Задание №1.
Классы, объекты и инкапсуляция

Цели задания:

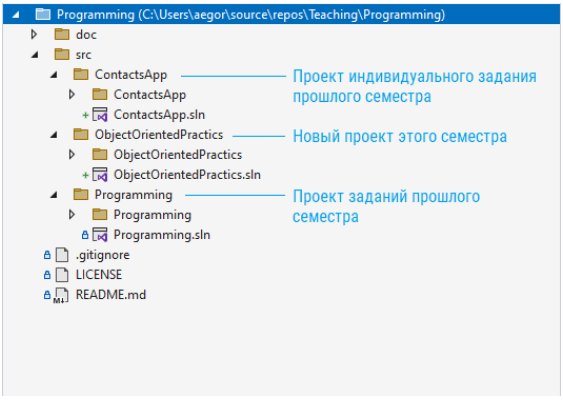
- Повторить принципы написания классов и их использования, включая абстрагирование, именование, оформление класса, автодокументирование.

- Освоить принципы рисования классов для UML-диаграмм.

В рамках первого задания мы начнем создавать приложение по управлению доставкой заказов. В рамках этого приложения мы будем разбирать механики объектно-ориентированного программирования. В этом задании мы подготовим ключевые классы приложения, а также пользовательский интерфейс. Вспомним принципы создания классов, инкапсуляции полей и методов, сокрытия реализации, валидации данных с помощью свойств, а также оформление и комментирование классов.

Создание классов

1. Склонировать ранее (в прошлом семестре) созданный для практических занятий репозиторий Programming на рабочий компьютер.
2. В репозитории в папке src создать новое решение ObjectOrientedPractics с одноименным проектом. Тип проекта Windows Forms Application. Таким образом, всего в папке src у вас должно быть три решения:



3. В проекте ObjectOrientedPractics создать папки Model, Services и View для классов бизнес-логики, сервисных классов и классов пользовательского интерфейса соответственно.
4. Класс главного окна Form1 переименовать в MainForm и перенести в папку View.
5. В папку Model добавить класс Item («Товар»). В классе должны быть следующие поля:
   - _id – целочисленное readonly-поле, хранящее уникальный номер товара.
   - _name – строковое поле с названием товара, до 200 символов.
   - _info – строковое поле с описанием товара, до 1 000 символов.
   - _cost – вещественное поле со стоимостью товара, от 0 до 100 000.
6. Поля класса должны быть закрытыми, а доступ к значениям полей и их валидация должна быть реализована через свойства.
7. Генерацию уникальных Id для товаров можно реализовать либо с помощью статического поля-счетчика внутри класса Item, либо с помощью созданного сервисного класса IdGenerator с методом GetNextId().
8. Добавьте в класс Item конструктор по трём аргументам name, info и cost. Важно: конструктор должен использовать свойства для присвоения значений в поля, а не прямое обращение к полям.
9. В папку Model добавить класс Customer («Покупатель») со следующими полями:

a. _id – целочисленное readonly-поле, хранящее уникальный номер товара.
b. _fullname – строковое поле с полным именем покупателя (Фамилия имя отчество), до 200 символов.
c. _address – строковое поле с адресом доставки для покупателя, до 500 символов.

10. Аналогично классу Item, реализуйте закрытые поля, свойства с валидацией и конструктор класса. Для генерации id можно использовать один класс IdGenerator, генерируя общие (сквозные) идентификаторы в программе.

11. Так как валидация строковых полей в обоих классах одинаковая, создайте сервисный класс ValueValidator с методом AssertStringOnLength(string value, int maxLength, string propertyName). Метод проверяет строку на длину, и, если строка длиннее заданного maxLength, выбрасывает исключение. Текст исключения должен формироваться на основе имени свойства, в котором его вызвали (propertyName) и значения максимальной длины, то есть, если метод вызван из сеттера свойства Fullname, тогда текст сообщения должен быть «Fullname должен быть меньше 500 символов». Используйте сервисный класс во всех строковых свойствах обоих классов.

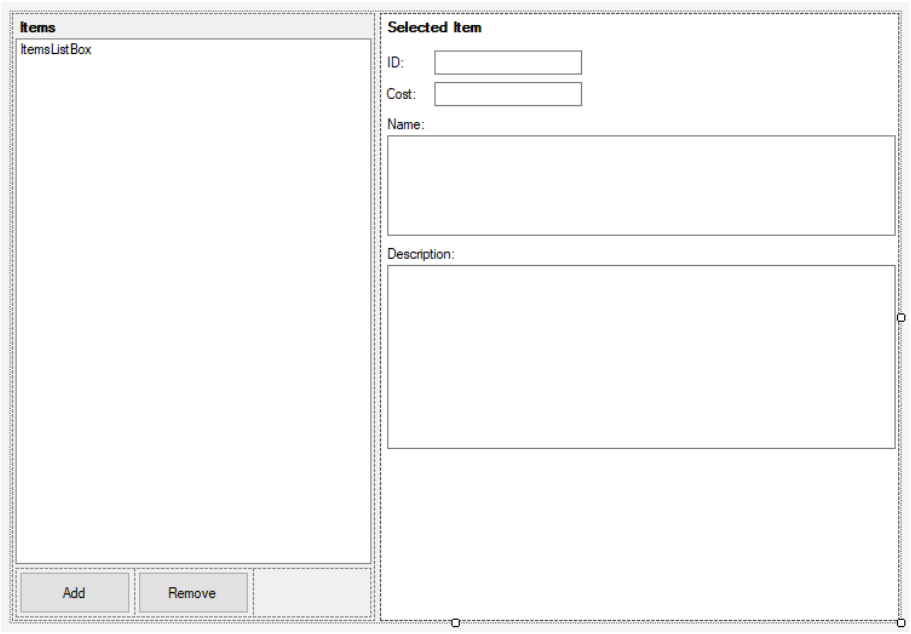
12. Убедитесь, что все члены созданных классов расположены в правильном порядке: константы, статические поля, readonly-поля, обычные поля, свойства, конструкторы, открытые методы, закрытые методы.

13. Убедитесь, что названия классов, папок, пространств имен, а также членов классов соответствуют заданию и требованиям RSDN. Проверьте именование полей, свойств, методов, локальных переменных.

14. Добавьте xml-комментарии для созданных классов: комментарии для классов, полей, свойств и других методов. Текст комментариев должен соответствовать ***ранее изученным правилам комментирования***.

Создание пользовательского интерфейса

1. В папке View создайте подпапку Tabs. Добавьте в папку Tabs новый пользовательский элемент управления ItemsTab.
2. Сверстайте ItemsTab так, как показано на рисунке.

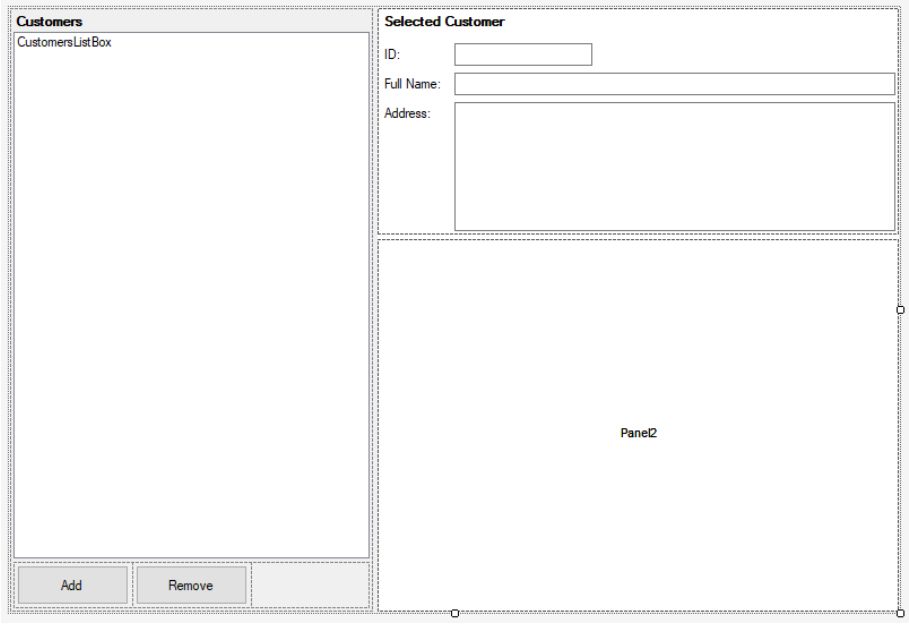


3. При верстке не забудьте о: отступах между элементами; отступах от краев элемента; выравнивании элементов по направляющим относительно друг друга; адаптивной верстке; переименовании элементов управления.
4. В главном окне MainForm добавьте элемент управления TabControl с одной вкладкой «Items». На вкладку Items добавьте экземпляр ItemsTab, примените свойство Dock в значение Fill как для TabControl, так и для ItemsTab (сделайте элемент управления автоматически растягивающимся вместе с родительским элементом). Запустите программу, убедитесь, что верстка выполнена правильно.
5. Добавьте в класс ItemsTab поле List<Item> _items = new().
6. По аналогии с заданиями прошлого семестра, реализуйте на вкладке ItemsTab логику создания новых товаров и добавление их в ListBox, логику выбора и редактирования товара в списке ListBox, логику удаления.

7. Текстовое поле для Id должно быть доступным только для чтения в пользовательском, текстовые поля Name, Info и Cost должны обеспечивать валидацию – в случае ввода неправильного значения поля должны подсвечиваться красным фоновым цветом.

8. Запустите программу и убедитесь, что функциональность вкладки работает корректно.

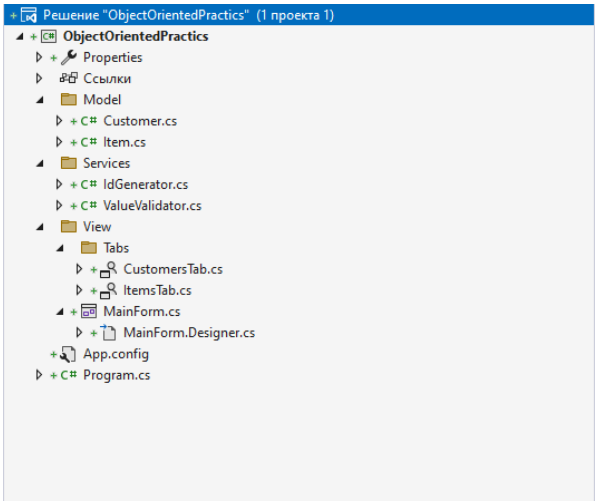
9. По аналогии с ItemsTab создайте пользовательский элемент управления CustomersTab и вкладку «Customers»:



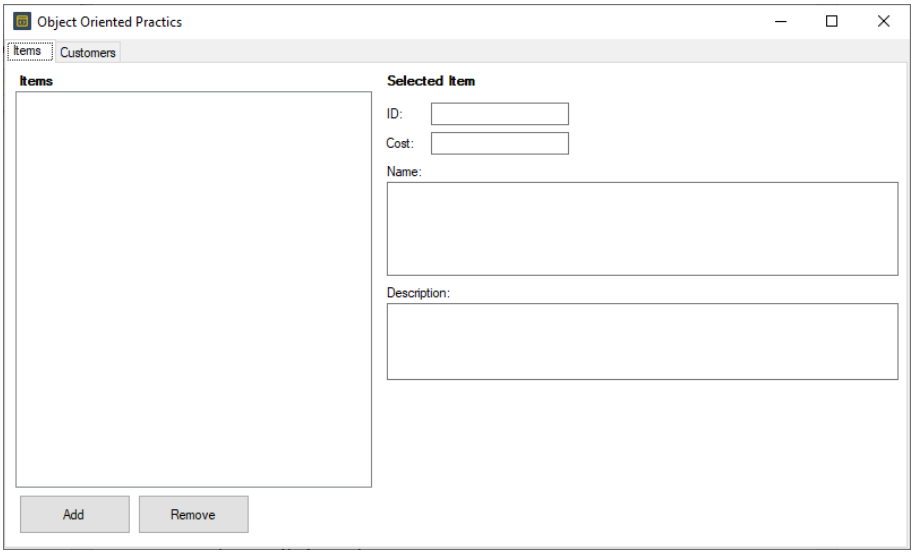
10. Проверьте правильность оформления классов пользовательского интерфейса – именование, порядок членов класса, наличие комментариев.

11. Добавьте иконку главному окну, исправьте название главного окна на “Object Oriented Practics”.

12. Структура проекта после выполнения всех заданий должна выглядеть примерно следующим образом:



13. Пример верстки главного окна после выполнения всех заданий:



14. Не забывайте делать коммиты в репозиторий, чтобы не потерять написанный код.

Проектная документация

1. Для разработанных вами классов бизнес-логики (Item, Customer, ValueGenerator и, возможно, IdGenerator) нарисуйте UML-диаграмму классов.
2. Проконтролируйте, чтобы на диаграмме были показаны все члены каждого класса.
3. Проконтролируйте, что статические поля и методы правильно нарисованы на диаграмме класса (обозначаются подчеркиванием).
4. Проконтролируйте, что связи между классами нарисованы верно. В данном задании, классы Item и Customer используют сервисные классы ValueGenerator и IdGenerator. Укажите правильное направление связи и само её обозначение.
5. Для создания диаграммы классов используйте десктоп-приложения StarUML, Sparx Enterprise Architect или онлайн-редактор, например, draw.io. Рекомендуется использовать десктоп-приложения, так как не все бесплатные онлайн-версии редакторов позволяют сохранять диаграммы и вносить в них изменения после, например, в случае если вы получите замечания по диаграмме.
6. Созданную диаграмму сохраните в репозитории. Для этого рядом с папкой src создайте папку doc, а в ней – подпапку ObjectOrientedPractics. Диаграмму сохраните под названием «Practics1.*». 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. Во время отладки программы вам придется постоянно создавать множество товаров и покупателей. Чтобы сэкономить время для себя во время разработки и во время сдачи лабораторных работ преподавателю, создайте классы ItemFactory и CustomerFactory, выполняющих генерацию случайных товаров и покупателей. Генераторы должны вызываться с помощью дополнительных кнопок под ListBox на вкладках соответствующих данных.

Рекомендуем использовать не генерацию случайных строк, а использовать реальные названия и описания товаров, реальные имена и адреса. Описания товаров можно найти в любом интернет-магазине, имена и адреса можно найти по поисковому запросу «генерация пользовательских данных». Например, сайт https://randomdatatools.ru/ позволяет генерировать полный набор пользовательских данных, включая адрес. Использование таких данных сделает тестирование программы более приближенным к реальным условиям.

2. Опять же, чтобы упростить дальнейшую отладку программы и не вводить данные каждый раз снова и снова при каждом запуске программы, вы можете

реализовать сохранение и загрузку данных в файл, аналогично заданиям с прошлого семестра. Для этого подключите библиотеку Newtonsoft JSON .NET (или другую библиотеку сериализации), и создайте сервисный класс ProjectSerializer.