Семинар 4.

Задачи для самостоятельного решения.

№1. Разработать Windows-приложение с двумя элементами пользовательского интерфейса **Button** и **Label**. При каждом нажатии кнопки выводить в текстовое поле элемента **Label** значение очередного члена ряда Пелла:

Для представления членов ряда использовать переменные типа **int**. При возникновении переполнения выдать в диалоговом окне **MessageBox** сообщение *"Переполнение! Ряд начнем с начала!"*.

**Дополнительное задание**

На основе задачи 1 разработайте программу с двумя кнопками (**Button**): первая – для вывода членов ряда Пелла; вторая – для вывода членов ряда Фибоначчи. Вывод выполнять в одно и то же (единственное) текстовое поле элемента **Label**. Нажатие другой кнопки должно приводить к переключению на вывод членов другого ряда. Можно при каждом переходе на другую кнопку соответствующий ей ряд начинать с начала. Можно продолжать вывод с предыдущего члена ряда. По вашему усмотрению.

№2. Разработать Windows-приложение. В поле **TextBox** (*в режиме ввода многострочных данных и их редактирования*) вывести в виде списка элементы массива строк. Изменяя, добавляя или удаляя элемент списка, изменять, добавлять или удалять его в массиве. Вывести в окно **MessageBox** массив после изменений (для вывода используйте метод string.Join(“ ”, **textBox1.Lines**), т.е. **separator** – пробел). Обеспечить возможность восстановления начального состояния списка.

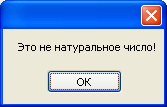
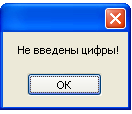
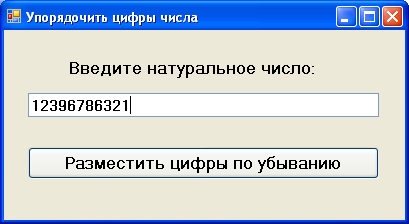
**Описание элемента управления TextBox**:

<http://msdn.microsoft.com/ru-ru/library/system.windows.controls.textbox(v=VS.95).aspx>

**Описание метода Join:**

<http://msdn.microsoft.com/ru-ru/library/57a79xd0.aspx>

№3. Упорядочить по убыванию цифры натурального числа, вводимого с клавиатуры в текстовое поле, представленное элементом **TextBox**. Предусмотреть и обработать случаи, когда ввод был некорректным.



№4. Разработать Windows-приложение, в котором программно изменяются значения свойств элементов управления.

Поместить в центр формы одну кнопку и в обработчике события «нажатие на кнопку» изменять размеры формы.

В начале при каждом нажатии на кнопку уменьшать размеры формы, как только форма достигнет минимальных размеров – увеличивать ее размеры при нажатии на ту же кнопку.

Когда форма достигнет максимальных размеров – переключить кнопку на уменьшение и т. д. Начальный вид формы – минимальный размер.

№5. Разработать игру на основе оконного приложения. Кнопка появляется и исчезает на форме. Цель игры пока кнопка видна, кликать по ней мышью. Клики по кнопки учитываются как попадания, клики мимо – промахи.

<http://www.aimbooster.com/>

Подсчёт очков ведётся в объекте класса **Rate**. Описание класса разместить в отдельном файле в проекте оконного приложения.

Создать форму с одной кнопкой (**Button**) и двумя метками (**Label**). Изменить названия для кнопки на **button**, для меток на **failsLabel** и **hitLabel**. Для управления временем отображения кнопки использовать компонент **Timer**. Для этого добавить его к форме и установить свойство **Interval** = *1000*, **Enabled** = *true*.

*Задания со звёздочкой*

№1. Оконное приложение предназначено для отображения на экране окружности. Окружность – объект класса **Circle** (п. 2). Приложение должно использовать события формы и объекта **Circle** так, чтобы при нажатии кнопки «Изменить» в **PictureBox** отображалась окружность с текущим значением радиуса. Установка флажка «Цвет» приводит к изменению цвета отображения окружности, снятие флажка – восстанавливает умалчиваемый (чёрный) цвет.

Тип-делегата public delegate void **RadiusChanged ()**

**Описание типов**

1. Класс **Circle** – представляет окружность.
   1. Поле **r** – целое неотрицательное число, радиус окружности
   2. Класс является источником события **OnRadiusChanged**. Событие запускается при изменении значения радиуса окружности
   3. Конструктор с вещественным параметром (радиусом окружности). Если значение параметра – число меньшее нуля, конструктор создаёт исключение **ArgumentOutOfRangeException**
2. Класс **CircleVizualizator** – представляет объект, отвечающий за представление окружности в поле **PictureBox**
3. Поле **c** – окружность типа **Circle**
4. Поле **trg** типа **System.Windows.Forms.PictureBox** – поле, в котором осуществляется рисование
5. Поле **pen** – объект типа **Pen**
6. Конструктор с параметрами

**CircleVizualizator** (*System.Windows.Forms.PictureBox* **pb**, *Circle* **c**)

1. Инициализирует поля
2. Связывает событие **OnRadiusChanged** с методом-обработчиком
3. Свойство **PenColor** служит для изменения цвета ручки
4. Метод **Draw ()** рисует окружность с текущими настройками **pen**
5. Метод **Refresh ()** обновляет поле рисования **trg**

№2. Написать программу, моделирующую поведение цепочки из **N** бусин, нанизанных на нить длины **len**. Радиус бусин одинаков и равен целому числу .

* Бусины цепочки должны реагировать на событие «изменение длины нити» и настраивать свой размер (количество бусин не изменилось).
* Бусины цепочки должны реагировать на событие «изменение количества бусин» и также настраивать свой размер (длина нити не изменилась).

**Описание типов**

1. Тип-делегат public delegate void **ChainLenChanged** (*double* **r**);
2. Класс **Bead** – бусина
3. Поле **r** – вещественное число, радиус бусины
4. Конструктор с вещественным параметром – радиусом бусины. Если радиус меньше или равен нулю, конструктор создаёт исключение **ArgumentOutOfRangeException**
5. Класс **Chain** – цепочка бусин
6. Поле **len** – вещественное число, длина нити, на которую нанизаны бусины
7. Поле **beads** – список ***List*** <**Bead**>, составленный из бусин, нанизанных на нить
8. Событие **ChainLenChangedEvent**, определённое типом-делегатом **ChainLenChanged**
9. Свойство **Len**. Обеспечивает доступ к полю – длина нити. При изменении длины нити активируется событие **ChainLenChangedEvent**
10. Конструктор с двумя параметрами – вещественной длиной нити **len** и целым числом **N** бусин в цепочке. Создание бусин выполняет вспомогательный метод **CreateBeads ()**.
    1. Метод **CreateBeads ()** – создаёт объекты-бусины и добавляет их методы-обработчики в список обработчиков события **ChainLenChangedEvent**

**События**

1. Добавить в код событие, возникающее при изменении **N** – количества бусин на нити, предполагается, что длина нити не изменяется, а размеры бусин «подстраиваются» под длину нити так, чтобы занять её.
2. В обработчике этого события добавить код (в классе **Bead**) пересчёта и изменения радиуса бусин
3. Добавить в код событие, возникающее при изменении радиуса бусин
4. Подписать объект **Chain** на *события 2* (событие, возникающее при изменении радиуса бусин)
5. В обработчике пересчитывать количество бусин, которые могут поместиться на нити заданной длины, удалять/добавлять бусины

**Отладка**

Тестирование кода выполните в консольном приложении. Получить от пользователя количество бусин и длину нити. Создать объект **Chain**. Вывести информацию о цепочке бусин: количество бусин, длина нити (с точностью до двух знаков после запятой), радиус бусины. Предложить пользователю экранное меню: 1) изменить длину нити; 2) изменить количество бусин на нити. После выбора пункта выводить информацию об обновлённой цепочке бусин.

(\*) Создайте оконное приложение, в поле **pictureBox** визуализируйте цепочку бусин. Добавьте возможность изменения параметров цепочки и бусин. Свяжите изменения в интерфейсе с изменениями бусин и цепочки.

Дополнительно для задачи 2

// 1 шаг. Определяем класс-наследник EventArgs

// с аргументами для своего события

// у нас это событие изменения длины нити и аргумент - радиус

public class ChainLenChangedEventArgs : EventArgs

{

public readonly double rad;

public ChainLenChangedEventArgs(double r)

{

rad = r;

}

}

public class Chain

{

// 2 шаг описываем событие.

// Используем предусмотренный обобщённый делегат

public event EventHandler<ChainLenChangedEventArgs> ChainLenChanged;

//...

}

// 3 шаг. В класс Chain добавляем метод запуска события

protected virtual void OnChainLenChanged(ChainLenChangedEventArgs e)

{

if (ChainLenChanged != null) ChainLenChanged(this, e);

}

// 4 шаг. Изменяем код запуска события, там, где он производился

OnChainLenChanged(new ChainLenChangedEventArgs(newR));

// 5 шаг. Изменяем код обработчика.

// Добавляем параметры "источник" события и "параметры"

public void OnChainLenChangedHandler(object sender,

ChainLenChangedEventArgs e)

{

R = e.rad;

}