

Exception



- A exceptional circumstance
- Error condition or unexpected behavior while executing a code statement
- Examples:
 - Division by zero
 - Using a reference variable that points to null
 - Parsing "Dan" to a decimal

Exceptions are a good thing



- A mechanism for API developers to alert other developers that they are using class behaviors incorrectly
- Generated by
 - OS
 - API Developers


Identify Potential Misuse



```
public class Person
{
    private decimal amountOfMoney;

    public void AddMoney(decimal amountOfMoneyToAdd)
    {
        this.amountOfMoney += amountOfMoneyToAdd;
    }
}
```

How to handle error?




```
public class Person
{
    private decimal amountOfMoney;

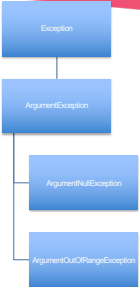
    public void AddMoney(decimal amountOfMoneyToAdd)
    {
        if (amountOfMoneyToAdd < 0)
        {
            // ?
        }

        this.amountOfMoney += amountOfMoneyToAdd;
    }
}
```

Exception Hierarchy




- Exceptions are objects
- System.Exception
 - base class for all exceptions



```
graph TD
    Exception[Exception] --> ArgumentException[ArgumentException]
    ArgumentException --> ArgumentOutOfRangeException[ArgumentOutOfRangeException]
    ArgumentOutOfRangeException --> ArgumentOutOfRangeException
```

Generate an Exception



- Two Steps
 1. Construct an Exception instance
 2. Throw the instance

Throwing Exceptions

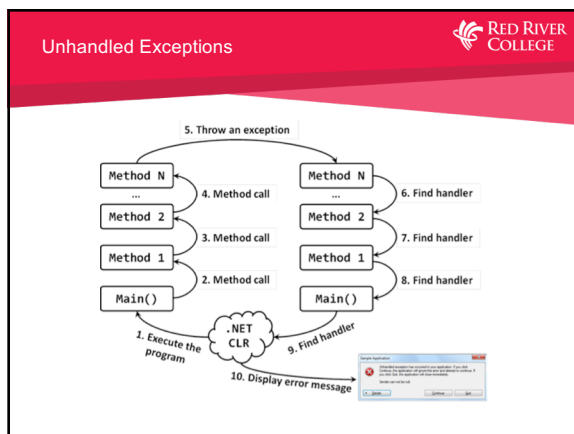
RED RIVER COLLEGE

```

public class Person
{
    private decimal amountOfMoney;

    /// <summary>
    /// Increments the amount of money the Person has.
    /// </summary>
    /// <param name="amountOfMoneyToAdd">The amount of money to increment by.</param>
    /// <exception cref="System.ArgumentOutOfRangeException">Thrown when ...</exception>
    public void AddMoney(decimal amountOfMoneyToAdd)
    {
        if (amountOfMoneyToAdd < 0)
        {
            throw new ArgumentOutOfRangeException("amountOfMoneyToAdd",
                "Argument cannot be less than zero.");
        }
        this.amountOfMoney += amountOfMoneyToAdd;
    }
}

```



Control the Flow

RED RIVER COLLEGE

- Exceptions interrupt flow
- Exceptions not handled will **abend** the program
- Handling exceptions controls the flow

Handling Exceptions



- try...catch construct
- Try to execute code you know could cause an exception
- Catch an Exception when one is thrown

Try...Catch Syntax




```
try
{
    // code that could cause an exception
    // code I don't want to execute if an exception occurs
}
catch(ExceptionType [identifier])
{
}
[catch(ExceptionType identifier)
{
}]
}}
```

Try...Catch Example



```
try
{
    me.AddMoney(-100.50m);
}
catch (ArgumentOutOfRangeException ex)
{
    // Error message to the user
    // Write to an error log
}
```

StackTrace




- Property of an exception
- Provides information about the exception
 - Where in the code did the exception happen (line number)?
 - What method did it occur in?
 - What variables/values were used?

Best Practices



- DO NOT wrap all your code in a try...catch
 - Coded when you anticipate an exception
 - Coded where the exception is to be handled
- Don't display exception information to the user
- Do not ignore exceptions you catch
 - never have an empty catch block

Summary



- Exceptions = objects
- Constructed and thrown
- Unhandled exceptions will abend app
- Handled by try...catch

