



# Collections & Generics



# CONTENTS



- **Objectives**

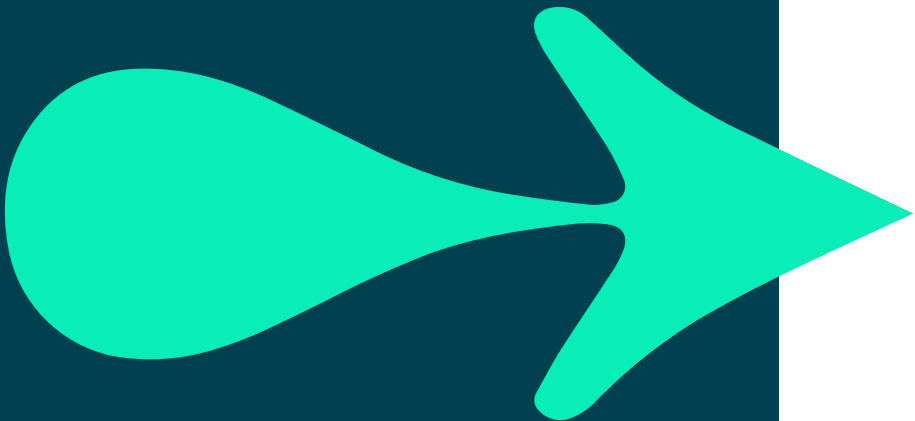
- Compare functionality offered by arrays & collections
- Understand generic concepts, use generic types & syntax



- **Contents**

- Recap arrays, introduce collection classes
- Generic concepts
  - Collections framework, generic classes

- **Hands-on labs**



# Arrays vs collection classes




- **Limitations of arrays**
  - Fixed size, can't append, insert or delete
  - No built-in method to reject duplicates
  - But are type-safe! An **int** array can only contain **integers**
- **C# offers collection classes**
  - List, Queue, Stack, Dictionary...
    - Know their capacity
    - Support append / insertions / deletions / searching
  - Generic version of these are type-safe

# Collection classes before .NET 2 (2006)

- Classic `ArrayList` could only hold a collection of `Object` type

```
ArrayList myList = new ArrayList();  
  
myList.Add(123);  
  
myList.Add("Bob");
```

using System.Collections

<code>int id = myList[0];</code>	 needs casting. <code>get( )</code> returns an <code>Object</code>
<code>int id = (int)myList[0];</code>	 Cast is valid but it takes time to cast <code>Object</code>
<code>int id = (int)myList[1];</code>	 Compiles but crashes during runtime

# Generic collection classes

- Can hold a collection of a specific type and always returns the expected type

## Ứng Dụng Hệ Thống Collections Generic

```
List<int> myList = new List<int>();  
  
myList.Add(123);    ✓ can add an integer  
  
myList.Add(123.5);   ✗ double is not allowed  
myList.Add("Bob");   ✗ only integers  
  
int k = myList[0];    ✓ no casting is required
```

```
List<double> salaries = new List<double>();  
  
salaries.Add(123.5);    ✓ can add a double  
  
double x = salaries[0];  ✓ no casting is required
```

# Iterating through a generic List

```
List<string> friends = new List<string>();  
friends.Add("Tom");  
friends.Add("Sue");  
friends.Add("Sanjeev");
```

add( ) expects a string

```
foreach (string name in friends)  
    Console.WriteLine(name);
```

enumerable

```
for(int i = 0; i < friends.Count; i++)  
    Console.WriteLine( friends[i] );
```

Note "Count"

```
friends[1] = "Susan";
```

Susan replaces Sue

# Filtering items – an example of using Linq

```
class Person
{
    public int Age { get; set; }
    public string Name { get; set; }
}
```

```
List<Person> people = new List<Person>();
// fill the list
```

Given a list of objects like List<Person>

```
using System;
using System.Collections.Generic;
using System.Linq; ←
```

Use Linq

```
List<Person> peopleOver20 = people.Where(p => p.Age > 20).ToList();
peopleOver20.Count will be 0 if no person is over 20
```

Find people over 20

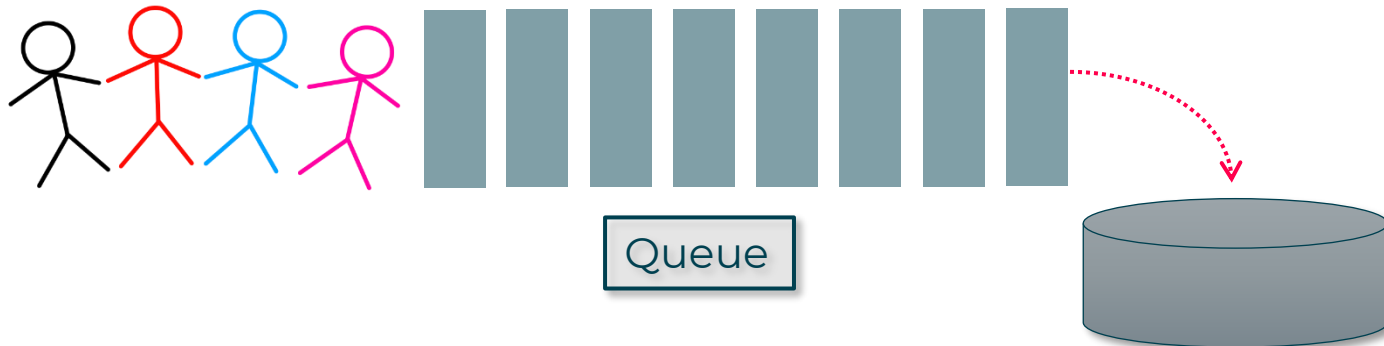
```
Person p1 = people.Find(p => p.Name == "Bob");
p1 will be null if Bob is not in the list
```

Find the first person with the name of Bob  
(there could be more than one)

# Generic Collections framework types

- ứng dụng Hệ thống Collections Generic

```
public class List<T> { ... }  
public class LinkedList<T> { ... }  
public class Queue<T> { ... }  
  
public class Dictionary<K,V> { ... }  
public class TreeMap <K,V> { ... }  
public class HashSet<T> { ... }  
public class SortedList<T> { ... }
```





## Queue – FIFO (first in, first out)

Řuêuê şţşîngû ruêuê ngêx Řuêuê şţşîngû

Đầu tiên, chúng ta cần xác định các thành phần của bài toán. Bài toán yêu cầu tìm số lượng các số nguyên dương  $n$  sao cho  $n$  chia hết cho tất cả các số nguyên dương  $a$  và  $b$  sao cho  $a + b = n$ . Điều này có nghĩa là  $n$  phải là bội chung nhỏ nhất (BCNN) của tất cả các số nguyên dương  $a$  và  $b$  sao cho  $a + b = n$ .

ryêê Éryêê Ñîlê

ruêuê Éruêuê L'îηđầ

# xhîlê rụêụê Cộụựự

Cộng số lẻ và số chẵn

Dave  
Mike  
Linda

Řuêüê şŧŧsîŋğ řuêüê ŋêş Řuêüê şŧŧsîŋğ

ruêuê Énruêuê Dăwê

ruêuê Énruêuê Nîlê

ryêuê Êryêuê L'inhđả

**g**ỗ**e**ắ**ç**    wắ*s*   i**tj**ê*n*   **i**n   rụêuê

Cộng số lẻ và số chẵn

Cộng số là viết tắt của Cộng số và Cộng số

Dave  
Mike  
Linda  
3

## Just read

# LABS



## Part 1 – Using Lists



## Part 2 – Using Queues



Duration 30 minutes

# KEY-VALUE PAIRS



## C# Dictionary class

- Stores and manages any type of item
- Each item has a unique key
- You can get an element using its key
- The key can be of any type, not just integers
- More efficient than managing two Lists

Key	Value
"AB102"	200.5
"JF786"	984.9
"KE900"	765.75

# Usage of Dictionary

```
Dictionary<String, Car> cars = new Dictionary<String, Car>();
```

```
cars.Add("sam", new Car("Ford"));
```

```
cars.Add("joe", new Car("BMW"));
```

```
cars.Add("joe", new Car("Tesla"));
```



Keys must be unique

```
Car car = cars["sam"];
```

```
cars["sam"] = new Car("Honda");
```

Using the same key replaces the item

```
if(!cars.ContainsKey("mike"))  
{  
    cars.Add("mike", new Car("Ferrari"));  
}
```

Searching and  
inserting an item

# Getting the keys/values

Dĩçťĩộặầỷ şťsîng Cầs ặầsặ ặềx Dĩçťĩộặầỷ şťsîng Cầs

ặầsặ Add ặần ặềx Cầs Gộsđ  
ặầsặ Add ặồề ặềx Cầs BÑŴ

ặộsềắặ ắắ ặầ ỉặ ặầsặ Ầắặềs

Cộặộặề ỠsỉặềẦềề Ầắặề ặầ ặộđềầ

Value: Ford  
Value: BMW

ặộsềắặ ắắ ặề ỉặ ặầsặ ặềỷ

Cộặộặề ỠsỉặềẦềề ặềỷ ặề

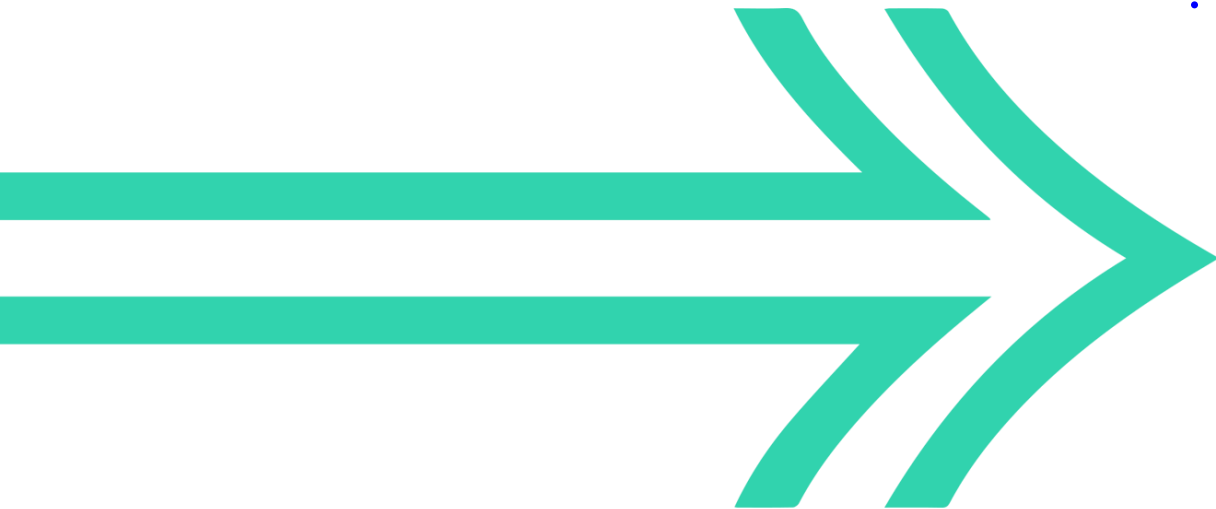
Key: sam  
Key: joe



## C# 2.0 (2006) introduced generic types

- Improve performance and type safety
- `List<T>` & `Dictionary<TKey, TValue>`
- Stack and Queue provide LIFO and FIFO behaviour

usîng Sýstêm Cöllêctîonş Gênesîç



# LABS



**Zoo Animals**  
**Using Dictionary<K, V>**



**Duration 1 hour**