Экзаменационная работа

Студента групи КС-32

Безрука Юрия Руслановича

Билет №13

**1. Фази та дисципліни ЖЦ RUP**

Основным принципом RUP (Rational Unified Process) является принцип итеративной разработки в рамках которой разработка ведется в виде кратковременных мини-проектов фиксированной длительности (например, по 3-4 недели), которые называются итерациями. Каждая итерация состоит из собственных фаз анализа требований, проектирования, реализации, тестирования, интеграции и созданием рабочей версии программной системы (ПС). Такой итеративный цикл базируется на постоянном расширении и дополнении проекта ПС в процессе нескольких итераций с периодическим обратной связью и адаптацией дополнительных компонентов к существующему ядра ПС.

Основные принципы RUP:

1 — ранняя идентификация и непрерывное (до окончания проекта) устранение основных рисков

2 — концентрация на выполнении требований заказчиков к исполняемой программе

3 — ожидание изменений в требованиях, проектных решениях и реализации в процессе разработки

4 — компонентная архитектура, реализуемая и тестируемая на ранних стадиях проекта

5 — постоянное обеспечение качества на всех этапах разработки проекта (продукта)

6 — работа над проектом в сплочённой команде, ключевая роль в которой принадлежит архитекторам

Полный жизненный цикл разработки ПО состоит из 4 этапов:

1. Исходное стадия (Inception)

В фазе начальной стадии:

Формируются видение и границы проекта.

Создается экономическое обоснование (business case).

Определяются основные требования, ограничения и ключевая функциональность продукта.

Создается базовая версия модели прецедентов.

Оцениваются риски.

При завершении начальной фазы оценивается достижения этапа жизненного цикла цели, которое предусматривает соглашение заинтересованных сторон о продолжении проекта.

2. Уточнение (Elaboration)

В фазе «Уточнение» проводится анализ предметной области и построение исполняемой архитектуры. Это включает в себя:

Документирования требований (включая детальное описание для большинства прецедентов).

Спроектированную, реализованную и тестированы выполняемую архитектуру.

Обновлено экономическое обоснование и более точные оценки сроков и стоимости.

Снижение основных рисков.

Выполнение фазы разработки означает достижение этапа жизненного цикла архитектуры

3. Построение (Construction)

В фазе «Построение» происходит реализация большей части функциональности продукта. Фаза Построение завершается первым внешним релизом системы и вехой начальной функциональной готовности.

4. Внедрение (Transition)

В фазе «Внедрение» создается финальная версия продукта и передается от разработчика к заказчику. Это включает в себя программу бета-тестирования, обучение пользователей, а также определение качества продукта. В случае, если качество не соответствует ожиданиям пользователей или критериям, установленным в фазе Начало, фаза Внедрение повторяется снова. Выполнение всех целей означает достижение вехи готового продукта и завершения полного цикла разработки.

**2. Шаблон Iterator**

Iterator предоставляет способ последовательного доступа ко всем элементам составного объекта, не раскрывая его внутреннего представления, придает обходу коллекции "объектно-ориентированный статус", применяет полиморфный обход.

Iterator решает проблему "абстрактного" обхода различных структур данных так, что могут определяться алгоритмы, способные взаимодействовать со структурами прозрачно.

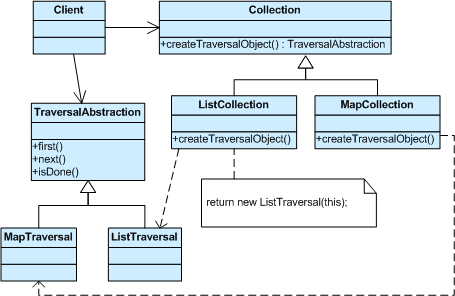
Составной объект, такой как список, должен предоставлять способ доступа к его элементам без раскрытия своей внутренней структуры. Более того, иногда нужно перебирать элементы списка различными способами, в зависимости от конкретной задачи. Но не хорошо раздувать интерфейс списка операциями для различных обходов, даже если они необходимы. Кроме того, иногда нужно иметь несколько активных обходов одного списка одновременно. Было бы хорошо иметь единый интерфейс для обхода разных типов составных объектов (т.е. полиморфная итерация).

Паттерн Iterator позволяет это делать. Ключевая идея состоит в том, что ответственность за доступ и обход перемещается из составного объекта на объект Iterator, который будет определять стандартный протокол обхода.

Абстракция Iterator имеет основополагающее значение для технологии, называемой "обобщенное программирование". Эта технология четко разделяет такие понятия как "алгоритм" и "структура данных".

Для манипулирования коллекцией клиент использует открытый интерфейс класса Collection. Однако доступ к элементам коллекции инкапсулируется дополнительным уровнем абстракции, называемым Iterator. Каждый производный от Collection класс знает, какой производный от Iterator класс нужно создавать и возвращать. После этого клиент использует интерфейс, определенный в базовом классе Iterator.

Пример диаграмы классов для шаблона Итератор:



На данной диаграмме показано, как различные варианты обхода инкапсулируются в объекты-наследники абстрактного обхода.

**3. Реалізувати дизайн згідно з завданням**

**Необхідно реалізувати компоненти інтерфейсу користувача для крос платформного додатки (може бути запущено під Linux, Windows). Зокрема такі компоненти як: Button, Label, Table, ImageView. Для відображення компонентів ви припускаєте використовувати API операційної системи.**

Для создания элементов интерфейса и удобной реализации их отображения на экране будет уместно применить шаблон проектирования GOF-коллекции Composite. Он заключается в том, что мы позволяем обращаться к составным элементам некоего семейства так же, как и к одному элементу. В данном случае понятно, что все графические компоненты должны реализовывать метод draw() для отрисовки самих себя, поэтому они все реализуют абстрактный интерфейс компонента. Реализуют его так же и группы компонентов – и кнопка и група кнопок должны являться графическим компонентом, точно такж же и с таблицей и ее элементами.

Для определения реакций на события нажатия кнопок можно использовать шаблон Observer. В этом случае кнопка будет выступать Publisher-ом, у которого будет список абстрактных подписчиков. При нажатии кнопки (click()), она оповещает своих подписчиков о событии вызовом update(). Каждый конкретный подписчик должен при этом реализовывать интерфейс абстрактного подписчика.

Примерная диаграмма классов для реализации задачи:

