Expensive v0.2 Developer Guide

EzXpns

Contents

Chapter 1

Developer Guide

EzXpns is a personalised expense management system that allows users to record all their expenditure and do some simple management of their budget and expenses. It aims to provide the user with the platform for stress-free budget management. With a simple click of the button and some simple data entry, the user would be provided with a detailed overview of his expenditure and saving trends and patterns over a duration of a week, a month, or even a year.

This guide aims to help developers like you to understand the design of EzXpns so that you may begin to work on improving it. We will assume that you are already familiar with Java as well as object-oriented programming concepts. In addition, as EzXpns is a graphical user interface (GUI) application, you may find it beneficial to do some background reading on GUI application development concepts, such as the Model-View-Presenter software design pattern.

1.1 Getting started

1.1.1 Prerequisites

Before you begin to work on the source code, you will need to install on your system a few tools that you will use extensively later on. They are:

- **Java Development Kit (JDK)**[1]
  Used for compiling Java source code. The minimum version required to build the source code is version 7u2.
- **JavaFX Software Development Kit (SDK)**
  JavaFX is the graphical toolkit used in EzXpns. It provides a set of UI controls and graphics and has native support for rendering charts, in addition to Cascading Style Sheets (CSS) support. The JavaFX SDK comes included with latest releases of the JDK.
- **Eclipse**[2]
  A popular integrated development environment (IDE) for Java projects. While there are many IDEs available on the market and every developer has their own preferences, using Eclipse is recommended to avoid potential interoperability problems during development.
- **Mercurial**[3]
  Used for version control. It efficiently handles projects of any size and offers a powerful command line interface to manage changes in the source tree.

[1] `http://www.oracle.com/technetwork/java/javase/downloads/index.html`
[2] `http://www.eclipse.org`
[3] `http://mercurial.selenic.com`

- **Apache Maven**[4]
  This is the build and packaging tool for the project. As a command-line tool, building the project and performing test runs can be quickly done without the need for an IDE. Its automatic handling and downloading external dependencies speeds up the process of using external libraries by not needing to check them into version control.

EzXpns also depends on a few third-party libraries. You should not need to install them manually as they are automatically managed by the Maven build tool. The libraries are:

- **XStream**[5]
  Extensible Markup Language (XML) is the format used for on-disk storage of expense information. The XStream library handles the conversion of Java objects into pretty-printed XML.
- **JUnit**[6]
  The testing environment for the project's unit and integration tests.
- **Logback**[7]
  Logback is a logging framework for Java. It is used to record and trace major code activity, which is useful for debugging purposes.

1.1.2 Obtaining the source code

There are a few simple steps you can follow to get the latest copy of the project source code, and set up an IDE project on your system.

1. Ensure that you have commit access to the project's code repository on Google Code. The name of the project is "cs2103jan13-w11-1j".
2. Choose a location for your local copy of the repository. Using your existing Eclipse workspace directory should suffice since you will be adding the project to Eclipse later on.
3. Clone the repository. You should end up with a new directory `ezxpns` containing `.hgignore`, `pom.xml`, and a subdirectory `src` with all the Java source files.
4. Now you can add the Maven project to Eclipse. Launch Eclipse and open your workspace. Select File › Import...

   In the "Import" dialog, select Maven › Existing Maven Projects. Click Next.

   In the new "Import Maven Projects" dialog, browse for the root directory of your remote repository. Click Finish.
5. Eclipse will now proceed with the import. It may take a while to download the third-party dependencies.

[4] `http://maven.apache.org`
[5] `http://xstream.codehaus.org`
[6] `http://junit.org`
[7] `http://logback.qos.ch`

1.2 Use cases

The vision for EzXpns came about when it was found that existing expense tracking solutions were complicated to use or involved tedious registration processes. Therefore EzXpns concentrates on providing only the most essential expense tracking features, exposed to the user through an inviting and easy-to-use user interface. Figure 1.1 shows the use cases that are currently supported. Do keep the idea of simplicity in mind when contemplating adding a new feature to the software.

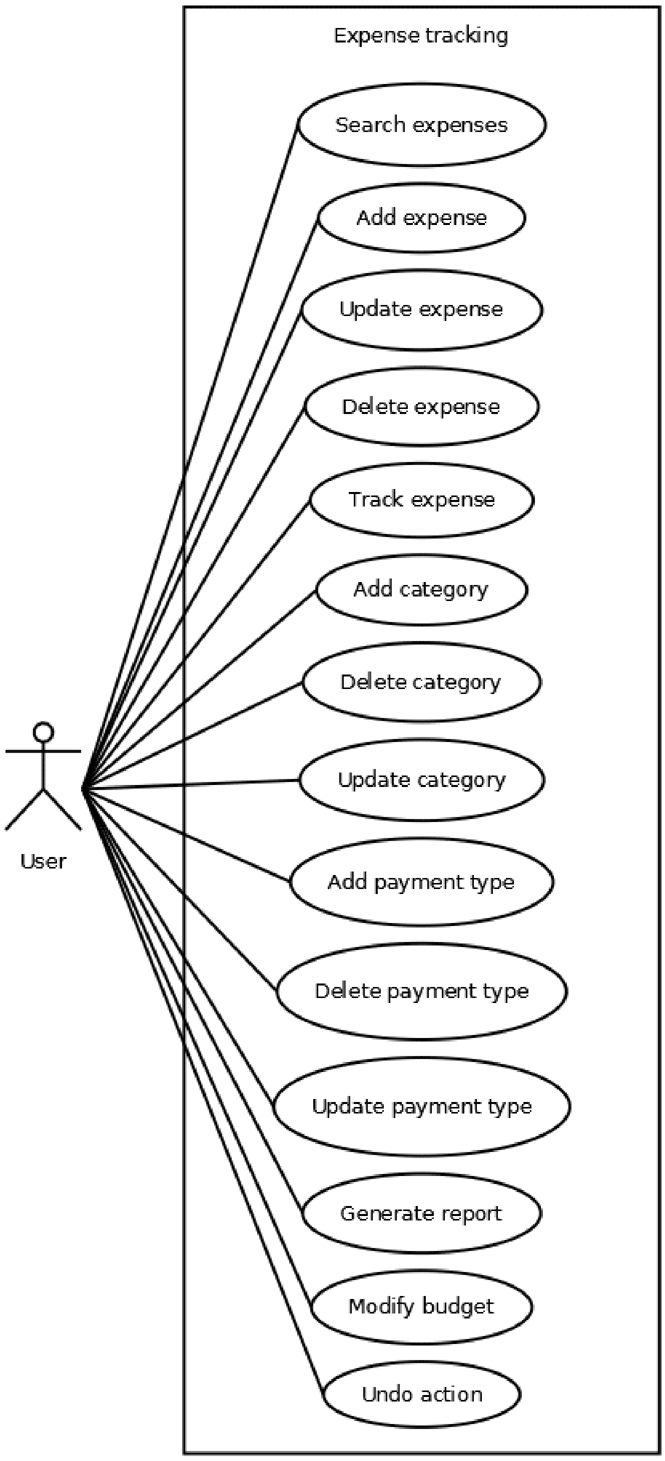


Figure 1.1: Use case diagram

1.3 Architecture

EzXpns is separated into several components, each consisting of a number of classes. This enhances code modularity and also makes it easier for developers to work in parallel.

1.3.1 Components

Figure 1.2 shows a high-level overview of the components and the relationships between them.

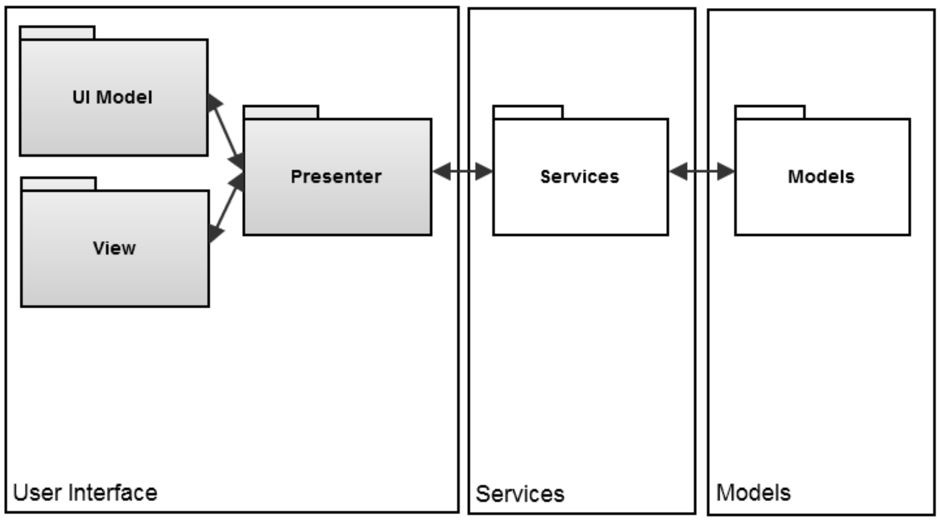


Figure 1.2: Architecture diagram

There are three components: User Interface (UI), Services, and Models. You can identify which component any class belongs to by just checking which package it is in.

UI

The UI component is broken up into three sub-components: the UI model, the view, and the presenter. The UI model holds data relevant to its related view. Values in the UI model are bound to UI components in the view. When any value in the UI model is changed, the view is notified and is automatically updated as well. The presenter links the UI model and the view together and also sends commands to the Services. The `net.mysoc.w111j.ui` package holds all UI classes.

Services

Services contain the business logic for manipulating (e.g., CRUD) model objects. They act as the interface between the UI component and the Models component. Services belong to the `net.mysoc.w111j.service` package.

Models

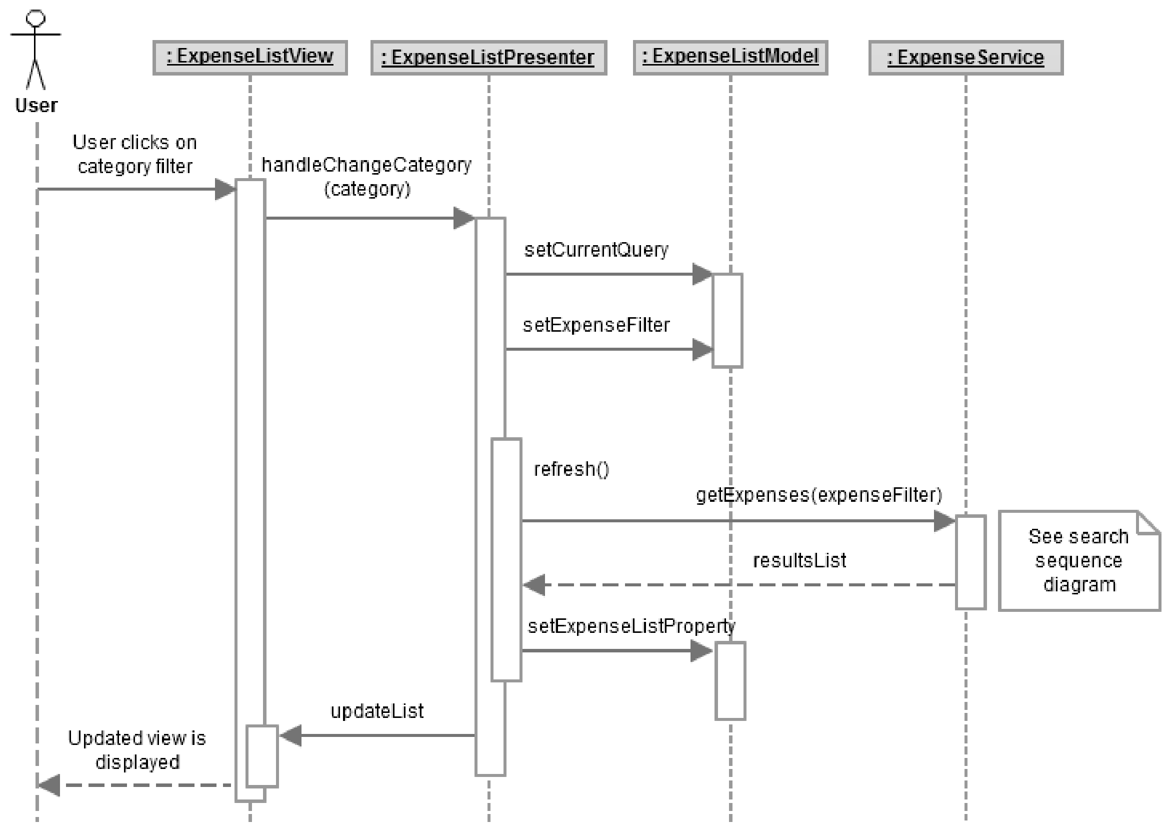
Models are data objects used to store or represent information. These are passive models and do not notify the UI when they are updated. Do not confuse these data models with UI models – UI models only store data to be displayed in the UI view. All models are defined in the `net.mysoc.w111j.model` package.

1.3.2 Component interaction

The typical flow of interaction between the components goes as follows:

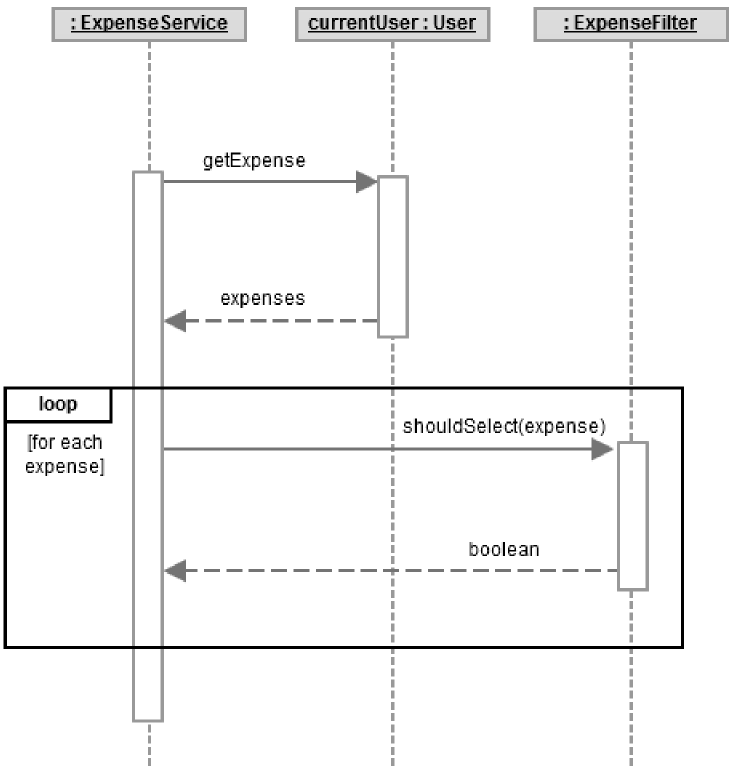
- The UI (typically the presenter) sends commands to Services.
- Services receive the commands and takes the appropriate action (such as retrieving `Expense` objects from Models).
- The response is then returned to the caller.

To help you better understand how this works, we include the following two examples. These examples are chosen because they are representative of most operations found in EzXpns.



Example 1.3: Listing expense by category sequence diagram

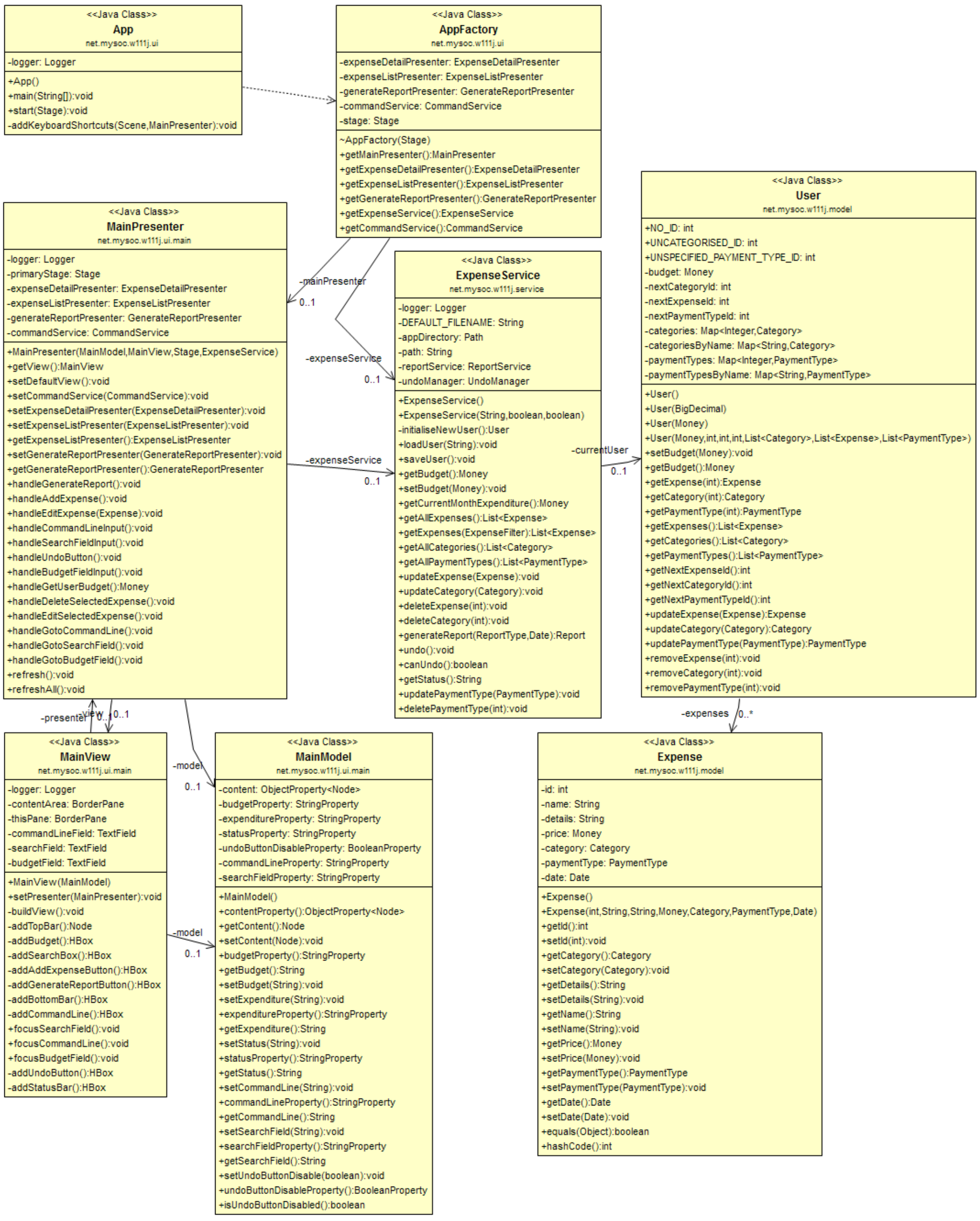
When the user clicks on one of the categories in the UI, `ExpenseListView` invokes the `handleChangeCategory(category)` method in `ExpenseListPresenter`. The presenter then sets the current query and expense filter in the `ExpenseListModel`. A refresh is done to update UI models with new data. During the refresh, the presenter calls `ExpenseService` to do a search based on the filter stored in the UI model. (Details of the search performed by `ExpenseService` are illustrated in Example 1.4.) The updated list returned by `ExpenseService` is stored in the UI model. The presenter then calls the view to update itself by retrieving the list from the UI model.



Example 1.4: Search sequence diagram

To perform a search based on an `ExpenseFilter`, `ExpenseService` first calls `currentUser.getExpense()` to retrieve all the current user's expenses. It then iterates through every expense, checking whether each expense matches the search filter criteria by calling the `shouldSelect(expense)` method. The matching expenses are returned as a `List<Expense>`.

1.3.3 Class design



<<Java Class>>
App
net.mysoc.w111j.ui
-logger: Logger
+App()
+main(String[]):void
+start(Stage):void
-addKeyboardShortcuts(Scene,MainPresenter):void
<<Java Class>>
AppFactory
net.mysoc.w111j.ui
-expenseDetailPresenter: ExpenseDetailPresenter
-expenseListPresenter: ExpenseListPresenter
-generateReportPresenter: GenerateReportPresenter
-commandService: CommandService
-stage: Stage
~AppFactory(Stage)
+getMainPresenter():MainPresenter
+getExpenseDetailPresenter():ExpenseDetailPresenter
+getExpenseListPresenter():ExpenseListPresenter
+getGenerateReportPresenter():GenerateReportPresenter
+getExpenseService():ExpenseService
+getCommandService():CommandService
<<Java Class>>
MainPresenter
net.mysoc.w111j.ui.main
-logger: Logger
-primaryStage: Stage
-expenseDetailPresenter: ExpenseDetailPresenter
-expenseListPresenter: ExpenseListPresenter
-generateReportPresenter: GenerateReportPresenter
-commandService: CommandService
+MainPresenter(MainModel,MainView,Stage,ExpenseService)
+getView():MainView
+setDefaultView():void
+setCommandService(CommandService):void
+setExpenseDetailPresenter(ExpenseDetailPresenter):void
+setExpenseListPresenter(ExpenseListPresenter):void
+getExpenseListPresenter():ExpenseListPresenter
+setGenerateReportPresenter(GenerateReportPresenter):void
+getGenerateReportPresenter():GenerateReportPresenter
+handleGenerateReport():void
+handleAddExpense():void
+handleEditExpense(Expense):void
+handleCommandLineInput():void
+handleSearchFieldInput():void
+handleUndoButton():void
+handleBudgetFieldInput():void
+handleGetUserBudget():Money
+handleDeleteSelectedExpense():void
+handleEditSelectedExpense():void
+handleGotoCommandLine():void
+handleGotoSearchField():void
+handleGotoBudgetField():void
+refresh():void
+refreshAll():void
-mainPresenter
0..1
-expenseService
0..1
-expenseService
0..1
<<Java Class>>
ExpenseService
net.mysoc.w111j.service
-logger: Logger
-DEFAULT_FILENAME: String
-appDirectory: Path
-path: String
-reportService: ReportService
-undoManager: UndoManager
+ExpenseService()
+ExpenseService(String,boolean,boolean)
-initialiseNewUser():User
+loadUser(String):void
+saveUser():void
+getBudget():Money
+setBudget(Money):void
+getCurrentMonthExpenditure():Money
+getAllExpenses():List<Expense>
+getExpenses(ExpenseFilter):List<Expense>
+getAllCategories():List<Category>
+getAllPaymentTypes():List<PaymentType>
+updateExpense(Expense):void
+updateCategory(Category):void
+deleteExpense(int):void
+deleteCategory(int):void
+generateReport(ReportType,Date):Report
+undo():void
+canUndo():boolean
+getStatus():String
+updatePaymentType(PaymentType):void
+deletePaymentType(int):void
-currentUser
0..1
<<Java Class>>
User
net.mysoc.w111j.model
+NO_ID: int
+UNCATEGORISED_ID: int
+UNSPECIFIED_PAYMENT_TYPE_ID: int
-budget: Money
-nextCategoryId: int
-nextExpenseId: int
-nextPaymentTypeId: int
-categories: Map<Integer,Category>
-categoriesByName: Map<String,Category>
-paymentTypes: Map<Integer,PaymentType>
-paymentTypesByName: Map<String,PaymentType>
+User()
+User(BigDecimal)
+User(Money)
+User(Money,int,int,int,List<Category>,List<Expense>,List<PaymentType>)
+setBudget(Money):void
+getBudget():Money
+getExpense(int):Expense
+getCategory(int):Category
+getPaymentType(int):PaymentType
+getExpenses():List<Expense>
+getCategories():List<Category>
+getPaymentTypes():List<PaymentType>
+getNextExpenseId():int
+getNextCategoryId():int
+getNextPaymentTypeId():int
+updateExpense(Expense):Expense
+updateCategory(Category):Category
+updatePaymentType(PaymentType):PaymentType
+removeExpense(int):void
+removeCategory(int):void
+removePaymentType(int):void
-expenses
0..*
-presenter
-view
0..1
0..1
<<Java Class>>
MainView
net.mysoc.w111j.ui.main
-logger: Logger
-contentArea: BorderPane
-thisPane: BorderPane
-commandLineField: TextField
-searchField: TextField
-budgetField: TextField
+MainView(MainModel)
+setPresenter(MainPresenter):void
-buildView():void
-addTopBar():Node
-addBudget():HBox
-addSearchBox():HBox
-addAddExpenseButton():HBox
-addGenerateReportButton():HBox
-addBottomBar():HBox
-addCommandLine():HBox
+focusSearchField():void
+focusCommandLine():void
+focusBudgetField():void
-addUndoButton():HBox
-addStatusBar():HBox
-model
0..1
-model
0..1
<<Java Class>>
MainModel
net.mysoc.w111j.ui.main
-content: ObjectProperty<Node>
-budgetProperty: StringProperty
-expenditureProperty: StringProperty
-statusProperty: StringProperty
-undoButtonDisableProperty: BooleanProperty
-commandLineProperty: StringProperty
-searchFieldProperty: StringProperty
+MainModel()
+contentProperty():ObjectProperty<Node>
+getContent():Node
+setContent(Node):void
+budgetProperty():StringProperty
+getBudget():String
+setBudget(String):void
+setExpenditure(String):void
+expenditureProperty():StringProperty
+getExpenditure():String
+setStatus(String):void
+statusProperty():StringProperty
+getStatus():String
+setCommandLine(String):void
+commandLineProperty():StringProperty
+getCommandLine():String
+setSearchField(String):void
+searchFieldProperty():StringProperty
+getSearchField():String
+setUndoButtonDisable(boolean):void
+undoButtonDisableProperty():BooleanProperty
+isUndoButtonDisabled():boolean
<<Java Class>>
Expense
net.mysoc.w111j.model
-id: int
-name: String
-details: String
-price: Money
-category: Category
-paymentType: PaymentType
-date: Date
+Expense()
+Expense(int,String,String,Money,Category,PaymentType,Date)
+getId():int
+setId(int):void
+getCategory():Category
+setCategory(Category):void
+getDetails():String
+setDetails(String):void
+getName():String
+setName(String):void
+getPrice():Money
+setPrice(Money):void
+getPaymentType():PaymentType
+setPaymentType(PaymentType):void
+getDate():Date
+setDate(Date):void
+equals(Object):boolean
+hashCode():int

Figure 1.5: General overview of classes

Figure 1.5 shows a partial class diagram to help you get an idea of the basic structure of the classes in each architectural component. A few things deserve special mention:

- Observe the relationship between `MainPresenter`, `MainView`, `MainModel`, in particular the object references held by each class.
- Notice that `MainPresenter` does not make any references to `User`. This is because any changes made to Models should be done by Services.

1.4 User interface design and implementation

The guiding principle for our UI design is *simplicity*. We strongly believe a simple UI lends well to a great, efficient, and enjoyable user experience.

1.4.1 Design inspirations

Metro

The Metro design language is a bold, clean style with roots in Swiss graphic design. We decided to implement Metro as we are targeting the Windows platform. It also provides a clean look for busy interfaces.

Metro is distinct from Modern, which is Microsoft's Metro interpretation for Windows 8.

Strict adherence to Metro-defined typography styles and the 20px grid gives EzXpns a consistent look throughout. When designing new UI elements, consider using CSS styles already defined in the file `ezxpns.css`. `ezxpns.css` is located in the `resources` subdirectory. See Section 1.4.2 for more details on CSS usage.

GitHub for Windows

We borrow design styling metrics from GitHub for Windows[8], GitHub's Metro-styled application for the desktop. Some of Metro's guidelines do not translate well to the desktop experience. Metrics taken from GitHub for Windows includes the 40px left margin instead of the 140px recommended for full-screen Metro applications.

1.4.2 Implementation

JavaFX

JavaFX is a modern software platform for the development of UIs in Java. JavaFX is bundled with Java versions 7u2 and above, and is designed to replace the ageing Swing Toolkit (SWT). JavaFX supports FXML (an XML derivative) for the declaration of UI controls, but we currently do not use this feature.

However, we do use CSS styling support in JavaFX for UI elements. You can assign CSS classes to JavaFX `Node` objects. These objects are then styled by rules defined in `ezxpns.css`.

> Note: CSS rules in JavaFX have an `-fx` prefix. For example, the CSS rule `font-family` is replaced by `-fx-font-family` in JavaFX CSS.

Please see the JavaFX CSS Reference Guide[9] for more specifics on CSS styling.

FX Charts

JavaFX has built-in chart support. Charts in the reports window are all graphical charts available in the `javafx.scene.chart` package. You can easily create a new chart by utilising one of these charts. Data for charts can be obtained from methods in the `GenerateReport` service.

[8] `http://windows.github.com`
[9] `http://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html`

1.4.3 Design resources

Not all Metro styles are pre-defined in `ezxpns.css`. When in doubt, consult the following online resources and make a reasoned decision:

- Metro presentation[10]
- Windows 8 UX Guidelines[11]
- Microsoft Modern Design Assets[12]
- JavaFX Metro CSS Styles[13]
- Modern Form Layout Guidelines[14]
- Windows Store Dev Guidelines[15]

1.5 Important APIs

1.5.1 Expense search

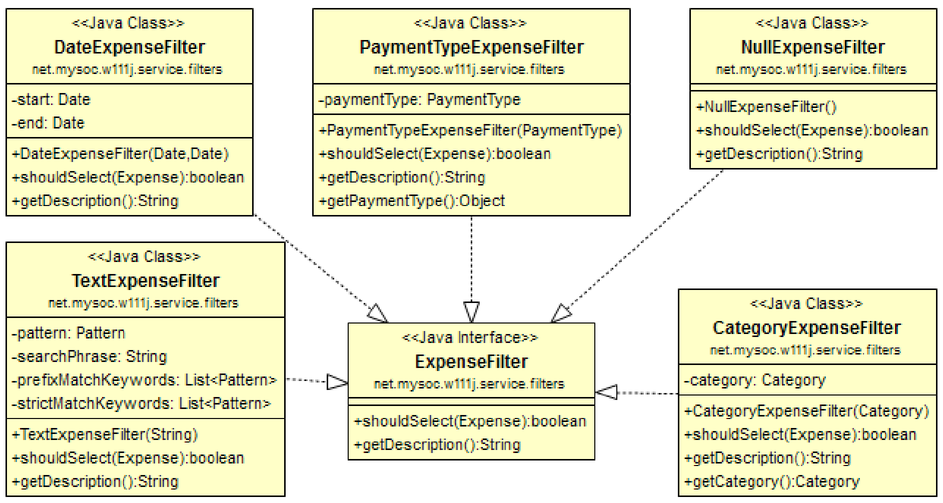

<<Java Class>>
DateExpenseFilter
net.mysoc.w111j.service.filters
-start: Date
-end: Date
+DateExpenseFilter(Date,Date)
+shouldSelect(Expense):boolean
+getDescription():String
<<Java Class>>
PaymentTypeExpenseFilter
net.mysoc.w111j.service.filters
-paymentType: PaymentType
+PaymentTypeExpenseFilter(PaymentType)
+shouldSelect(Expense):boolean
+getDescription():String
+getPaymentType():Object
<<Java Class>>
NullExpenseFilter
net.mysoc.w111j.service.filters
+NullExpenseFilter()
+shouldSelect(Expense):boolean
+getDescription():String
<<Java Class>>
TextExpenseFilter
net.mysoc.w111j.service.filters
-pattern: Pattern
-searchPhrase: String
-prefixMatchKeywords: List<Pattern>
-strictMatchKeywords: List<Pattern>
+TextExpenseFilter(String)
+shouldSelect(Expense):boolean
+getDescription():String
<<Java Interface>>
ExpenseFilter
net.mysoc.w111j.service.filters
+shouldSelect(Expense):boolean
+getDescription():String
<<Java Class>>
CategoryExpenseFilter
net.mysoc.w111j.service.filters
-category: Category
+CategoryExpenseFilter(Category)
+shouldSelect(Expense):boolean
+getDescription():String
+getCategory():Category

Figure 1.6: Expense search filters

An expense search is performed when you want to retrieve a filtered set of `Expense` objects. You might use it to change what is displayed in the expense list panel. The search function works by iterating through all existing expenses and invoking the `shouldSelect()` method of the `ExpenseFilter` class. `Expense` objects are retrieved from the active `User`.

To search through expenses, call the `getExpenses(ExpenseFilter)` of the `ExpenseService` class. The `ExpenseFilter` passed to `getExpense` will determine the expense filtering behaviour.

Some available filters are:

- `NullExpenseFilter()`
  matches all `Expense` objects
- `TextExpenseFilter(String)`
  matches all `Expense` objects that contain the given keywords

[10]`https://skydrive.live.com/view.aspx?resid=40CFFDE85F1AB56A%211284`
[11]`http://go.microsoft.com/fwlink/p/?linkid=258743`
[12]`http://msdn.microsoft.com/en-us/library/windows/apps/hh779072.aspx`
[13]`https://github.com/JFXtras/jfxtras-styles/blob/master/src/jmetro/JMetroDarkTheme.css`
[14]`http://msdn.microsoft.com/en-us/library/windows/apps/jj839734.aspx`
[15]`http://msdn.microsoft.com/en-us/library/windows/apps/br229519.aspx`

- `CategoryExpenseFilter(Category)`
  matches all `Expense` objects in the given `Category`
- `DateExpenseFilter(Date start, Date end)`
  matches all `Expense` objects in date range
- `PaymentTypeFilter(PaymentType)`
  matches all `Expense` objects with the given `PaymentType`

To add a new filter, create a new class under the `net.mysoc.w111j.service.filters` package and implement the `ExpenseFilter` interface. It should override the `shouldSelect(Expense)` method, which will be called by `ExpenseService` when filtering expenses. An expense is included in the filtered results if `shouldSelect()` returns `true`.

For example, the following statement returns a `List` of `Expense` objects with text matching the `String` "foobar":

```
expenseService.getExpenses(new TextExpenseFilter("foobar"));
```

1.5.2 Saving and loading

User profiles are saved to and loaded from disk through the `UserXMLHelper` class located in the `net.mysoc.w111j.io` package. There are two main APIs in this class:

- `saveUser(String, User)` – saves the passed-in `User` object as a XML file located at the path specified as a `String`.
- `loadUser(String)` – load and return a `User` object from a XML file. Pass in the location of the file as a `String`.

1.5.3 Undoable actions

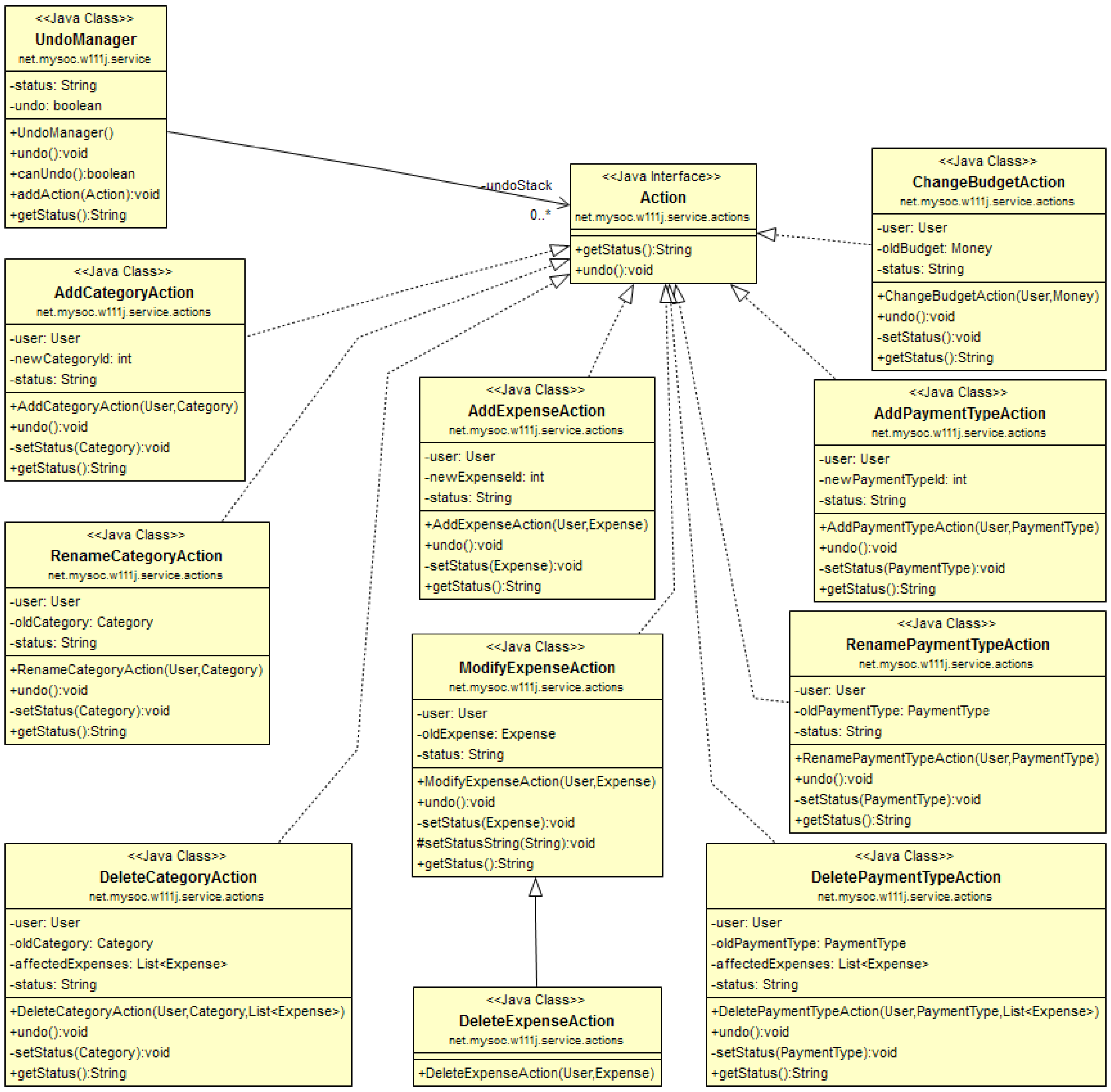


Figure 1.7: Undoable actions

To allow the user to undo their last operation, an `Action` is created and stored in a stack within an UndoManager whenever the user performs an undoable action.

The current list of `Action` classes are:

- `AddCategoryAction`
- `RenameCategoryAction`
- `DeleteCategoryAction`
- `RenamePaymentTypeAction`
- `DeletePaymentTypeAction`
- `AddExpenseAction`

- `ModifyExpenseAction`
- `DeleteExpenseAction`
- `ChangeBudgetAction`

To add a new type of undoable action, create a new class and implement the `Action` interface. It should define an `undo()` method, which will be called by the `UndoManager` to reverse the effects of the last action.

1.6 Testing

Software testing is necessary to ensure that features work as intended and that new code changes do not cause regressions. Testing for EzXpns is done through both automated unit testing and manual testing (for sections which cannot, or are difficult to, be automated). Services and Models are unit tested while UI testing is mostly done manually.

1.6.1 Unit testing

The project uses the popular JUnit test harness to implement unit testing.

New code should be written in tandem with unit tests. For example, every public method in the `Money` class has a corresponding unit test in `MoneyTest`. Passing the unit test gives confidence that the code just written has some degree of correctness.

Here is an example unit test from `MoneyTest.java`. This unit test checks for the correct conversion of `Money` objects into a `BigDecimal` representation.

```
private final Money ZERO = new Money();
private final Money ONE = new Money("1");

@Test
public void testToBigDecimal() {
    assertEquals(0, ZERO.toBigDecimal().compareTo(new BigDecimal("0")));
    assertEquals(0, ONE.toBigDecimal().compareTo(new BigDecimal("1")));
    assertNotEquals(0, ONE.toBigDecimal().compareTo(new BigDecimal("0")));
}
```

Example 1.8: `Money` unit test

1.6.2 Manual testing

As it is difficult to automate tests for UI testing, it is currently done manually. To minimise the number of tests required, we have tried to move as much business logic out of the UI as possible into Services and Models.

Here are some example test cases for UI testing:

Test case	Description
TC001	**UI involved**: Main UI, ExpenseDetail UI **Description**: Adding a valid expense. **Objective**: To add the expense from user input **Input**: Click the `Add Expense` button. The expenses window is displayed. Enter "abc" as the name and "123" as the price. Click the `OK` button. **Expected output**: Expense is added successfully and displayed in the expense list.
TC001a	**UI involved**: Main UI, ExpenseDetail UI **Description**: Adding an invalid expense. **Objective**: To check if the user has input all required fields **Input**: Click the `Add Expense` button. The name and the price fields are empty. Click the `OK` button. **Expected output**: A new expense is not added. Validation warnings and errors appear next to invalid fields.
TC002	**UI involved**: ExpenseList UI **Description**: Deleting a category. **Objective**: To verify that categories can be deleted properly. **Input**: Click the `Delete` button which appears when your mouse is hovering over a category. **Expected output**: The category is deleted and removed from the side bar.
TC003	**UI involved**: Main UI, Report UI **Description**: Generating a report. **Objective**: To display the reports window. **Input**: Click the `Generate Report` button. Click the corresponding period tab (e.g. `DAY`, `MONTH`). **Expected output**: The reports window will be shown. The corresponding report for the selected report type is shown in the window.
TC004	**UI involved**: Main UI **Description**: Undo previous action. **Objective**: To undo the previous action. **Input**: Click the `Undo` button. **Expected output**: The previous action is undone and the status bar is updated.
TC005	**UI involved**: Main UI **Description**: Setting a valid budget. **Objective**: To change the current budget. **Input**: Select budget field and enter "173" as the value. **Expected output**: The budget is updated and changed in the field.
TC005a	**UI involved**: Main UI **Description**: Setting an invalid budget. **Objective**: To check whether the budget field is validated. **Input**: Select the budget field and enter "abc". **Expected output**: The characters "abc" cannot be entered into the budget field.
TC006	**UI involved**: Main UI **Description**: Searching for expenses (simple search). **Objective**: To search for expenses by keyword. **Input**: Select the search field and enter the name of an existing expense. **Expected output**: Matched expenses are displayed. An empty list is displayed if there are no matching results.
TC007	**UI involved**: ExpenseList UI **Description**: Renaming a category. **Objective**: To update a category's name. **Input**: Hover over a category in the side bar and click the `Edit` button. Enter "Food" into the field. **Expected output**: The category name is updated.

1.7 Change log

Version 0.1 (released on 18 March 2013)

- Implemented setting of budget
- Implemented expense and category management
- Implemented simple search
- Implemented basic report generation
- Implemented status bar
- Created a basic GUI to expose the features

Version 0.2 (released on 15 April 2013)

- Styled GUI based on Metro design
- Made improvements to reporting feature
- Implemented payment type management
- Hooked up a command line parser to the user interface
- Designed and implemented advanced search syntax

1.8 Known issues

- Only one expense can be selected at any time
- No visual feedback for invalid entry in the command line

1.9 Future work

1.9.1 Mobile version

Currently, EzXpns is only available as a desktop application. One possible way to expand on this project is to extend the service to other kinds of devices. Based on this idea, an app for smartphones can be developed that allows users to access their expense history on-the-go. Users should also be able to enter expenses on their phones right after making a purchase instead of having to wait till they get home. You can also explore ways to synchronise expense data between the different devices to further streamline the user experience.

1.9.2 Multiple user management

Presently, our software only allows a single user profile to be created. It can be made such that multiple users can use the software via a password-based login system.

1.9.3 Internationalisation

The current interface is limited only to the English language. The currency of expenses is also assumed to be in dollars. However, there are many potential users in non-English speaking regions, and also users that regularly shop online and pay for their items in various currencies. Support for more languages and currencies can be implemented to cater to the requirements of these users.

1.9.4 Automated UI testing

As mentioned in section 1.6.2, testing of the UI is done via manual testing. This takes up the developer's time and is prone to human error. To improve this, some research needs to be done into converting the UI tests into automated tests. One possible avenue is to activate user actions programmatically and compare the resulting UI changes with reference screenshots.