Exceptions

Is any error condition or unexpected behavior that is encountered by an execution program. The base exception or System.Exception will have the following parts:

Message

Describes reason for the exception.

Write for the developers who going to handle exception

Stack trace

Information about call stack

Helps to show the execution path/ flow that led to the exception

Data

Dictionary

Key is string, Value is the La mancha.

It can contain multiplication

Inner exception

Capture the preceding exceptions in a new Existing when he returns.

Information

Application/object names that caused error. Defaults contain by default the name of the originating assembly

HResult

Type In32, represents HRESULT numeric value.

Often used in COM

Help link

Link to associate help file.

URN and URL

Target site

System Reflection Method base

Method that threw exc Name, return type, is public/private, etc

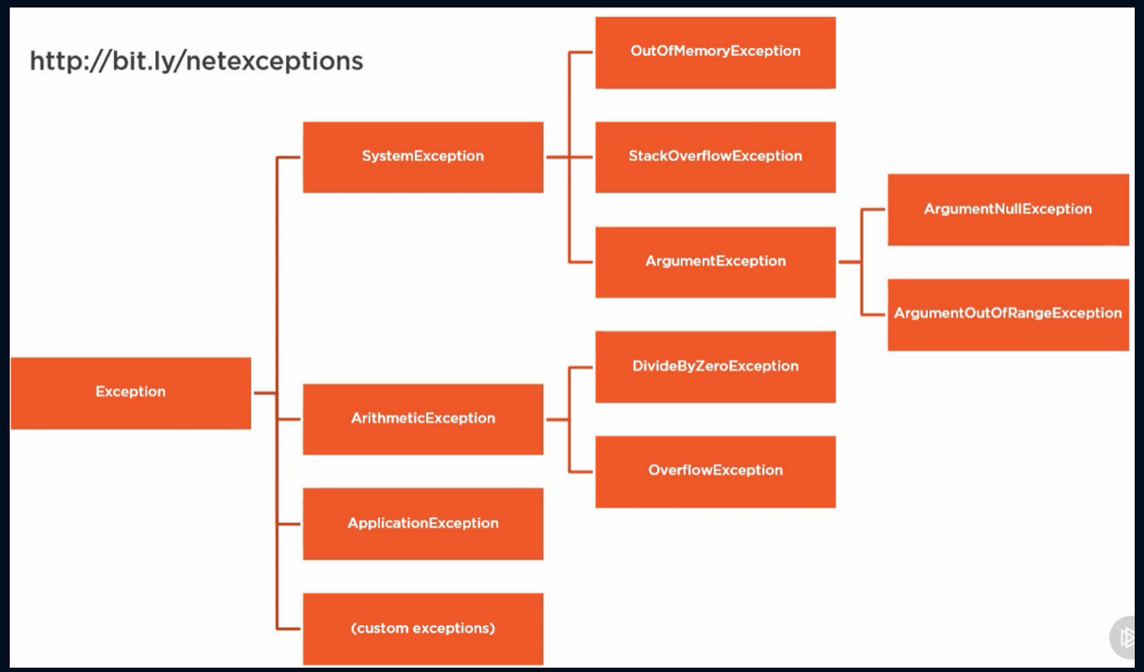
Source

Application/object name that caused error.

Defaults to name of originating assembly

Exception hierarchy

The actual type of the exception class represents the kind of error that occurred.



Exception hierarchy

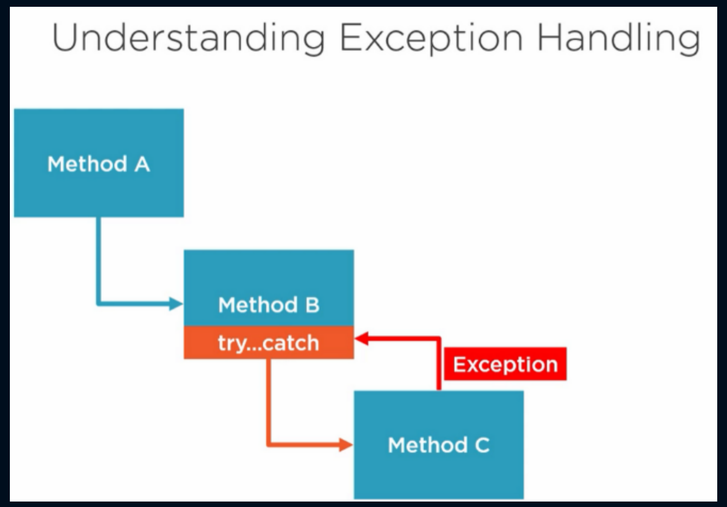
Exceptions meaning

Exception: Represents execution errors
SystemException: Base class for system exceptions namespace
IndexOutOfRangeException: when attempting to access a collection item outside its bounds
StackOverflowException: when too many nested methods calls cause the execution stack to overflow
OutOfMemoryException: when there is not enough memory to continue executing the program

InvalidOperationException: when a invalid state of an object is invalid for a specific method being called.
ArgumentException: when a method argument is invalid
ArgumentNullException: when a null is passed to a method argument and it cannot accept nulls
ArgumentOutOfRangeException: when a method argument is outside of an allowable range
NullReferenceException: when an attempt is made to deference a null object reference

Try, catch and finally

- try: Defines a block of code where exceptions might occur.
- catch: Catches and handles specific exceptions that occur within the try block.
- finally: Defines a block of code that is always executed, regardless of whether an exception occurred or not.



How try catch handles exceptions

```
try
{
    // Some operation
}

catch (ArgumentNullException ex)
{
    // Handle ArgumentNullException
}

finally
{
    // Always executed when control leaves try block
}
```

Use finally

```
try
{
    // Some operation
}

catch (ArgumentNullException ex)
{
    // Handle ArgumentNullException
}

catch (InvalidOperationException ex)
{
    // Handle InvalidOperationException
}

catch (Exception ex)
{
    // Handle all other exceptions
}
```

Most specific

Least specific

How to handle multiple cases

Custom exceptions

Custom Exceptions are created by deriving from the base Exception class to handle specific application errors.

Use custom exceptions when you need to represent specific error conditions that are unique to your application or domain.

```
namespace Packt.Shared;
public class PersonException : Exception
{
  public PersonException() : base() { }
  public PersonException(string message) :
base(message) { }
  public PersonException(string message, Exception
innerException)
    : base(message, innerException) { }
}
```

Create custom exception

It is recommended when possible to use a more specific Exception type

```
public void TimeTravel(DateTime when)
{
  if (when <= DateOfBirth)
  {
    throw new PersonException("If you travel back in
time to a date earlier than your own birth, then the
universe will explode!");
  }
  else
  {
    WriteLine($"Welcome to {when:yyyy}!");
  }
}
```

Throw custom exception

```
try
{
  john.TimeTravel(when: new(1999, 12, 31));
  john.TimeTravel(when: new(1950, 12, 25));
}
catch (PersonException ex)
{
  WriteLine(ex.Message);
}
```

```
Welcome to 1999!
If you travel back in time to a date earlier than
your own birth, then the universe will explode!
```

Catch custom exception

.NET Collections

C# provides a rich set of collection support to manage and manipulate data efficiently.

ArrayList,
BitArray,
Hastable

Collections

Generic Collections

Concurrent Collections

Immutable Collections

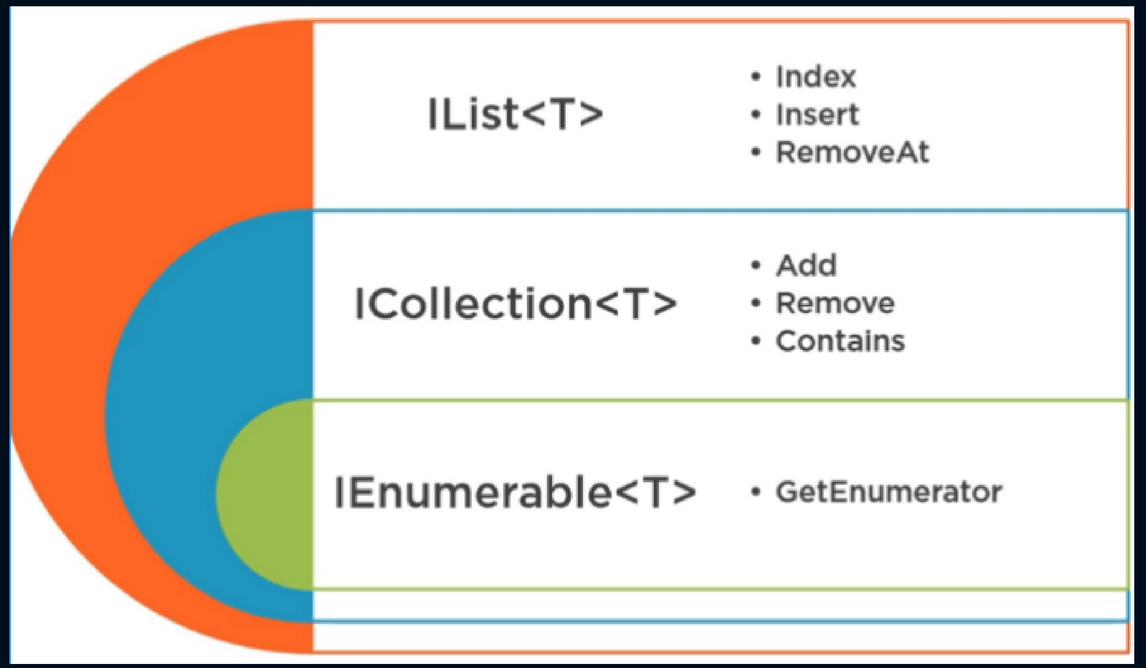
Specialized Collections

IEnumerable

IEnumerable : represents a generic sequence of elements that can be enumerated. It defines a single method, GetEnumerator(), which returns an IEnumerator<T> object for iterating over the collection.
IEnumerator: provides methods for iterating over a collection of elements. It defines properties like Current to get the current element in the collection, and methods like MoveNext() to move to the next element and Reset() to reset the enumerator.

These interfaces provide abstractions for working with collections in a generic and type-safe manner.

Generic collection interfaces implementation

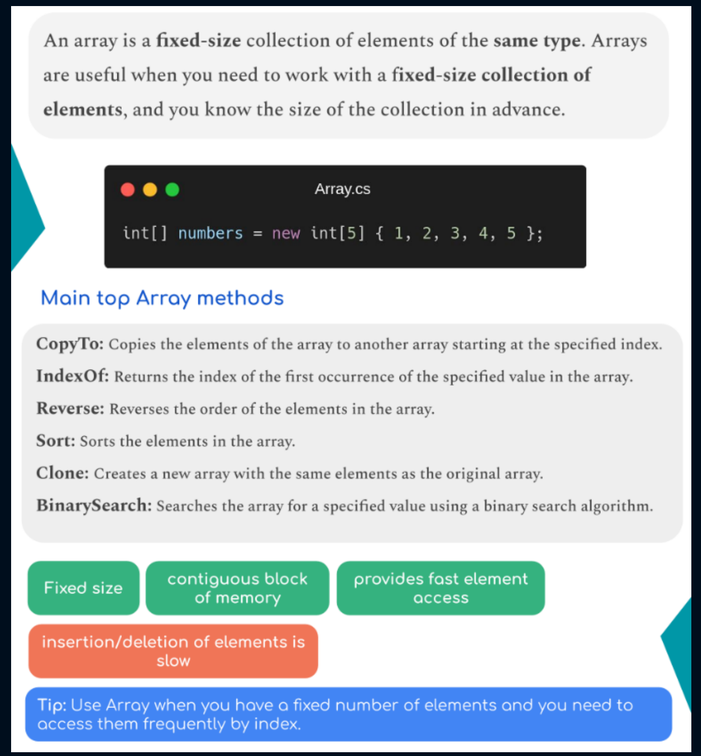


IList, ICollection, IEnumerable

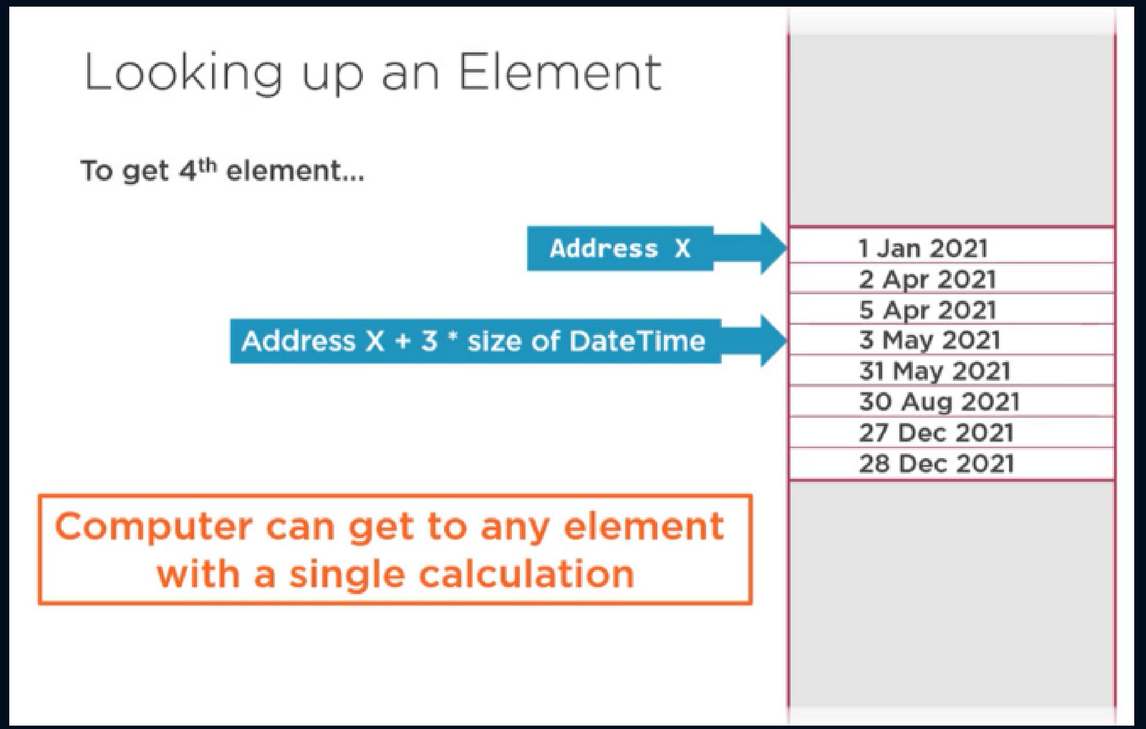
Array

Fixed-size, ordered collection of elements of the same type. It is a fundamental data structure that allows you to store and manipulate multiple values in a single variable.

Reference types, numeric indexing, static or dynamic sizing, fixed once created. Arrays are structures that perform great



Arrays



Array under the hood

List

is a generic class that represents a dynamic, resizable collection of elements of the same type.
Adding can be slow

A List<T> is a **generic dynamically sized** collection of objects of a specific type. Lists are useful when you need to work with a collection of objects that **can grow or shrink** in size.

List.cs

```
List<string> names = new List<string>();
names.Add("Ali");
names.Add("Fatima");
```

Main top List<T> methods

Add(): Adds an object to the end of the List<T>.

Contains(): Determines whether an element is in the List<T>.

Count: Gets the number of elements actually contained in the List<T>.

Insert(): Inserts an element into the List<T> at the specified index.

Remove(): Removes the first occurrence of a specific object from the List<T>.

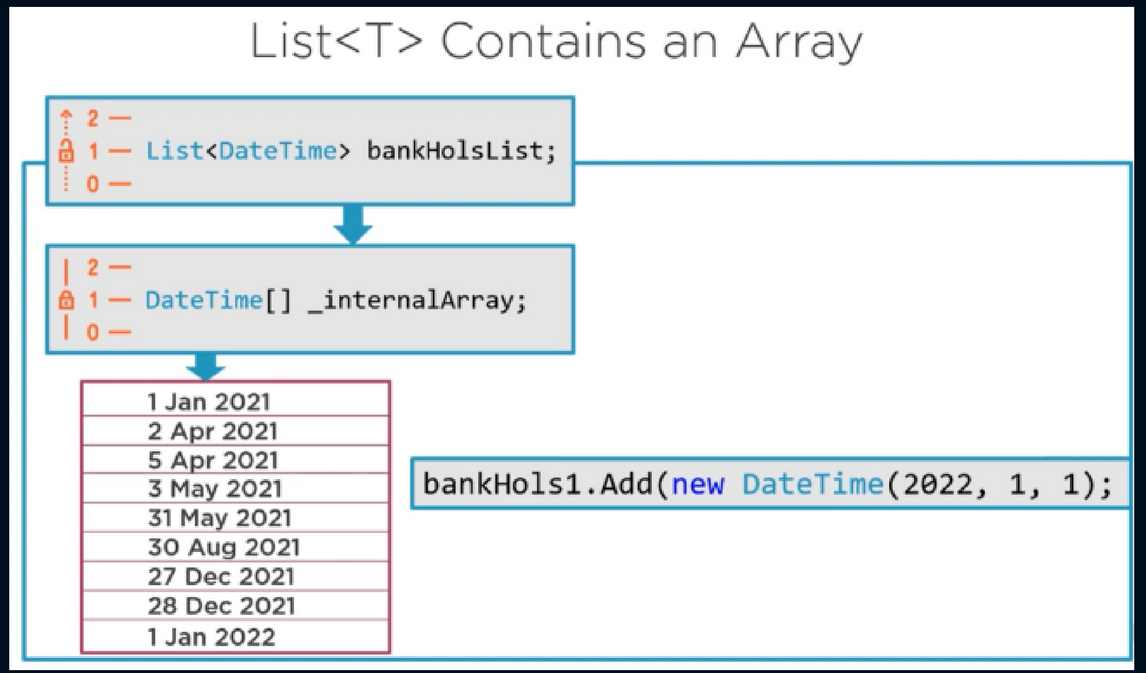
ToArray(): Copies the elements of the List<T> to a new array.

Resizable array | stores elements of a specific type | fast element access

Occupy more memory than arrays

Tip: Use List<T> when you have a collection of objects whose size can change dynamically and you need to perform type-safe operations like add, remove, and search frequently.

Lists



Lists under the hood

Demo

We will check the following:

- Take a look at the Exception hierarchy
- Try, catch, finally
- Throw and Rethrow exceptions
- Custom exceptions
- Arrays
- Lists

Task

Exercise/ Homework

Using the Quiz Game application created on class 5 add the following

- Add limitations on the symbols the user can input to fulfil the quiz, for example restrict the use of the following symbols ()$%[]~” ” ”.
- Use the try, catch statement to handle potential exceptions. For example input errors from the user like when you are asking a numeric option
  - Create custom exceptions that are related to the context of the application if needed
- Handle the exceptions generated from bad processing of data.
- Avoid terminating the execution of the application

NOTE: Saving all exception errors in a txt file is a plus, add the date and time in which the exception occurred when writing this information in the txt file