# Week 3 Generics

## Video 1

class Product

- Name
- Price
- PrintProduct()

*عندنا ٤ طرز منتجات

- عايز int
- x عايز double
- x عايز ......

use Generics

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

---

```
Product<String,int> my-Product = new ____ ();
                                   Product<String,int>
```

Why use Generics

- Code usability (*)

- (*) 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 <u>generic method</u> in <u>non generics</u> class.

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

}
```

---

(*) مش محتاج تكتب نوع ال generics عادي

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

T ↓ String

---

* مش محتاج تحدد نوع الـ generics

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

compiler will know the datatype

---

* Interface can also be generics

* لازم تخلى الـ class إلى هيستعمل الـ interface يبقى generic

```
IContainer<String> newContainer = new Box<Strin
```

---

Constrains on Generics

* `IContainer<T> where T : struct` → value type

* و الكلاس عليه نفس القيد

`where T : class` → ref type

---

# عمل كلاس هياخد منك اى نوع من أنواع الكلاسات الى يستخدم الـ interface المعين (راجع الكود)

Date: ____ no: ____

## video 02

Lists ArrayLists ⊛

System.Collections نوعين
- 1- generic collections
- 2- non-generic collections

| non generic | generic |
| --- | --- |
| - ArrayList ⇓ dynamic * مش لازم عدد العناصر | - list |

① ArrayList array2 = new ArrayList;

⊛ array2.Add(1); // boxing adding element
حول الـ value type لـ ref type

⊛ array2[0] // unboxing أعمل العكس
حول الـ object لـ value type

⊛

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

(*) Array List  تفتقد لـ → Type Saftey

(*) ممكن تعمل error متبانش | في Programming الأفضل تحدد type

```
array2.Remove(15);   remove element
array2.RemoveAt(0);  Remove by index

array2.addRange(array ممكن اخرى)  add range at one time

array.Count          عدد العناصر

array2.Insert(□, □)
       the Index ↲   ↳ object
```

---

{ Lists } Generics

```
List<int> my_list = new List<int>
```

(*) dynamic | مش لازم احدد حجم

(*) my_list.add(5); ✓

(*) my_list.add("13"); ✗

List<class> ممكن اخلي

.Add(new Person() { });

(*) AddRange()  (*) Count()

(*) foreach  (*) Sort()

(*) Remove()  (*) Contains()

(*) RemoveAt()  (*) IndexOf()

(*) Insert()  (*) Clear()

# Video 03

Dictionary and Sortedlist and hashtable

key-value Pair ⇐ Data ال يحفظوا شكل معين من ال (*)

Dictionary → ماشي 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 ---- )
{
    CW(item.key);        → KeyValuePair [object type]
    CW(item.Value);
}
```

---

[ ]
{ }
< >

Date: ____ no: ____

SortedList لو عايز أعرض الداتا أو أخزنها مترتبة

↓

نفس ال Dictionary

بس مرتب

---

## Hash table

non-generic

→ ArrayList شبه ال كلاكان بيخزن زي ( as Declartion)

```
Hashtable employee2 = new Hashtable();
employee2.Add( 120, "Ali")
```

object بيخزن

(*) بيخزن ك object وما بيسترجعه بيعمل Unboxing

---

(*) بيستخدموا ملا تكتبه تخزن Pair values

configuration settings

(*) في ال caching

Hashtable بلاش

non-generics بلاش

Date: ____ no: ____

# Video 04

## Stack and queue

stack (LIFO) [Last In First Out]

generic فيه واحد non-generic وواحد
→ best to use

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

| | |
|---|---|
| ① ints1.Push(1); ② ints1.Pop(); | foreach (var item in ints1) CW(items) ↓ |
| ممكن أعرف collection new Stack<int>( ▪ ); ↳ اداله array | هيعرضهم كـ (LIFO) |

④ ints.Count

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

تقل Error عشان حجم الـ ints1 بيتغير
↳ (نستخدم while)

while (ints1.Count > 0)

Date: ____ no: ____

ints1.Peek(); Pop بيشوف آخر عنصر من غير

*(*) ولو في ال app عايز استعمل undo

Queue FIFO First In First Out

في منها generic و non generic
أفضل

(*) Queue<int> ints1 = new Queue<int>();

(*) ints1.Enqueue زي ال Push

(*) foreach (var item in inst1) {
    cw(item);
}

(*) ints1.Dequeue();

Dynamic size

نفس قصة ال while في ال Dequeue

(*) ints1.Peek()

* نستخدم في حجات فيها ترتيب (مثلا app بيطبع ورقة أو نظام انتظار للشركات

Date: ____ no : ____

## video 05

Tuple    ValueTuple ] ——> مش collection . بس مهم

موجودة جوا ال 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

---

### ازاى أوصل للعناصر جوا ال Tuple

| | |
|---|---|
| Tuple1.Item1<br>.Item2<br>.Item3<br>⋮ | طريقة تانية احرف بيها Tuple<br>var my_Tuple2 = Tuple.Create(1, "Bassem", 4.5);<br>(*) // Tuple is refrence Type |

### ايه هو ال TRest

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

8 → Rest.item1

```
cw(myTuple.Rest.item1) output => 8
```

ROX

لو عايز أخزن أكثر من 8 عناصر

لا يفضل

* في الحالة دي بنستعمل Nested Tuples

في العنصر الثامن ⟸ Tuple.Create()

Create()

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

```
myTuple.Rest.Item1.Item2   output => 9
```

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

الاستخدام العملي لل Tuple

(*) عايز return من method أكثر من عنصر (راجع الكود)

value 1 يرجع كدا

(*) في ال Line

(*) ممكن ياخد Parameter كـ Tuple

ValueTuple ⟹ بيعالج قصور ال Tuple

generic

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

Struct    ValueType

Date: ____ no : ____

(*) ممكن أسمي ال items

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

employee.Id

---

(*)

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

(*) مش محدود بال 8 items

---

Public (String, age) Get---(){ راجع الكود
return (Name, age)
}

---

// Deconstraction

(*) استقبل حاجة او فككها واحطها في variable جديدة

---

IENUMERABLE
IENUMERATOR

**video 06**

(*) كل ال collection بيعتمدوا عليهم (ال collection مش collection
لومستعملهاش)

(*) دول بيسمحوا لل foreach تشتغل على ال collection
← IEnumrable

(*) عبارة عن Interface — Ienumerable

(*) عملت class ا كده myCollection بيستغل Ienumerable
جواه method واحدة بـ return type → IEnumerator
→ GetEnumerator()

ايه الفرق بين Enumerable و Enumerator

مثال الطلبات

هي اللستا نفسها
List

أنت
اللي بتلف على الطلبات
عشان يقروا

(*) ده يعتبر نوع من أنواع ال design Pattern
الفصل بين الحاجة إلى بلف عليها وبين إلى بيقوم باللف.

Cint
en
جديدة

class ShoppingListEnumrator : IEnumerator {
(1) public object Current — 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 element
return bool لو اتحرك true ولو معرفش
يرجع False next
(3) Set enumerator to first Postion.

---

{ راجع الكود }

## video07

* هعرف ازاى ال index بيشتغل
* هعرف ازاى أخلى class عادى إلى كأنه Array or collection
*

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

ازاى الكلام ده بقى

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

(يعتبر نوع ال Array) (ده إلى عايز أعمله Indexing)

* ممكن استعمل idexer غير ال int
* بعمل overload ال index

راجع الكود