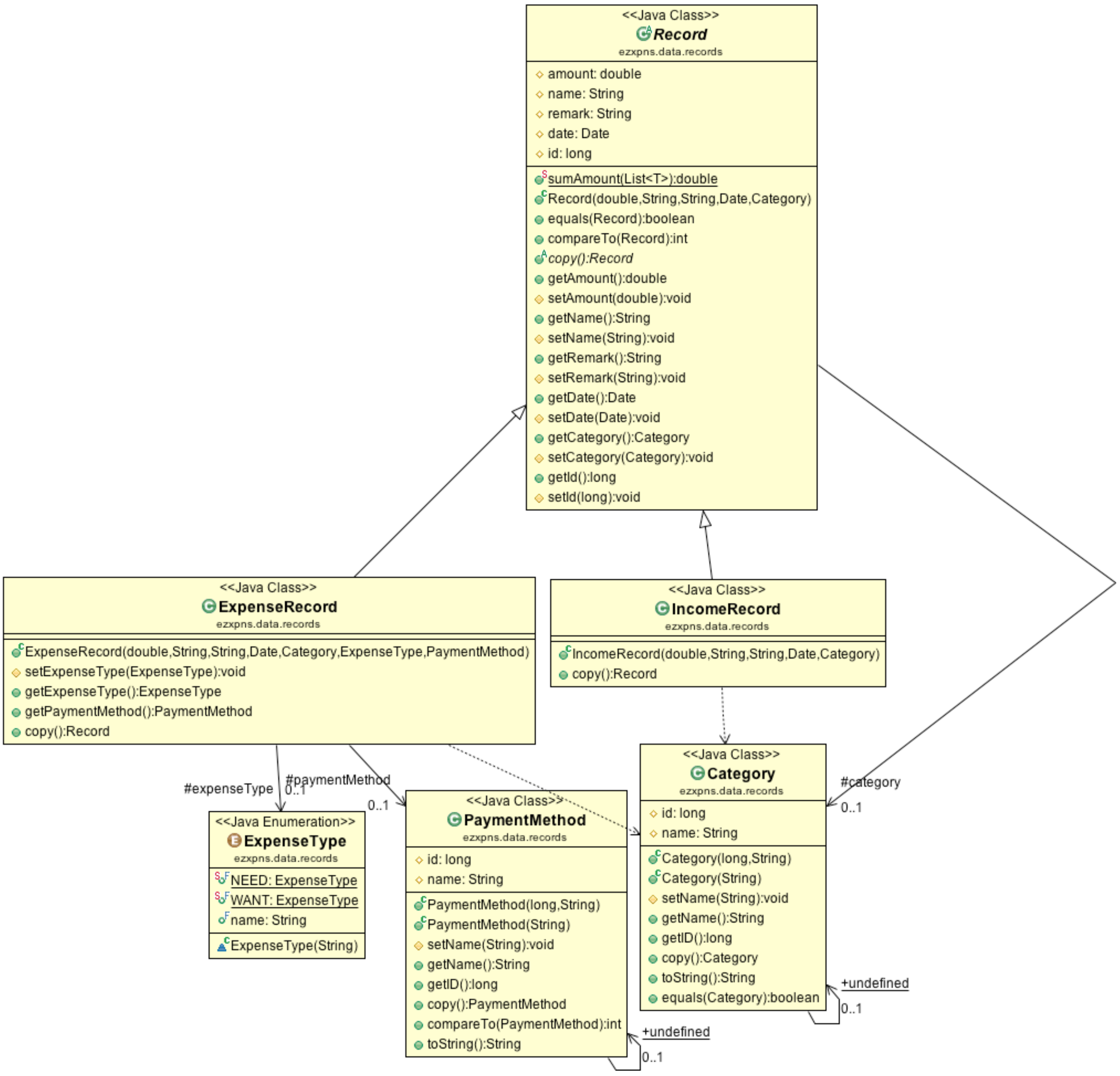
Below are the details for `RecordManager` classes.



<<Java Class>>
Storable
ezxpns.data
updated: boolean
Storable()
isUpdated():boolean
saved():void
markUpdate():void
afterDeserialize():void
<<Java Class>>
RecordManager<T>
ezxpns.data.records
categories: HashMap<Long,Category>
recordsByCategory: HashMap<Long,TreeSet<T>>
recordsByDate: TreeMap<Date,Vector<T>>
recordsByName: HashMap<String,TreeSet<T>>
recordsById: TreeMap<Long,T>
ran: Random
allTimeSum: double
monthlySum: double
dailySum: double
yearlySum: double
monthlySumByCategory: HashMap<Long,Double>
cal: Calendar
today: Date
startOfYear: Date
startOfMonth: Date
getAllTimeSum():double
getMonthlySum():double
getDailySum():double
getYearlySum():double
getMonthlySum(Category):double
RecordManager()
afterDeserialize():void
addSums(Record):void
removeSums(Record):void
findRecord(long)
updateRecord(long,T)
removeRecord(long):void
addNewRecord(T)
removeCategory(Category):boolean
updateCategory(long,String):void
addNewCategory(Category):Category
getCategory(Long):Category
getRecordBy(long)
getRecordsBy(Category,int):Vector<T>
getRecordsBy(String,int):Vector<T>
getRecordsBy(Date,Date,int,boolean):Vector<T>
removeCategory(long):boolean
getAllCategories():List<Category>
addNewCategory(String):Category
updateCategory(long,Category):Category
<<Java Class>>
CategoryUpdateException
ezxpns.data.records
CategoryUpdateException(String)
<<Java Class>>
RecordUpdateException
ezxpns.data.records
RecordUpdateException()
RecordUpdateException(String)
<<Java Class>>
ExpenseRecordManager
ezxpns.data.records
payms: TreeMap<Long,PaymentMethod>
ExpenseRecordManager()
afterDeserialize():void
getAllPaymentMethod():Vector<PaymentMethod>
addNewPaymentMethod(PaymentMethod):boolean
removePaymentMethod(PaymentMethod):boolean
updatePaymentMethod(PaymentMethod):boolean
<<Java Interface>>
RecordQueryHandler<T>
ezxpns.data
getRecordsBy(String,int):Vector<T>
getRecordsBy(Category,int):Vector<T>
getRecordsBy(Date,Date,int,boolean):Vector<T>

RecordManager Class Diagram

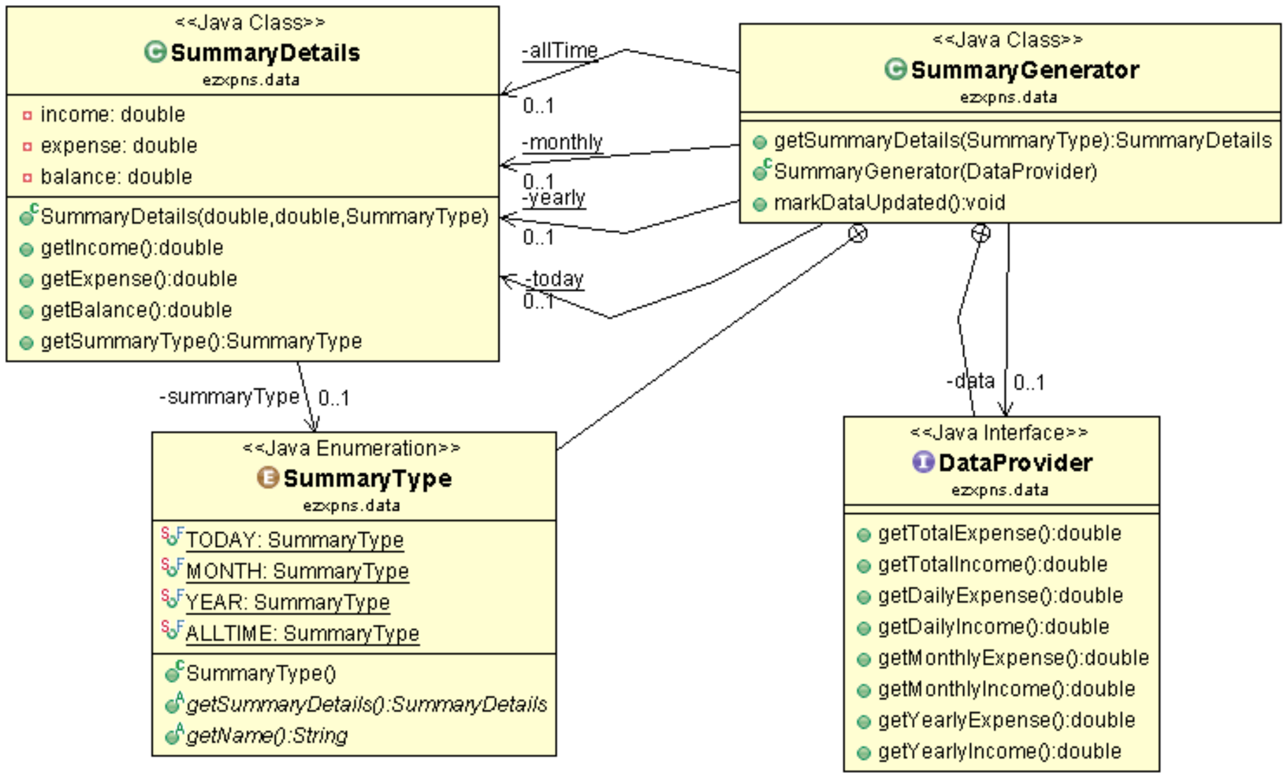
Below are the atomic data units, note that all data units - `Record`, `Category` and `PaymentMethod` are immutable outside `ezxpns.data.record` package.



Atomic Data Unit Class Diagram

The immutability allows us to optimize queries in `RecordManager` without circular references, or copying of object instances for output. Moreover, when methods like `createRecord(Record r)` are called, the record that got created and inserted into the data manager is not guaranteed to be the same instance as the parameter, it may contain different id, and can be a copy of the original object, though the other attributes should be the same.

4.3 Summary System



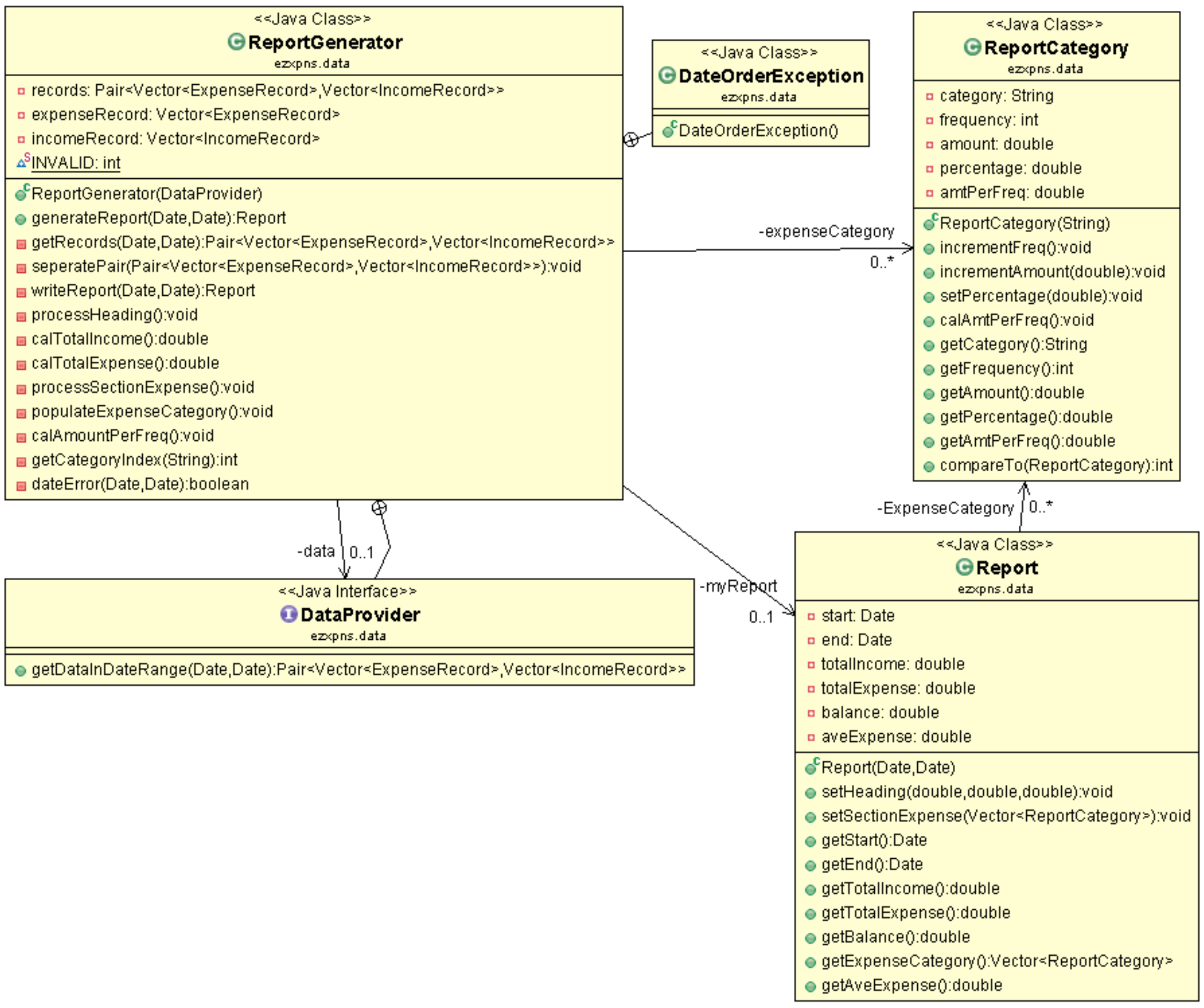
Summary System Class Diagram

The Summary System will retrieve and store information to be displayed in the `OverviewPanel` in the main window.

The main class for the Summary System is the `SummaryGenerator`. `SummaryGenerator` will create 4 `SummaryDetails`, a data structure containing overall financial information over a specific time frame. This is done by calling `getSummaryDetails(SummaryType)`. There are 4 time frame, expressed as `enum SummaryType`. There will be 1 `SummaryDetails` for each SummaryType. `SummaryGenerator` will retrieve relevant informations for the 4 `SummaryType` through the `DataProvider` interface.

`markDataUpdate()` (in `SummaryGenerator`) should be called when you want to refresh the information (eg. when there is a change in the records).

4.4 Report System



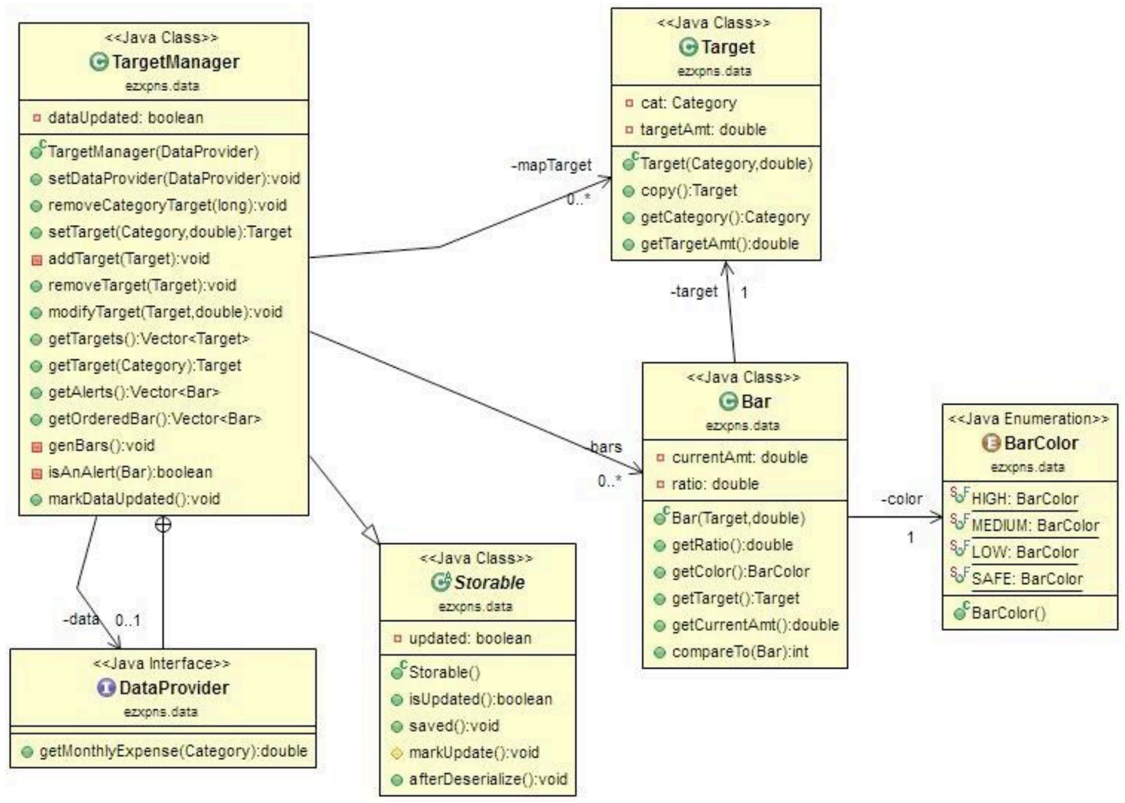
Report System Class Diagram

The Report System will retrieve and store information to be displayed for `ReportFrame`.

The main class for the Report system is `ReportGenerator`. `ReportGenerator` will generate a `Report`, a data structure containing information regarding user between a specific `start` date and `end` date. This is done through `generateReport(Date start, Date end)`. `ReportGenerator` will get the records within the timeframe through `DataProvider`. `ReportGenerator` will then process the records and store them in a `Report`.

`ReportCategory` is another data structure representing the rows in an expense category table. Similarly, it is created by `ReportGenerator` and stored in `Report`.

4.5 Target System (with alerts)



Target System Class Diagram

The main class for Target system is the `TargetManager`. `TargetManager` handles the creation, removal and modification of a `Target`. It is also used to retrieve information to be displayed in the `TargetOverviewPanel`.

`TargetManager` requires a `DataProvider` interface in which it will retrieve the total monthly expense for a particular `Category`. It also implements `Storable` since the `Tar-get` are stored in the database, and the storage manager needs to check if the targets are changed.

The `Target` is a data structure that represents the target that the user set for a particular expense `Category`. It contains a reference to the expense `Category` and the targeted amount of money (`double`).

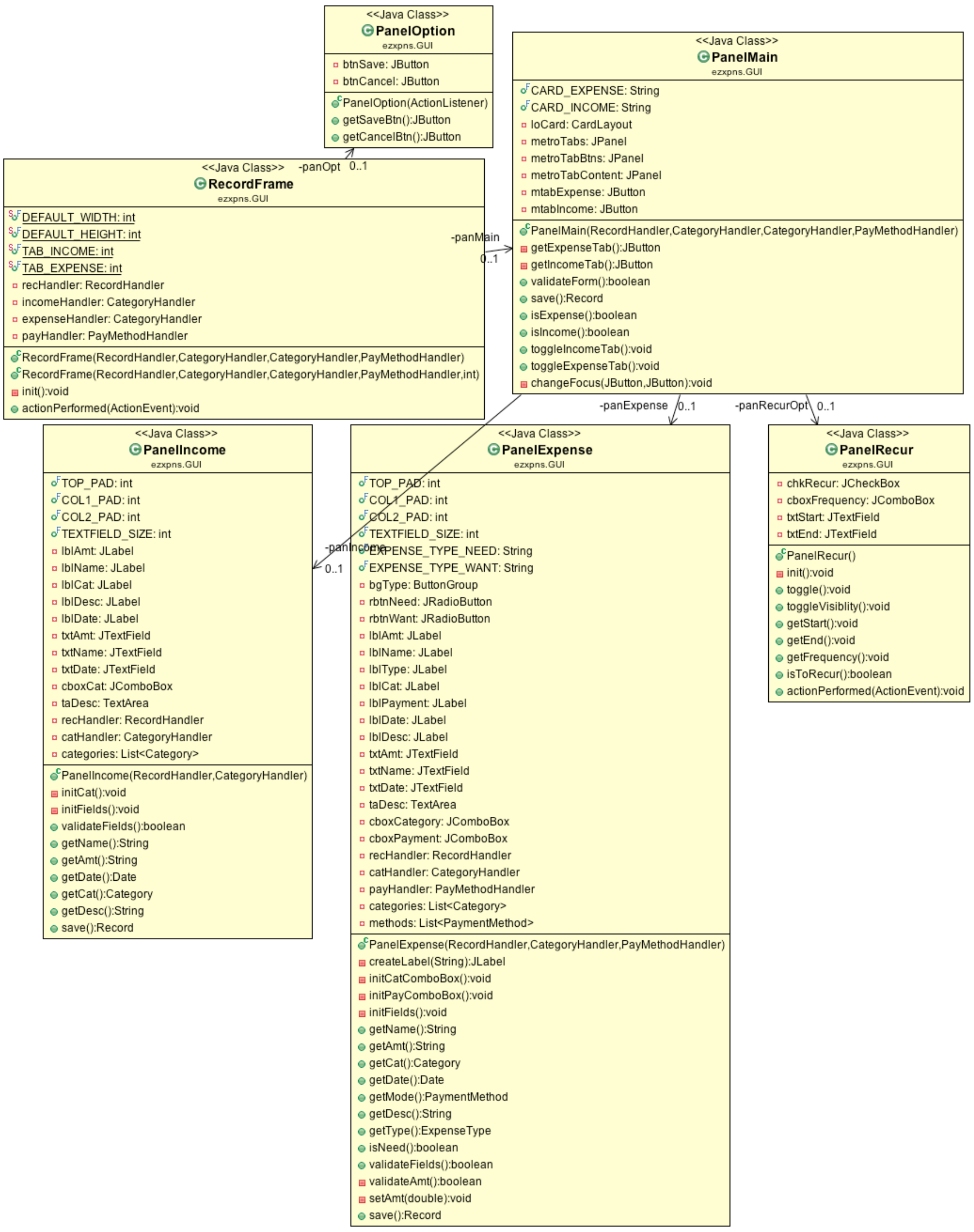
The `Bar` is a data structure used to display the user's target progress in the `TargetO-verviewPanel` of the GUI. A `Bar` has a `BarColor` enum. It is defined as either *`HIGH`*, *`MEDIUM`*, *`LOW`* or *`SAFE`*. Each of this enum is assigned an actual color in the `TargetOver-viewPanel` of the GUI so that the `Bar` is color-coded when displayed. (It is named "Bar" because the progress will be displayed as a bar chart in later iterations)

`TargetManager` also manages the alerts. The `isAnAlert(Bar)` classifies a `Bar` as an alert if its ratio of total monthly expense (`currentAmt`) to target amount is considered

`HIGH` or `MEDIUM`. `TargetManager` is able to generate a Vector of Bar that are classified as alerts. The number of alerts is displayed in the `TargetOverviewPanel` of the GUI.

Only the `Target` object is stored in the database. Alerts and `Bar` are generated at run-time.

4.6 Graphical User Interface (GUI)



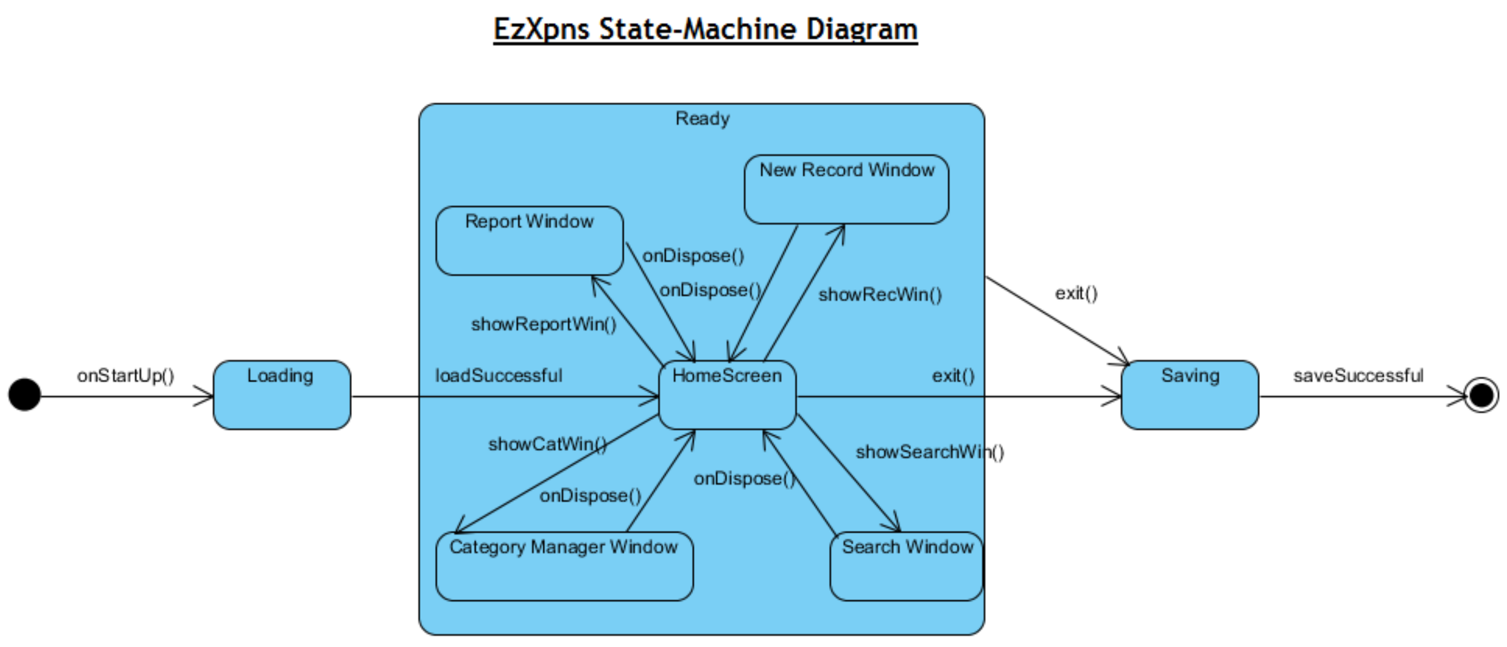
GUI Class Diagram

The current GUI is designed to provide a simple single streamlined modular window interface - because multiple windows are harder to manage, easy to miss and at times, distracting and messy. For that reason, all the GUI components are controlled by the class `UIControl`, where `UIControl` has the ability to manipulate all the windows, such as when they appear, as well as disabling or removing it from the user's view. Therefore, `UIControl` handles window events (such as closing of window) and related areas of functionality. The only exception is the `ShutDownHook` that will be placed by the `StorageManager` on startup; That is to handle unexpected exits, if any.

The GUI components uses java libraries such as `javax.swing` for the visual components and `java.awt` for the event handling component. However, at the current iteration as the priority was placed on functionality over visual aesthetics, the GUI for v0.1 does not have the full functionality of event handlers (such as mouse hovering, resizing window, etc) and visual aids (tooltips, error messages, etc). More event handlers will be implemented in the next iteration.

To enforce a homogeneous look and feel, it is recommended to use a generic method or class to create GUI components such as buttons and textfields. An example of such would be in `RecordFrame`, `createLabel(String lblText)` is used to create labels with the same font.

Below is the State-Machine Diagram for *EzXpns*.



5. Testing Framework

JUnit is used for all unit testing. All testing code is under the package `ezxpns.test`, with self explanatory names. Currently only test cases for data management is implemented. It is recommended that the developer should write additional tests for any new features he creates.

6. API

6.1 Package ezxpns

Class EzXpns

Method Summary	
Category	addNewCategory(Category newCat) Create a new category, note that the new category will be a copied with perhaps different id
Category	addNewCategory(java.lang.String catName) Create a new category with the name
boolean	createRecord(ExpenseRecord newRecord) Create a new expense record
boolean	createRecord(IncomeRecord newRecord) Create a new income record
java.util.List<Category>	getAllCategories() Get all user defined categories
DataManager	getDataMng()
Record	getRecord(long identifier) Retrieve a specific record based on the identifier given
java.util.List<Record>	getRecords(int n) Get some records stored
StorageManager	getStore()
TargetManager	getTargetManager()
ExpenseRecord	lastExpenseRecord(java.lang.String name) Return the latest expense record matching the name, or null
IncomeRecord	lastIncomeRecord(java.lang.String name) Return the latest expense record matching the name, or null
boolean	modifyRecord(long id, ExpenseRecord r) Modify an expense record
boolean	modifyRecord(long id, IncomeRecord r) Modify an income record

boolean	**removeCategory**(long identifier) Remove a user defined category based on the given identifier
boolean	**removeRecord**(long identifier) Remove record based on an identifier
java.util.Vector<Record>	**search**(SearchRequest req)
Category	**updateCategory**(long identifier, Category selectedCat) Modify Category Method

Class Main

Method Summary	
static void	**main**(java.lang.String[] args)

6.2 Package ezxpns.data

Class Bar

Method Summary	
int	**compareTo**(Bar other) compareTo() will return an integer value that represents the comparison between two Bar objects -1 will represent that this object is lesser compared to the other object 0 will represent that both objects have equal values 1 will represent that this object is larger compared to the other object
BarColor	**getColor**() This returns the enum of the BarColor
double	**getCurrentAmt**() This returns the current amount of monthly expenses for the target of this bar
double	**getRatio**() This returns the ratio of currentAmt/targetAmt
Target	**getTarget**() This returns the Target of this bar

double	**getTargetAmt**() This returns the target amount that the user has set for the target of this bar

Enum BarColor

Enum Constant Summary
HIGH
LOW
MEDIUM
SAFE

Method Summary	
static BarColor	**valueOf**(java.lang.String name) Returns the enum constant of this type with the specified name.
static BarColor[]	**values**() Returns an array containing the constants of this enum type, in the order they are declared.

Class DataManager.CombinedRecordsQueryHandler

Method Summary	
java.util.Vector<Record>	**getRecordsBy**(Category category, int max) Search for records matching the category
java.util.Vector<Record>	**getRecordsBy**(java.util.Date start, java.util.Date end, int max, boolean reverse) Search for records in the date range, inclusive of both ends
java.util.Vector<Record>	**getRecordsBy**(java.lang.String name, int max) Search for records matching the name
java.util.Vector<Record>	**getRecordsByCategory**(java.lang.String name)

Class DataManager

Method Summary	
void	**afterDeserialize**() Optional method to populate transient attributes after deserializing data from json.
DataManager.CombinedRecordsQueryHandler	**combined**()
ExpenseRecordManager	**expenses**()
double	**getDailyExpense**()
double	**getDailyIncome**()
Pair<java.util.Vector<ExpenseRecord>,java.util.Vector<IncomeRecord>>	**getDataInDateRange**(java.util.Date start, java.util.Date end) Returns a Pair of ExpenseRecord and IncomeRecord vectors within the start date and end date
double	**getMonthlyExpense**()
double	**getMonthlyExpense**(Category cat)
double	**getMonthlyIncome**()
double	**getTotalExpense**()
double	**getTotalIncome**()
double	**getYearlyExpense**()
double	**getYearlyIncome**()
RecordManager<IncomeRecord>	**incomes**()
boolean	**isUpdated**()
void	**saved**() Tells the object that it has been stored
TargetManager	**targetManager**()

Interface RecordQueryHandler<T extends Record>

Method Summary	
java.util.Vector<T>	getRecordsBy(Category category, int max) Search for records matching the category
java.util.Vector<T>	getRecordsBy(java.util.Date start, java.util.Date end, int max, boolean reverse) Search for records in the date range, inclusive of both ends
java.util.Vector<T>	getRecordsBy(java.lang.String name, int max) Search for records matching the name
java.util.Vector<T>	getRecordsByCategory(java.lang.String name)

Class Report

Method Summary	
double	getAveExpense() Get the average expense for the period between start date and end date
double	getBalance() Get balance for the period between start date and end date
java.util.Date	getEnd() Get end date
java.util.Vector<ReportCategory>	getExpenseCategory() Get a vector of ReportCategory
java.util.Date	getStart() Get start date
double	getTotalExpense() Get total expense for the period between start date and end date
double	getTotalIncome() Get total income for the period between start date and end date
void	setHeading(double income, double expense, double balance) Set 'Heading' of the report, namely: total income, total expense, and balance.

void	`setSectionExpense(java.util.Vector<ReportCategory> expenseCategory)` Set 'Expense' of the report, namely: Expense table, and average expense per day

Class ReportCategory

Method Summary	
void	`calAmtPerFreq()` Calculate the amount per frequency ratio
int	`compareTo(ReportCategory anotherCategory)` Comparable, for sorting Vector in descending order according to percentage
double	`getAmount()` Get the total expense for this category
double	`getAmtPerFreq()` Get the amount per frequency ratio for this category
java.lang.String	`getCategory()` Get the name of this category
int	`getFrequency()` Get the requency of this category
double	`getPercentage()` Get the percentage of expense of this category
void	`incrementAmount(double amount)` Increment amount
void	`incrementFreq()` Increment frequency by 1
void	`setPercentage(double percentage)` Set percentage for expense of this category

Class ReportGenerator

Method Summary	
Report	`generateReport(java.util.Date start, java.util.Date end)` Get the required records within date range Returns Report object.

Interface ReportGenerator.DataProvider

Method Summary	
`Pair<java.util.Vector<ExpenseRecord>, java.util.Vector<IncomeRecord>>`	**`getDataInDateRange`**`(java.util.Date start, java.util.Date end)` Returns a Pair of ExpenseRecord and IncomeRecord vectors within the start date and end date

Class Storable

Method Summary	
`void`	**`afterDeserialize`**`()` Optional method to populate transient attributes after deserializing data from json.
`boolean`	**`isUpdated`**`()`
`void`	**`saved`**`()` Tells the object that it has been stored

Class StorageManager

Method Summary	
`void`	**`addEventListener`**`(StorageManager.StorageEventListener listener)` Need this to handle some exceptions during timer IO
`DataManager`	**`getDataManager`**`()`
`void`	**`read`**`()` Deserialize all data from the json file and start the timer that will attempt to save for each interval
`void`	**`save`**`()` Attempt to save the data.

Interface StorageManager.StorageEventListener

Method Summary	
`void`	**`readFail`**`(java.io.IOException e)`

void	**writeFail**(java.io.IOException e)

Class SummaryDetails

Method Summary	
double	**getBalance**() Get balance value for this SummaryDetails
double	**getExpense**() Get expense value for this SummaryDetails
double	**getIncome**() Get income value for this SummaryDetails
SummaryGenerator.SummaryType	**getSummaryType**() Get the SummaryType for this SummaryDetails

Class SummaryGenerator

Method Summary	
SummaryDetails	**getSummaryDetails**(SummaryGenerator.SummaryType myType) Returns a SummaryDetails object based on which SummaryType is in the parameter
void	**markDataUpdated**() Generate 4 SummaryDetails objects for the different time ranges

Interface SummaryGenerator.DataProvider

Method Summary	
double	**getDailyExpense**()
double	**getDailyIncome**()
double	**getMonthlyExpense**()
double	**getMonthlyIncome**()

double	**getTotalExpense**()
double	**getTotalIncome**()
double	**getYearlyExpense**()
double	**getYearlyIncome**()

Enum SummaryGenerator.SummaryType

Enum Constant Summary
ALLTIME
MONTH
TODAY
YEAR

Method Summary	
abstract java.lang.String	**getName**() Returns the string to be displayed on Main Window based on which time range is selected
abstract SummaryDetails	**getSummaryDetails**() Returns the enum type's SummaryDetails
static SummaryGenerator.SummaryType	**valueOf**(java.lang.String name) Returns the enum constant of this type with the specified name.
static SummaryGenerator.SummaryType[]	**values**() Returns an array containing the constants of this enum type, in the order they are declared.

Class Target

Method Summary	
Target	copy() This returns a copy of the Target
Category	getCategory() This returns the category of the target
double	getTargetAmt() This returns a the target amount that is set by the user

Class TargetManager

A generator that takes in targets and data and produce alert info

Method Summary	
java.util.Vector<Bar>	getAlerts()
java.util.Vector<Bar>	getOrderedBar() This generates an ordered vector of bars in increasing order of ratio of currentAmt/targetAmt
Target	getTarget(Category cat)
java.util.Vector<Target>	getTargets() This returns a copy of the internal targets
void	markDataUpdated()
void	modifyTarget(Target oldTarget, double targetAmt) Removes oldTarget and create a new one using setTarget(Category param1, double param2)
void	removeCategoryTarget(long identifier) This removes the target that has the category ID from the TreeMap This is called when a Category is deleted
void	removeTarget(Target target) Removes target from the tree map
void	setDataProvider(TargetManager.DataProvider data)
Target	setTarget(Category cat, double targetAmt) This returns the target with the same Category and double attribute

Interface TargetManager.DataProvider

Method Summary	
double	**getMonthlyExpense**(Category cat)

6.3 Package ezxpns.data.record

Class Category

An immutable class to represent the current category
meant to be used in Record as a reference

Method Summary	
Category	**copy**()
boolean	**equals**(Category oCat)
long	**getID**()
java.lang.String	**getName**()
java.lang.String	**toString**()

Class ExpenseRecord

Record with two additional attribute: expenseType{NEED, WANT} and PaymentMethod

Method Summary	
Record	**copy**()
ExpenseType	**getExpenseType**()
PaymentMethod	**getPaymentMethod**()

Class ExpenseRecordManager

A special record manager for expense records, since we need to store all payment methods for it

Method Summary

boolean	**addNewPaymentMethod**(PaymentMethod paymentRef) To create a new user defined payment mode
void	**afterDeserialize**() Optional method to populate transient attributes after deserializing data from json.
java.util.Vector<PaymentMethod>	**getAllPaymentMethod**() Get all defined payment modes
boolean	**removePaymentMethod**(PaymentMethod paymentRef) To remove a user defined payment mode
boolean	**updatePaymentMethod**(PaymentMethod paymentRef) To modify a user defined payment mode

Enum ExpenseType

Enum Constant Summary
NEED
WANT

Method Summary	
static ExpenseType	**valueOf**(java.lang.String name) Returns the enum constant of this type with the specified name.
static ExpenseType[]	**values**() Returns an array containing the constants of this enum type, in the order they are declared.

Class IncomeRecord

A Record object that specifically stores income details

Method Summary	
Record	**copy**()

Class Payment

A category-like class that stores payment method

Method Summary	
int	**compareTo**(PaymentMethod o)
PaymentMethod	**copy**()
long	**getID**()
java.lang.String	**getName**()
java.lang.String	**toString**()

Class Record

A container for some basic record attributes

Method Summary	
int	**compareTo**(Record other)
abstract Record	**copy**()
boolean	**equals**(Record other) A method to check if the other Record supplied is the same as this Record object
double	**getAmount**()
Category	**getCategory**()
java.util.Date	**getDate**()
long	**getId**()
java.lang.String	**getName**()
java.lang.String	**getRemark**()
static <T extends Record> double	**sumAmount**(java.util.List<T> rs) A helper function to calculate sum of all records

Class RecordManager<T extends Record>

A java Generic to manage records

Method Summary	
Category	**addNewCategory**(Category toAdd) Create a new category, note that the new category will be a copied with perhaps different id
Category	**addNewCategory**(java.lang.String catName) Create a new category with the name
T	**addNewRecord**(T toAdd) Add a new record, returns the record added.
void	**afterDeserialize**() Optional method to populate transient attributes after deserializing data from json.
java.util.List<Category>	**getAllCategories**() Get all user defined categories
double	**getAllTimeSum**()
java.util.Vector<Category>	**getCategoriesBy**(java.lang.String name)
Category	**getCategory**(java.lang.Long id)
double	**getDailySum**()
double	**getMonthlySum**()
double	**getMonthlySum**(Category cat)
T	**getRecordBy**(long id)
java.util.Vector<T>	**getRecordsBy**(Category category, int max) Search for records matching the category
java.util.Vector<T>	**getRecordsBy**(java.util.Date start, java.util.Date end, int max, boolean reverse) Search for records in the date range, inclusive of both ends
java.util.Vector<T>	**getRecordsBy**(java.lang.String name, int max) Search for records matching the name
java.util.Vector<T>	**getRecordsByCategory**(java.lang.String name)
double	**getYearlySum**()
boolean	**removeCategory**(Category category)

boolean	removeCategory(long identifier) Remove a user defined category based on the given identifier
void	removeRecord(long id) Remove a record
Category	updateCategory(long identifier, Category selectedCat) Modify Category Method
void	updateCategory(long id, java.lang.String newName)
T	updateRecord(long id, T updated) Update a record.

6.4 Package ezxpns.GUI

Interface CategoryHandlerInterface

Interface to handle the categories between the Graphical User Interface and the data storage (upon GUI Exit)

Method Summary	
Category	addNewCategory(Category newCat) Create a new category, note that the new category will be a copied with perhaps different id
Category	addNewCategory(java.lang.String name) Create a new category with the name
java.util.List<Category>	getAllCategories() Get all user defined categories
boolean	removeCategory(long identifier) Remove a user defined category based on the given identifier
Category	updateCategory(long identifier, Category selectedCat) Modify Category Method

Class CategoryModel

A model used to display list of category in category frame

Method Summary	
java.lang.Object	getElementAt(int arg0)
int	getSize()

void	**update**() Refresh the whole list If the size of categories gets large, this can be slow But given that we are not even using a data base, who cares about this small one

Interface ConfigManagerInterface

To manage the configurations of the GUI - place where all the constants are stored
(i.e. height and width of the window)
Should also be writing or accessing the written or previously saved configurations

Field Summary	
static java.awt.Font	**DEFAULT_BTN_FONT**
static int	**DEFAULT_BTN_FONT_SIZE**

Method Summary	
void	**load**() To load the configuration from the file xyz.ini (example)
void	**save**() To save the configuration from the file xyz.ini (example)

Interface RecordHandlerInterface

To handle the records between the Graphical User Interface and the data storage (upon GUI Exit)

Method Summary	
boolean	**createRecord**(ExpenseRecord newRecord) Create a new expense record
boolean	**createRecord**(IncomeRecord newRecord) Create a new income record
Record	**getRecord**(long identifier) Retrieve a specific record based on the identifier given
java.util.List<Record>	**getRecords**(int n) Get some records stored
ExpenseRecord	**lastExpenseRecord**(java.lang.String name) Return the latest expense record matching the name, or null

IncomeRecord	**lastIncomeRecord**(java.lang.String name) Return the latest expense record matching the name, or null
boolean	**modifyRecord**(long id, ExpenseRecord selectedRecord) Modify an expense record
boolean	**modifyRecord**(long id, IncomeRecord selectedRecord) Modify an income record
boolean	**removeRecord**(long identifier) Remove record based on an identifier

Interface SearchHandlerInterface

This interface denotes the possible searches the user may do as well as assists the UI side in disseminating the search request, and returning formatted, relevant search results back to the UI component.

Method Summary	
java.util.Vector<Record>	**search**(SearchRequest req)

Class SearchRequest

Wrapper class to contains search query parameters
May support multiple queries (future iterations)
but not handled in the master class for now.

Method Summary	
Category	**getCategory**()
Pair<java.util.Date,java.util.Date>	**getDateRange**()
java.lang.String	**getName**()
SearchRequest.RecordType	**getType**()
boolean	**isMultiple**()
void	**setType**(SearchRequest.RecordType type)
java.lang.String	**toString**()

Enum SearchRequest.RecordType

Enum Constant Summary
BOTH
EXPENSE
INCOME

Method Summary	
static SearchRequest.RecordType	**valueOf**(java.lang.String name) Returns the enum constant of this type with the specified name.
static SearchRequest.RecordType[]	**values**() Returns an array containing the constants of this enum type, in the order they are declared.

Class UIControl

To assist EzXpns in managing all the GUI Windows

Method Summary	
void	**closeHomeScreen**() Closes the main/home screen of EzXpns
void	**showCatWin**() Displays the category handler window
void	**showHomeScreen**() Display the main/home screen of EzXpns
void	**showRecWin**() Displays the new record handler window
void	**showRecWin**(int recordType) Displays a new record handler window with the chosen tab Use RecordFrame.TAB_INCOME or TAB_EXPENSE to choose
void	**showReportWin**() Displays the report handler window
void	**showSearchWin**() Displays the search handler window

6.5 Package ezxpns.util

Class Pair<L,R>

Method Summary	
boolean	**equals**(java.lang.Object o)
L	**getLeft**() Get the left object
R	**getRight**() Get the right object
int	**hashCode**()

Project Process & Administration

Responsibilities

Andrew: GUI and documentation diagrams
Shu Zhen: Target/alert system
Ting Zhe: Report and Summary system, graphics
Yujian: Overall architecture, backend and data management

Timeline

Week	Task
7	• Basic storage and retrieval • Basic search in backend • GUI and implementation for alert • Report fully designed and implemented
8	• All components integrated • Basic features done • Redesign/optimisation of GUI
9	Version 0.1 is due
9-11	Finish all the to-dos (see Apendix)
11	Version 0.2 due on Monday 15 April

Appendix

Future works

1. Refactoring of `DataManager`

    Currently, this class use data structures like `TreeMap` or `HashMap` to index fields of classes inheriting `Record`. This makes the code messy as each additional indexing field requires changes to functions that add, remove and load records. This can be handled by a indexing framework with help of classes that perhaps inherit a `AbstractIndexer`.
2. Implement undo
3. Implement validation for every form fields for a new Expense / Income record
4. Add more unit tests
5. Make the amount input support simple math equation
6. Summary panel for needs and wants, with simple intelligence

======= End of Report =======