Bachelor of Cyber Security - Pathway Program (Year 1)

SIT232 Object‑Oriented Development Assignment 4.1P

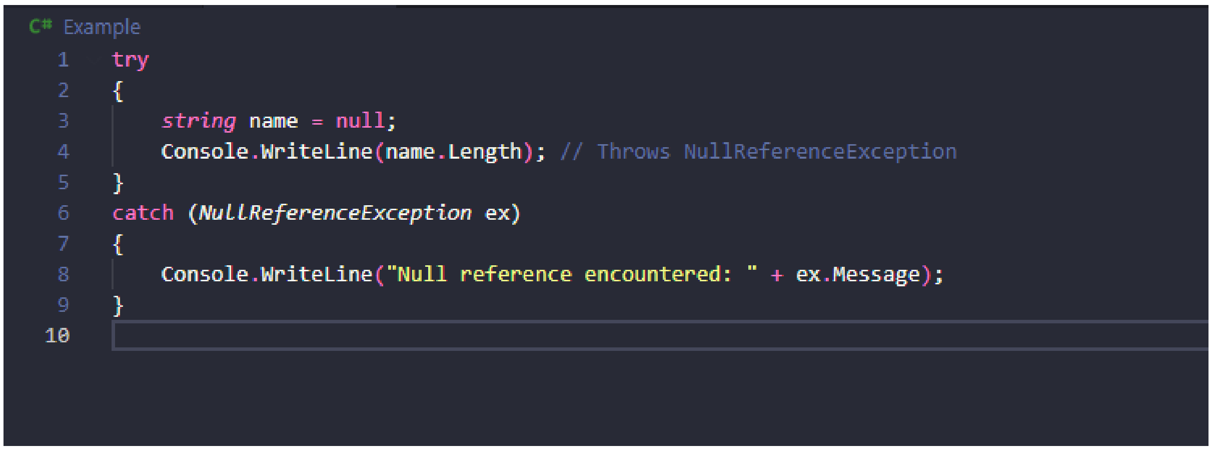
Prepared by :

Charitha B. Pieris | BSCP|CS|62|114

NullReferenceException:

- Possible Situation: This exception occurs when you try to access or call a member (property, method, or field) on an object that is null.
- Thrower: The programmer typically throws this exception by attempting to perform operations on a null reference.
- Message to User: The message should explain which object is null and what the user can do to prevent it.
- Catchability: It can be caught and handled, but it's generally better to avoid it through proper null-checking.
- Handling: You should generally catch and handle this exception because it often indicates a bug or unexpected condition.
- Avoidance: Avoidance is preferred by checking for null references before using them.

Example:



C# Example

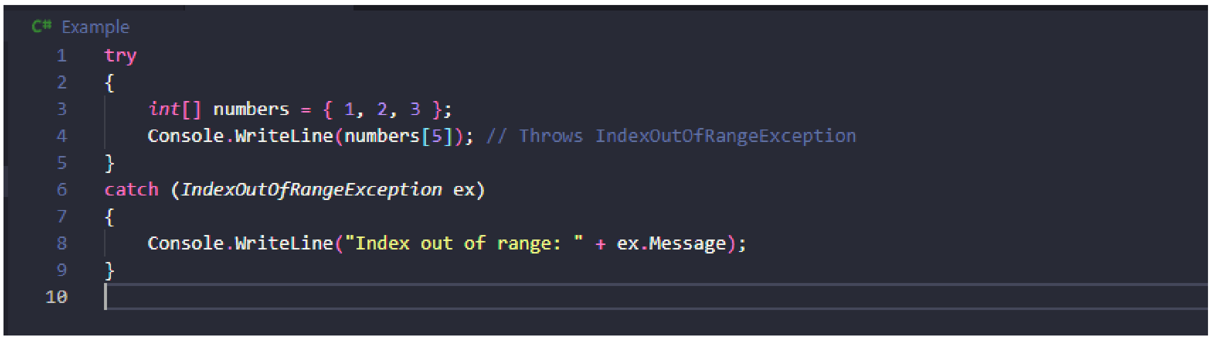
```csharp
try
{
    string name = null;
    Console.WriteLine(name.Length); // Throws NullReferenceException
}
catch (NullReferenceException ex)
{
    Console.WriteLine("Null reference encountered: " + ex.Message);
}

```

IndexOutOfRangeException:

- Possible Situation: Occurs when trying to access an array or collection element with an index that is outside the valid range.
- Thrower: Programmers typically throw this exception by providing an invalid index.
- Message to User: Include the index that caused the exception and specify the valid range.
- Catchability: It can be caught and handled.
- Handling: You should catch and handle it to provide user-friendly feedback or take corrective actions.
- Avoidance: Avoid by ensuring indices are within the valid range before accessing elements.

Example:



C# Example

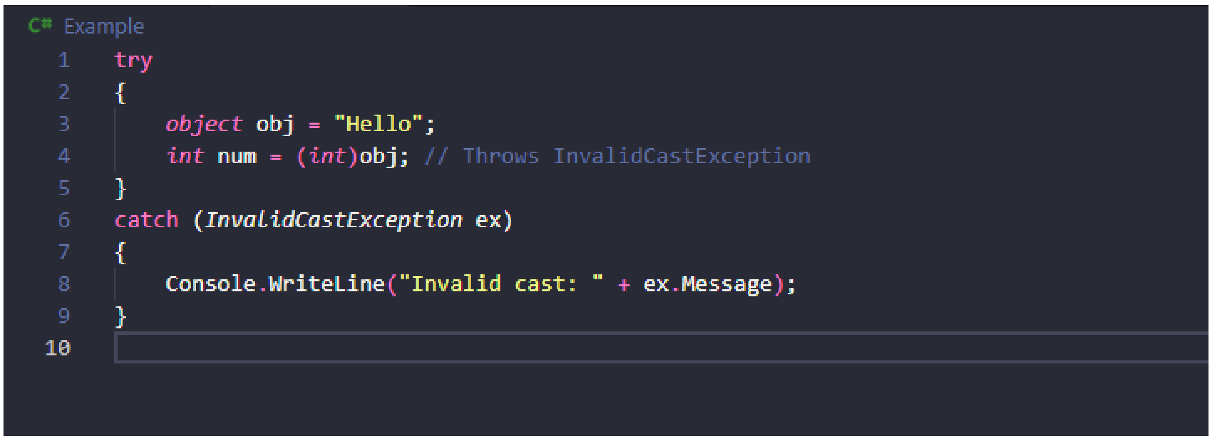
```csharp
try
{
    int[] numbers = { 1, 2, 3 };
    Console.WriteLine(numbers[5]); // Throws IndexOutOfRangeException
}
catch (IndexOutOfRangeException ex)
{
    Console.WriteLine("Index out of range: " + ex.Message);
}

```

StackOverflowException:

- Possible Situation: Occurs when the program's call stack exceeds its limit, often due to recursive method calls.
- Thrower: Usually, it's the runtime system.
- Message to User: Typically, you wouldn't catch this for end-users; it indicates a serious program error.
- Catchability: It's generally not caught but left for the runtime to handle.
- Handling: In most cases, it's not handled directly; you should refactor code to prevent stack overflow.
- Avoidance: Avoid by using iterative approaches instead of excessive recursion.

Example:



```
try
{
    object obj = "Hello";
    int num = (int)obj; // Throws InvalidCastException
}
catch (InvalidCastException ex)
{
    Console.WriteLine("Invalid cast: " + ex.Message);
}
```

OutOfMemoryException:

- Possible Situation: Occurs when there is insufficient memory to allocate an object.
- Thrower: Usually, the runtime system.
- Message to User: Typically not caught for end-users; it signifies a system-level resource problem.
- Catchability: It's usually not caught but left for the runtime to handle.
- Handling: Rarely handled directly; you should optimize memory usage.
- Avoidance: Avoid by managing memory efficiently, disposing of objects, and using appropriate data structures.

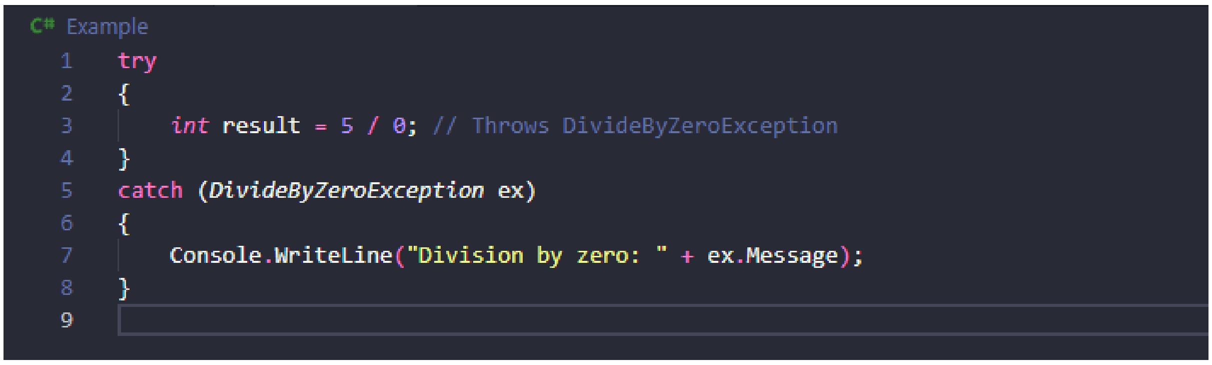
InvalidCastException:

- Possible Situation: Occurs when an invalid cast is attempted, such as casting an incompatible type.
- Thrower: Programmer throws this exception when trying to perform an invalid cast.
- Message to User: Specify the types involved in the cast and why it's invalid.
- Catchability: It can be caught and handled.
- Handling: Catch and handle it to provide meaningful feedback or alternative actions.
- Avoidance: Avoid by checking types before casting or using safe type conversion methods.

DivideByZeroException:

- Possible Situation: Occurs when attempting to divide a number by zero.
- Thrower: Programmer throws this exception by performing the division.
- Message to User: Inform the user about the division operation and why it's not possible.
- Catchability: It can be caught and handled.
- Handling: Catch and handle it to prevent application crashes when division by zero is a possibility.
- Avoidance: Avoid by checking for zero before performing division.

Example:



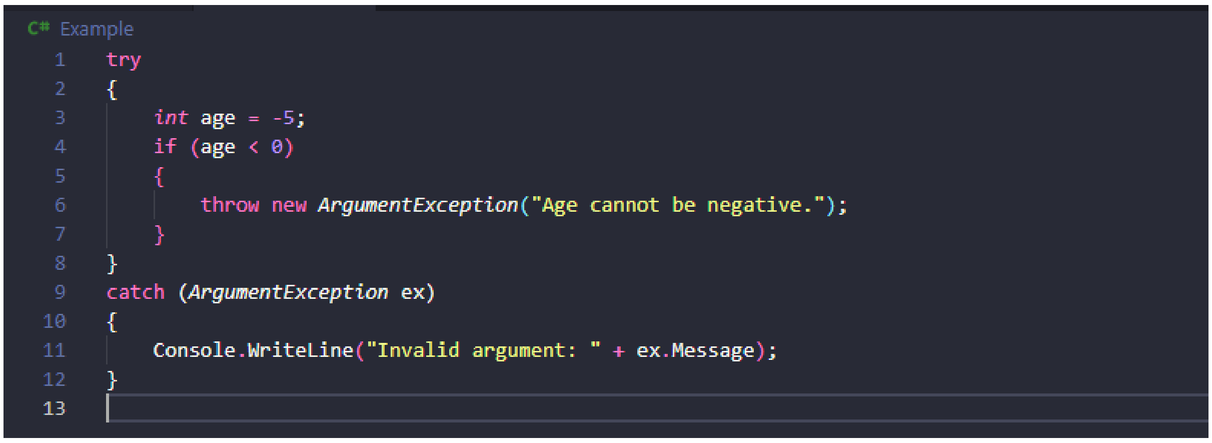
```csharp
try
{
    int result = 5 / 0; // Throws DivideByZeroException
}
catch (DivideByZeroException ex)
{
    Console.WriteLine("Division by zero: " + ex.Message);
}

```

ArgumentException:

- Possible Situation: Occurs when an argument provided to a method is invalid.
- Thrower: Programmers typically throw this exception when validating method arguments.
- Message to User: Explain why the argument is invalid and what valid values should be.
- Catchability: It can be caught and handled.
- Handling: Catch and handle it to provide clear feedback on invalid inputs.
- Avoidance: Avoid by validating input parameters before using them.

Example:



C# Example

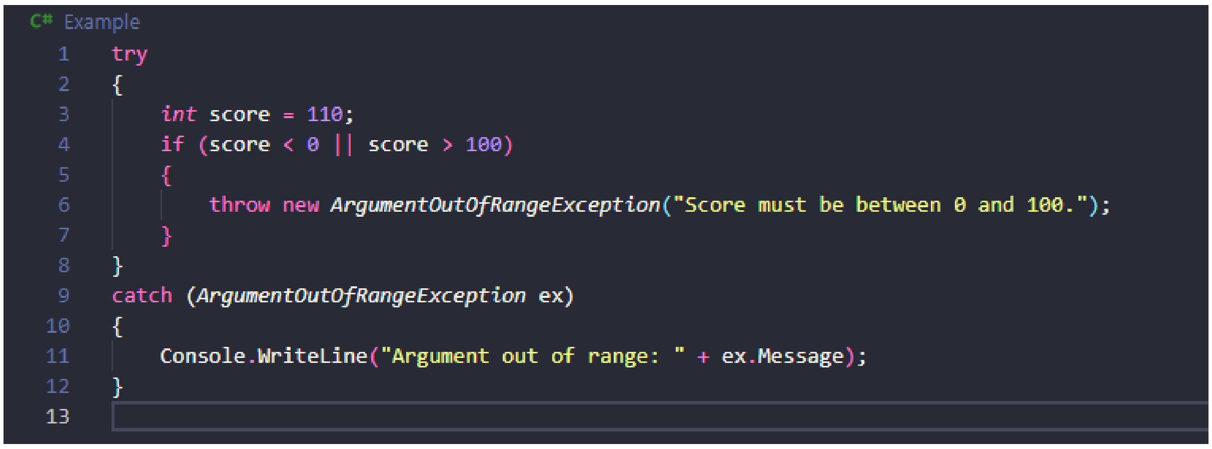
```csharp
try
{
    int age = -5;
    if (age < 0)
    {
        throw new ArgumentException("Age cannot be negative.");
    }
}
catch (ArgumentException ex)
{
    Console.WriteLine("Invalid argument: " + ex.Message);
}

```

ArgumentOutOfRangeException:

- Possible Situation: Occurs when an argument value is outside the acceptable range.
- Thrower: Programmer throws this exception during argument validation.
- Message to User: Indicate the valid range and why the argument is out of range.
- Catchability: It can be caught and handled.
- Handling: Catch and handle to inform users about valid input ranges.
- Avoidance: Avoid by validating input within the acceptable range.

Example:



C# Example

```csharp
try
{
    int score = 110;
    if (score < 0 || score > 100)
    {
        throw new ArgumentOutOfRangeException("Score must be between 0 and 100.");
    }
}
catch (ArgumentOutOfRangeException ex)
{
    Console.WriteLine("Argument out of range: " + ex.Message);
}

```

SystemException:

- Possible Situation: It's a base class for all predefined system exceptions in .NET.
- Thrower: Both the runtime system and programmers can throw exceptions derived from this class.
- Message to User: Depends on the specific derived exception; handle based on the exception type.
- Catchability: Depends on the specific derived exception; some can be caught and handled, others left for the runtime.
- Handling: Handle based on the specific exception and its impact on the application.
- Avoidance: Avoid by following best practices and handling specific exceptions appropriately.