Задание №4.
Приёмы разработки классов

Закрытые методы класса:

1. В ранее созданный класс контакта (Contact) добавьте свойства Name и Surname, если они не были добавлены ранее.
2. Добавьте в свойства проверку, что строка содержит только символы английского алфавита.
3. Так как проверка в обоих свойствах требуется одинаковая, реализуйте проверку в виде закрытого метода AssertStringContainsOnlyLetters(string value) класса Contact, который будет вызываться в обоих свойствах.
4. Закрытый метод валидации имени и фамилии должен выбрасывать исключение типа ArgumentException. При этом, если метод вызывается в свойстве Name, то текст исключения должен указывать на некорректное значение в свойстве Name. Если же метод вызывается в свойстве Surname, то текст исключения должен указывать на некорректное значение в свойстве Surname. Подумайте, как это реализовать.

Сервисные классы:

5. Если рассмотреть ранее созданные классы, то в них часто реализуется проверка на неотрицательное значение. Так как одинаковая проверка реализуется не в одном классе, а в нескольких, проверку логично вынести не в закрытый метод каждого класса, а в отдельный класс.
6. Создайте в папке Model статический класс Validator. Добавьте в класс открытый статический метод AssertOnPositiveValue(int value). Метод должен выбрасывать исключение ArgumentException, если входное значение не является положительным. Также текст сообщения в исключении должен явно указывать название свойства, в которое пытались присвоить некорректное значение.
7. Замените все проверки на неотрицательные значения в ранее реализованных классах на вызовы класса Validator.
8. Запустите программу и убедитесь, что валидация в пользовательском интерфейсе (подсветка элементов управления) работает также корректно, как и до использования класса Validator.

9. Реализуйте перегрузку метода AssertOnPositiveValue(double value) для вещественных полей, и также замените проверки в исходных классах. Проверьте правильность работы валидации в пользовательском интерфейсе.

10. Аналогично методу AssertOnPositiveValue(), реализуйте метод проверки значения, входящего в диапазон AssertValueInRange(int value, int min, int max). Метод также должен сообщать о названии свойства, из которого его вызвали.

11. Замените в исходных классах все проверки диапазона значений на вызов метода AssertValueInRange(). Запустите программу и убедитесь, что валидация в пользовательском интерфейсе работает правильно.

Поля, доступные только на чтение

12. Создайте класс точки в двумерном пространстве Point2D, хранящей координаты X и Y. Реализуйте класс таким образом, чтобы поля координат X и Y были доступны только для чтения (их нельзя было изменить вне класса). Значения полей должны задаваться через конструктор и более не меняются. Другими словами, объекты класса Point2D должны быть неизменяемыми (immutable). Небольшие структуры и классы типа Point2D часто делают неизменяемыми.

13. Несмотря на то, что значения полей устанавливаются только в конструкторе и потом не меняются, поля всё равно должны проходить проверки (например, это может быть проверка на неотрицательность координат, или координаты не должны превышать каких-нибудь значений, если мы говорим о координатах объектов на картах, игровых объектов или объектах на чертежах и 2D- 3D-моделях). Для этого свойства для полей X и Y также должны обладать сеттерами, но только с модификатором доступа private. Конструктор класса должен инициализировать поля через сеттеры.

14. В ранее созданный класс Rectangle добавьте свойство Center типа Point2D (свойство не требует валидации), и добавьте инициализацию точки в конструктор класса Rectangle.

15. Модифицируйте пользовательский интерфейс программы таким образом, чтобы при выборе прямоугольника для него в отдельных текстовых полях отображались значения координат центра прямоугольника.

16. Текстовые поля для центра прямоугольника не требуют дополнительной валидации, так как являются доступными только для чтения (пользователь не сможет их изменить для существующего прямоугольника).

17. Так как поля центра прямоугольника являются доступными только для чтения, необходимо запретить изменение текста в текстовых полях. Самый простой способ – установить текстовым полям свойство Readonly в true. WinForms реализует такие свойства не у всех элементов управления, поэтому в таких случаях приходиться либо устанавливать свойство Enabled в значение false

(простой, но не корректный способ), либо использовать такие обработчики событий, как KeyPressed для текстового поля, чтобы отменить любые нажатия клавиш в текстовом поле (более сложный, но правильный подход).

Зависимые и взаимозависимые поля

18. Создайте класс Ring, описывающий геометрическое кольцо. Кольцо должно хранить в себе центр координат (ранее созданный класс Point2D), а также значения внешнего и внутреннего радиуса. Реализуйте свойства для перечисленных данных с валидацией. Внешний и внутренний радиус должны быть вещественными положительными числами, внутренний радиус не может быть больше внешнего, а внешний – меньше внутреннего. То есть эти поля являются взаимозависимыми. В случае неправильных значений полей, сеттер должен генерировать исключение. Реализуйте конструктор класса.
19. Добавьте в класс Ring доступное только на чтение свойство double Area, возвращающее площадь кольца. Площадь кольца равна разности площади круга, образуемого внешним радиусом, и площади круга, образуемого внутренним радиусом.
20. Свойство Area является зависимым от значений радиусов, поэтому значение свойства не должно храниться в виде поля, а должно рассчитываться заново каждый раз, когда свойство Area вызывается в клиентском коде.

Статические поля и методы

21. Создайте в классе Rectangle закрытое статическое целочисленное поле _allRectanglesCount. Поле предназначено для подсчета всех существующих объектов прямоугольников.
22. Создайте в классе Rectangle открытое статическое свойство AllRectanglesCount(), возвращающий значение статического поля _allRectanglesCount. Обратите внимание, что поле _allRingsCount фактически является доступным только на чтение (не путать с модификатором readonly).
23. Добавьте в конструктор класса Rectangle строку, увеличивающую значение статического поля _allRectanglesCount на единицу. То есть, при каждом вызове конструктора должно происходить увеличение счетчика.
24. Добавьте в класс Rectangle поле и соответствующее свойство ему Id типа int. Поле должно быть доступным только для чтения.
25. Поле _id должно инициализироваться в конструкторе (или конструкторах) класса значением из поля _allRectanglesCount. Поскольку статическое поле _allRectanglesCount в конструкторе каждый раз увеличивается на единицу, то у каждого объекта класса Rectangle должен быть свой уникальный Id, каждый раз больше на единицу, чем у ранее созданного объекта.

26. Добавьте в пользовательский интерфейс вывод свойства Id для прямоугольника. Текстовое поле также должно быть доступно только для чтения, и запрещено для ввода новых значений Id.

27. Запустите программу и убедитесь, что текстовое поле работает правильно. Убедитесь, что у каждого прямоугольника новый Id, отличный от Id других прямоугольников.

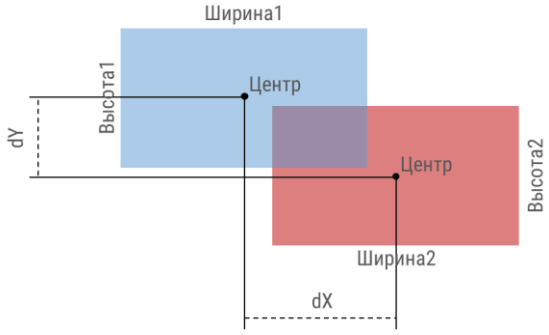
28. Убедитесь, что в ходе выполнения всех заданий, ранее реализованная функциональность не повредилась – выполняется валидация значений в текстовых полях, работает поиск, доступно изменение открытых свойств для прямоугольников и фильмов, работа перечислений на первой вкладке интерфейса работает правильно.

29. Отдельно проверьте правильность верстки новых элементов в интерфейсе, их выравнивание, а также их правильное именование.

30. Проверьте оформление кода классов – именование полей, свойств, методов, порядок членов класса.

31. Создайте статический класс CollisionManager - класс, выполняющий проверку о пересечении/столкновении геометрических фигур. Классы по проверке столкновений применяются в геометрических САПР, 2D- и 3D-редакторах и компьютерных играх. Класс должен реализовать два метода:

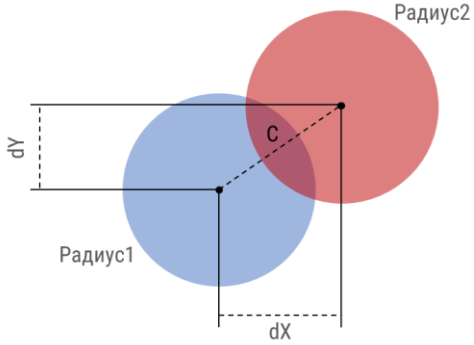
- bool IsCollision(Rectangle rectangle1, Rectangle rectangle2) – метод принимает два экземпляра прямоугольника и проверяет не пересекаются ли они. Прямоугольники считаются пересекающимися если разница их координат по X (по модулю) меньше суммы половин их ширин и разница их координат по Y (по модулю) меньше суммы половин их высот. Если прямоугольники пересекаются, метод возвращает true, иначе возвращает false.



Условия пересечения:

$$\begin{cases} dX < |Ширина1 - Ширина2| / 2 \\ dY < |Высота1 - Высота2| / 2 \end{cases}$$

- bool IsCollision(Ring ring1, Ring ring2) – метод принимает два экземпляра колец и проверяет не пересекаются ли они. Кольца, как и круги, считаются пересекающимися, если гипотенуза треугольника, образующегося между координатами центров колец меньше суммы их внешних радиусов. По желанию можете реализовать учет коллизии, при котором одно кольцо вписано внутрь другого кольца.



Условия пересечения:

C < (Радиус1 + Радиус2)

где C – гипотенуза треугольника, образованного катетами dX и dY