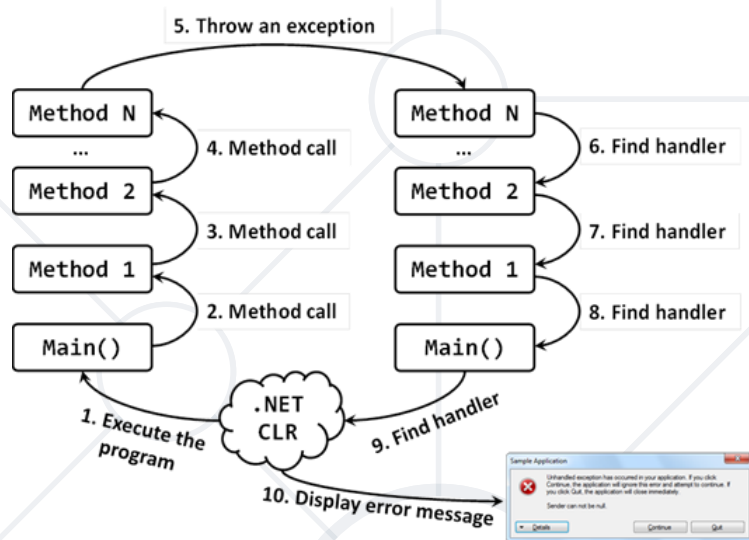


# Exception Handling

## Handling Errors During the Program Execution



SoftUni Team  
Technical Trainers



SoftUni

Software University

<https://softuni.bg>

sli.do

**#prgm-for-qa**

## 1. What are Exceptions?

- The **System.Exception** Class
- Types of Exceptions and their Hierarchy

## 2. Handling Exceptions

- **try-catch-finally**

## 3. Raising (Throwing) Exceptions

- **throw new Exception(...)**





# **The Paradigm of Exceptions in OOP**

What Are Exceptions?

# What Are Exceptions?

- **Exceptions** handle errors and problems at runtime
- **Throw** an exception to signal about a problem

```
if (size < 0)
    throw new Exception("Size cannot be negative!");
```

- **Catch** an exception to handle the problem

```
try {
    size = int.Parse(text);
} catch (Exception ex) {
    Console.WriteLine("Invalid size!");
}
```



# How Do Exceptions Work?

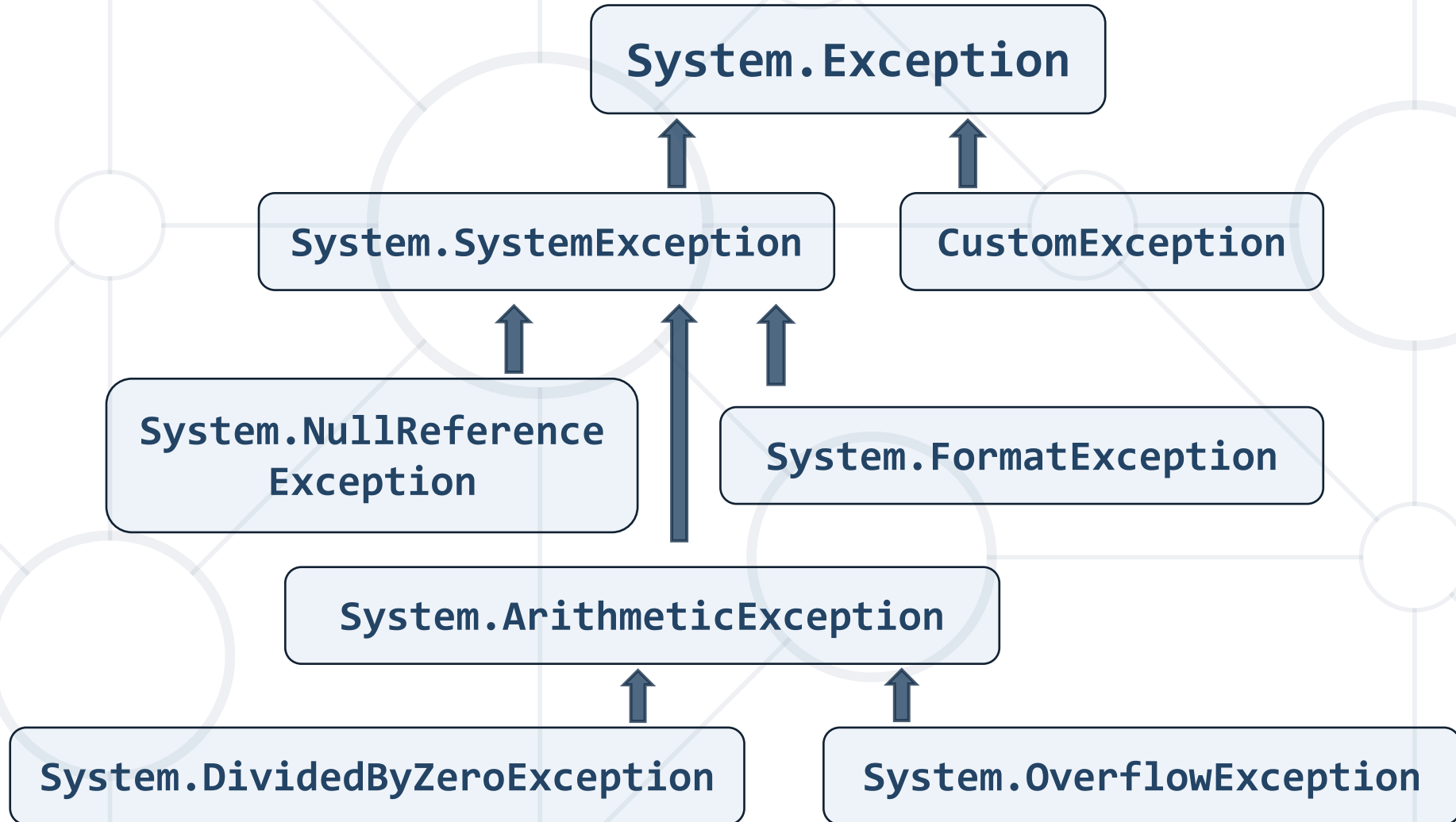


# The System.Exception Class

- Exceptions in C# are **objects**
- The **System.Exception** class is a base for all exceptions in CLR
  - Holds information about the **error**
  - **Message** – a text description of the exception
    - **StackTrace** – the snapshot of the stack at the moment of exception throwing
    - **InnerException** – exception that caused the current exception (if any)



# Exception Hierarchy in .NET







# Catching and Processing Errors

Handling Exceptions

# Handling Exceptions

- In C# exceptions can be handled by the try-catch construction



```
try {  
    // Do some work that can raise an exception  
}  
catch (SomeException) {  
    // Handle the caught exception  
}
```

- **catch** blocks can be used multiple times to process different exception types

# Multiple Catch Blocks – Example

```
string s = Console.ReadLine();
try {
    int.Parse(s);
    Console.WriteLine(
        "You entered a valid Int32 number {0}.", s);
}
catch (FormatException) {
    Console.WriteLine("Invalid integer number!");
}
catch (OverflowException) {
    Console.WriteLine(
        "The number is too big to fit in Int32!");
}
```



- When catching an exception of a particular class, all its inheritors are caught

```
try {  
    // Do some work that can cause an exception  
}  
catch (ArithmeticException ae) {  
    // Handle the caught arithmetic exception  
}
```

- Handles **ArithmeticException** and its descendants **DivideByZeroException** and **OverflowException**

- For **handling all exceptions** (disregarding the exception type, even unmanaged) use the construction:

```
try
{
    // Do some work that can raise any exception
}
catch
{
    // Handle the caught exception
}
```

# The Try-finally Statement

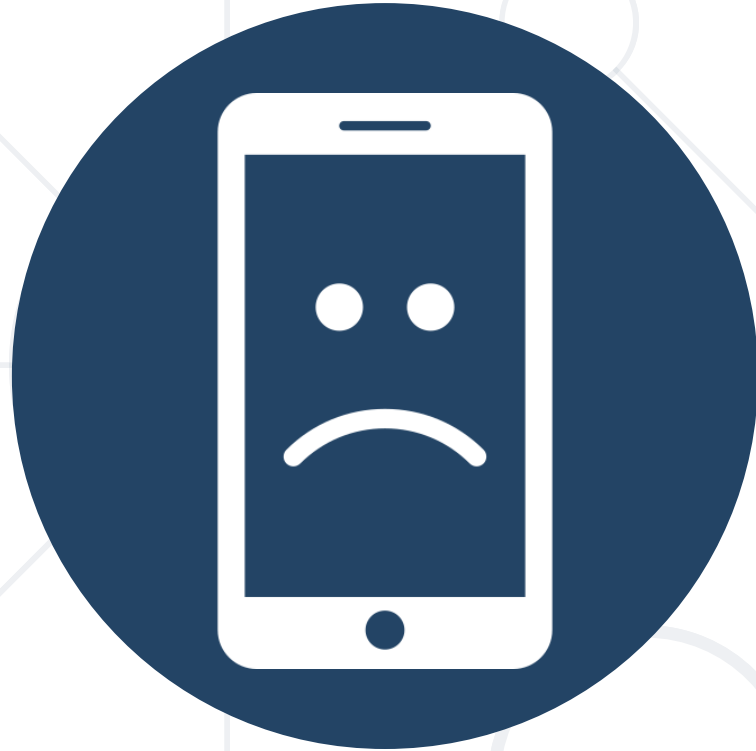
- The **try-finally** statement ensures the **finally** block is **always executed**:

```
try {  
    // Do some work that can cause an exception  
}  
finally {  
    // This block will always execute  
}
```

- Used for execution of **cleanup code**, e. g. releasing resources

# Try-finally – Example

```
static void TestTryFinally() {  
    Console.WriteLine("Code executed before try-finally.");  
    try {  
        string str = Console.ReadLine();  
        int.Parse(str);  
        Console.WriteLine("Parsing was successful.");  
        return; // Exit from the current method  
    } catch (FormatException) {  
        Console.WriteLine("Parsing failed!");  
    } finally {  
        Console.WriteLine("This cleanup code is always executed.");  
    }  
    Console.WriteLine("This code is after the try-finally block.");  
}
```



# Throwing Exceptions

Using the "throw" Keyword



- **Throwing an exception** with an error message:

```
throw new ArgumentException("Invalid amount!");
```

- Exceptions can accept **message** + **another exception** (cause):

```
try {  
    ...  
}  
catch (SQLException sqlEx) {  
    throw new InvalidOperationException("Cannot save invoice.",  
sqlEx); }
```

- This is called "**chaining**" exceptions

- Exceptions are thrown (raised) by the **throw** keyword
- Notify the calling code in case of an error or problem
- When an exception is thrown:
  - The program execution stops
  - The exception travels over the stack
    - Until a matching **catch** block is reached to handle it
- Unhandled exceptions display an error message

- Caught exceptions can be **re-thrown** again:

```
try {  
    Int32.Parse(str);  
}  
catch (FormatException fe) {  
    Console.WriteLine("Parse failed!");  
    throw fe; // Re-throw the caught exception  
}
```

```
catch (FormatException) {  
    throw; // Re-throws the last caught exception  
}
```

# Throwing Exceptions – Example

```
public static double Sqrt(double value) {  
    if (value < 0)  
        throw new System.ArgumentOutOfRangeException("value",  
            "Sqrt for negative numbers is undefined!");  
    return Math.Sqrt(value);  
}  
static void Main() {  
    try {  
        Sqrt(-1);  
    }  
    catch (ArgumentOutOfRangeException ex) {  
        Console.Error.WriteLine("Error: " + ex.Message);  
        throw;  
    }  
}
```

- Exceptions provide a **flexible** error handling mechanism

```
throw new Exception("Invalid size!");
```

- **Try-catch** allows exceptions to be handled

```
try { ... } catch (Exception ex) { ... }
```

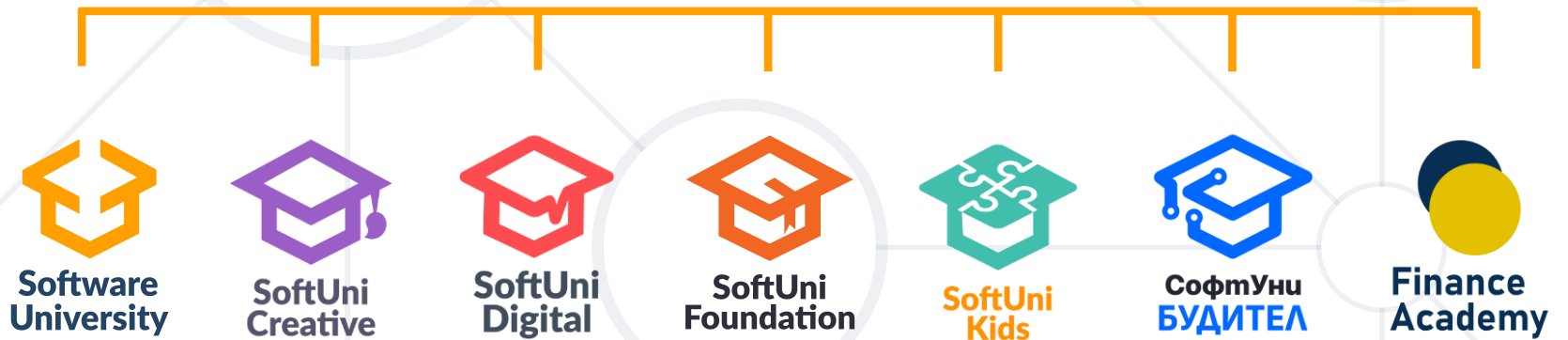
- Unhandled exceptions cause error messages
- **Try-finally** ensures a given code block is always executed



# Questions?



SoftUni



# SoftUni Diamond Partners



- Software University – High-Quality Education, Profession and Job for Software Developers
  - [softuni.bg](http://softuni.bg), [about.softuni.bg](http://about.softuni.bg)
- Software University Foundation
  - [softuni.foundation](http://softuni.foundation)
- Software University @ Facebook
  - [facebook.com/SoftwareUniversity](https://facebook.com/SoftwareUniversity)





- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg>
- © Software University – <https://softuni.bg>

