POWERING POTENTIAL

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
```

```
int id = myList[0];
```
needs casting. get( ) returns an Object

```
int id = (int)myList[0];
```
Cast is valid but it takes time to cast Object

```
int id = (int)myList[1];
```
Compiles but crashes during runtime

Generic collection classes

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

```
using System.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

- ṵṣîŋĝ Ṣỳṣţêṇ Cȯḷḷêçţîȯṇṣ Ǧêŋêsîç

```
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)

```
Řụêụê ṣʃsîŋĝ  ṛụêụê   ŋêx Řụêụê ṣʃsîŋĝ
ṛụêụê Énṛụêụê  Dǎŵê
ṛụêụê Énṛụêụê  Ňîlê
ṛụêụê Énṛụêụê  Ľîŋđǎ

xḥîľê ṛụêụê Cộụŋʃ
    Cộŋṣộľê Ẅsîʃêľîŋê̊ ṛụêụê Dêṛụêụê
```

```
Dave
Mike
Linda
```

```
Řụêụê ṣʃsîŋĝ  ṛụêụê   ŋêx Řụêụê ṣʃsîŋĝ
ṛụêụê Énṛụêụê  Dǎŵê
ṛụêụê Énṛụêụê  Ňîlê
ṛụêụê Énṛụêụê  Ľîŋđǎ

ǧộsêǎçḥ  ŵǎs îʃêṇ îŋ ṛụêụê
            Cộŋṣộľê Ẅsîʃêľîŋê îʃêṇ

Cộŋṣộľê Ẅsîʃêľîŋê ṛụêụê Cộụŋʃ
```

```
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"));

Car car = cars["sam"];

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

Keys must be unique

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

```
Dîçƭîộŋǎsỳ ṣƭsîŋĝ  Cǎs  çǎsṣ   ŋêx Dîçƭîộŋǎsỳ ṣƭsîŋĝ  Cǎs

çǎsṣ Ađđ  ṣǎṇ   ŋêx Cǎs  Gộsđ
çǎsṣ Ađđ  ḳộê   ŋêx Cǎs  BŇẄ

ğộsêǎçḥ  ŵǎs çǎs îŋ çǎsṣ Λǎĺụêṣ

    Cộŋṣộĺê ẄsîƭêĹîŋê  Λǎĺụê      çǎs ṇộđêĺ


ğộsêǎçḥ  ŵǎs lêỳ îŋ çǎsṣ Ḳêỳṣ

    Cộŋṣộĺê ẄsîƭêĹîŋê  Ḳêỳ      lêỳ
```

Value: Ford
Value: BMW

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

```
using System.Collections.Generic
```

LABS

Zoo Animals

Using Dictionary<K, V>

Duration 1 hour