Введение

В данной работе вы сделали свой вариант списка, способный работать только с целыми числами `int` - `class MyIntList` (или как вы его назвали у себя). А если завтра нам понадобится список для дробных чисел? Первая идея, появившаяся в голове: "я скопирую весь код из класса целых чисел, поменяю везде все на `double` и назову новый класс `MyDoubleList`. ЛаДнО, а если нам нужны списки для всех остальных типов данных? Ну сделали мы 10+ почти одинаковых классов, ну нашли ошибку в одном, теперь во всех 10 классах нужно исправить эту ошибку. Это не задача программиста, это задача оператора `Ctrl+C` `Ctrl+V`.

Что такое generic

И эту задачу стандартный класс `List<T>` решает той самой `<T>`. `<T>` (type) - это дженерик (generic). То есть, когда мы напишем `MyList<T>` мы обозначаем, что при создании объекта этого класса мы сможем указать какой-то тип данных. Какой именно - мы укажем при создании объекта (пример: `MyList<int> ints = new MyList<int>()`). В этот момент, когда мы создаем объект класса, наш компилятор подставит вместо `<T>` -> `int`. То есть, если в нашем классе есть массив дженериков (`T[]`), то во время создания объекта эта `T` будет заменена на `int`.

Также дженерик решает проблему безопасности: создав объект `MyList<int>` мы не сможем добавить в этот список другие типы данных.

Не обязательно указывать именно `<T>`, внутри скобок может быть любой текст: `<A>`, `<B>`, `<SomeType>`.

Очень подробно про generic (обобщенный тип) можно почитать на сайте доки микрософта а потом здесь.

В части 1 вы сделали основу класса списка, в данной части мы усовершенствуем этот класс, добавив возможность нашему классу работать с любым типом данных.

Примеры и использование

Много теории

Выше мы рассмотрели одну проблему которую решают генерики (обобщение, далее везде будет генерик): указание не явного типа данных, а обобщенного.

Вспомним метод `Swap(ref int a, ref int b)` из третьей лабы. Суть работы метода заключена в том, что метод меняет местами переменные `a` и `b`. Но в данной конкретной реализации подойдут только переменные типа `int`. В следующих заданиях этой же лабы использовалась перегрузка методов, но также это решение не будет подходящим.

Нам не важно какие типы данных мы хотим менять местами, мы хотим просто переложить из одной переменной в другую. Использование генериков будет самым подходящим.

Пример кода на функциях

Было:

```
void Swap(ref int a, ref int b)
{
    int temp = a;
    a = b;
    b = temp;
}

// Вызов метода

int x = 5, y = 10;

Swap(ref x, ref y);
//Строки передать не получится :(
```

Стало:

```
void Swap<T>(ref T a, ref T b)
{
    T temp = a;
    a = b;
    b = temp;
}

// Вызов метода

int x = 5, y = 10;

Swap<int>(ref x, ref y);

string a = "Физика";
string b = "не страшно";

Swap<string>(ref a, ref b);
```

После имени функции мы можем явно указать тип, который будет использоваться внутри функции.

Пример кода на классах

Как уже говорилось выше: стандартный список умеет работать с любым типом данных. Свой список мы тоже можем сделать универсальным.
Для этого у нашего названия класса мы должны добавить `<T>` (ну или что угодно что вы придумаете внутри `< >`, только это должно быть осознанное и понятное название).

Было:

```
// ...самый верх вашего файла с классом списка, выше тут using-и и namespace;

public class MyOwnList
{
```

```
    private int[] _data; //оставим для примера массив
    //Тут весь ваш остальной большой и страшный класс
списка, который умеет только в int
}
```

Стало:

```
// ...самый верх вашего файла с классом списка, выше тут
using-и и namespace;

public class MyOwnList<T>
{
    private T[] _data;  //Теперь буковка T на месте типа
данных переменной или аргумента
                        // будет тем самым типом данных,
который вы объявите при создании объекта.

    //Тут весь ваш остальной большой и крутой класс
списка, который теперь умеет в любой тип данных (почти
любой...)
}
```

Пример вызова функций:

Было:

```
// метод не полный и не является рабочим!!!
// сохранена только часть для демонстрации генериков!!!
public void Add(int item)
{
    _data[_count] = item;
}
```

Стало:

```
// метод не полный и не является рабочим!!!
// сохранена только часть для демонстрации генериков!!!
public void Add(T item)
{
    _data[_count] = item;
}
```