**Debugger Review**

Learning Outcomes

* Define a debugger
* Explore debugging best practices
* Identify the tools and features of the debugger in Visual Studio
* Practice utilizing the debugger to walk through code
* Identify and fix bugs in pre-existing code

What is a debugger?

The debugger is a specialized developer tool that attaches to the running application and allows you to inspect your code line by line.

Why use a debugger?

The debugger gives us two major benefits. The first benefit is that it enables the developer to isolate bugs so that they can be fixed. The second benefit is that it allows the developer to walk through the flow of the code base.

Here is the scenario, you run your code you are expecting a specific behavior and the application does something totally different. You have recognized that there is a bug in your application and now you need to find exactly where the bug or unexpected behavior is occurring.

So now you ask yourself two questions:

* What did you expect your code to do?
* What happened instead?

Here are a few things to consider before debugging your code:

* Did you check the errors panel?
* Does your code contain any typos?
* Did you expect the object or variable to have a different value then what it gave?
* Do you know the intent of the code?

By asking yourself these questions it may help reduce the time it takes to fix the problem in your code.

Debugger Overview

The debugger inside Visual Studio offers many features and tools. These features allow us to set breakpoints, look at the values of variables and objects within scope, as well as watch any specific value as the code runs. The tool also offers various commands to navigate through your code.

Breakpoints

In order to enable the debugger within Visual Studio you must set a breakpoint. A breakpoint is an indicator for the compiler to stop at a specific line of code which then gives you, as the developer, the ability to step through the code using the navigation commands.

Breakpoints can be set by clicking in the margin to the left of a line of code. Once the breakpoint is set you can click the start button **(f5)** and the compiler will run your code until it reaches the breakpoint. Visually, you should see a yellow line highlighting where the compiler has stopped. From this point you can then begin to use the navigation commands to walk through the code.

Navigation Commands

**Step Over** **(F10)–** this command will allow you to step through the code line by line.

**Step Into** **(F11)–** similar to the step over command this command will allow you to step through the code line by line. The difference here is that once you come to a method, this feature will go and find where the method is defined, and then execute the methods body line by line before continuing the flow of the application.

**Run to Cursor** – this neat feature allows you to set a temporary breakpoint and start the debugger at the same time. This can be enabled by right-clicking the line of code you want the debugger to stop on and navigating to the run to cursor option.

Inspecting variables

There are several features within the debugging tool that allows us to inspect or view the value of a variable. These features assist you in ensuring that the application is running as expected.

Hover

While the debugger has the execution of your code paused, you can then hover over any variable or object to see its value. When it is an object, you can tick the drop-down menu to view other property values. There’s also the option to pin the variable or object so that it is visible without hovering.

Locals window

The locals window can be found with the debugger running, by navigating to View > Debug Windows > Locals. The locals window will show all the variables within the given scope. This becomes useful if you do not want to hover over each of the variables to see values.

Setting a watch

During your debugging session you can use the watch window to keep track of variables and expressions. The watch window is very helpful when you need to evaluate a variable or an expression multiple times.

Code Snippet

static void Main(string[] args)

{

int firstNum = 1;

int secondNum = 2;

int thirdNum = 0;

for (int i = 0; i < 10; i++)

{

firstNum++;

secondNum \*= 2;

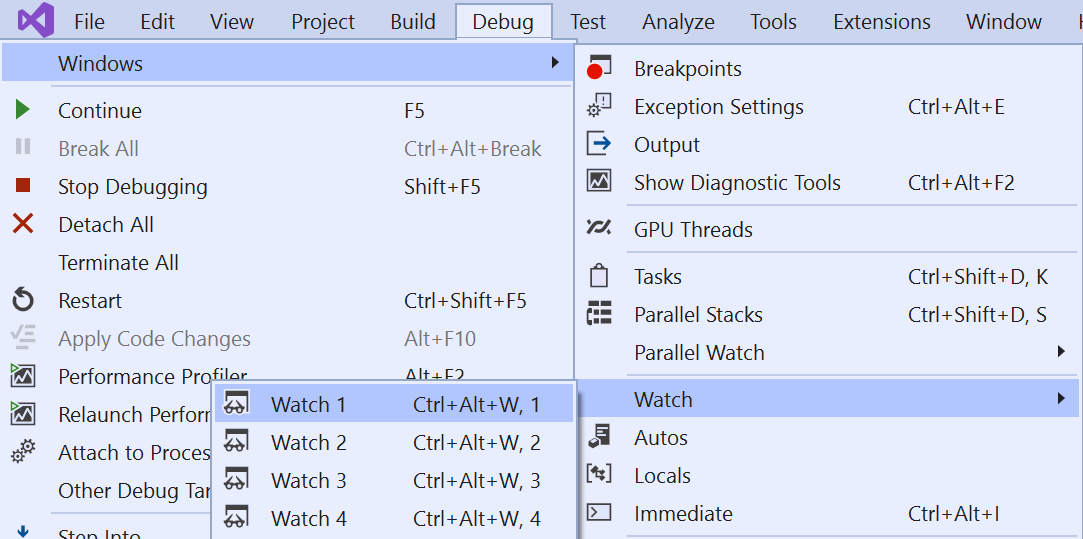
thirdNum = firstNum + secondNum;

}

}

Exercise

* Copy the code snippet into your IDE
* Add a breakpoint at the start of the *for* loop
* Start debugging the application
* Right-click the *firstNum* variable and navigate to the *Add a Watch* option
* Locate the watch Window by going to **Debug > Windows > Watch > Watch1**



* Add a watch for *secondNum* and *thirdNum* using the Watch window
* Step through the code a examine the values of each of the variables changing

Examining the call Stack

One way the debugger helps examine and understand the execution flow of an application is by using the call stack. The call stack shows the order in which methods are getting called. The topmost line will show the current method and then any preceding line will show where that method has been called from.

Conditional Breakpoints?

Conditional breakpoint are a great feature to use when you are iterating through a loop with many iterations. It allows you to set a condition when the breakpoint should be hit as opposed to forcing you to step though the many iterations before you get to your desired point.

Code Snippet

for (int i = 2; i < 50; i++)

{

Console.WriteLine(i\*i);

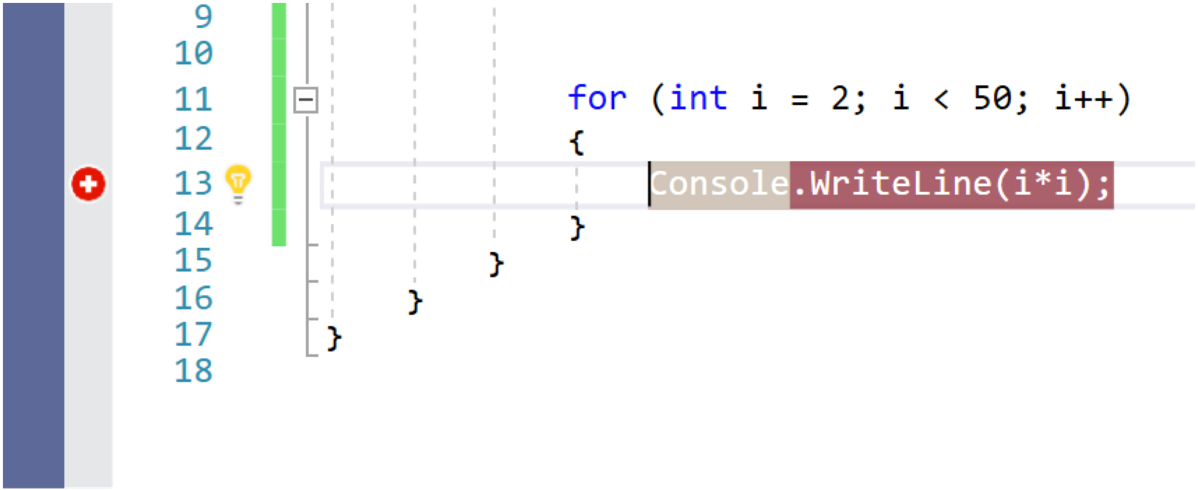
}

Exercise

* Copy the code snippet into your IDE
* Set a breakpoint on the *Console.WriteLine()*
* Hover over the breakpoint and click on the gear (settings) icon

Graphical user interface

Description automatically generated with medium confidence

* Click the *Conditions* option in the Breakpoint Settings
* Use a Conditional Expression to hit the breakpoint **when i is equal to 49**
* Once you have your condition set, close the Breakpoint Settings
* Take notice of the changed icon
* Start the debugger
* Add a watch to the *i* variable
* Notice that the value of *i* is currently at 49. The breakpoint did not halt the program until the condition we set **i==49 was true**.

Graphical user interface, application

Description automatically generated

Review Exercise

Create a new console application and replace the default code with the following:

|  |
| --- |
| class Program     {         static void Main(string[] args)         {             Console.WriteLine("Welcome to the Puppy Depot!");             IterateThroughList();             Console.ReadKey();         }          private static void IterateThroughList()         {             var theTypes = new List<Dog>         {             new Dog() { Name="Spunky", Color = "grey", BreedType=new BType('C')},             new Dog() { Name="Jax", Color = "black", BreedType=new BType('R')},             new Dog() { Name="Puddles", Color = "golden", BreedType=new BType('P')},             new Dog() { Name="Wow wow", Color = "beige", BreedType=new BType('C')},             new Dog() { Name="Gus", Color = "tan", BreedType=new BType('G')},         };              foreach (Dog theType in theTypes)             {                 Console.WriteLine($"{theType.Name} {theType.Color} {theType.BreedType}");             }         }     }      public class Dog     {         public string Name { get; set; }          public string Color { get; set; }         public object BreedType { get; set; }      }      public class BType     {         public BType(char type)         {             switch (type)             {                 case 'G':                     MyBType = Type.German\_Shepherd;                     break;                 case 'R':                     MyBType = Type.Rottweiler;                     break;                 case 'C':                     MyBType = Type.Chihuahua;                     break;                 case 'P':                     MyBType = Type.Poodle;                     break;                 default:                     break;             }         }         public object MyBType { get; set; }         private enum Type { German\_Shepherd, Rottweiler, Chihuahua, Poodle }     } |

* Run the application and review the output
* The objective is to find the bug that is causing the incorrect output of the data to the console.

**Expected Output:**

Welcome to the Puppy Depot!

Spunky grey Chihuahua

Jax black Rottweiler

Puddles golden Poodle

Wow wow beige Chihuahua

Gus tan German\_Shepherd

Questions

What was your process in finding the bug?

What tools within in the debugger did you leverage?

Explain the bug that was causing the incorrect output.