Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Факультет инфокоммуникационных технологий

Домашняя работа.

«Разработка программного прототипа по проекту

инфокоммуникационной системы»

Выполнил: студент группы К4113с Тимошкина Анна Викторовна

Проверил: к. т. н., доцент ФИКТ   
Осипов Никита Алексеевич

Санкт-Петербург

2020 г.

Цель:

Реализация инфокоммуникационной системы учета рабочего времени.

Задачи:

1. Разработать уточненную диаграмму классов.
2. Применить шаблоны GoF в тех случаях, где это полезно.
3. Реализовать классы как можно подробнее.

Ход работы:

1. Применение порождающих шаблонов GoF.
   1. Abstract Factory (Абстрактная фабрика).

Разрабатываемая система на первоначальном этапе разработки предполагает использование двумя типами сотрудников предприятия: офисного отдела и производственного цеха. Но в дальнейшем это может измениться, и система должна поддерживать подобные изменения, например, если будет решено добавить другие должности или сотрудников, учет рабочего времени которых осуществляется по другим правилам.

Решить поставленную задачу помогает паттерн проектирования «Абстрактная фабрика», который позволяет создавать сочетающиеся группы объектов. Его использование упростит добавление новых типов объектов и позволит использовать их, не изменяя уже существующий код.

Для этого создается абстрактный класс-фабрика EmployeeFactory, работающий с абстрактными полями Card, Work, WorkRules. Реализация представленных классов осуществляется для двух типов сотрудников: офисных – OfficeEmployeeFactory, OfficeCard, OfficelWorkRules и производственных – ProductionEmployeeFactory, ProductionCard, ProductionWork, ProductionWorkRules.

* 1. Factory Method (Фабричный метод).

Добавление новых типов объектов не должно приводить к изменению в коде, что может произойти при создании новых экземпляров.

С данной проблемой позволяет справиться паттерн «Фабричный метод», входящий в состав Абстрактной фабрики.

Для этого в конструкторе абстрактной фабрики при заполнении полей используется не стандартная инициализация c new(), а абстрактный метод create(), который для каждого типа сотрудников возвращает требуемый тип объекта.

protected override Card createCard(int id)

{

return new ProductionCard(id);

}

* 1. Singleton (Одиночка).

Система будет поддерживать работу с базой данных, поэтому требуется обеспечить к ней доступ. При этом следует организовать единую точку доступа для предотвращения лишних соединений. Подобная единственность требуется и для самой системы учета времени, так как хранение локальных данных также должно быть централизовано.

Решить поставленную задачу помогает паттерн проектирования «Одиночка», который гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.

Для этого, в классах Database и TimeTrackingSystem требуется создать закрытый конструктор и экземпляр самого класса, обеспечив к нему доступ через свойство get.

1. Применение структурных шаблонов GoF.
   1. Adapter (Адаптер).

Терминал, необходимый для считывания данных с карты, предоставляет их в строковом формате, но разработанная система поддерживает работу с данными целочисленного типа.

Обеспечить работу двух несовместимых систем помогает «Адаптер».

Для этого создается класс AdapterHandler, который содержит метод работы fixGo(string id), конвертирующий идентификатор в требуемый формат и вызывающий метод работы системы, передавая ему преобразованное значение.

* 1. Façade (Фасад).

Поскольку система имеет сложную реализацию, а пользователь «умеет» только прикладывать смарт-карту к терминалу, то требуется скрыть он него внутреннюю реализацию.

Подобную задачу решает паттерн «Фасад», предоставляющий простой интерфейс к сложной системе классов.

Для этого требуется создать класс, содержащий методы, понятные для пользователя fixGo(), в которых будет происходить вызов остального функционала программы (addEmployee(), updateEmployee(), saveEmployee(), deleteEmployee).

* 1. Proxy (Заместитель).

Разрабатываемая система часто обращается к базе данных, проверяя наличие сотрудника в базе данных, получая информацию о нем или сохраняя данных. Следовательно, требуется разгрузить базу данных, используя коммуникацию с ней только для прямого назначения (передача данных).

Правильно организовать данный процесс позволяет паттерн «Заместитель», создающий вместо реального объекта объект-заместитель, выполняющий часть задач первого.

Для этого требуется создать отдельный класс ProxyDatabase, реализующий тот же интерфейс, что и реальный класс Database, но выполняющий в своих методах, помимо вызова замещаемого объекта, вспомогательные методы, обеспечивающие корректную работы, но не относящиеся к передаче данных (например, проверка соединения с базой данных или передаваемой информации).

1. Применение поведенческих шаблонов GoF.
   1. Chain of Responsibility (Цепочка обязанностей).

Прикладывая смарт-карту к терминалу, пользователь либо фиксирует время своего прихода, либо осуществляет обратную операцию. При этом данные действия для одного пользователя не могут проходить параллельно и вызывается только одно из них.

Следовательно эти операции можно поместить в одну цепочку обязанностей по обработке прикладывания карты. Шаблон проектирования «Цепочка обязанностей» позволит не только создавать и определять такую цепь, но и обеспечит на программном уровне определение того, кто сможет обработать данный запрос, что еще больше отдалит пользователя от реализации программы.

Для этого создаются 2 дочерних класса (HandlerIn, HandlerOut) от Handler, каждый из которых содержит ссылку на следующий элемент цепочки обязанностей и при невозможности обработать запрос, передает обработку ему.

* 1. Template Method (Шаблонный метод).

После прикладывания сотрудником смарт-карты на выходе с предприятия происходит процесс завершения работы, который состоит из нескольких шагов: фиксирование времени окончания работы и расчета требуемых параметров рабочего времени.

Пошаговую реализацию содержит и паттерн «Шаблонный метод», определяющий скелет алгоритма и позволяющий базовым класса переопределять входящие в него методы.

Для этого создается шаблонный метод saveWork() в абстрактном классе EmployeeFactory, включающий фиксирование окончания рабочего сеанса и расчет его параметров, который может быть изменен в любом из дочерних классов.

* 1. Strategy (Стратегия).

Расчет параметров рабочего сеанса зависит от типа сотрудника, выполняющего работу, и рассчитывается для каждой должности по-разному.

Обеспечить разные пути решений в зависимости от типа сотрудника, а значит и дочернего класса, позволяет шаблон «Стратегия», который позволяет решать одну задачу разными способами.

Для этого в абстрактном классе создается метод calculateWork(), который переопределяется в дочерних класса и выполняет задачу для конкретной ситуации наилучшим образом.

Вывод:

В ходе выполнения данной работы были изучены шаблоны проектирования GoF, позволяющие решать базовые задачи оптимально и с наименьшими затратами. Также использование шаблонов в большинстве случаем улучшает читабельность кода и обеспечивает масштабируемость программы, поэтому предполагая дальнейшее ее развитие, можно выбрать подходящий шаблон, который позволит произвести требуемые изменения с наименьшими затратами.

Стоит заметить, что в некоторых случаях эффективно именно совместное применение шаблонов проектирования, в частности одни шаблоны проектирования могут включать другие (самый простой пример: абстрактная фабрика – фабричный метод).

Диаграмма классов, разработанная согласно шаблонам проектирования GoF представленна в файле *Диаграмма классов.drawio*.