CSI 120 Week 3

LECTURE NOTES

“Any code of your own that you haven’t looked at for six or more months might as well have been written by someone else.”

- *(EAGLESON’S LAW)*

More Whole Number Data Types

sbyte - 8-bit whole number data type

- sbyte can store values from: -128 to 127
- A total of 256 different numbers or 2 to the 8th power

short – 16-bit whole number data type

- short can store values from: -32,768 to 32,767
- A total of 65,536 different numbers or 2 to the 16th power

More Whole Number Data Types

int - 32-bit whole number data type

- int can store values from: -2,147,483,648 to 2,147,483,647
- A total of 4,294,967,296 different numbers or 2 to the 32nd power

long – 64-bit whole number data type

- long can store values from: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
- A total of 18,446,744,073,709,551,616 different values or 2 to the 64th power

More Decimal Number Data Types

float - 32-bit decimal number data type

- float can store roughly 7-8 significant digits. Also known as single.
- 1/3 as a float is 0.3333333

double – 64-bit decimal number data type

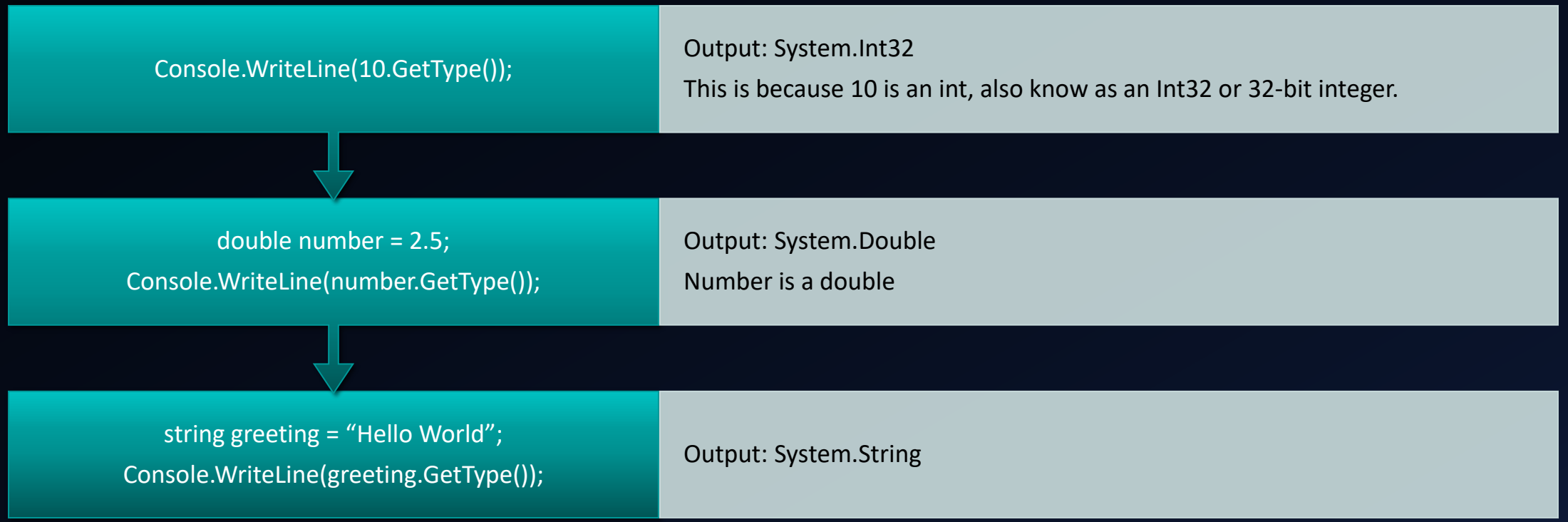
- double can store roughly 16 significant digits
- 1/3 as a double is 0.3333333333333333

decimal – 128-bit decimal number data type

- Decimal can store roughly 27-28 significant digits. Decimal is the preferred datatype for money in C#
- 1/3 as a decimal is 0.3333333333333333333333333333

Checking the data type of a variable

The **GetType**() method will get you the data type of a variable.



String Interpolation

String **interpolation** is an easier way to output a string to the console

This is an alternative to concatenation, Consider this example

- double number = 3.295;
- Console.WriteLine(“The number is ” + number + “ The data type is ” + number.GetType());

The same example using String **Interpolation**

- double number = 3.295;
- Console.WriteLine($“The number is {number} The data type is {number.GetType()}”)

LET’S CODE IT! – MoreDataTypes

The Convert Library

Convert is an alternative to **Parsing** and can be used in its place.

In addition to being able to **convert** a string to a number. **Convert** can also do the reverse operation and convert a number back to a string. This is not possible with **Parsing**.

Getting a string input and converting to a double

- Console.WriteLine(“Please enter a number”);
- string input = Console.ReadLine();
- double number = Convert.ToDouble(input);

LET’S CODE IT! - ConvertLibrary

Try and Catch

Try and **Catch** give us a way of running code that we know might fail or cause an exception.

- The code that may cause the error know as an **exception** is wrapped in a **Try** Block
- The code that will run if the error or **exception** occurs is in the **Catch** Block

A common operation that may have an exception is converting from a string to a number

If the string does not contain a number, the conversion or parsing will cause an exception

If the string does contain a number but the number is too large to fit in the numeric variable this will also cause an exception

Basic Try and Catch Example

- The syntax for a try and catch statement

```
int number;
Console.WriteLine(“Please enter a number”);
try
{
            number = Convert.ToInt32(inputString);
            //Do Some calculation with number and output the result
}
catch
{
            Console.WriteLine($”The parse failed {number} could not be parsed”);
}
```

LET’S CODE IT! – TryCatch

Variable Scope

- When working with try and catch you must be careful to consider **variable scope.**
- A **code block** in C# starts with an opening curly bracket {
- Each opening curly bracket has a corresponding ending bracket }
- The variables declared in within the block are no longer accessible after the closing bracket

```
{//start of code block
    int number = 5;
    number++;
}//end of code block

//number cannot be accessed ouside of the code block
Console.WriteLine(number);
```

CS0103: The name 'number' does not exist in the current context

Show potential fixes (Alt+Enter or Ctrl+.)

LET’S CODE IT! – VariableScope

Increment/Decrement Operators

The Increment and Decrement operators are a commonly used shortcut in C#. They are used to either add or subtract 1 from a numeric variable

int number = 10;

number++; - After this runs the value of number would be 11

++ is the increment operator. number++; is a shortcut for number = number +1;

number--; After this runs the vale of number would be back to 10 again

-- is the decrement operator. number– is a shortcut for number = number -1

Compound Assignment Operators

Remember that the = operator is **assignment**. It assigns a value to a variable.

In addition to the = operator there are also the following assignment operators.

int number = 10; declare the variable number and assign the value 10 to it.

+=

```
number += 5; is a shortcut for number = number + 5; the value of number now would be 15
```

-=

```
number -= 4; is a shortcut for number = number – 4; the value of number now would be 11
```

*=

```
number *= 3; is a shortcut for number = number * 3; the value of number now would be 33
```

/=

```
number /= 3; is a shortcut for number = number / 3; the value of number now would be 11
```

%=

```
number %= 2; is a shortcut for number = number % 2; the value of number now would be 1
```

LET’S CODE IT! – CompoundOperators