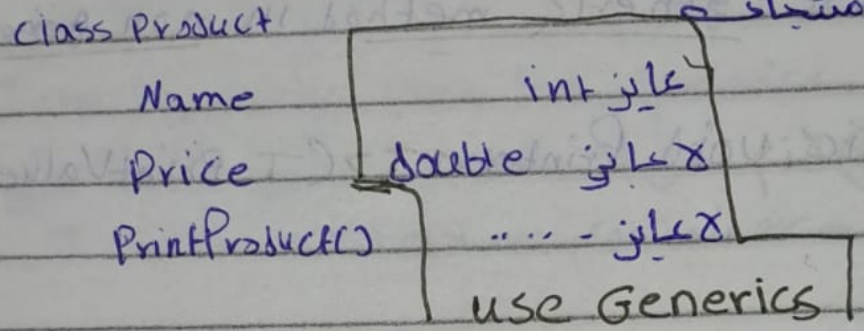
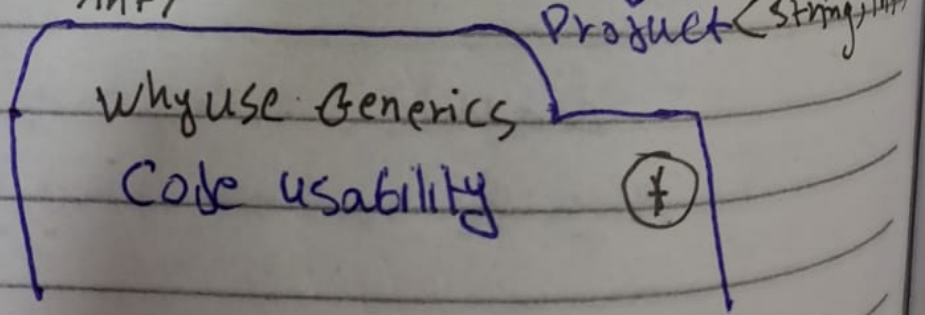
# Week 3 GenePICS

## Video 1




```
Public class Product <T,U>
{
  Public T Name { set; get; }
  Public U Price { set; get; }
}
```

Product<string,int> my_Product = new Product<string,int>();



Date: ____ no : ____

(*) Fields can be generics

(*) array can be generics / Public T[] Product = new T[10];

(*) can use generics ~~and normal type~~ in methods

(*) Can use generics and normal type together

(*) we can use generic method in non generics class.

- Public void Printer<T>(T PrintValue){

}

---

(*) generics لأ نوع بكتب مش

Printer ~~Printer~~ . Printer.Print<T>(sentence)

String

Date: ____ no: ____

* مش محتاج تحدد نوع الـ generics

```
Printer.Print("Ahmed");
```

Compiler will know the ← datatype

* Interface can also be generics

* لازم تحلى الـ class اللى بيستقبل الـ interface اللى بيقبل generic

```
IContainer<String> newContainer = new Box<String>();
```

## Constrains on Generics

* IContainer<T> where T : struct
  ↳ value type

* و الكلاس عليه نفس القيد

where T: class
↳ ref type

* معمل كلاس هياخدمنك اى نوع من أنواع الكلاسات
الى بيستقبل الـ interface المعين ( راجع الكود )

Date: ____ no: ____

Video 02

Lists ArrayLists ⊛

System.Collections تنقسم إلى:
- 1-generic collections
- 2-non-generic collections

non generic
- ArrayList
↓
dynamic
* حجمها يزيد عدد العناصر

generic
- list

① ArrayList array2 = new ArrayList;

⊛ array2.Add(1); // boxing adding element
ref type // value type ← ول

② array2[0] // unboxing أخرج العنصر
value type // object ← ول

⊛ foreach(var item in ____)
____ اطبع كل العناصر

⊛

Date: ____ no: ____

* Array List [لا تعتمد] ← Type Saftey

* ممكن تعمل error متبانش في Programming type الأوغل تحدد

array2.Remove(15); remove element

array2.RemoveAt(0); Remove by index

array2.AddRange(array تاخد ممكن) add range at one time

array.Count عدد العناصر

array2.Insert( □ , □ )
the Index ↓ ↓ object

---

Lists Generics

List<int> my_list = new List<int>

* Dynamic<> متحدش احدد حجم

* my_list.add(5); ✓
* my_list.add("13"); ✗

List<Class> ممكن أعمل
.Add(new Person() { });

* .AddRange() * Count()
* Foreach * Sort()
* Remove() * Contains()
* RemoveAt() * IndexOf()
* Insert() * Clear()

Date: ____ no: ____

# Video 03

Dictionary and SortedList and hashtable

Key-value Pair ⇐ Data ال بتحفظوا شكل معين من (*)

Dictionary ← جاية ← System.Collections.Generic

key — No-Repeat

value — Repeat

```
Dictionary<string, int> ages = new Dictionary<string, int>();
```

```
ages.Add("Ali", 30);
```

ages["key"] ← return

```
foreach(var item in ages)
{
    Console.WriteLine(item.Key);
    Console.WriteLine(item.Value);
}
```

item ← KeyValuePair[objectType]

[ ]

{ }

( )

ROX

Date: no:

لو عايز أعرض الداتا أو أحزنها مترتبة

SortedList
↓
Dictionary نفس الـ
بس مرتبة

Hash table

non-generic

ArrayList زي ما كنا بنعمل (as Declartion)

Hashtable employee2 = new Hashtable();

Employee2.Add (120, "Ali")
↓
object يخزن

* يخزن object وما بيسترجعه بيعمل unboxing

* يستخدموا لما يكون تخزن Pair values
Configuration settings

* caching في الـ

Hashtable بلاش
non-generics بلاش

Date: ____ no: ____

# Video 04

## Stack and queue

stack (LIFO) [Last In First Out]

generic وفيه واحد اسمه non-generic
↳ best to use

```
Stack<int> ints1 = new Stack<int>();
```

| | |
|---|---|
| ① ints1.Push(1);<br>② ints1.Pop(); | foreach (var item in ints1)<br>Console.WriteLine(item);<br>↓<br>(LIFO) يعني هيطبعهم بـ |
| ممكن أعرف الـ collection<br>new Stack<int>( ); ;<br>array ↳ | |

③ ints.Count

```
foreach (var item in ints1)
{
    ints1.Pop();
}
```

نفس الـ Error كان حجم الـ ints1 بيتغير

(مستحيل) while

```
while (ints1.Count > 0)
```

Date: no:

intS1.Peek(); بيشوف آخر عنصر من غير Pop

* ⓐ في الـ app بتاعي استخدمت undo

## Queue FIFO First In First Out

generic عن non generic فيها

ⓐ Queue<int> intS1 = new Queue<int>();

ⓐ intS1.Enqueue Push زي الـ

ⓐ foreach(var item in intS1) {
    Console.WriteLine(item);
}

ⓐ intS1.Dequeue();

Dynamic size

Dequeue الـ while نفس قصة الـ

* تستخدم في حجات فيها ترتيب (مثلا app بيطبع ورقة أو نظام انتظار للعربيات)

ⓐ intS1.Peek()

Date: ____ no : ____

# Video 05

Tuple  ValueTuple ] ⟶ Collection · مش بين مهم

موجودة جوا الـ System على طول

```
Tuple<int, int> Tuple1 = new Tuple<int, int>(5,10);
```

`<int, String>`

`Tuple<1, 2, 3, 4, 5, 6, 7, TRest>` لغاية 8 عناصر

Tuple ازاى أوصل للعناصر جوا الـ

| Tuple1.Item1 | var my_Tuple = Tuple.Create(1, "Bassem", 4.5); |
|---|---|
| .Item2 | طريقة تانية اخف بيها Tuple |
| .Item3 | (*) // Tuple is reference Type |
| ⋮ | |

TRest ايه هو الـ

(*) `var myTuple = Tuple.Create(1, 2, 3, 4, 5, 6, 7, 8);`

8 ⟵ Rest.Item1

`cw(myTuple.Rest.Item1)` output => 8

لو عايز أخزن أكتر من 8 عناصر

## يفضل

* في الحالة دي بنستعمل Nested Tuples

Tuple.Create() ⇐ في الحيز الثامن

Create()

var myTuple = Tuple.Create(1,2,...,7, Tuple.Create(8,9));

myTuple.Rest.Item1.Item2 Output => 9

(*) مش شرط Nested Tuple تبقى في الآخر بس ده يفضل

Tuple الاستخدام العملي لـ

(*) عايز return من method أكتر من عنصر (رابع الكود)

(*) في الـ Line يرجع كذا value

(*) ممكن ياخد Parameter كـ Tuple

ValueTuple ⟹ Tuple بيعالج قصور الـ

generic

```
ValueTuple<int,int> employee = (6, 36);
Cw(employee.Item1);
(int, string) employee1 = (1, "A");
```

Struct ValueType

Date: ____ no : ____

(*) ممكن ا ك items لا

(int Id, String FName) employee = (123, "Bassem")

employee.Id

(*)

var x = (1, "ali", "mohamed")

(*) items مش محدود بـ 8 لا

Public (string,age) Get---(){ راجع الكود

return (Name, age)

}

// Deconstraction

(*) استقبل حاجة وا وكمها واحطها فـ variable جديدة

# IENUMERABLE
# IENUMERATOR

video 06

* كل الـ Collection بيعتمدوا عليها (الـ Collection هي الـ Collection اللي بتستعملها)

* دول بيسمحوا للـ foreach تشتغل على الـ Collection
↑ IEnumrable ↓

* Ienumerable عبارة عن Interface

Ienumerable بيستقبل myCollection كـ class حاجة

GetEnumerator()

return type واحدة method جواه

IEnumerator

Enumerator و Enumerable ايه الفرق بين

مثال العمليات

أنت

(الي بتتلف على العمليات)

عشان يقرا

هي اللستا نفسها

List

* ده يعتبر نوع من أنواع الـ design Pattern

الفصل بين الحاجة الي بلف عليها وبين الي بيقوم باللف .

class ShoppingListEnumrator : IEnumerator {

(1) public object Current  (IEnumerator ← Interface)

(2) Public bool MoveNext() {

}

(3) Public void Reset() {

}

---

(1) Gets element in the collection at current (1) Position of enumerator.

(2) Advances the enumerator to the next elem
return bool لو انتهى ولو مفيش true
False next ←

(3) Set enumerator to First Postion.

---

راجع الكود

# video 07

* تعرف ازاى ال index بيشتغل
* تعرف ازاى جوا class عادى إنى كأنه Array of collection

```
List<int> list1 = new List<int>();
Cw(list[1]);      geting
list[1] = 6;      setting
```

ازاى الكلام ده يبقى

```
Public class BookCollection {
    Public Book this[int index] {
        get
        { return Books[index]; }
        set
        { Books[index] = value; }
    }
```

* ممكن استقبل indexer غير ال int

راجع الكود

index ال overload بعمل ✗