Part 14 - Exceptions

Part 14 - Exceptions

> **Definition: Exception**
> A mechanism designed to handle runtime errors or other problems (exceptions) inside a computer program.

Exceptions are very important, as they are raised whenever an error occurs in the system. (Or at least they should be.)

Catching Exceptions

An exception stops the program if it is not caught.

Division by Zero

```
print 1 / 0
```

Output

```
System.DivideByZeroException: Attempted to divide by zero.
   at Test.Main(String[] argv)
```

Which stopped the program.

To handle the situation, exceptions must be caught.
`Exceptions` are either caught in a `try-except` statement, a `try-ensure` statement, or a `try-except-ensure` statement.
Also, all `Exceptions` are derived from the simple `Exception`.

try-except example

```
import System

try:
    print 1 / 0
except e as DivideByZeroException:
    print "Whoops"
print "Doing more..."
```

Output

```
Whoops
Doing more...
```

This prevents the code from stopping and lets the program keep running even after it would have normally crashed.

There can be multiple `except` statements, in case the code can cause multiple `Exceptions`.

`Try-ensure` is handy if you are dealing with open streams that need to be closed in case of an error.

try-ensure example

```
import System

try:
    s = MyClass()
    s.SomethingBad()
ensure:
    print "This code will be executed, whether there is an error or not."
```

Output

```
This code will be executed, whether there is an error or not.
System.Exception: Something bad happened.
   at Test.Main(String[] argv)
```

As you can see, the `ensure` statement didn't prevent the `Exception` from bubbling up and causing the program to crash.

A `try-except-ensure` combines the two.

try-except-ensure example

```
import System

try:
    s = MyClass()
    s.SomethingBad()
except e as Exception:
    print "Problem! $(e.Message)"
ensure:
    print "This code will be executed, whether there is an error or not."
```

Output

```
Problem: Something bad happened.
This code will be executed, whether there is an error or not.
```

Recommendation
If you don't solve the problem in your `except` statement, use the `raise` command without any parameters to re-`raise` the original `Exception`.

Raising Exceptions

There are times that you want to `raise` `Exceptions` of your own.

Raising an Exception

```
import System

def Execute(i as int):
    if i < 10:
        raise Exception("Argument i must be greater than or equal to 10")
    print i
```

If `Execute` is called with an improper value of `i`, then the `Exception` will be `raised`.

Recommendation
In production environments, you'll want to create your own `Exceptions`, even if they are just wrappers around `Exception` with different names.

Recommendation
Never use `ApplicationException`.

Exercises

1. Think of an exercise

Go on to **Part 15 - Functions as Objects and Multithreading**