Institute of
Technology
Development
of Canada

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

Parameterized Queries, Using Statements and Exception Handling

Institute of Technology Development of Canada

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

Exception Handling in C#

Institute of Technology Development of Canada

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

Learning Objectives

- Understand the need for handling errors in software and the concept of exceptions.
- Learn the basic structure of try, catch, and finally blocks for managing exceptions.
- Recognize some common exception types in C# and when to use them.
- Get familiar with best practices for handling exceptions to create stable and reliable applications.
- Explore simple techniques for logging and displaying error messages to users.

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

Introduction to Exceptions

- In any software program, it's essential to handle errors and unexpected situations that might occur during runtime. Exceptions are a mechanism in C# that allows you to deal with these situations and ensure your program continues to execute properly.

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

What are Exceptions?

- Exceptions are runtime errors that occur during the execution of a program. These errors can be caused by a variety of reasons such as incorrect input, invalid calculations, or other unexpected situations that the program may encounter. In C#, when an exception is thrown, it causes the program to halt its execution and looks for an exception handler that can handle the exception.

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

Why are Exceptions Important?

- Exceptions are important because they help maintain the stability and reliability of your program. Without proper exception handling, your program could crash, or worse, produce incorrect results. Proper exception handling ensures that your program can handle errors gracefully and continue to function as intended.

Basic Concepts

- The try-catch block is the most common way to handle exceptions in C#. The try block contains the code that may potentially throw an exception. The catch block contains the code that handles the exception. If an exception is thrown within the try block, the program jumps to the catch block to handle the exception.

Institute of Technology Development of Canada

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

Basic Concepts

- The try-catch block is the most common way to handle exceptions in C#. The try block contains the code that may potentially throw an exception. The catch block contains the code that handles the exception. If an exception is thrown within the try block, the program jumps to the catch block to handle the exception.

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

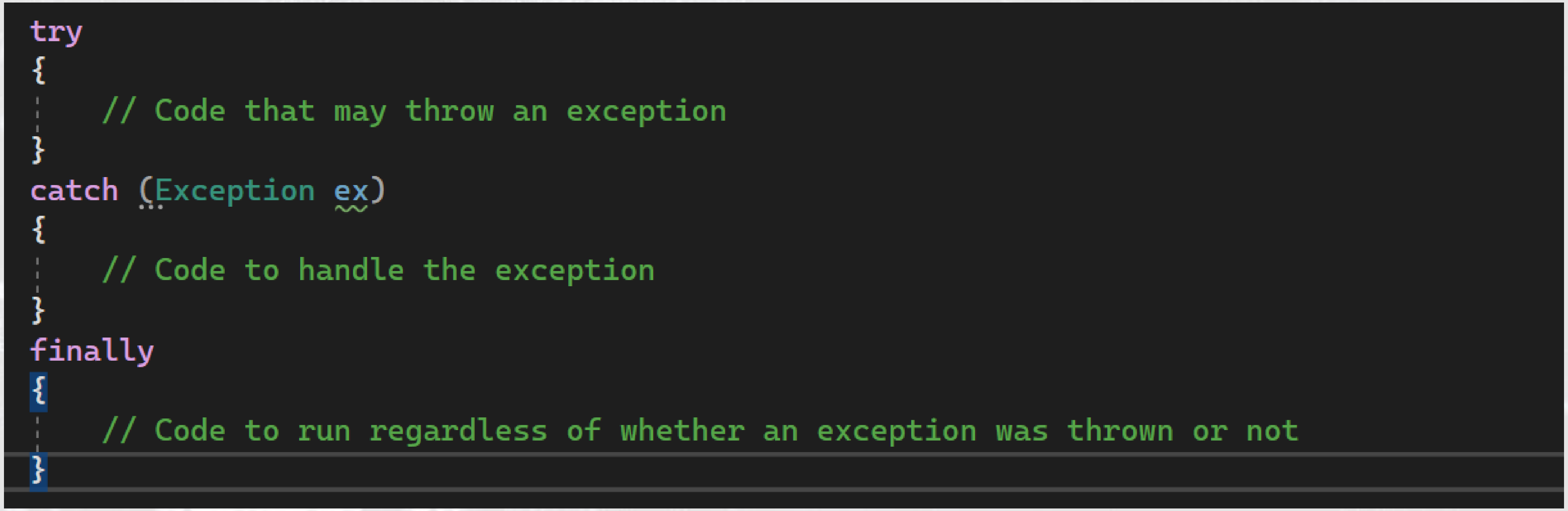
Try-Catch Blocks

- We wrap a block of code that might throw an exception inside a try block.
- If an exception is thrown within the try block, the program immediately jumps to the catch block.
- The catch block contains code that handles the exception. This can include displaying an error message to the user, logging the error for later analysis, or attempting to recover from the error and continue running the program.

Institute of
Technology
Development
of Canada

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

Try Block Anatomy



```
try
{
    // Code that may throw an exception
}
catch (Exception ex)
{
    // Code to handle the exception
}
finally
{
    // Code to run regardless of whether an exception was thrown or not
}
```

Institute of Technology Development of Canada

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

Try-Catch Blocks

- In this example, we put the code that might cause an unexpected event inside a try block. If something unexpected happens, the program will jump to the catch block, which is where we put the code to handle the problem. In the catch block, we specify what kind of problem we're ready to handle (in this case, it's called Exception), and give it a name (ex) that we can use to find out more about what went wrong.
- It's important to note that catch blocks should be used sparingly and only for exceptions that we can handle.

Institute of
Technology
Development
of Canada

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

Try-Catch Blocks – Rethrowing

- Overall, the try-catch block is an essential tool for writing robust and reliable code in C#. By handling exceptions effectively, we can prevent our programs from crashing and provide a better user experience for our use

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

Anti Patterns

- As we all know, software development can be a rollercoaster ride, and unexpected errors are a fact of life. However, fear not, for there is a powerful tool at our disposal to help us navigate these treacherous waters - exception handling. Today, let us explore the best practices in exception handling, so that you may avoid falling into the trap of becoming an exception handling exception.

Institute of Technology Development of Canada

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

Anti Patterns

- Common try-catch anti-patterns in C#:
  1. Swallowing exceptions without logging or handling them properly
  2. Catching all exceptions with a generic catch block instead of specific exceptions
  3. Relying on exceptions to control program flow, instead of using conditional statements
  4. Catching exceptions but not re-throwing them or propagating the error up the call stack
  5. Overusing try-catch blocks, which can lead to verbose and confusing code
  6. Not properly disposing of resources in a finally block, leading to memory leaks and other issues
  7. It's important to be aware of these anti-patterns and strive to avoid them in your code to ensure proper exception handling and reliable software.
- We are going to cover the first three.

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

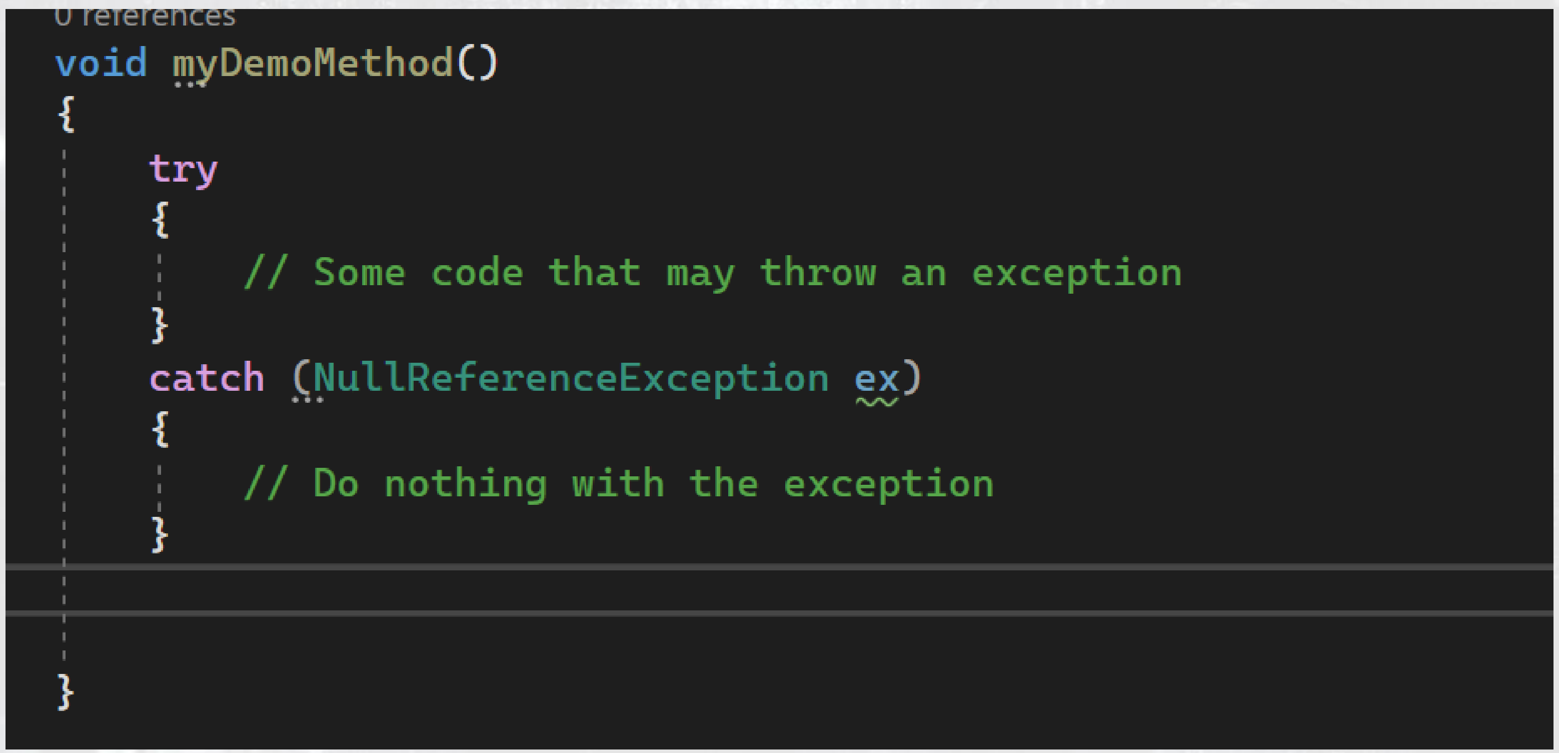
Swallowing exceptions

- One of the most common anti-patterns in exception handling is swallowing exceptions without proper logging or handling. It's like trying to find a needle in a haystack without a magnet - a frustrating and time-consuming task. By logging exceptions and handling them properly, we can make finding and fixing errors much more efficient and effective. Remember, exception handling is like a magnet for your code - it attracts and helps you identify the needles in the haystack.

Institute of Technology Development of Canada

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

Swallowing exceptions



```
0 references
void myDemoMethod()
{
    try
    {
        // Some code that may throw an exception
    }
    catch (NullReferenceException ex)
    {
        // Do nothing with the exception
    }


}
```

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

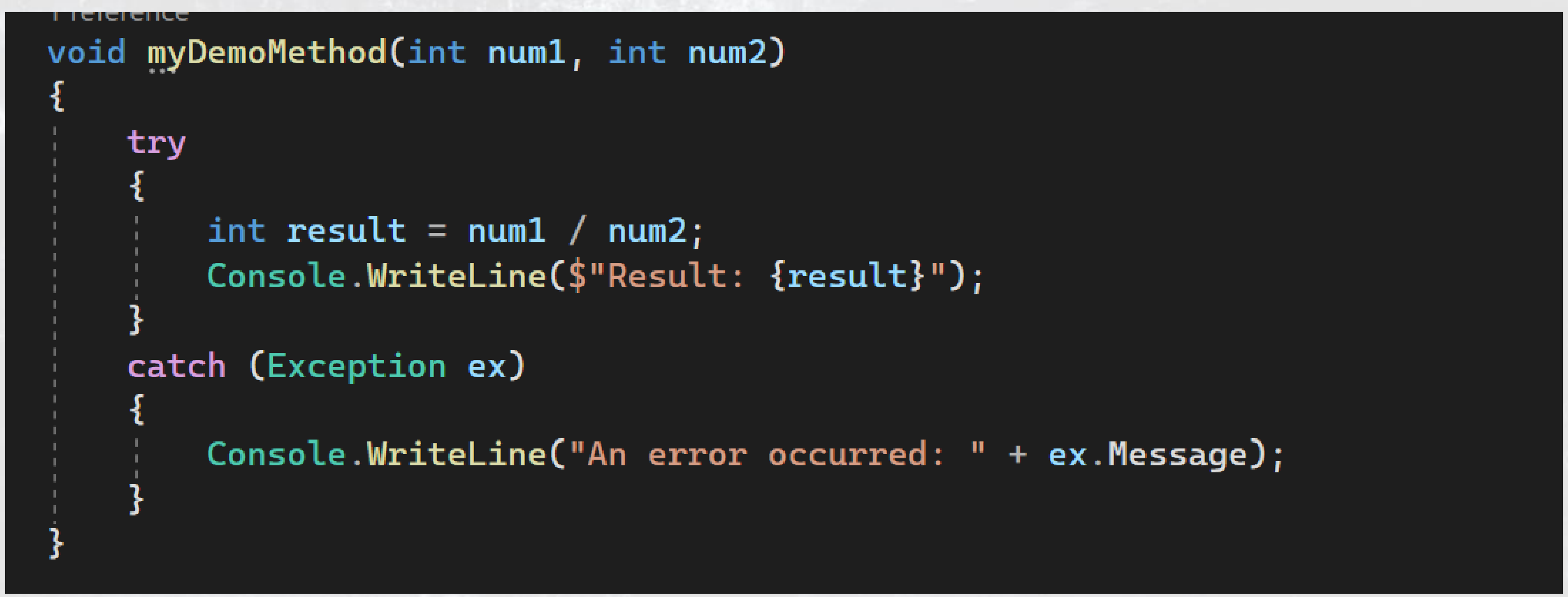
Generic Catch Blocks

- Let's talk about another common programming mistake - using a generic catch block to handle all exceptions, instead of specific ones. This approach can hide crucial information about the error, making it harder to diagnose and fix. It can also make your code harder to maintain, as it's unclear what exceptions are being handled. The best practice is to catch specific exceptions by identifying potential errors in your code and handling them accordingly. This approach improves error handling and makes your code more maintainable. T

Institute of Technology Development of Canada

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

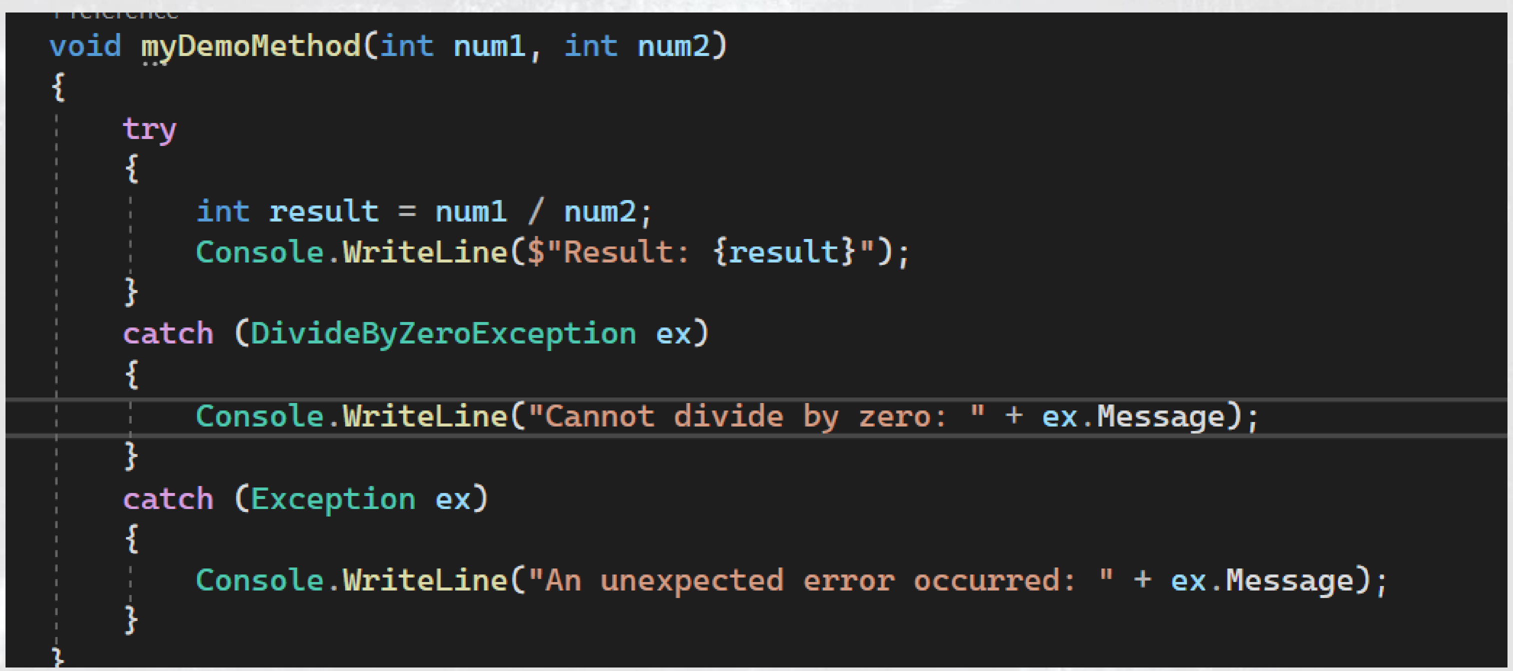
Generic Catch Blocks - Pattern



```
void myDemoMethod(int num1, int num2)
{
    try
    {
        int result = num1 / num2;
        Console.WriteLine($"Result: {result}");
    }
    catch (Exception ex)
    {
        Console.WriteLine("An error occurred: " + ex.Message);
    }
}
```

475 GRANVILLE STREET, VANCOUVER, BC, V6C 1T1
PHONE: +1(604)558-8727, +1(604)409-8200
TOLL FREE: +1(888) 880-4410
FAX: +1(888) 881-6545
WEB: WWW.ITDCANADA.CA
EMAIL: STUDYING@ITDCANADA.CA

GENERIC CATCH BLOCKS - SOLUTION



```
void myDemoMethod(int num1, int num2)
{
    try
    {
        int result = num1 / num2;
        Console.WriteLine($"Result: {result}");
    }
    catch (DivideByZeroException ex)
    {
        Console.WriteLine("Cannot divide by zero: " + ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine("An unexpected error occurred: " + ex.Message);
    }
}
```

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

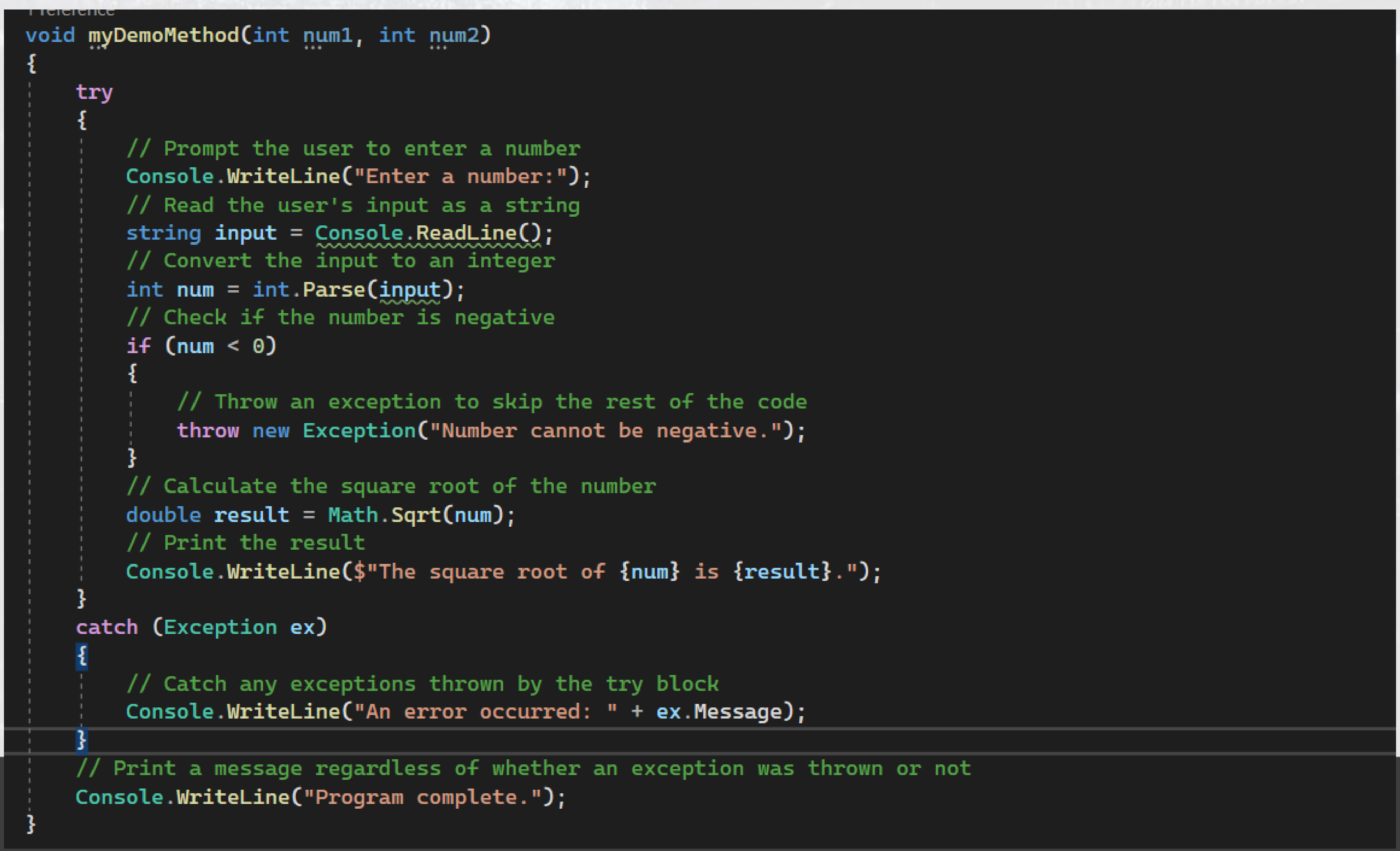
Avoiding Exceptional Control Flow

- You see, an anti-pattern is a common solution to a problem that initially seems correct, but in reality, it's a flawed approach that creates more problems than it solves. In the case of exception handling, some common anti-patterns include ignoring exceptions, catching exceptions and doing nothing with them, or even creating custom exceptions when they're not necessary.

Institute of Technology Development of Canada

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

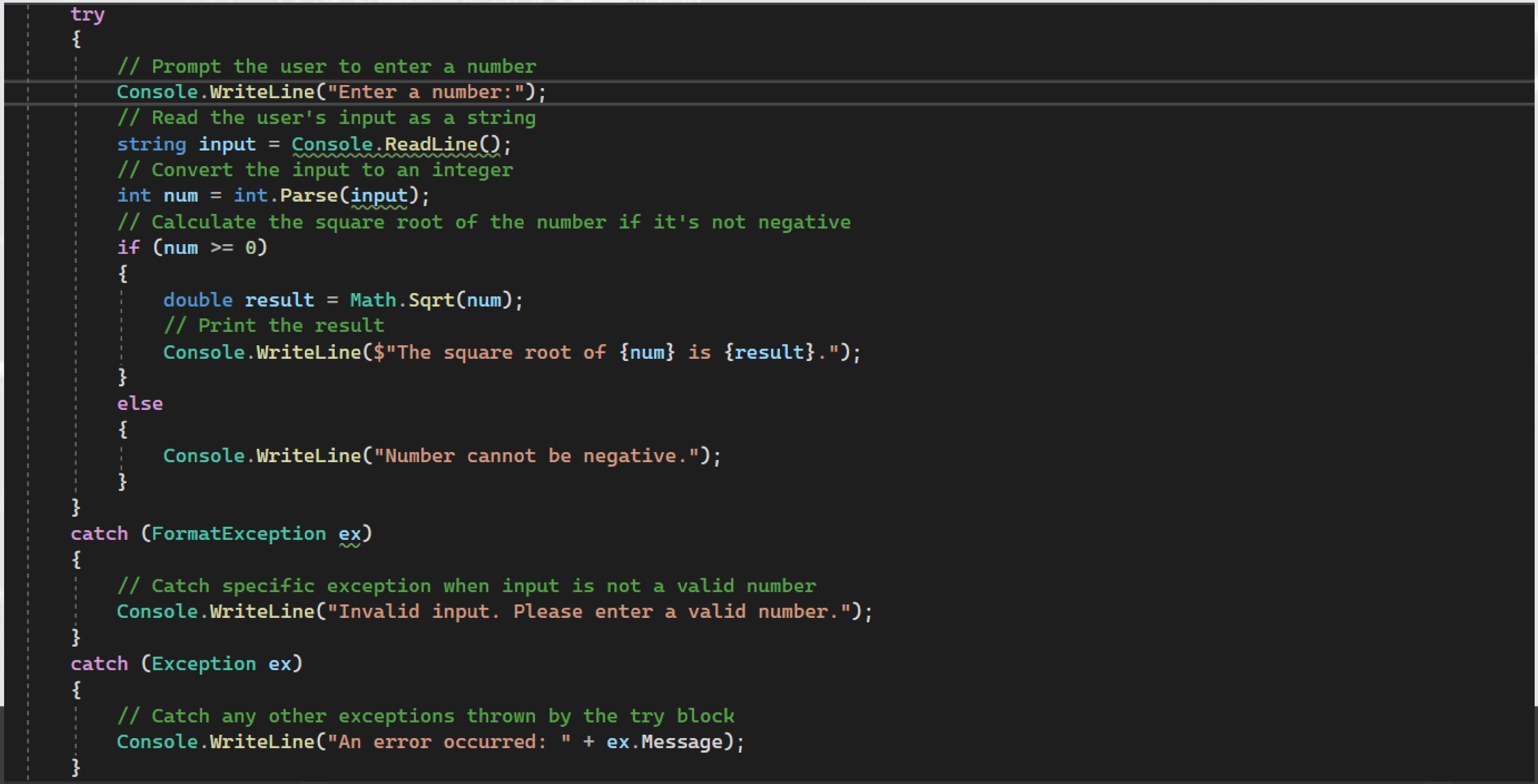
Control Flow – Anti-Pattern



```
1 reference
void myDemoMethod(int num1, int num2)
{
    try
    {
        // Prompt the user to enter a number
        Console.WriteLine("Enter a number:");
        // Read the user's input as a string
        string input = Console.ReadLine();
        // Convert the input to an integer
        int num = int.Parse(input);
        // Check if the number is negative
        if (num < 0)
        {
            // Throw an exception to skip the rest of the code
            throw new Exception("Number cannot be negative.");
        }
        // Calculate the square root of the number
        double result = Math.Sqrt(num);
        // Print the result
        Console.WriteLine($"The square root of {num} is {result}.");
    }
    catch (Exception ex)
    {
        // Catch any exceptions thrown by the try block
        Console.WriteLine("An error occurred: " + ex.Message);
    }
    // Print a message regardless of whether an exception was thrown or not
    Console.WriteLine("Program complete.");
}
```

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

Control Flow - Solution



```
try
{
    // Prompt the user to enter a number
    Console.WriteLine("Enter a number:");
    // Read the user's input as a string
    string input = Console.ReadLine();
    // Convert the input to an integer
    int num = int.Parse(input);
    // Calculate the square root of the number if it's not negative
    if (num >= 0)
    {
        double result = Math.Sqrt(num);
        // Print the result
        Console.WriteLine($"The square root of {num} is {result}.");
    }
    else
    {
        Console.WriteLine("Number cannot be negative.");
    }
}
catch (FormatException ex)
{
    // Catch specific exception when input is not a valid number
    Console.WriteLine("Invalid input. Please enter a valid number.");
}
catch (Exception ex)
{
    // Catch any other exceptions thrown by the try block
    Console.WriteLine("An error occurred: " + ex.Message);
}
```

Institute of Technology Development of Canada

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

Best Practices

- Now, I know it can be tempting to take shortcuts or try to be clever when it comes to exception handling. But it's not worth it. Following best practices and avoiding anti-patterns will not only make your code more robust and easier to maintain, but it will also save you time and headaches in the long run.
- So, let us all strive for excellence in exception handling, and remember the wise words of Benjamin Franklin: "An ounce of prevention is worth a pound of cure."

Institute of
Technology
Development
of Canada

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

Using Statements

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

"Using" Statements for Resource Management

- Resource management is a critical aspect of programming, especially when working with SQL connections. Resource leaks can cause program instability and unexpected behavior. To prevent these issues, we can use the "using" statement in C# to automatically dispose of SQL resources. By using "using" statements, we can ensure that the resources we use are properly managed and disposed of when no longer needed, leading to more stable and efficient programs.

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

"Using" Statements for Resource Management

- Using "using" statements is an effective way to manage SQL connections and other resources in C#. Examples include SqlConnection, SqlCommand, and SqlDataReader objects. By wrapping database operations in a "using" statement, we can ensure that resources are properly disposed of when no longer needed. Additionally, using parameterized queries can prevent SQL injection attacks. Overall, using "using" statements and following best practices can lead to more efficient and secure SQL-connected C# programs.

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

How "Using" Statements Work

- Imagine you are baking a cake in your kitchen. After you are done with the mixing bowl and spatula, you don't just leave them on the counter to collect dust. Instead, you wash them and put them back in their designated places in your kitchen. Similarly, when your program is done using a resource, such as a SQL connection or a file stream, it needs to be properly "cleaned up" and put back where it belongs in order to prevent issues like memory leaks and resource exhaustion.

How "Using" Statements Work

- This is where "using" statements in C# come into play. They help manage resources efficiently by automatically disposing of them when they are no longer needed. It's like having a personal assistant in your kitchen who takes care of washing and putting away your mixing bowl and spatula after you're done using them, without you even having to think about it.

How "Using" Statements Work

- In C#, when you wrap an object in a "using" statement, the compiler generates code that ensures that the object's "Dispose" method is called when the block of code is exited, whether it exits normally or due to an exception. This means that resources are automatically released and cleaned up when the "using" block is finished, without requiring the developer to explicitly call the "Dispose" method.

How "Using" Statements Work

- Think of it like a bouncer at a nightclub who ensures that everyone who enters the club also exits the club at the end of the night. The "using" statement is like the bouncer who ensures that every resource that enters the block of code is also properly disposed of when the block of code is finished.

How "Using" Statements Work

- In conclusion, "using" statements are a powerful tool in C# for managing resources efficiently and preventing issues like memory leaks and resource exhaustion. They act like personal assistants or bouncers, taking care of the details so that developers can focus on writing high-quality code without worrying about resource management.

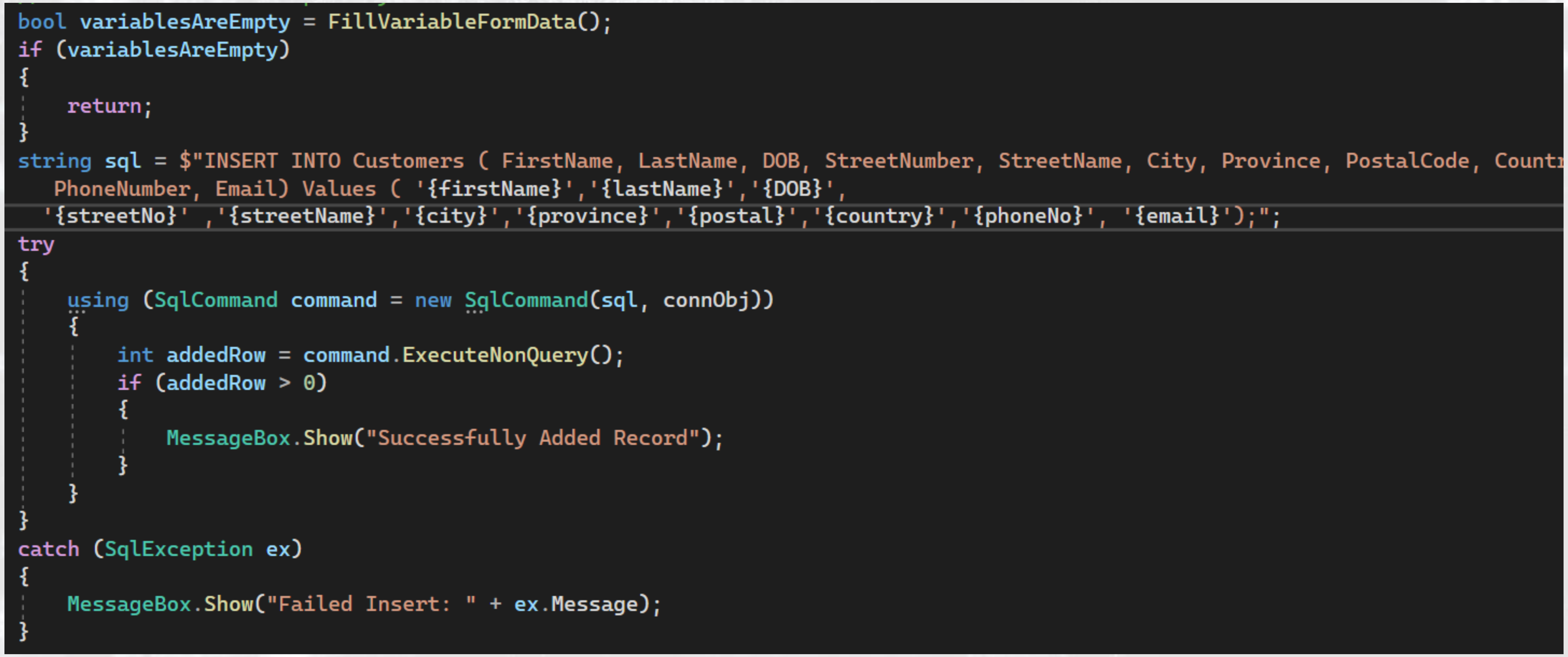
Efficient Resource Practice with 'Using'

- Now, we are going to examine a common mistake in programming, where resources like SQL connections are not disposed of properly, leading to potential memory leaks and other issues. To address this, take a look at the SQL code I have given you for the Insert function. Your task will be to modify the code by incorporating the 'using' statement, which ensures the proper disposal of resources like the SQL connection. By the end of this exercise, you'll gain a deeper understanding of the 'using' statement and be able to effectively create more reliable and efficient software.

Institute of Technology Development of Canada

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

Update you code with the "using"



```
bool variablesAreEmpty = FillVariableFormData();
if (variablesAreEmpty)
{
    return;
}
string sql = $"INSERT INTO Customers ( FirstName, LastName, DOB, StreetNumber, StreetName, City, Province, PostalCode, Countr
   PhoneNumber, Email) Values ( '{firstName}','{lastName}','{DOB}',
  '{streetNo}' ,'{streetName}','{city}','{province}','{postal}','{country}','{phoneNo}', '{email}');";
try
{
    using (SqlCommand command = new SqlCommand(sql, connObj))
    {
        int addedRow = command.ExecuteNonQuery();
        if (addedRow > 0)
        {
            MessageBox.Show("Successfully Added Record");
        }
    }
}
catch (SqlException ex)
{
    MessageBox.Show("Failed Insert: " + ex.Message);
}
```

Institute of
Technology
Development
of Canada

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

Parameterized Queries

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

Introduction to parameterized queries

- Now, we're going to talk about parameterized queries, a fundamental concept in SQL databases that allows us to execute queries with parameters that can be easily customized.
- First, let's discuss why parameterized queries are so important. When we execute queries in SQL databases, we often need to include parameters, such as user input, to retrieve specific data. Without parameterized queries, these parameters can be exploited by malicious users who can execute SQL injection attacks, which can be extremely dangerous for any system.

Introduction to parameterized queries

- To prevent SQL injection attacks, we use parameterized queries, which provide a safer way to execute queries with parameters by binding them to a specific data type, such as a string or integer. This ensures that any user input is properly sanitized and prevents malicious code from being injected into the query.

Institute of
Technology
Development
of Canada

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

SQL query string with parameters

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

SQL query string with parameters

- Imagine you're a composer writing a symphony. Instead of writing every instrument's part down in full detail, you use a sheet with placeholders like "@violin1" and "@cello2". This allows you to focus on the overall structure of the symphony and fill in the specific details later, ensuring each instrument is played correctly. This is similar to using parameter placeholders in SQL queries with C#, where you can focus on the structure of the query and fill in the specific values later, ensuring the query runs correctly.

Institute of Technology Development of Canada

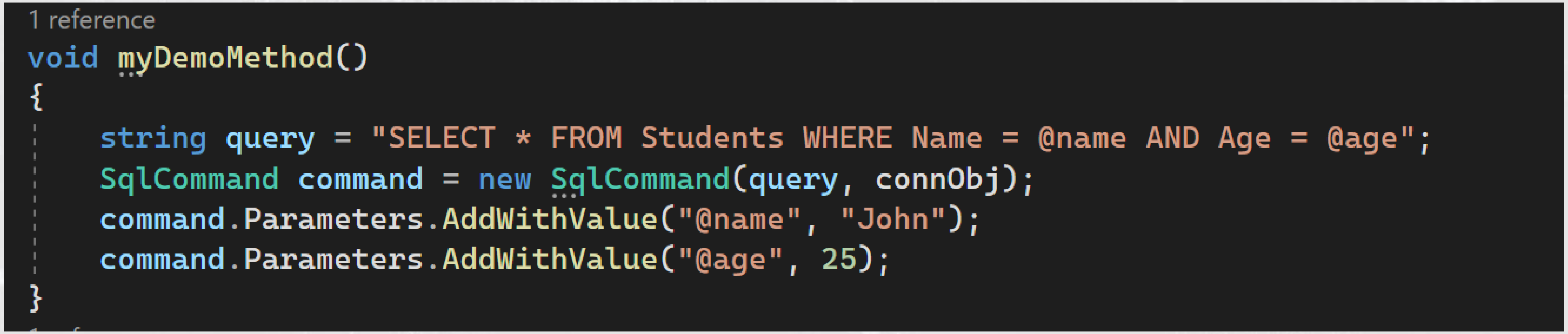
475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

SQL query string with parameters

- Now, let's talk about how we do this in C#. When you define a SQL query string, you use the '@' symbol followed by a name to create a parameter placeholder, like '@name' or '@age.' This enables you to substitute specific values later, keeping your code safe from SQL injection attacks.

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

SQL query string with parameters



```
1 reference
void myDemoMethod()
{
    string query = "SELECT * FROM Students WHERE Name = @name AND Age = @age";
    SqlCommand command = new SqlCommand(query, connObj);
    command.Parameters.AddWithValue("@name", "John");
    command.Parameters.AddWithValue("@age", 25);
}
```

475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

SQL query string with parameters

- Here, we have two parameter placeholders, '@name' and '@age.' When we want to fill in these placeholders, we use the SqlCommand object and its 'Parameters.AddWithValue()' method. It's as simple as passing in the parameter name and its value

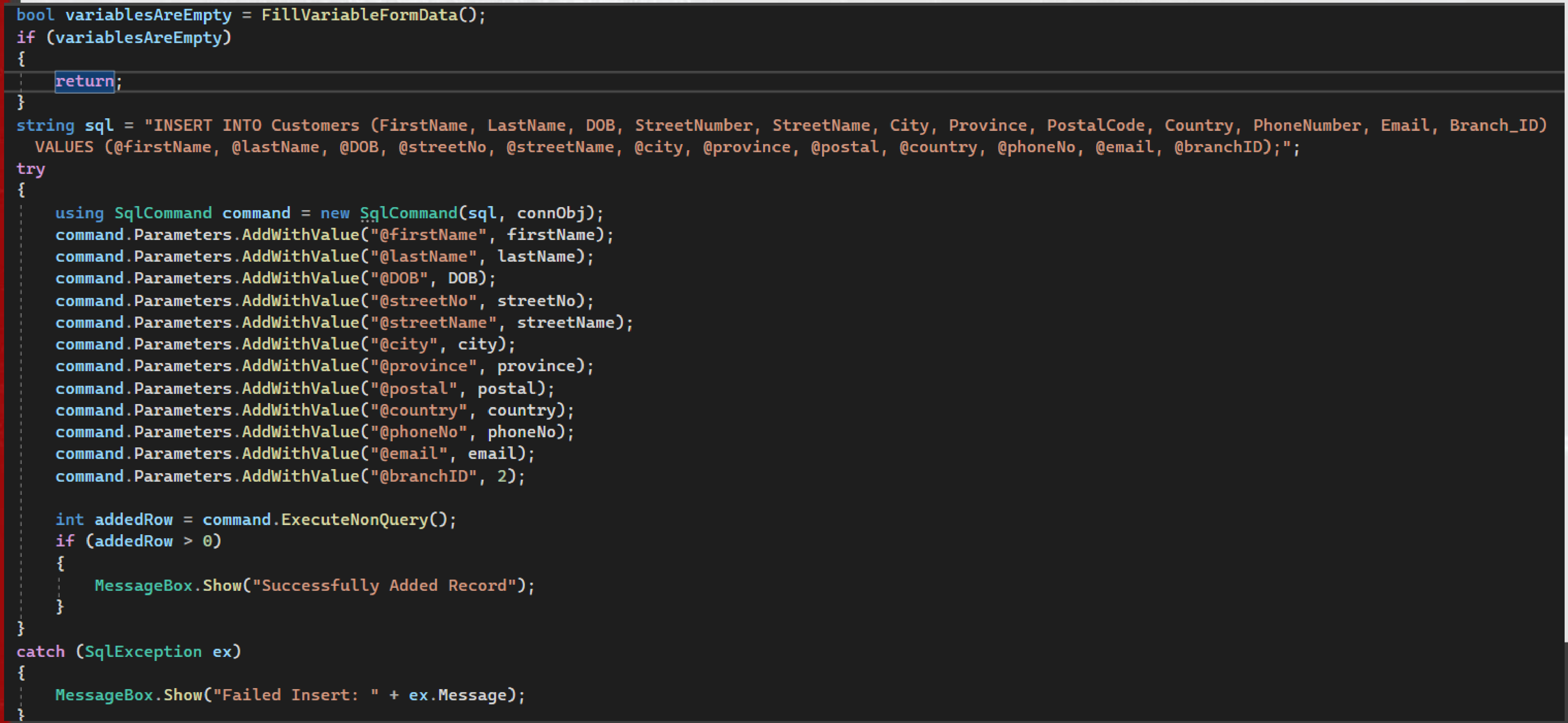
475 Granville Street, Vancouver, BC, V6C 1T1
Phone: +1(604)558-8727, +1(604)409-8200
Toll Free: +1(888) 880-4410
Fax: +1(888) 881-6545
Web: www.itdcanada.ca
Email: studying@itdcanada.ca

SQL query string with parameters

- This approach allows you to use different data types as parameters, such as integers, strings, booleans, and more. Just like in the restaurant example, where the waiter could use the placeholders for various dishes and sides, you can use parameter placeholders to create flexible and secure queries in your C# code. And remember, always be cautious with your queries, just as a waiter would double-check their order notes to ensure customer satisfaction!

475 GRANVILLE STREET, VANCOUVER, BC, V6C 1T1
PHONE: +1(604)558-8727, +1(604)409-8200
TOLL FREE: +1(888) 880-4410
FAX: +1(888) 881-6545
WEB: WWW.ITDCANADA.CA
EMAIL: STUDYING@ITDCANADA.CA

PUTTING IT ALL TOGETHER



```csharp
bool variablesAreEmpty = FillVariableFormData();
if (variablesAreEmpty)
{
    return;
}
string sql = "INSERT INTO Customers (FirstName, LastName, DOB, StreetNumber, StreetName, City, Province, PostalCode, Country, PhoneNumber, Email, Branch_ID)
  VALUES (@firstName, @lastName, @DOB, @streetNo, @streetName, @city, @province, @postal, @country, @phoneNo, @email, @branchID);";
try
{
    using SqlCommand command = new SqlCommand(sql, connObj);
    command.Parameters.AddWithValue("@firstName", firstName);
    command.Parameters.AddWithValue("@lastName", lastName);
    command.Parameters.AddWithValue("@DOB", DOB);
    command.Parameters.AddWithValue("@streetNo", streetNo);
    command.Parameters.AddWithValue("@streetName", streetName);
    command.Parameters.AddWithValue("@city", city);
    command.Parameters.AddWithValue("@province", province);
    command.Parameters.AddWithValue("@postal", postal);
    command.Parameters.AddWithValue("@country", country);
    command.Parameters.AddWithValue("@phoneNo", phoneNo);
    command.Parameters.AddWithValue("@email", email);
    command.Parameters.AddWithValue("@branchID", 2);

    int addedRow = command.ExecuteNonQuery();
    if (addedRow > 0)
    {
        MessageBox.Show("Successfully Added Record");
    }
}
catch (SqlException ex)
{
    MessageBox.Show("Failed Insert: " + ex.Message);
}
```