EXCEPTIONS AND ERROR HANDLING

TYPES OF ERRORS

- Syntax errors violate the rules for how C# statements must be written.
- Runtime errors don't violate syntax rules, but cause exceptions to be thrown which stop the execution of the application.
- Logic errors don't cause syntax or runtime errors, but produce incorrect results.

COMMON SYNTAX ERRORS

- Misspelling keywords
- Forgetting to declare a data type for a variable
- Forgetting an opening or closing parenthesis, bracket, quotation mark, or comment character
- Forgetting to code a semicolon at the end of a statement

HANDLING EXCEPTIONS

How Exceptions Work

An exception is thrown when the application can't perform an operation. An exception is an object created from one of several different Exception classes.

HANDLING EXCEPTIONS

How Exceptions Work

An application is said to catch any thrown exceptions and handle them, which may involve simply informing the user they need to provide valid input or more complicated action.

HANDLING EXCEPTIONS

In order to catch an exception, we use blocks of statements, called the try statement (which contains code that may throw an exception) and exception handler (which details the appropriate response to an exception).

SYNTAX

```
try
{ statements
}
catch (ExceptionClass exceptionName)
{
  statements
}
```

EXAMPLE

```
double subtotal = 0.0;
    try{
    Console.WriteLine("Enter subtotal:");
    subtotal = Double.Parse(Console.ReadLine());
    }
    catch (FormatException e)
    {
        Console.WriteLine("Error! Invalid number.");
    }
```

HAVING MULTIPLE CATCHES

```
try{ // statements causing exception
}
catch( ExceptionName e1 )
{   // error handling code
}
catch( ExceptionName e2 )
{   // error handling code
}
catch( ExceptionName e3 )
{   // error handling code
}
```

THROW STATEMENT

Gives the programmer the ability to generate an exception.

```
throw new Exception("Interst rate must be > 0.");
```

VALIDATING DATA

Preventing Exceptions

We will generally wish to prevent exceptions being thrown due to invalid data. We can use data validation to display an error message when the user inputs invalid content until all entries are valid.

NUMERIC ENTRIES

- Make sure the entry has a valid numeric format
- Make sure that the entry is within a valid range. This is known as range checking.

VALIDATING STRING INPUT

Use the TryParse!

```
double price;
bool isDouble = Double.TryParse(inputString, out price);
if(isDouble) {
// process the input here
}
```

SOFTWARE TESTING

To test an application is to run it to make sure it works correctly, trying every possible combination of input data and user actions to be certain it work in every case.

SOFTWARE TESTING PHASES

- Check the user interface to make sure that it works correctly.
- Test the application with valid input data to make sure the results are correct.
- Test the application with invalid data or unexpected user actions. Try everything you can think of to make the application fail.

SOFTWARE DEBUGGING

Debugging an application means actually fixing the errors (or bugs) identified by testing the it. The app must then be tested again to make sure the fix hasn’t caused more errors.

LOGICAL ERRORS

Logical errors are errors that will not make your program run as expected.

EXAMPLES:

- Initializing variables to the wrong value.
- Doing the wrong operation (doing addition instead of a difference).

RECAP

WHAT YOU SHOULD KNOW AT THIS POINT

- Error Types.
- How to handle exceptions and runtime error.
- How to validate input.
- How to test and debug your code.
- How to fix syntax errors.
- How to detect logical errors.