CS 3307A Object-Oriented Design & Analysis

Design Rationale

The Dollar Tracker

Western University

Dr. Umair Rehman

Kareem Rahme	Ahmad Omran
251219015	251220813
krahme@uwo.ca	aomran3@uwo.ca

Table of Contents

Purpose of the Document

This document provides an overview of the principal design choices that were made in Deliverable 2 of *The Dollar Tracker* project. It details the structure, explains the rationale for the use of particular design patterns, discusses modifications made since Deliverable 1, and outlines the planned actions for Deliverable 3 in a nutshell. The intention is to offer a clear explanation of the system's structure and why these choices support maintainability, clarity, and scalability.

Architectural Overview

A. Layered Structure

The system is organized into two main layers:

- **UI Layer (`ui/`)** – Handles interaction with managers and employees, collects input, and presents data. A Qt-based user interface provides role-specific windows.

- **Core Layer (`core/`)** – Contains the system's logic, calculations, file access mechanisms, and data parsing.

This separation keeps the GUI independent from the underlying logic and prevents mixing interface code with domain-specific behavior. It results in cleaner code, easier debugging, and a structure that supports incremental expansion in later deliverables.

B. Text-Based Persistence

All data is stored in human-readable text files, such as:

- `emp_clock.txt`
- `Ids.txt`
- `jobs.txt`
- `mg_ids.txt`
- `PayStubs.txt`
- `schedule.txt`

The utilization of plain text files guarantees complete portability among different systems and also facilitates the verification and inspection of data by the instructors without having to run the program. This approach is simple, transparent, and appropriate for a project of this scale.

C. Consistent File Format

The system standardizes formats such as:

```
emp312, Service$20/H/
 emp312, 2025.11.09 -12:00:00/17:00:00
```

This reduces parsing complexity, maintains consistency across features, and allows new features to build on predictable data.

Use of Design Patterns

A. Singleton — FileGateway

`FileGateway` manages all file operations and is implemented as a Singleton, so the entire system interacts with a single, consistent file interface. This prevents conflicts from multiple components writing to the same files and also centralizes all I/O logic in one place.

Where: core/FileGateway.h/.cpp

B. Strategy-Like Behavior in Payroll

The payroll logic distinguishes between hourly ("H") and flat-rate ("F") roles. Each pay type has its own calculation rule. Even though the logic is not formally implemented via Strategy classes, it still adheres to the same design principle - different behaviors hidden behind a single interface.

C. Signal/Slot Mechanism (Qt Event Model)

The GUI uses Qt's signal/slot system to handle button clicks and other user actions. This keeps the windows loosely connected because each window only reacts to the specific events it needs. This setup works well with an MVC-style approach, where the UI triggers actions without tightly depending on other parts of the program. For example, when the user clicks '*clock In',* the button sends a signal, which then triggers a slot called `clockIn()` in the *EmployeeWindow.*

D. Model View Controller (MVC) Inspired Organization

Although simplified, the structure aligns with MVC principles:

Model

- Stores and manages the data and logic of the program
- Example: payroll calculations, employee roles, read/write files

View

- Everything the user sees - the GUI screens, buttons, and text fields.
- Example: LoginWindow, EmployeeWindow, ManagerWindow.

Controller

- Handles user actions and connects the ‘view’ to the ‘model’
- Example: When the user clicks `clockIn()`, which updates the Model (emp_clock.txt).

This is essential because it keeps the GUI separate from the logic; UI can be changed without breaking payroll logic, and the program becomes easier to extend and debug.

Changes Implemented Since Deliverable 1

A. Feature Additions

Deliverable 2 introduces several new capabilities in the program:

From the *ManagerWindow*:

- View Employees
- Add/remove Employees
- Add/remove Managers
- View Schedule
- Preview/export Payroll
- Add/remove/list/assign Roles
- Create Shift (date, start, end, employee ID)
- View Logs
- Filter Schedule by employee or date range
- Sign-out functionality

From the *EmployeeWindow:*

- View Shifts
- Clock-in and clock-out system
- Status function that displays the time when clocked in/ out.
- Sign-out functionality

These features demonstrate substantial expansion from the initial proposal.

B. Improved Domain Logic

Major improvements include:

- Rewriting the payroll engine for accurate hour calculation
- Parsing enhancements and defensive error checks
- Centralization of file operations

- Introduction of the data directory structure for consistent file locations

C. UI and Interaction Improvements

- Clearer window titles and button names
- Additional dialogs for viewing personal shifts
- Better separation of responsibilities between windows
- A more intuitive sign-out flow returning to login

Design Justifications

A. Why Text Files Instead of a Database

The use of text files corresponds to what the project needs, and this approach also minimizes dependencies, while making the grading and debugging og the system easy. The complexity is avoided, but the features required are still supported.

B. Why Centralized Payroll Logic

Locating payroll computation in a single class:

- Ensures consistency across preview and export functions
- Prevents duplicated logic in the UI
- Makes the feature easier to test and modify
- Allows additional pay structures to be added later

C. Why Separate Manager and Employee Windows

This cleanly enforces role-specific permissions and prevents accidental access to functions by the wrong user type. It also keeps the interface simple by showing only relevant options.

Video Walk-through Link

Link: https://youtu.be/gVKN-shCQIg

Milestone Plan

Deliverable 3 will focus on refinement, added features, UI improvements, and final testing to complete the system.

Refinement and Code Cleanup

- Add validation for IDs, rates, and shift data
- Expand comments and documentation
- Improve error messages and input handling

UI Enhancements

- Improved layout spacing
- Add table-based viewing for schedules
- Add confirmation popups for critical actions

Logic Extensions

- Support more pay types
- Add weekly or monthly payroll summaries
- Implement editing of shifts and roles

Testing Work

- Multi-user testing
- Edge cases, such as invalid logs or duplicate roles
- Repeated clock-in/out attempts

Implementing Additional Functions

- Automatic Overtime Calculations
- Export Schedule to file
- View Manager Activity Log
- Weekly/ Monthly Payroll Summary

Conclusion

Deliverable 2 gives *TheDollarTracker* a solid and manageable base. The system now supports all of the main features needed for an employee management and payroll application, and the architecture is modular with clearly defined roles. Singleton, strategy-like computation, and Qt's signal/slot mechanism are some of the design patterns that help create a clear and expandable structure. The system will be finished with refinement, interface polish, and other improvements in the upcoming deliverable.