Задание №3.
Агрегация

Цели задания:

- Изучить принципы агрегации классов, научиться различать агрегацию и композицию.

Данное задание логически продолжает предыдущую работу. Ранее мы узнали, что ***агрегирование*** – это взаимодействие двух объектов, при котором один объект является частью состояния другого класса. Агрегирование различается двух типов – композиция и агрегация. ***Композиция*** – связь между двумя объектами «часть-целое», при котором время жизни объекта-части совпадает со временем жизни объекта-целого. ***Агрегация*** – связь между двумя объектами «часть-целое», при котором время жизни объекта-части и объекта-контейнера отличаются. Другими словами, при уничтожении объекта-контейнера, объект-часть не уничтожается вместе с ним.

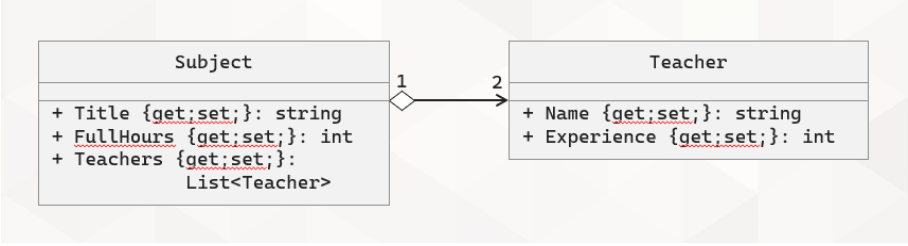
Агрегация является менее жесткой связью по сравнению с композицией, потому что агрегация не является определяющей связью для объекта-части. Например, связь между классом магазина Store и классом товара Item является определяющей – без магазина перечень товаров не имеет смысла, а потому при уничтожении магазина логично, если объекты товаров также будут уничтожены. Следовательно, это композиция. Но если мы создадим класс заказа Order, который формируется из товаров Item, то это связь неопределяющая – заказ может быть отменен, а объект класса удален. Но при отмене заказа уничтожать объекты товаров не логично. Более того, одни и те же товары могут одновременно находится в нескольких заказах. Таким образом, заказ хранит в себе товары (агрегирует), но их время жизни отличается – это агрегация.

С точки зрения синтаксиса, реализация агрегации ничем не отличается от композиции – мы также создаем поля типа одного класса в другом классе. Разницу можно определить, отследив в алгоритме момент создания и уничтожения объектов:

- Объект-часть создается совместно с объектом-контейнером?

- Объект-часть связан только с одним объектом-контейнером или может быть связан с несколькими контейнерами?
- При уничтожении объекта-контейнера объекты-части также будут (должны быть) уничтожены?

На диаграмме классов связь агрегации обозначается в виде сплошной незакрытой стрелки, направленной на композируемый класс. Со стороны композирующего класса на стрелке рисуется ***незакрашенный*** ромб (в отличии от композиции, где ромб должен быть закрашенным):



Для агрегации также указывается количественное соотношение объектов между собой – кардинальность связи – с помощью двух чисел над стрелкой.

Агрегация на примере создания класса Cart

1. Создать из ветки develop ветку Tasks/3_agregation, перейти в новую ветку.
2. Добавить в проект класс Cart. Класс должен описывать корзину товаров покупателя, которая потом будет использоваться для формирования заказов. В классе Cart должно быть поле списка товаров (Item) и соответствующее ему открытое свойство.
3. Добавить в класс Cart открытое свойство Amount, которое возвращает общую стоимость всех товаров в корзине. У свойства не должно быть сеттера. В геттере свойства должен быть организован цикл с подсчетом суммы стоимости товаров. Если список пустой или равен null, геттер возвращает 0.0.
4. Добавить в класс Customer поле типа Cart и соответствующее ему открытое свойство. У покупателя должна быть только одна корзина, экземпляр объекта должен создаваться в конструкторе класса Customer.
5. Теперь в нашей архитектуре организована связь между покупателями и товарами: объект покупателя ***композирует*** объект корзины, а корзина ***агрегирует*** список товаров.
6. Проверьте правильность оформления кода в классе Cart, добавьте xml-комментарии, сделайте коммит в репозиторий.

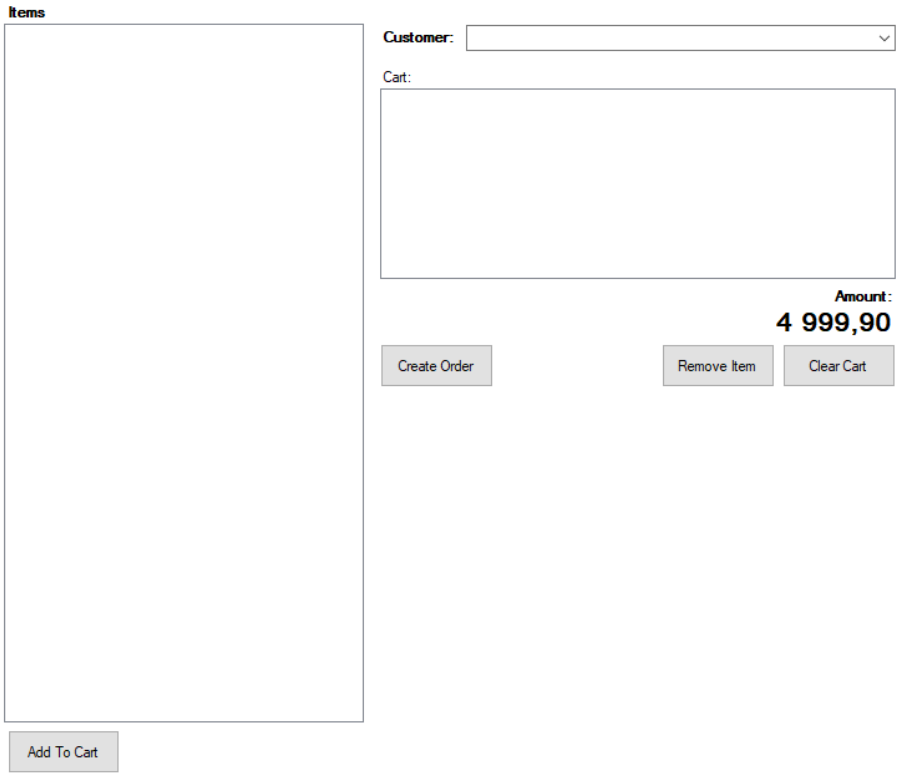
Агрегация на примере создания класса Order

1. Корзина позволяет выбрать товары для покупки, а далее на её основе должен формироваться заказ. Создадим класс, который отвечает за описание заказа.
2. Для начала создадим перечисление OrderStatus, описывающее состояние заказа в процессе его выполнения. Перечисление должно содержать следующие пункты:
    - New – Новый заказ.
    - Processing – Обрабатывается.
    - Assembly – Собирается на складе.
    - Sent – Отправлен.
    - Delivered – Доставлен.
    - Returned – Возврат.
    - Abandoned – Отменен (со стороны магазина).
3. Создайте класс заказа Order. Класс должен хранить следующую информацию: уникальный Id, дату создания заказа, адрес доставки, список товаров, общую стоимость. Id и дата создания должны быть доступными только на чтение. Общая стоимость должна быть аналогична общей стоимости из корзины.

4. Добавьте в класс покупателя Customer список его заказов (поле и свойство). Один покупатель может хранить в себе целый список заказов. Экземпляр пустого списка
5. Таким образом, класс Order связывает большое количество классов воедино: его композирует покупатель, сам заказ композирует статус и агрегирует список товаров.
6. Убедитесь, что код компилируется и работает верно.
7. Проверьте правильность оформления кода новых типов данных, добавьте xml-комментарии, сделайте коммит в репозиторий.

Пользовательский интерфейс для управления корзиной

1. В следующем блоке мы создадим вкладку для управления корзинами.
2. Создайте новый пользовательский элемент CartsTab, представляющий новую вкладку в главном окне.
3. Сверстайте элемент управления согласно макету (адаптивная верстка):



4. Добавьте в логику элемента управления два открытых свойства – список товаров Items и список покупателей Customers.
5. На главном окне создайте новую вкладку Carts и добавьте туда элемент CartsTab. В логике главного окна, после создания экземпляра Store и инициализации данных других вкладок, добавьте инициализацию данных элемента CartsTab – присвойте в свойства Items и Customers данные из объекта Store.
6. Таким образом, например, один и тот же экземпляр списка товаров одновременно хранится в Store, на вкладке Items и на вкладке Carts. Аналогично один и тот же экземпляр списка покупателей хранится в Store, CustomersTab и

CartsTab. Это создаёт проблему необходимости ручного обновления интерфейса, но эту проблему мы решим чуть позже.

7. Добавьте в логику класса CartsTab инициализацию левого списка ListBox товарами из Items.

8. Добавьте в логику класса CartsTab инициализацию выпадающего списка ComboBox покупателями из Customers.

9. Реализуйте логику, что при выборе покупателя в выпадающем списке, в правом списке ListBox показываются товары из корзины выбранного покупателя. Если покупатель не выбран, то список должен быть пуст. Для удобства дальнейшей работы с выбранным покупателем, храните его в дополнительном закрытом свойстве CurrentCustomer.

10. Реализуйте логику добавления выбранного товара в корзину с помощью кнопок, а также удаления товаров из корзины.

11. Реализуйте логику отображения общей стоимости корзины. Стоимость должна автоматически обновляться после добавления или удаления товара в корзину.

12. Реализуйте логику кнопки создания заказа Create Order. По нажатию на кнопку, программа должна создать экземпляр класса Order и поместить в него все товары из текущей корзины пользователя. Корзина пользователя при этом очищается. Новый Order также должен быть проинициализирован Id, датой создания, адресом доставки текущего покупателя и статусом New. Объект заказа помещается в список заказов пользователя.

13. Запустите программу и проверьте правильность работы. Проверьте правильность верстки и её адаптивность. Проверьте структуру проекта, правильность оформления кода и наличие всех комментариев.

14. Во время тестирования и отладки программы вы могли заметить одну проблему. Если запустить программу и добавить вручную товары на вкладке Items, а затем переключиться на вкладку Carts, то добавленные товары не будут отображаться на вкладке. Аналогичная ситуация со списком покупателей и даже товарами в корзине – если данные были изменены на одной из вкладок, то другие вкладки не знают об этих изменениях и продолжают отображать устаревшие данные. Необходимо придумать механизм, который бы выполнял обновление вкладки по новым данным. Попробуем реализовать его далее.

15. Добавьте в класс CartsTab открытый метод void RefreshData(). Метод должен заново перезаполнить левый список ListBox, показывающего товары, по списку товаров Items. Аналогично, должно происходить обновление для выпадающего списка покупателей, сброс выбранного покупателя или обновление правого ListBox, показывающего товары из корзины.

16. Теперь, вызывая у вкладки CartsTab метод RefreshData(), можно вызвать обновление интерфейса в случае изменения данных. Но кто и когда должен вы-

зывать этот метод? Вкладки ItemsTab и CustomersTab не могут вызывать данный метод, так как они никак не связаны с вкладкой CartsTab, а добавлять в них ссылку на CartsTab было бы ошибкой проектирования. К тому же обновлять данные на CartsTab после каждого добавления или удаления данных на вкладке ItemsTab или CustomersTab было бы избыточным.

17. Архитектурно все три вкладки связаны через два класса – класс бизнес-логики Store и класс главного окна MainForm. Бизнес-логика не должна ничего знать о пользовательском интерфейсе, иначе это будет нарушением принципа разделения Model-View. Следовательно, обновление данных на вкладке должно вызываться в классе главного окна.

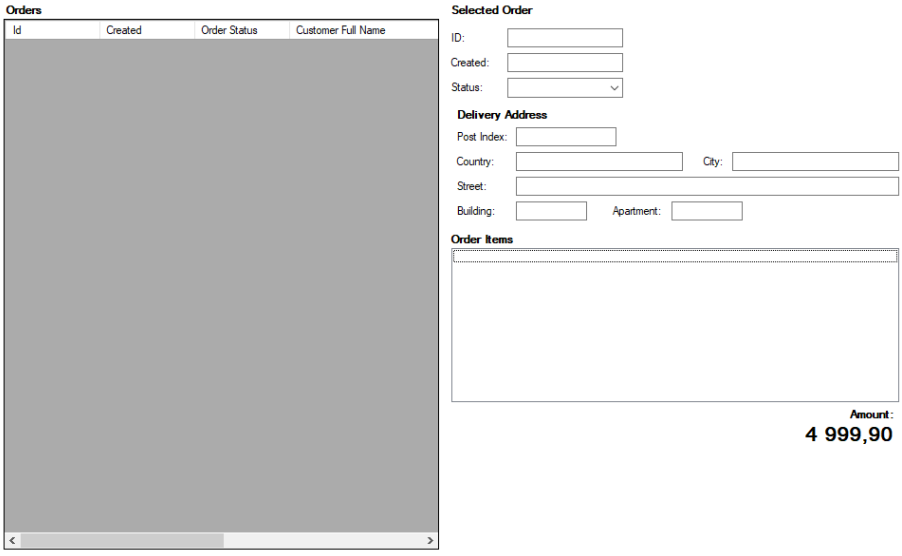
18. Теперь определимся с моментом. Обновлять данные на CartsTab после каждого добавления или удаления данных на других вкладках – слишком часто и создаёт лишнюю нагрузку на ПО. Данные достаточно обновлять тогда, когда пользователь переключается с любой другой вкладки на вкладку CartsTab. Реализуем эту логику.

19. В главном окне создадим обработчик события SelectedTabChanged (SelectedIndexChanged) для TabControl. Если пользователь сделал переход на вкладку Carts (например, SelectedIndex равен 2), то необходимо вызвать метод RefreshData у элемента CartsTab.

20. Если программа работает верно, сделайте коммит в репозиторий.

Пользовательский интерфейс для управления заказами

1. Создайте в проекте новый элемент OrdersTab, предназначенный для отображения всех существующих заказов в системе.
2. Сверстайте элемент согласно макету:



3. Для верстки элементов адреса доставки используйте уже готовый элемент управления AddressControl.
4. В отличие от предыдущих вкладок, для отображения данных здесь должен использоваться элемент DataGridView – таблица. Работа с таблицей может осуществляться двумя способами – без привязки данных (когда разработчик самостоятельно инициализирует все столбцы и строки таблицы) и с привязкой данных (когда все столбцы и строки таблицы формируются автоматически при изменении данных в бизнес-логике). Первый способ более простой в реализации, но требует большего количества кода. Второй способ более лаконичный, так как формирование таблицы выполняется автоматически, но концепция привязки данных более сложная для понимания. Подробно работа с DataGridView описана на сайте learn.microsoft.com [1].
5. В таблице должны быть показаны следующие столбцы: Id заказа, дата создания, ФИО покупателя, адрес доставки в виде одной строки, общая стоимость, статус.

6. В классе OrdersTab добавьте открытое свойство Customers (список покупателей). Главное окно при инициализации данных вкладок, должно передать на вкладку OrdersTab список всех покупателей из объекта Store.

7. DataGridView на вкладке OrdersTab должен предоставлять выбор только одной строки, режим мультивыбора (Multiselect) должен быть отключен. Также необходимо запретить пользователю создавать новые строки в таблице, менять высоту строк, переставлять столбцы, сортировать данные по клику на заголовок столбца. Сортировка запрещается по тому, что в случае сортировки в таблице могут перепутаться индексы для выбора текущего заказа и отображения его на панели справа. Если вы используете механизм привязки данных, можете не блокировать сортировку.

8. Реализуйте логику инициализации таблицы данных – необходимо перебрать всех пользователей в списке Customers и забрать у них все заказы. Заказы всех пользователей необходимо поместить в один общий список Orders, и на его основе создать строки в таблице DataGridView. Обновление списка заказов по списку Customers и данных в таблице поместите в отдельный метод UpdateOrders().

9. Реализуйте логику выбора текущего заказа. При выборе заказа в таблице, должна инициализироваться панель справа. Все отображаемые данные на правой панели, включая список товаров, должны быть доступны только на чтение – пользователь не может изменить поля. Единственное изменяемое поле – поле статуса заказа OrderStatus. При выборе нового значения в выпадающем списке, новое значение статуса тут же присваивается в статус выбранного товара.

10. Аналогично вкладке CartsTab, реализуйте метод RefreshData() на вкладке OrdersTab и вызов метода из главного окна.

11. Проверьте правильность работы программы, её верстки. Проверьте правильность оформления кода, наличия комментариев, структуры проекта. Сделайте коммит.

Проектная документация

1. В результате выполнения всех заданий, структура проекта должна быть примерно следующей:

2. По результатам всех заданий нарисуйте новую диаграмму классов.
3. На ней должны быть отражены новые перечисление и классы, а также связи между ними.
4. Для связей композиции и агрегации на диаграмме обязательно подпишите кардинальность. Обозначение агрегации на диаграммах классов см. в книге Буча «Язык UML. Руководство пользователя».
5. Диаграмму сохраните под названием «Practics3.*» в папке doc. Сохраните диаграмму как в формате программы, которую вы будете использовать, так и в формате png или jpeg, чтобы диаграмму можно было просмотреть онлайн через GitHub.

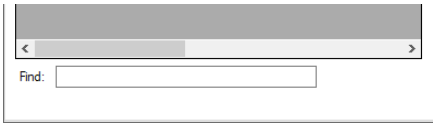
Дополнительные задания

1. Для удобства отладки, чтобы не вводить данные вручную, реализуйте генерацию случайных заказов у пользователей. Для вызова функциональности добавьте в интерфейс специальную кнопку на вкладке Customers. Аналогично можете реализовать генерацию случайных адресов доставки.
2. Аналогично AddressControl, созданному в прошлом задании, создайте отдельный элемент управления OrderControl.
3. Реализуйте сохранение и загрузку экземпляра Store в файл и из файла. При реализации данного пункта при агрегации возникает проблема: одни и те же экземпляры товаров одновременно хранятся в списке товаров в объектах Store, Cart и Order. То есть один объект товара может быть сразу в трёх списках. Обычная сериализация запишет эти данные в сохраняемом файле три раза. однако при загрузке данных из файла, десериализация может воспринять данные как три ***разных*** объекта, что может привести к неправильной работе другой функциональности.

Во-первых, это дублирование данных товаров, что само по себе делает информационную систему избыточной. Во-вторых, редактирование товаров на вкладке Items не будет приводить к обновлению товаров на вкладке Carts или Orders – они работают с разными объектами товаров. В-третьих, дублирование данных в дальнейшем затруднит реализацию сортировок, поиска, фильтраций и категоризаций товаров и других объектов.

Таким образом, сложность данного задания заключается в том, чтобы придумать такой механизм сохранения, когда при загрузке данных из файла, не создавались дубликаты исходных объектов в разных списках. Реализовать это можно двумя способами: а) настройки сериализатора от Newtonsoft позволяют для сериализуемых объектов сохранять id, по которым сериализатор сам понимает, что объекты дублируются; б) создать отдельную прослойку классов, отвечающих за сохранение данных (Data Transfer Object - DTO).

4. Сделать хранение истории смены статусов товара (добавить поле типа Dictionary<DateTime, OrderStatus>). С точки зрения бизнеса нужно отслеживать всю историю изменений статусов, чтобы видеть, на каком этапе происходят задержки. При создании нового заказа, вместо задания поля «Время создания», необходимо просто добавлять в словарь истории статусов новую запись с текущим временем и статусом «New». В таблице заказов показывать не время создания, а время последнего изменения статуса.
5. Если взглянуть на макет вкладки Orders, можно заметить, что под таблицей осталась небольшая свободная полоса на вкладке. Добавьте в это место следующие элементы и реализуйте поиск заказов по подстроке (например, имени покупателя, статусу или адресу доставки).



Если пользователь вводит подстроку в текстовое поле, то должны остаться только те заказы, которые соответствуют поисковому запросу. Если поисковая строка пустая, то показываются все заказы. Если какой-то заказ подходит под поисковый запрос сразу по нескольким критериям (например, подстрока встретилась и в имени покупателя, и в адресе), то этот заказ должен выводится в таблице только один раз. Убедитесь, что при выборе заказа в таблице среди найденных заказов, правая панель действительно инициализируется выбранным заказом, а не другим.

6. Реализовать сортировку в таблице заказов по Id и по общей стоимости. Сортировка должна работать по клику на имя столбца в таблице. Убедитесь, что при выборе заказа в таблице среди отсортированных заказов, правая панель действительно инициализируется выбранным заказом, а не другим.