CHEATSHEET - Debugging in C#

DEBUGGING IN C#

tutorials.EU

Debugging Basics

Breakpoints: Points in code where execution pauses to inspect the state.
Stepping: Step Over, Step Into, and Step Out commands to navigate through code.
Variable Inspection: Checking values of variables during execution.

Locals and Autos

Locals Window: Displays local variables in the current scope.
Autos Window: Shows variables used around the current line of execution.

Essential Topics

List Handling: Techniques for creating and managing copies of lists.
Bug Resolution: Common bugs and debugging solutions.
Call Stack: Trace function calls leading to the current point of execution.
Throwing Errors: Generate exceptions intentionally for testing.
Defensive Programming: Writing code to anticipate and handle potential errors.

Try and Catch

Syntax: Structure for handling exceptions.

```
try {
  // Code that may throw an exception
} catch (ExceptionType e) {
  // Code to handle the exception
}
```

Flow: Execution moves to the catch block if an exception is thrown in the try block.

The Finally Keyword

Usage: Code that executes regardless of an exception.

```
try {
  // Code that may throw an exception
} catch (ExceptionType e) {
  // Code to handle the exception
} finally {
  // Code that always executes
}
```

Try Catch vs. If Statements

Use try-catch for exceptions and if statements for logical conditions.

Debug Log

Techniques to log information for debugging.
Syntax: Debug.WriteLine("Log message");

Throw Keyword

Manually throwing exceptions.
Syntax : throw new Exception("Error message");

Managing Multiple Types of Exceptions

Multiple Catch Blocks: Handling different exceptions separately.

Syntax :

```
try {
  // Code that may throw exceptions
} catch (ArgumentException e) {
  // Handle ArgumentException
} catch (InvalidOperationException e) {
  // Handle InvalidOperationException
}
```

Exception Handling Techniques

Custom Exceptions: Creating user-defined exception classes

Syntax :

```
public class CustomException : Exception {
  public CustomException(string message)
: base(message) { }
}
```

Why the Default Exceptions Rock:
Advantages: Reliability and comprehensive coverage of default .NET exceptions.

How Exceptions Work with the Call Stack:
Propagation: How exceptions move up the call stack until caught.

How to Handle Exceptions Properly:
Best Practices: Ensure proper handling and logging of exceptions to maintain application stability.