Задание №5.
Полиморфизм

Цели задания:

- Изучить механизм полиморфизма объектно-ориентированного программирования.

Полиморфизм – возможность работы с объектами с общим интерфейсом, но без знания о том, какой объект скрывается за интерфейсом. Классы, реализующие общий интерфейс, называются **полиморфными классами**.

Полиморфные классы реализуются с помощью виртуальных функций и их переопределения. **Виртуальная функция** – это метод базового класса, реализация которого может быть изменена в дочерних классах. Для того, чтобы сделать метод класса виртуальным в базовом классе, необходимо после модификатора доступа указать ключевое слово virtual. Для того, чтобы переопределить реализацию виртуального метода в дочернем классе, необходимо создать метод с аналогичной сигнатурой (именем, набором входных аргументов и выходным типом данных), при этом после описания входных аргументов указав слово override. Далее в фигурных скобках необходимо описать новую реализацию метода.

Иногда в базовом классе не нужна реализация виртуальной функции, а нужно только её объявление – например, если мы создаем виртуальную функцию только для переопределения в дочерних классах, но не планируем её использовать как часть базового класса. Для этого можно создать **чисто виртуальные функции** – виртуальные функции, не имеющие реализации в базовом классе.

В данном задании мы рассмотрим механику полиморфизма, добавив в приложение систему скидок. Магазины и доставки создают разные виды скидок: накопительные баллы, процентные скидки, сезонные скидки, скидки в честь дня рождения и т.п. Каждая скидка имеет разную расчетную механику. Благодаря инкапсуляции, механизм расчета каждой скидки может быть скрыт внутри отдельного класса, в то время как пользователи класса будут работать со всеми видами скидок через единый интерфейс. В качестве единого интерфейса (описания открытых методов класса) можно использовать либо абстрактный базовый класс (abstract class), либо специальную сущность языка C# (interface).

Реализуя общий интерфейс, мы можем использовать разные классы через ссылку на базовый класс (или ссылку на интерфейс).

Таким образом, полиморфизм позволяет избавиться от дублирования ***в клиентском коде***. Это важное отличие от наследования, которое позволяет избавиться от дублирования при ***создании новых классов***. Если мы напишем код, который работает с интерфейсом скидок, в дальнейшем мы сможем добавлять новые классы скидок без изменений в клиентском коде – программа автоматически будет работать с любой новой скидкой.

Важный момент – как мы знаем, объекты любого класса создаются с помощью вызова конструкторов. Конструктор – это специальный метод класса, который обязан называться по имени класса. Это значит, что конструкторы классов нельзя вынести в общий интерфейс. Другими словами, создание объектов полиморфных классов всегда будет оставаться не полиморфным. Весь остальной код, который работает с интерфейсом, будет общим для всех классов. В том числе и деструкторы – в языках Си++, C#, Java разработчик не вызывает явно деструкторы классов. Они вызываются автоматически при освобождении памяти – с помощью оператора delete в Си++ или сборщиком мусора в C# и Java.

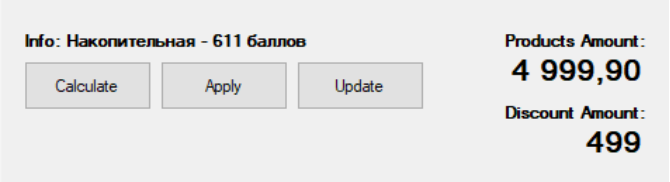
Полиморфизм является мощнейшим механизмом в ООП, возможности которого сложно осознать начинающим разработчиком. Технически, полиморфизм позволяет сначала написать клиентской код для использования классов, а затем писать реализацию классов. Например, с помощью полиморфизма можно реализовать систему плагинов – систему расширения функциональности приложения с помощью дополнительных модулей. Если в основном приложении реализовать общий интерфейс для плагинов, которые могут загружаться из произвольных dll-файлов, позже мы сможем в отдельном решении создавать плагины и добавлять их в приложении ***без перекомпиляции основного приложения***. Основное приложение будет работать с плагинами, ничего не зная об их реализации и не подключая их явно. Таким образом, любое приложение может расширяться другими разработчиками даже через десятки лет после его релиза.

Создание классов скидок

1. Создать из ветки develop ветку Tasks/5_polymorphism, перейти в новую ветку.
2. Сначала мы создадим отдельные классы скидок и с помощью специальной вкладки убедимся, что классы правильно работают. Затем мы создадим для них общий интерфейс и добавим их в основное приложение.
3. Создайте класс скидки накопительных баллов PointsDiscount («points» – баллы, «discount» - скидка). Скидка должна хранить количество накопленных баллов. Количество баллов должно быть положительным, иметь открытый геттер и закрытый сеттер. Также в классе должны быть реализованы два метода:
   - public double Calculate(List<Item> items), который принимает на вход список продуктов и возвращает размер скидки, доступной для этого списка продуктов с текущим количеством баллов.
   - public double Apply(List<Item> items), который применяет скидку к товарам. Метод также должен вернуть размер скидки, но при этом списать накопленные баллы.
   - public void Update(List<Item> items), который добавляет баллы на основе полученного списка товаров.
4. Таким образом, перед оформлением заказа будет вызываться метод Calculate() для оценки возможной скидки. Метод Apply() будет вызываться в том случае, если скидка действительно применяется к будущему заказу. Метод Update() будет вызываться после каждой покупки, вне зависимости от того, была применена скидка или нет – с любого заказа покупатель должен получить баллы.
5. Скидка с накопительными баллами должна работать по следующим правилам:
   - Один балл – это один рубль скидки.
   - Скидка не может быть больше 30% от общей стоимости продуктов.
   - Если количество накопленных баллов превышает 30% от стоимости продуктов, то скидка предоставляется только на 30%. То есть и методы возвращают значение не больше 30%, и сумма баллов уменьшается не более чем на 30% от стоимости товаров.
   - Если количество накопленных баллов меньше 30%, то списываются все баллы.
   - Каждая покупка увеличивает количество накопленных баллов на 10% от общей стоимости товаров.
   - Количество баллов – это целое число. Если 10% от стоимости – это дробное число, то количество баллов округляется в большую сторону.

6. Добавьте строковое свойство Info, которое возвращает название скидки «Накопительная – {Баллы} баллов», где вместо {Баллы} подставляется текущее количество баллов в скидке. Свойство должно иметь только геттер.

7. Проверьте правильность работы класса. Для этого можете создать временную вкладку DiscountsTab. Убедитесь, что работают все вышеперечисленные правила расчета. Пример верстки элементов для DiscountsTab:



8. Теперь создайте класс процентной скидки на конкретную категорию товаров PercentDiscount. Класс должен хранить категорию товаров, на которую он предоставляется. Скидка может быть от 1 до 10%, накапливается на протяжении совершения покупок. Каждую 1000 рублей, на которую покупатель совершает покупки, скидка увеличивается на 1%. То есть, в самом начале скидка имеет значение 1%, но при покупке на 1000 рублей, скидка увеличивается до 2%. При покупке еще на 1000 рублей, скидка будет составлять уже 3%. Скидка не может быть больше 10%. При покупке учитываются только те товары, которые относятся к категории самой скидки.

9. Таким образом, класс должен хранить: 1) текущую скидку в процентах; 2) категорию товаров, на которую она предоставляется; 3) сумму, на которую покупатель уже сделал покупки данной категории товаров.

10. Аналогично PointsDiscount, добавьте в класс три метода Calculate(), Apply() и Update(). Методы также должны принимать список продуктов и рассчитывать на их основе размер скидки. Во время своей работы, методы должны перебирать список продуктов, находить среди них продукты той же категории, что и скидка, и применять текущий процент ***только*** к этим товарам. Если в списке товаров нет товаров нужной категории, то размер скидки равен 0.

11. Аналогично PointsDiscount, добавьте в класс строковое свойство Info, которое возвращает название скидки «Процентная «{Категория}» - {Процент}%», где вместо {Категория} подставляется название категории скидки, а вместо {Процент} подставляется текущее значение процента.

12. Проверьте правильность работы класса. Верстка аналогична верстке PointsDiscount.

13. Если оба класса скидки работают верно, проверьте оформление кода, правильность именования, наличие комментариев. Сделайте коммит.

Создание общего интерфейса

1. Мы не зря в реализации классов скидок старались обеспечить одинаковые открытые методы и их названия. Все методы и свойства, которые в обоих классах называются одинаково, имеют одинаковую сигнатуру, но разную реализацию, могут быть выделены в общий интерфейс.
2. Добавьте в проект интерфейс IDiscount:

```
public interface IDiscount
{
   string Info {get;}
   double Calculate(List<Item> items);
   double Apply(List<Item> items);
   void Update(List<Item> items);
}
```

3. Сразу обратим внимание на несколько моментов. Во-первых, для методов и свойств интерфейса не указываются модификатор доступа. Предполагается, что все свойства и методы интерфейса должны быть реализованы как открытые (public) методы в дочерних классах. Во-вторых, методы в интерфейсе не содержат никакой реализации. Предполагается, что интерфейс определяет только сигнатуру методов, но реализация остается в ответственности дочерних классов. В-третьих, название интерфейса принято начинать с заглавной буквы «I». Так, интерфейс можно будет отличить от обычных классов. Далее после буквы указывается слово или фраза, которое описывает назначение интерфейса. Так как мы создаем интерфейс, который будет описывать какую-либо скидку, то подходящим названием будет IDiscount. Также в названиях любых классов, реализующих интерфейс IDiscount стоит в название добавлять слово «Discount» - это позволяет сразу по названию класса определять, какие классы реализуют тот или иной интерфейс.
4. В примере интерфейса выше не хватает xml-комментариев. Добавьте их.
5. Добавьте в объявление классов PointsDiscount и PercentDiscount так называемое ***приобретение интерфейса***:

```
public class PointsDiscount : IDiscount
```

Синтаксис аналогичен наследованию. Скомпилируйте программу. Если вы всё сделали правильно, и нет ошибок в названиях или сигнатуре методов (как в интерфейсе, так и в классах скидок), то проект скомпилируется без ошибок. Если же будут какие-то несоответствия, то компилятор сообщит что интерфейс IDiscount реализован не полностью.

6. Общий интерфейс предназначен для хранения объектов скидок по ссылке, аналогично ссылкам на базовый класс в предыдущем задании. В клиентском коде мы можем создать переменную типа IDiscount и поместить в неё объект PointsDiscount или PercentDiscount. Важно, что создать объект типа IDiscount мы не можем – это не класс, а описание интерфейса, и у него нет реализации, для которой можно вызвать конструктор.
7. Далее мы внедрим работу со скидками в основное приложение.

Хранение объектов по ссылке на интерфейс

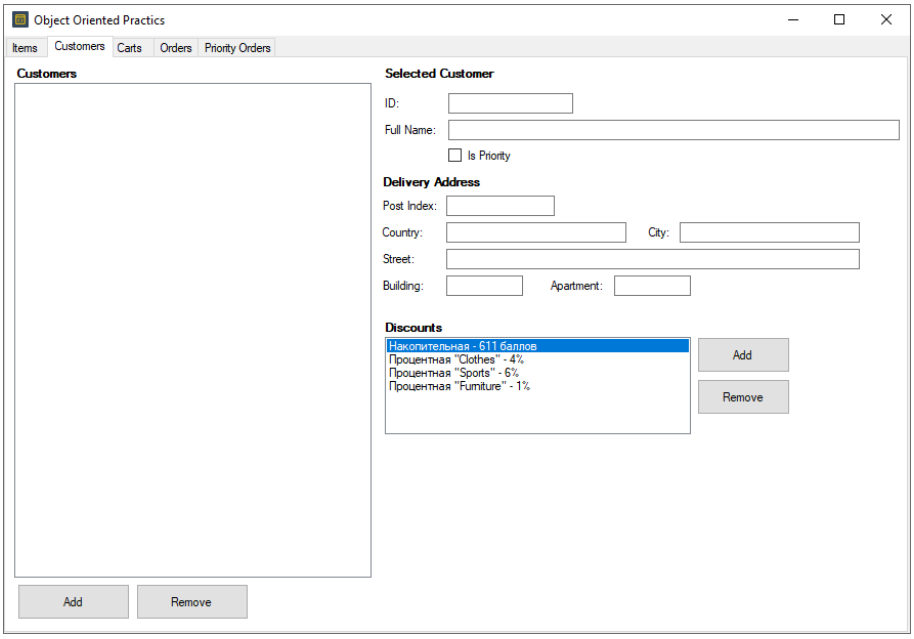
1. В основном приложении необходимо реализовать следующую механику. У каждого покупателя есть скидка с накопительными баллами. Кроме того, покупателю можно добавить произвольное количество процентных скидок на разные категории товаров. При оформлении корзины пользователь видит, какой бонус может дать каждая из скидок покупателя. Пользователь-оператор может выбрать, какие из скидок будут применены к корзине при создании нового заказа. Предполагается, что оператор связывается с покупателем по телефону для подтверждения заказа и озвучивает варианты скидок. Так как некоторые скидки могут не суммироваться или быть взаимоисключающими, покупатель выбирает нужный ему вариант, а оператор отмечает это в системе. После этого создается экземпляр заказа, в котором дополнительно хранится размер скидки.
2. Так как у одного покупателя может быть много скидок, добавьте в класс Customer свойство Discounts типа List<IDiscount>. Все скидки будут храниться списком по ссылке.
3. В конструктор (конструкторы) класса Customer добавьте инициализацию списка List<IDiscount> и добавьте в него экземпляр PointsDiscount. Скидка с накоплением баллов должна создаваться вместе с каждым покупателем.
4. Добавьте в класс Order открытое вещественное свойство DiscountAmount (размер примененной скидки). В него должен будет помещаться размер примененной скидки при создании заказа. Также добавьте вещественное свойство Total (конечная стоимость заказа). Оно должно возвращать не просто сумму стоимости всех товаров, но и вычитать из неё значение DiscountAmount. Свойство Amount, показывающее сумму всех товаров заказа, должно остаться без изменений.
5. Количество классов в папке Model становится слишком большим, что мешает навигации. Необходимо сделать более понятную структуру в проекте. Во-первых, создайте в папке Model подпапку Discounts и переместите туда классы PointsDiscount и PercentDiscount. Как правило, в подпапку помещают реализации интерфейса, а сам интерфейс остается в папке выше. Во-вторых, создайте

подпапку Orders и переместите туда классы Order и PriorityOrder. Иерархии наследования также можно поместить в подпапку. В-третьих, создайте подпапку Enums и переместите туда перечисления Category и OrderStatus. Перечисления могут размещаться либо в отдельной подпапке, либо рядом с классами, в которых они используются. Например, перечисление OrderStatus логично расположить как в подпапке Enums, так и в подпапке Orders.

6. После того, как классы были перемещены в подпапки, необходимо поменять указанные в них пространства имен. Зайдите в каждый из перемещенных классов и в строке объявления пространства имен укажите имя, соответствующее папке. Например, пространство имен для класса PriorityOrder должен смениться с namespace ObjectOrientedPractics.Model на namespace ObjectOrientedPractics.Model.Orders.
7. После изменения пространств имен в некоторых классах также надо будет поменять using – нужно добавить ссылки на правильные пространства имен, чтобы правильно обращаться к перемещенным классам. Такие изменения будут как классах бизнес-логики, так и в пользовательском интерфейсе.
8. Логика приложения готова, теперь надо реализовать часть пользовательского интерфейса. Проверьте оформление кода, наличие комментариев. Сделайте коммит.

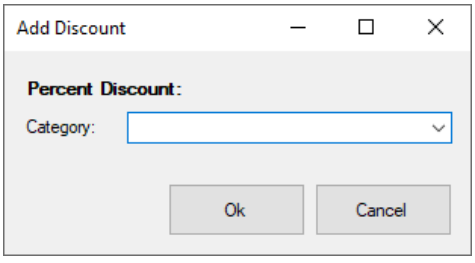
Создание пользовательского интерфейса

1. Изменения в пользовательском интерфейсе будут состоять из двух частей. Первая – это возможность удаления и добавления скидок покупателю на вкладке CustomersTab. Как говорилось ранее, создание объектов – это единственная операция, которая не может быть полиморфной и требует уникального кода для каждого полиморфного класса. То есть, для вкладки CustomersTab нормально, если в ней будет использоваться не только интерфейс IDiscount, но и явно упоминаться классы PointsDiscount и PercentDiscount для создания их объектов. Вторая – это применение скидок на вкладке CartsTab при оформлении нового заказа. Работа с уже созданными объектами скидок на этой вкладке должна делаться исключительно через интерфейс IDiscount без упоминания реализаций этого интерфейса. Если на вкладке CartsTab будут упоминаться конкретные классы PointsDiscount и PercentDiscount, это будет нарушением использования полиморфизма. Суть полиморфизма как раз и заключается в том, чтобы работать с объектами только через их интерфейс, не упоминая конкретных классов.
2. Добавьте на вкладку CustomersTab следующую панель для управления скидками:



3. Для каждой скидки, которая хранится в объекте покупателя, в ListBox должна отображаться строка со значением Info этой скидки. Накопительная скидка должна быть всегда самой первой.

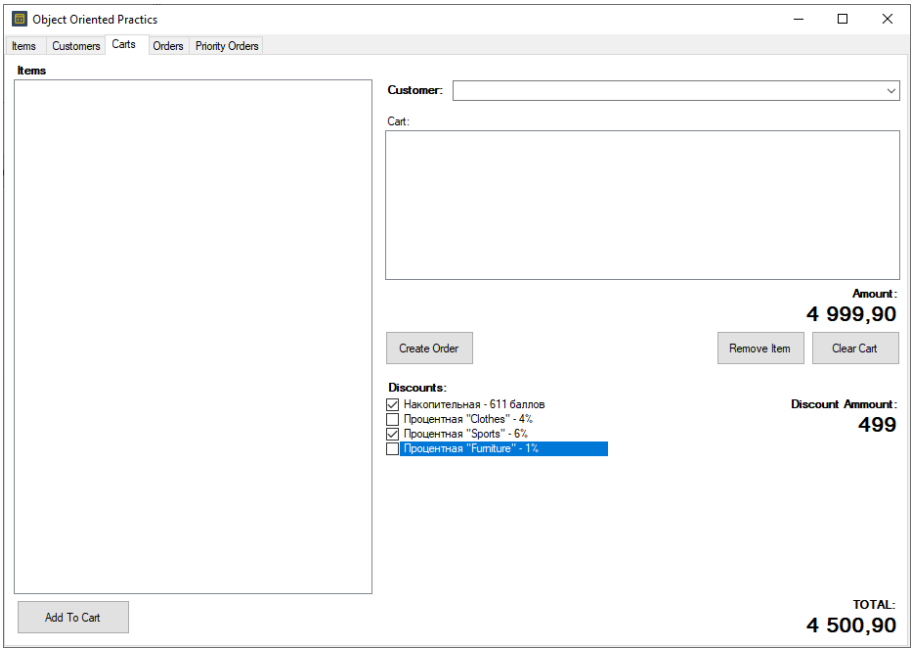
4. Реализуйте логику добавления новой процентной скидки покупателю. При нажатии на кнопку «Добавить» появляется всплывающее окно, в котором пользователь может выбрать категорию скидки. После нажатия на ОК скидка добавляется в общий список:



5. Реализуйте удаление выбранной скидки из списка. Накопительную скидку удалять нельзя – она обязательна для каждого покупателя.

6. Протестируйте приложение. Убедитесь, что скидки создаются и удаляются корректно. Проверьте оформление кода, верстку, именование элементов управления. Сделайте коммит.

7. Добавьте на вкладку CartsTab следующую панель для управления скидками:



8. Для отображения списка скидок используйте элемент CheckedListBox. При этом, чтобы он гармоничнее смотрелся в дизайне, можно убрать границу Border, фон сделать прозрачным. По высоте элемент сделать достаточным для отображения не менее 7 скидок. Также можете задать и другие свойства: CheckOnClick, SelectionMode, IntegralHeight.

9. Каждый элемент CheckedListBox показывает Info скидок выбранного покупателя. По умолчанию при выборе нового покупателя, все галочки всех скидок должны быть включенными – считается максимальная скидка.

10. При переключении галочек должно меняться значение предоставляемой скидки «Discount Amount» и значение итоговой стоимости «Total». Расчет предоставляемой скидки должен осуществляться на основе метода Calculate(), реализованного в скидках.

11. После нажатия на кнопку оформления заказа, для выбранных скидок выполняется вызов метода Apply(). После этого для всех скидок (вне зависимости от галочек) вызывается метод Update() – для начисления баллов или процентов с нового заказа. Значения Info в CheckedListBox обновляются согласно новым значениям накопленных баллов и процентов. Рассчитанное значение DiscountAmmount присваивается в новый заказ.

12. На вкладке OrdersTab добавьте новый столбец в таблицу Total, в котором должна отображаться итоговая стоимость заказа. Также добавьте значение итоговой стоимости и на панель справа.

13. Запустите программу, убедитесь, что программа работает корректно:

- У каждого нового покупателя всегда есть накопительная скидка.
- Каждому покупателю можно добавить или удалить скидки на конкретные категории товаров.
- При оформлении заказов можно выбирать нужные скидки.
- После оформления заказов баллы или проценты в скидках действительно накапливаются.
- и т.д.

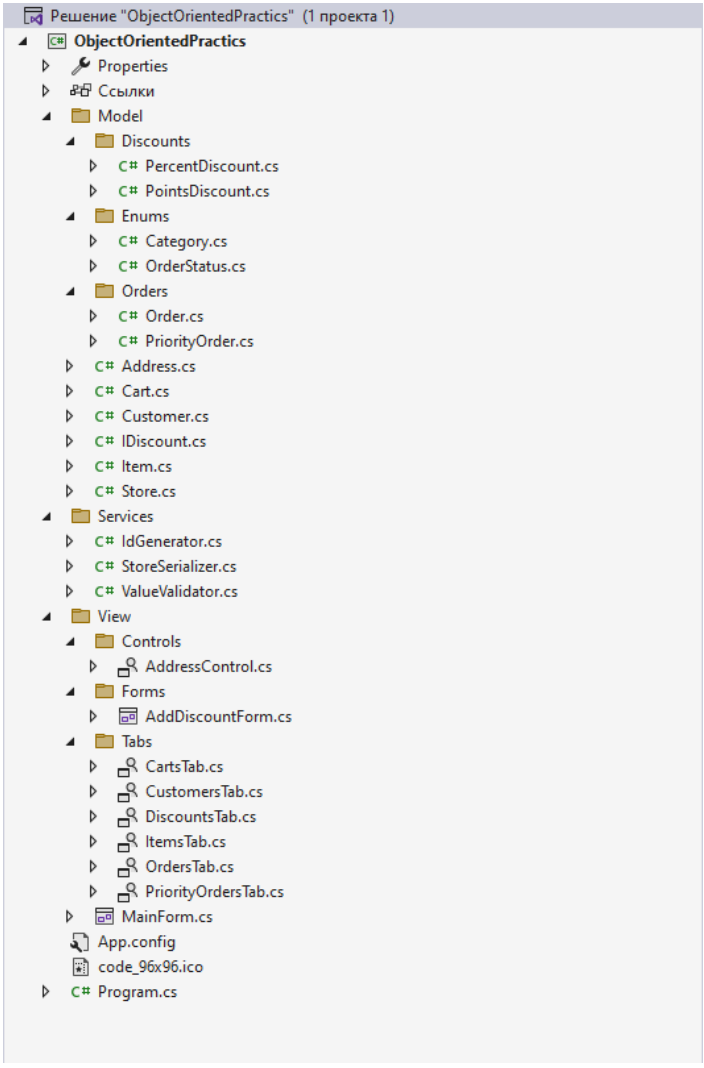
14. Убедитесь, что работает вся ранее реализованная функциональность. В процессе модификации программы могли сломаться другие части программы.

15. Убедитесь, что верстка верная, именование элементов управления и новых элементов в исходном коде правильное. Проверьте оформление кода и наличие комментариев. Сделайте коммит.

16. Стоит отметить, что предложенная реализация скидок подходит для данного задания, но ограничена в расширяемости. В реальных магазинах или доставках также существуют акции, бонусы, подарки, которые не только уменьшают стоимость товара, но могут увеличить количество накопленных баллов, добавить бесплатный подарок к заказу и т.д. Так как при проектировании приложения мы должны отталкиваться от требований ТЗ – в данном случае, для этих двух видов скидок решение вполне приемлемое. Для других требований реализация будет отличаться вплоть до отказа от общего интерфейса и полиморфизма. Важно понять, что полиморфизм применяется в тех случаях, когда у нас есть несколько классов, решающих одну задачу, но разными способами. Если классы не решают единой задачи, не надо стараться придумать для них общий интерфейс только ради полиморфизма. Полиморфизм должен решать архитектурные проблемы, а не создавать их.

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 классы, а также связи между ними.

4. Обозначение полиморфных классов на диаграммах классов см. в книге Буча «Язык UML. Руководство пользователя». Важно отметить, что связи «Полиморфизм» на диаграммах классов, но есть связь «Реализация». Реализация и полиморфизм – это не одно и то же. Реализация – это связь класса и интерфейса, при котором класс полностью или частично реализует методы, описанные в интерфейсе. Полиморфизм – это ***механика работы*** с полиморфных классов через общий интерфейс. Другими словами, реализация показывает как созданы полиморфные классы, а полиморфизм – как полиморфные классы используются в клиентском коде.
5. Обратите внимание, что новые поля и свойства добавились в ранее написанные классы – изменения должны быть показаны на диаграмме.
6. Диаграмму сохраните под названием «Practics5.*» 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.
7. Выполнить сливание текущей ветки с веткой develop.

Дополнительные задания

1. Реализуйте или исправьте ранее реализованный механизм сохранения и загрузки пользовательских данных. Если вы реализовывали сохранение и загрузку данных без использования сторонних библиотек, задача сохранения и загрузки списка List<IDiscount>, в котором одновременно хранятся объекты двух классов скидок, может оказаться нетривиальной. Кроме того, не все сторонние библиотеки умеют корректно сохранять наследуемые объекты. Библиотека Newtonsoft JSON.NET умеет правильно сохранять и загружать подобные коллекции, однако может потребоваться дополнительная настройка сериализатора – установить свойство ValueTypeHandling в значение All.
2. Добавьте в приложение скидку по дню рождения. День рождения должен храниться как поле у покупателя, но передаваться в качестве значения в объект скидки. Скидка по дню рождения составляет 10% на все продукты, применяется в течение одной недели до и после дня рождения, также является не удаляемой для покупателя. При реализации нового класса скидки обратите внимание, сколько изменений вам пришлось внести в остальное приложение, чтобы добавить новый тип скидки – изменений не должно быть совсем или они должны быть минимальными.