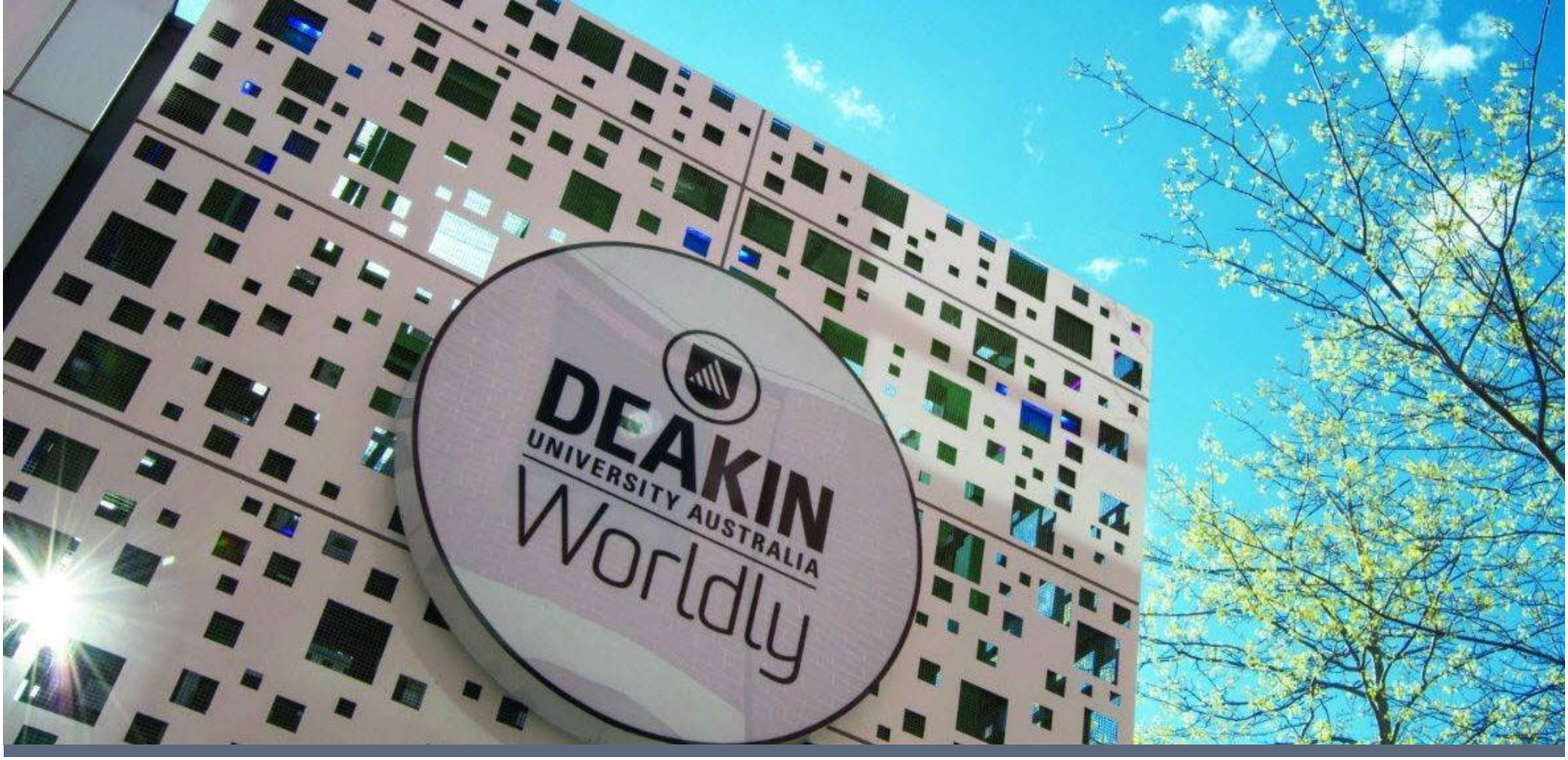


Lecture 03. Control Flow and Error handling

SIT232 Object-Oriented Development

Flow of Control

- Usual approach:
  - Program starts
  - We hit well-defined branch and loop points
  - Demanding user input at key points
  - Proceed to the end
  - Stop
- Event Driven Computing (flow is defined by events occurring):
  - Program starts
  - Set-up complete
  - Enter a main loop
  - Loop around waiting for events and then triggering code when they occur. Events may result from user actions such as key strokes or actions with mouse and etc.

Flow of Control: Switch Statement

C# switch statement works well for lots of choices:

```
bool forever = true; int kgfish1 = 0, kgfish2 = 0;
do {
     Console.WriteLine("1. Fishtype1");
     Console.WriteLine("2. Fishtype2");
     Console.WriteLine("3. Quit");
     int reply = Convert.ToInt32(Console.ReadLine());
     switch (reply) {
          case 1:
               Console.WriteLine("How many kg of fishtype1 do you want?");
               kgfish1 = Convert.ToInt32(Console.ReadLine());
               break;
          case 2:
               Console.WriteLine("How many kg of fishtype2 do you want?");
               kgfish2 = Convert.ToInt32(Console.ReadLine());
               break;
          case 3:
               Console.WriteLine("You've decided to quit.");
               forever = false;
               break;
          default:
               Console.WriteLine("Please insert either 1, 2 or 3.");
               break;
     }
} while (forever);
```

Flow of Control: Switches and Events

- Because the model is simple (set things up, run a loop and listen), many frameworks provide support for you to write part of the code.
  - Graphical User Interfaces are a very common example.
  - You write Event Handlers for well-defined events so if that event occurs, you control what happens.
- Switches are easy to extend.
- Matching multiple events to one piece of code is easy.
- Switch tests are evaluated in order, finishing with default (if present).
- If you reached default then none of the other matches can have occurred.

Error Handling

- Errors may affect the control flow and we have to manage errors towards improving the robustness of a program.
- This consists of
  - **Error Detection**: identify when an error has occurred
  - **Error Handling**: correct for that error

Potential Errors: Challenge

Does this code lead to an error? If so, in your groups, identify a scenario and the nature of the problem.

```
static void Recursive(int value) {
    Console.WriteLine(value);
    Recursive(++value);
}

static void Main() {
    Recursive(0);
}
```

1. This code is fine
2. NullReferenceException
3. OutOfMemoryException
4. OverflowException
5. IndexOutOfRangeException
6. StackOverflowException

Potential Errors: StackOverflowException

- The program stack has limited memory. It can overflow.
- The problem is caused by an infinite or uncontrolled recursion.
- The Recursive() method calls itself at the end of each invocation.
- https://www.dotnetperls.com/stackoverflowexception

```
static void Recursive(int value) {
    Console.WriteLine(value);
    Recursive(++value);
}

static void Main() {
    Recursive(0);
}
```

Potential Errors: Challenge

Does this code lead to an error? If so, in your groups, identify a scenario and the nature of the problem.

```
static void Main() {
    string value = null;
    if (value.Length == 0){
        Console.WriteLine(value);
    }
    }
}
```

1. This code is fine
2. NullReferenceException
3. OutOfMemoryException
4. DivideByZeroException
5. InvalidCastException
6. StackOverflowException

Potential Errors: NullReferenceException

- It indicates that you are trying to access member fields on an object reference that points to null.
- The string variable does not point to any object on the managed heap. It is equivalent to a null pointer.
- https://www.dotnetperls.com/nullreferenceexception

```
static void Main() {
    string value = null;
    if (value.Length == 0){
          Console.WriteLine(value);
       }
    }
}
```

Potential Errors: Challenge

Does this code lead to an error? If so, in your groups, identify a scenario and the nature of the problem.

```
static void Main(){
    int[] array = null;
    Test(array);
}

static void Test(int[] array) {
    if (array == null) return;
    Console.WriteLine(array.Length);
}
```

1. This code is fine
2. NullReferenceException
3. OutOfMemoryException
4. OverflowException
5. IndexOutOfRangeException
6. StackOverflowException

Potential Errors: NullReferenceException

- If the parameter points to null, the compiler will not know this at compile-time. This a runtime error.
- One may check for null at the start of use of the variable to avoid this error.
- https://www.dotnetperls.com/nullreferenceexception

```
static void Main(){
    int[] array = null;
    Test(array);
}

static void Test(int[] array) {
    if (array == null) return;
    Console.WriteLine(array.Length);
}
```

Potential Errors: Challenge

Does this code lead to an error? If so, in your groups, identify a scenario and the nature of the problem.

```
static void Main() {
    int[] array = new int[100];
    array[0] = 1;
    array[10] = 2;
    array[200] = 3;
}
```

1. This code is fine
2. NullReferenceException
3. OutOfMemoryException
4. OverflowException
5. IndexOutOfRangeException
6. FileNotFoundException

Potential Errors: IndexOutOfRangeException

- This error happens in C# programs that use arrays when a statement tries to access an element at an index greater than the maximum allowable index.
- This means that for an array of 100 elements, you can access array[0] through array[99].
- https://www.dotnetperls.com/indexoutofrangeexception

```
static void Main() {
    int[] array = new int[100];
    array[0] = 1;
    array[10] = 2;
    array[200] = 3;
}
```

Potential Errors: Challenge

Does this code lead to an error? If so, in your groups, identify a scenario and the nature of the problem.

```
static void Main()
{
    string value = new string('a', int.MaxValue);
}
```

1. This code is fine
2. NullReferenceException
3. OutOfMemoryException
4. OverflowException
5. IndexOutOfRangeException
6. StackOverflowException

Potential Errors: OutOfMemoryException

- Memory is limited.
- It can occur during any allocation call during runtime when program requests for RAM memory, but free memory is not available.
- This program attempts to allocate a string that is extremely large and would occupy four gigabytes of memory, that is something not possible.
- https://www.dotnetperls.com/outofmemoryexception

```
static void Main()
{
    string value = new string('a', int.MaxValue);
}
```

Potential Errors: Challenge

Does this code lead to an error? If so, in your groups, identify a scenario and the nature of the problem.

```
static void Main()
{
    checked {
        int value = int.MaxValue + int.Parse("1");
    }
}
```

1. This code is fine
2. NullReferenceException
3. OutOfMemoryException
4. OverflowException
5. IndexOutOfRangeException
6. StackOverflowException

Potential Errors: OverflowException

- An OverflowException is only thrown in a **checked** context.
- It alerts you to an integer overflow: a situation where the number becomes too large to be represented in the bytes.
- An int takes four bytes in size. More bytes would be needed to represent the desired number.
- https://www.dotnetperls.com/overflowexception

```
static void Main()
{
    checked {
        int value = int.MaxValue + int.Parse("1");
    }
}
```

Potential Errors: Challenge

Does this code lead to an error? If so, in your groups, identify a scenario and the nature of the problem.

```
static void Main()
{
    Object human = new Human();
    Panda panda = (Panda)human;
}
```

1. This code is fine
2. NullReferenceException
3. ArgumentException
4. InvalidCastException
5. IndexOutOfRangeException
6. StackOverflowException

Potential Errors: InvalidCastException

- The InvalidCastException occurs when an explicit cast is applied, but the type is not in the same path of the type hierarchy.
- It is generated by the runtime when a statement tries to cast one reference type to a reference type that is not compatible.
- https://www.dotnetperls.com/invalidcastexception

```
static void Main()
{
    // Creates a new object instance of type Human.
    // Then tries to use explicit cast to Panda, but fails.
    Object human = new Human();
    Panda panda = (Panda)human;
}
```

Error handling

C# supports different techniques for error handling, e.g. there are TryParse() and Parse() methods

```
Console.Write("Enter a number: ");
int value = 0;
if(int.TryParse(Console.ReadLine(), out value) == true)
    Console.WriteLine("Thank you.");
else
    Console.WriteLine("That wasn't a number!");
```

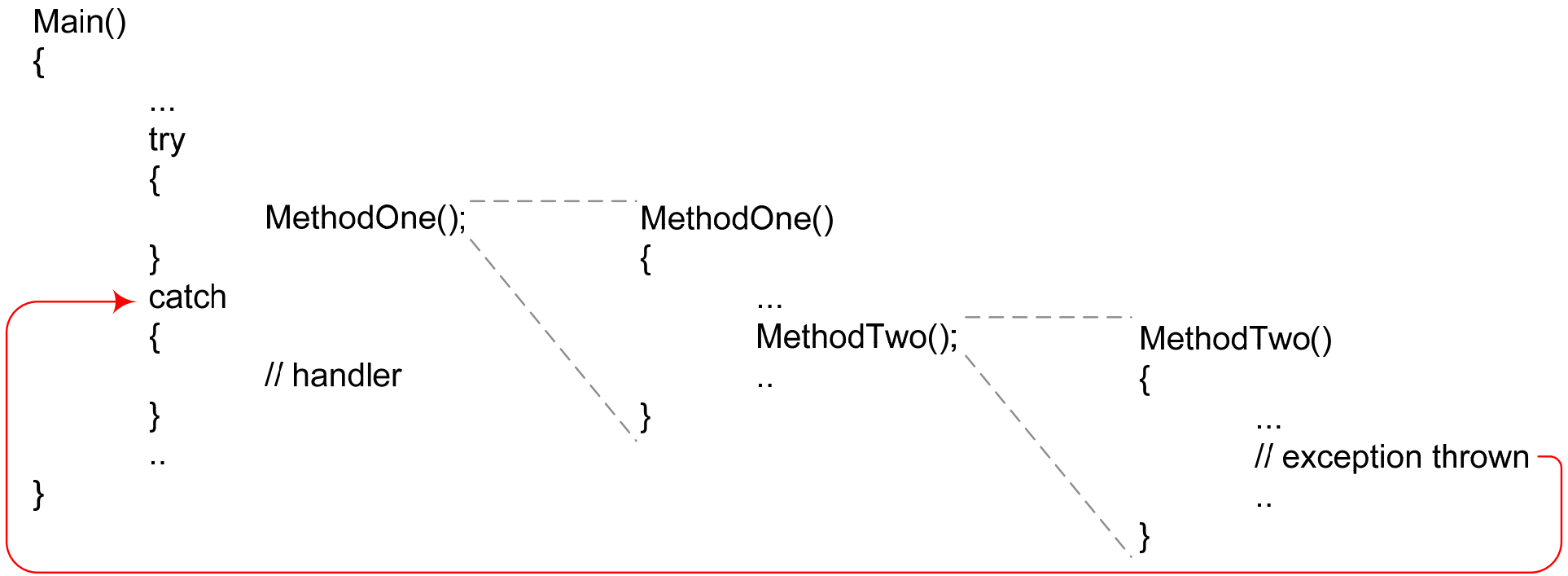
Error handling

C# supports different techniques for error handling, e.g. there are TryParse() and Parse() methods

```
Console.Write("Enter a number: ");
int value = 0;
try
{
    value = int.Parse(Console.ReadLine());
    Console.WriteLine("Thank you.");
}
catch(FormatException)
{
    Console.WriteLine("That wasn't a number!");
}
```

Exceptions

- When an **error** is detected an **object** is created with information about that error.
- Object is “thrown” directly to the error handling routine.



Catching Exceptions

We catch exceptions using a try-catch-finally block:

- **try** block contains code that may throw an exception
- **catch** blocks handle different exceptions that are thrown (zero or more)
- **finally** block executes regardless of exception or not

```
try {
    code that may generate an exception
}

[catch( Exception Type ) {
    error handling code
}]

[...]

[catch {
    error handling code
}]

[finally {
    code always executed after the try/catch
}]
```

- you may catch a particular error by specifying the **catch block**, or
- all potential (other) errors at once when the **catch block** remains unspecified

Note: Order is important!

Throwing Exceptions

Exceptions are thrown (and therefore special objects are created) when an error is detected

- throw new ExceptionType();
- throw new ExceptionType(message);
- throw new ExceptionType(message, inner_exception);

Rethrowing Exceptions

- Sometimes you also need to “rethrow” an exception you have caught:
  - Exception partially handled, changing the exceptional state
  - Need to tidy up/free resources that otherwise wouldn’t be
- Two options:
  - Create a new exception as per previous slide
  - Rethrow the caught exception: throw;

- What does happen to the program if you do not catch nor rethrow the exception?

Microsoft.Net Exception Classes

Microsoft.Net provides many different exception types that should be used where appropriate:

- **StackOverflowException** – the call stack cannot grow any larger;
- **OutOfMemoryException** – the system has run out of memory;
- **NullReferenceException** – when an attempt is made to access an attribute/operation of an object when the reference is set to null;
- **ArgumentException** – one or more arguments were invalid;
- **FileNotFoundException** – specified file does not exist;
- **InvalidCastException** – a data type casting is not valid (usually because the types are unrelated);
- **DivideByZeroException** – attempt made to divide by zero;
- **IndexOutOfRangeException** – array index is out of range;
- **OverflowException** – converting a value, such as with the Convert object, results in a loss of data, e.g., attempting to convert the value 123456 to a byte (which has range 0-255).

Custom Exception

- The provided exception classes will often not be adequate – this is not unusual or a fault
- One can create own exception classes instead by deriving the class representing the exception from the Exception (base) class via the mechanism of inheritance.

Throwing Exceptions: Guidelines

- Use exceptions to notify other parts of the program about errors that should not be ignored
- Throw an exception only for conditions that are truly exceptional
- Don’t use an exception to shift the responsibility for the error to someone else.
- Avoid throwing exceptions in constructors and destructors unless you catch them in the same place.

Throwing Exceptions: Guidelines

- Throw exceptions at the right level of abstraction.
- Include in the exception message all the information that led to the exception.
- Avoid empty catch blocks
- Know the exceptions your library code throws
- Standardize your project's use of exceptions
- Consider alternatives to exceptions