Лаба 8

Конкретизация параметров функции метода конструктора и атрибутов. Множественная конкретизация. Конкретизация с ограничениями

Конкретизация позволяет описывать классы и функции для работы с объектами, типы которых заранее не определены. С каким(и) именно типам(и) придется работать определяется в момент вызова – имя типа настраивает функцию\класс для работы.

Конкретизация – переход от абстрактного к частному. Один из столбов на котором стоит полиморфизм – то есть свойство системы использовать объекты с одним интерфейсом без обращения внимания на внутреннюю структуру  
общим является класс, шаблоны которые они используют  
\_Алгоритм который не зависит от типа класса – повторное использование алгоритма

Конкертизация параметров функции внутри класса в качестве параметров вместо Т внутри программы вызывать от разных классов  
мы конкретизируем Т другими классами которые определены в программе (Т должно быть классом то что мы прописываем называется конкретизаи я параметров функции ОГРАНИЧЕНИЕ мы ограничиваем что Т должно быть классом (можно прописать структуру организация по типу ) )

Конкертизация конструктура – внутрь констуруктора (в качестве параметра конструтктора) можно передавать разные классы

t is IA - проверить является ли t интерфейсом IA – реализует ли объект интерфейс IA

Конкретизация атрибута – мы можем конкретизировать атрибут

Кокнретизация метода – в ней мы работаем с атрибутом можем вызвать этот метод от атрибута который может быть разными классами

Множественная конкретизация – можно конкретизировать одновременно несколько параметров

Можем одновременно конкретизировать два параметра

Есть ограничения – т1 и т2 должны являться классами

Ref используется для того чтобы мы могли менять ссылки – оно передает адреса в памяти

Разница с агрегацией сслки по обхектам – там неизменяемые ссылки а тут мы меняем их значения

Там дубликаты создаются а там они не создаются

ЛАБА 9

Анонимные функции сигнатура функций (delegate) лямба выражения event

Delegate – указатели на методы, у котрых совпадает сигнатура возвращаемого щначения и количеству и типу параметров

Можно в явном виде прописать тело операции

Либо ожно прописать функцию или операцию в другом месте, например вынести за мейн

В качестве параметров конструктора делеагата пердать эту функцию или изначально при создании создать ссылку в null а потом уже присвоить ссылку на функцию

Пишем имя объекта и в скобках указываем конкретные параметры, которые вызываем

Step 2)Можно добавлять сразу ссылки на несколько функций в делегат и вызвать сразу для нескольких функций, тогда нам нужно писать Invoke – для того чтобы выполнялась проверка можно ли для данны параметров вызвать данную функцию – Invoke не возвращает (мы итерируем(+) или декремент инкремент)

-= мы удаляем ссылку на \*\* и теперь при использовании делегаты мы не будем вызывать эту функцию (исключили по имени)

Для определения делегата используется ключевое слово delegate, после которого идет сигнатура. Для использования делегата объявляется объект этого делегата. Объекту данного класса сигнатур можно присвоить метод со сходной сигнатурой. Для объектов класса сигнатур возможна операция декремента/инкремента. Также можно использовать анонимные делегаты путём объявления объекта класса сигнатур и присваивания ему анонимного метода со сходной у делегата сигнатурой. Делегаты можно передавать в качестве аргументов методу, что довольно удобно. Использование делегатов позволяет добиться более адекватного моделирования и повторного использования кода.

Лямбда выражаения

Лямбда-выражения представляют упрощенную запись анонимных методов. Лямбда-выражения позволяют создать емкие лаконичные методы, которые могут возвращать некоторое значение и которые можно передать в качестве параметров в другие методы. Ламбда-выражения имеют следующий синтаксис: слева определяется список параметров, в середине лямбда-оператор =>, а справа блок выражений, использующий эти параметры: (список\_параметров) => выражение. Как и делегаты, лямбда-выражения можно передавать в качестве аргументов методу для тех параметров, которые представляют делегат, что довольно удобно. Как и делегаты, лямбда-выражения позволяют добиться повторного использования кода и адекватного моделирования.

Схожи с делегатами, есть класс лямбда у которой мы можем созавать объекты и прописывать какие параметры будет принимать функции и тело это й функции, применяем оператор присваивания (=> оператор лябмда выражения)

Разница медлу делегатами для входных параметром

Мы не задаем конкретный тип параметров (уровень абстракции получается выше засчет того тчо не создаем конкретный тип )

Еще можно вычислятьс помозью класса функцию но там уже надо задвать тип входных и выходных параметров

Лямба выражения можно использовать в списке – искать индекс нечетного элемента

События (event)

Бывает нужно чтобы какой то алгоритм выполнился если произойдет что то определнное для этого нужны события мы их определяем с исопльзованием делегатов, с помозью которых передается сообщения что произошло событие

С помощью слово event создается событие

Event – внутрь него убираем вызов делегата (делегат можно использовать для методов как параметр) , а event инкапсулирует делегат – нельзя его передавать в качестве параметров для функции

* Ну синглтон гарантирует только один объект класса

То есть форма вызовется только один раз

Запускается форм 2

Потом форм 1

Создаю экземпляр класса и вызываю экземпляр через атрибут доступа

Функция инстанц атрбут доступа к которуму через ворм 2 обращаюсь в форм 1 помогает получить доступ к форм 1 из форм 2