Group 15

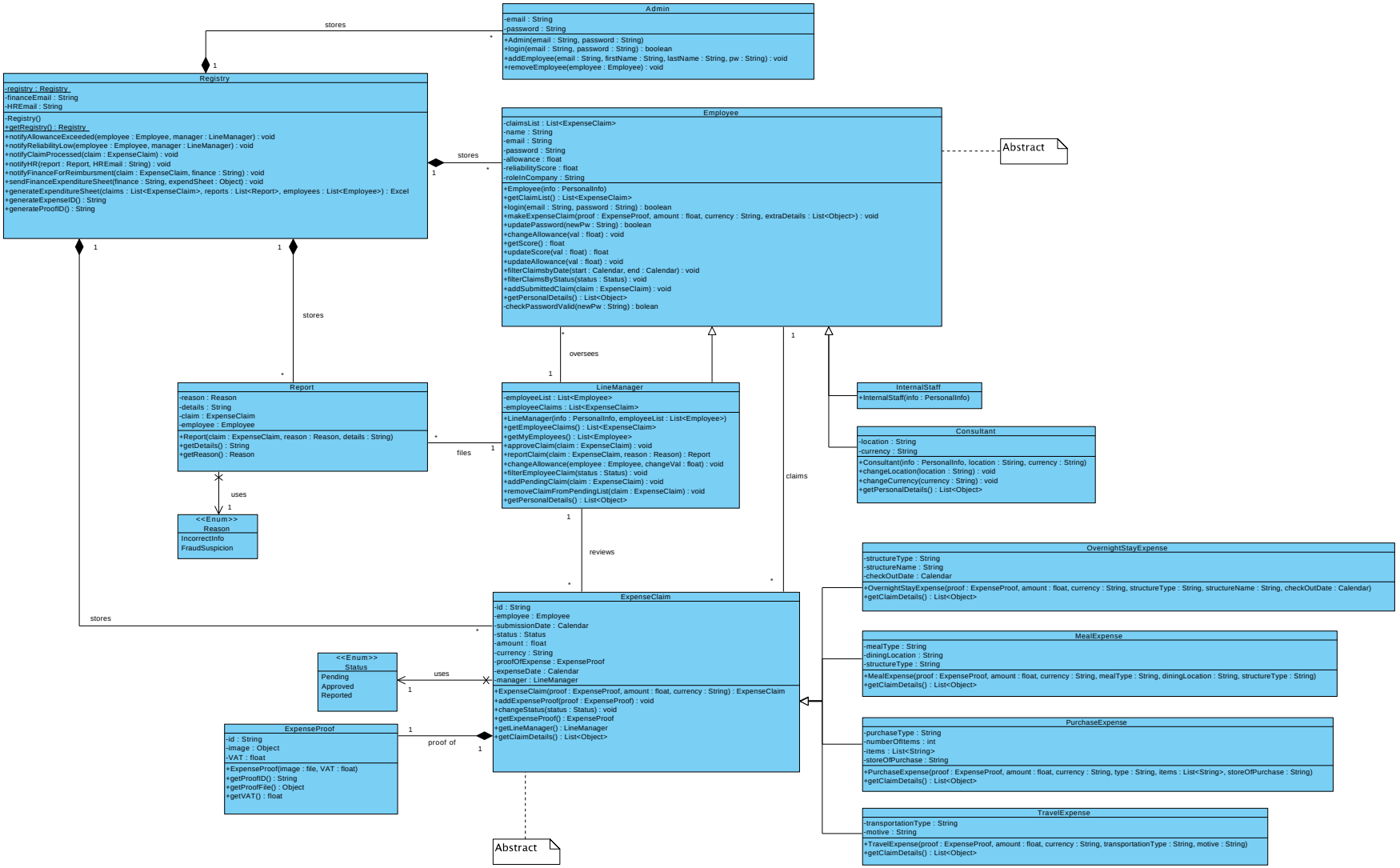
FDM Expenses Manager

ECS506U Software Engineering Group Project

Design Report

0. Contents

1. Class Diagram



2. Traceability matrix

Class	Requirement	Brief Explanation
Employee	FE2, FE3, FE8	All personal information for employee stored as attributes in Employee superclass
Consultant	FC1	Consultants additionally have their location and currency as separate attributes in the Consultant class
Line Manager	FLM1, FLM2	LM are identified by the list of employees they oversee and the claims they are in charge of processing, both represented as attributes in the LineManager class
Admin	FA4	Admin class has the attributes for email and password
Registry	FS4, FS5	Contact details (email) for finance department and HR are stored in the Registry (as attributes)
Expense Claim	FX1, FX2, FX3, FX4, FX5	All claim details (id, ExpenseProof, amount, etc.) are stored in ExpenseClaim class as attributes
Expense Claim	FX6, FX7	Every ExpenseClaim type instance has a Status (either Pending, Approved or Rejected), amount and currency
Expense Claim	FX8-19	Every subclass of ExpenseClaims has additional information (based on the type of claim), all stored as attributes in the relative subclass.
Expense Proof	FX20, FX21	Every ExpenseProof has an ID number and associated claim, as well as an (optional) typed in VAT value.
Report	FX24, FX25	Every Report has the associated claim (hence the claim ID) and the associated Employee/Line Manager who submitted/processed the claim. Each Report also has a Reason type attribute which identifies the claims who were reported for Fraud Suspicion or for Incorrect Information.

3. Design Discussion

The *Employee* class is an abstract class that represents all the common behaviours and attributes that the subclasses, LineManager, InternalStaff and Consultants, all have. This means that rather than creating instances of Employee, only Employee subclasses will be instantiated instead (Line Manager, Internal Staff and Consultant). The common attributes of all the Employee-type classes are a list of claims submitted and a list of personal information, while some common operations include: filtering claims, changing password, makeExpenseClaim, etc. The Employee class stores the employee's login details (email and password), first and last name, role in the company, personal reliability score and allowance given by the company.

The *Employee* class has a specific operation, (+getPersonalDetails():List<Object>), which conveniently returns all the Employee attributes as a List of Objects, mainly to make the displaying of personal information easier. Since the Employee subclasses have additional attributes, though, this method is overridden by both the LineManager and Consultant subclasses of Employee, as both have extra attributes (namely the list of employees and employee claims for LM, currency and location for Consultant) which are not depicted in the general superlcass Employee.

One main difference from the domain model in the Domain analysis report is the implementation of the Admin class. This class has been added to properly account for system maintenance and the requirements FA1-4 in the Requirements Elicitation Report. In the original Domain analysis report, in fact, administration roles were simply mentioned as non primary actors, but have now been properly implemented in the system, as they will use the backend of the system to add and remove Employee-type accounts in/from the system.

ExpenseClaim is an abstract class containing all the basic and general information about each claim type, such as the date of the claim submission, the id of the claim, the amount of the expense and more. Each ExpenseClaim-type entity has a ExpenseProof type attribute to store the information about the expense proof (id, jpg/pdf file, optional manually inserted VAT). Additionally, we have introduced an Enumeration class (Status) for:

- **Status Enum** → status attribute (-status:Status), identifies whether a claim has been approved, reported or is still pending. The Status enumeration enables us to efficiently distinguish between these three status types and enables us to easily interchange between the given values.

Similarly to the Employee subclasses, the ExpenseClaim class has an operation to get all the information of an ExpenseClaim (+getClaimDetails():List<Object>) as a list of Objects, which is overridden by each subclass of ExpenseClaim due to the presence of extra attributes depending on the type of Expense Claim submitted.

The Report class represents all claims that have not been approved by the Line Manager(s), as well as the reason and optional further details on the rejection. In particular, the reason attribute is a Reason-type instance, which we have introduced as an enumeration (Reason) class:

- **Reason Enum** → reason attribute (-reason:Reason); this was implemented because the system supports two types of rejection reasons: IncorrectInformation, for which reports are not sent to HR for review and which weigh less on the Employee's reliability score, or FraudSuspicion, for which reports are sent to HR and which weigh more on the employee's score. This distinction was implemented in accordance with the client following the Requirements Elicitation stage of the project.

Another difference from the original domain analysis class diagram is the changed naming for the (originally) Database class to Registry. We have chosen to change names to avoid any confusion during the implementation stage (as Database was not a proper and correct entity to include in the class diagram), as well as to showcase the automatic features in the system, like generation of IDs and expenditure sheets, automatic notification of HR and Finance and more.

Note that some operations in the classes in the diagram, like basic getters for some attributes (like name, role for Employee), were omitted at this Design stage because they are not essential to the operations currently being described. If needed, they will be included during the Prototype stage.

Design patterns discussion

We chose to apply the Singleton pattern into our class diagram, namely for the Registry class, which allows us to ensure that there is only one single registry instance within the entire system, thus removing the chances, for example, of duplicate employees or claims with the same id.

DESIGN DISCUSSION SUMMARY

1. **MAIN ASSOCIATIONS AND RELATIONS**

Association	Type	Explanation
Stores (Registry:Employee) → 1:* (one to many)	Composition	A single Registry (Database in domain analysis report) instance contains all instances of Employee subclasses, Report, and ExpenseClaim subclasses as a central location to store and organise all of them. These are all composition associations because the data stored for them is contained in the registry. If the registry is removed, so will the data stored in it
Stores (Registry:Report) → 1:* (one to many)	Composition	
Stores (Registry:ExpenseClaim) → 1:* (one to many)	Composition	
Stores (Registry:Admin) → 1:* (one to many)	Composition	
Claims (Employee: ExpenseClaim) → 1:* (one to many)	Association	Employees can submit many expense claims. Line managers can filter claims to narrow them down to the employee who submitted
Oversees (LineManager:Employee)	Association	The Line manager is in charge of “approving and reviewing expenses claims submitted by the

→ 1:* (one to many)		employees", in accordance with the domain analysis. Indicating that the Line manager is in charge of multiple employees, and multiple expense claims from those employees, giving us a 1:M relationship for both Line managers to employees, and line managers to expense claims
Reviews (LineManager: ExpenseClaim) → 1:* (one to many)	Association	
Files(LineManager: Report) → 1:* (one to many)	Association	Domain analysis report shows that line managers " if the line manager finds the claim inadequate they can reject the employee's request and file a formal report", indicating that the line manager will be able to report multiple expense claims, giving us a 1:M relationship
Proof of (ExpenseClaim: ExpenseProof) → 1:1 (one to one)	Composition	Each Expense Claim corresponds to exactly one Expense Proof for evidence. As specified in FX4, expense claims must have a proof of expense associated with them.

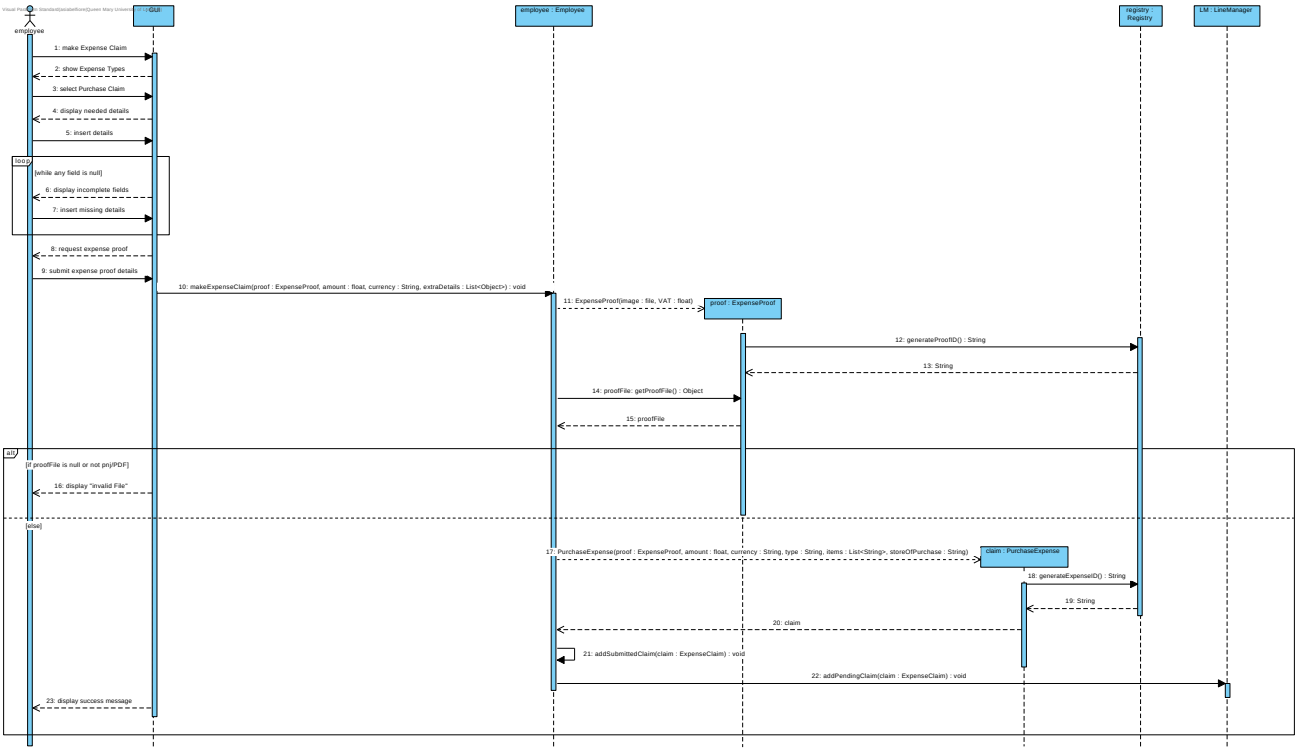
2. MAIN CHANGES FROM DOMAIN ANALYSIS

Changes from domain analysis	Explanation
Registry (was Database)	Singleton design implemented; Name changed to avoid misinterpretation
Status (Enum)	To ensure only certain values are taken
Reason (Enum)	To ensure only certain values are taken

4. Sequence Diagrams

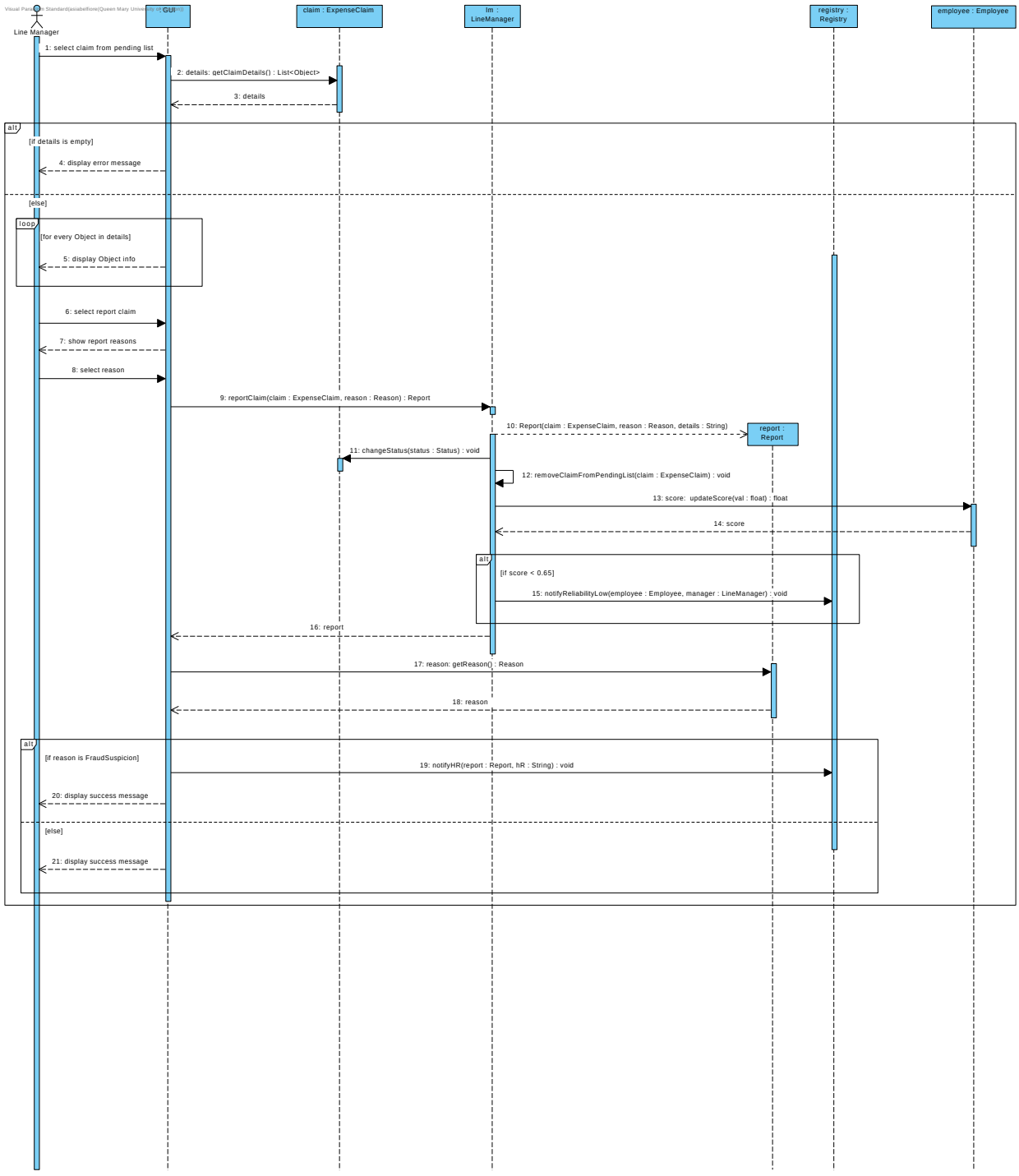
1. Submit (Purchase) Expense Claim

The Employee interacts with the system to submit an expense claim. As a prerequisite (in accordance with the Requirements Elicitation analysis), the Employee must be logged into the system and on the system's main page to begin this action.



2. Report Expense Claim

The Line Manager interacts with the system to report a claim submitted by an employee. As a prerequisite (in accordance with the Requirements Elicitation analysis), the Line Manager must be logged into the system and on the system's main page to begin this action.



3. Update Password

The Employee interacts with the system to change their password. As a prerequisite (in accordance with the Requirements Elicitation analysis), the Employee must be logged into the system and on the system's main page to begin this action.