AdIns
Advance Innovations

# Debugging

# Topics

- Overview
- Bug Categories
- Preventing Bugs
- Starting the Debugger
- Controlling Execution
- Observing States
- Projects Settings
- Advanced Debugging Techniques

# Overview

- In an ideal world, all programmers would be so skilled and attentive to detail that they would write bug-free code.
- Unfortunately, we do not live in an ideal world.
- Debugging is an important task that all developers need to perform before they allow end-users to use their applications

# Bug Categories

- Syntax errors
  - Occur when code breaks the rules of the language you're using.
  - The easiest to locate.
- Semantic errors
  - Occur in code that is correct according to the rules of the compiler, but that causes unexpected problems such as crashes or hanging on execution.
  - Harder to detect and are one type of runtime error.

# Bug Categories (Cont.)

- Logic errors
  - Like semantic errors, logic errors are runtime errors.
  - Occur while the program is running but unlike semantic errors, logic errors result in unexpected values or output.

# Preventing Bugs

- Write readable code
  - Choose (or develop) and make consistent use of naming and coding standards.
  - Anything that makes code more readable and easier to understand will help eliminate bugs from the get-go.

# Preventing Bugs (Cont.)

- Create effective test plans
  - Test every path of your application with every possible data value (or representative range of data) that a user could enter.
  - Test plans and other design documents can also help highlight design problems before they are implemented, preventing a wide array of bugs.

# Preventing Bugs (Cont.)

- Use a rich IDE
  - Consider developing in an IDE that provides syntax checking as you type.
- Get another pair of eyes
  - It is important to have someone else review your code and test it, if possible.
  - A review by a different pair of eyes can help you catch what you've missed.

# Starting the Debugger

- Launching to Debug
- Attaching to a Running Process
- Just-in-Time Debugging
- ASP.NET Debugging
- Client-Side Script Debugging

# Controlling Execution

- Breakpoints
  - Data breakpoints
  - The Breakpoints window
- Halting on Errors
- Single-Stepping
  - Stepping through multiple lines
  - Changing the current point of execution
  - Edit and continue

# Observing States

- Displaying Variables and Expressions
  - Watch windows
  - Autos, Locals, and This
- Registers
- The Call Stack
- Memory Windows
- The Output Window
- The Modules Window

# Projects Settings

- Release-Only Bugs
  - Some bugs occur only in release mode.
  - Fortunately, you can attach a debugger to a release build. However, you must be careful how you do so if you want the results to be useful.
  - If you build with a project configuration that has both debugging information and optimization enabled, you will still be able to use most of the debugger's normal functionality.

# Projects Settings (Cont.)

- Choosing Debugging Modes
  - You have the same flexibility when launching a program from within Visual Studio .NET as you do when attaching to an existing one.
  - For unmanaged projects, you can select whether you want CLR (Managed Only), Native, or both.

# Advanced Debugging Techniques

- Crossing Language and Technology Boundaries
- Multiple Threads
- Multiple Processes
- Cross-Machine Debugging
  - T-SQL debugging
- Alternative Debugging Protocols
- Symbol Servers
  - Using a symbol store
  - Creating and maintaining a symbol store