
WAUKESHA
COUNTY TECHNICAL
COLLEGE

Hands-on
Higher Ed

# .Net Database Week 1 Curriculum

*Instructor: Mark McArthey*

# Introduction to C#

## Overview of C# and Console Applications

## What is C#?

- C# is a modern, object-oriented, and type-safe programming language developed by Microsoft.
- It is part of the .NET framework, making it versatile for various types of applications.

# Basics of C# Programming

## Variables and Data Types

- **Variables** store data to be referenced and manipulated in programs.
- Common **data types** include `int`, `string`, `bool`, and `double`.

## Value vs Reference Types

- **Value Types**: Stored in the stack. Examples: `int`, `double`, `bool`.
- **Reference Types**: Stored in the heap. Examples: `string`, arrays, class objects.

# Operators

## Arithmetic Operators

- `+`, `-`, `*`, `/`

## Comparison Operators

- `==`, `!=`, `>`, `<`

## Logical Operators

- `&&`, `||`, `!`

## Control Structures

### Decision Making

- `if`, `else if`, `else`
- `switch` statements

### Loops

- `for`: Iterates a set number of times.
- `foreach`: Iterates over items in a collection.
- `while`: Continues as long as a condition is true.
- `do while`: Similar to `while`, but checks condition after the loop.

# Collections in C#

## Arrays

- Fixed size, same data type collection.

## Lists

- Dynamic size, same data type collection.
- Part of `System.Collections.Generic`.

## Namespaces

- **Namespaces** organize large code projects.
- Example: `System`, `System.Collections.Generic`.

```
using System;
using System.Collections.Generic;
```

## Classes and Objects

- Classes are blueprints for creating objects.
- Objects are instances of classes.

## Constructors

- Special methods called when an object is instantiated.
- Used to initialize objects.

```
public class Student
{
    public string Name;
    public int Age;

    // Constructor
    public Student(string name, int age)
    {
        Name = name;
        Age = age;
    }
}
```

## Functions in C#

- Encapsulate reusable code blocks.
- Can return a value or be void.

```
public int Add(int a, int b)
{
    return a + b;
}
```

## Exception Handling

- `try`, `catch`, `finally` blocks handle exceptions.
- Ensures graceful handling of runtime errors.

```
try
{
    // Code that may throw an exception
}
catch (Exception ex)
{
    // Code to handle the exception
}
finally
{
    // Code that runs regardless of an exception
}
```

## File I/O Operations

- Reading and writing files is essential for data management.
- `System.IO` namespace provides tools for file operations.

```
using System.IO;

string text = File.ReadAllText("file.txt");
File.WriteAllText("output.txt", text);
```

# Value vs Reference Types

## Value Types

- **Stored in the Stack**: Value types are stored in a region of memory called the stack. The stack is a simple, last-in-first-out (LIFO) storage structure.
- **Examples**: `int`, `double`, `bool`, `char`, `struct`.
- **Characteristics**:
  - **Direct Storage**: The actual data is stored directly in the variable.
  - **Memory Allocation**: Memory is allocated at compile time.
  - **Scope**: Value types are destroyed when they go out of scope.
  - **Performance**: Accessing value types is generally faster because they are stored in the stack.

```
int a = 10;
int b = a; // b gets a copy of a's value
b = 20;    // Changing b does not affect a
```

heap. The heap is a more complex memory structure used for dynamic memory allocation.

- **Examples:** string, arrays, class objects, delegate.
- **Characteristics:**
  - **Indirect Storage:** The variable stores a reference (or address) to the actual data, which is stored in the heap.
  - **Memory Allocation:** Memory is allocated at runtime.
  - **Scope:** Reference types are managed by the garbage collector, which automatically frees memory when it is no longer needed.
  - **Performance:** Accessing reference types is generally slower because it involves dereferencing a pointer to the heap.

```
class Person
{
    public string Name;
}

Person person1 = new Person();
person1.Name = "Alice";
```

## Stack and Heap: Memory Management in C#

- **The Stack:** Fast access memory area that manages function call and local variables in a very organized way. The stack is self-maintaining, meaning that data is automatically removed when it is no longer needed. As a function call is made, a block of the stack is reserved for it; as the function execution ends, its block is freed in a very straightforward manner.
- **The Heap:** This is a more loosely organized memory area which is generally larger than the stack. Memory allocation and deallocation from the heap are controlled via application instructions or garbage collection in managed languages like C#. The heap is necessary for accommodating dynamic memory allocation, which allows for flexible storage needs but at the cost of slower memory access compared to the stack, and the need for eventual garbage collection to reclaim unused memory.

# Boxing and Unboxing

## Boxing

- **Definition:** Boxing is the process of converting a value type to a reference type by wrapping the value inside an object.
- **Memory Utilization:** When a value type is boxed, it is moved from the stack to the heap.
- **Performance:** Boxing incurs a performance penalty due to the additional memory allocation on the heap.

```
int num = 123;         // Value type stored in the stack
object obj = num;      // Boxing: num is wrapped in an object and moved to the heap
```

# Boxing and Unboxing

## Unboxing

- **Definition:** Unboxing is the process of converting a reference type back to a value type.
- **Memory Utilization:** When a reference type is unboxed, the value is copied from the heap back to the stack.
- **Performance:** Unboxing also incurs a performance penalty due to the additional operation of copying the value.

```
object obj = 123;     // Boxing: 123 is wrapped in an object and moved to the heap
int num = (int)obj;  // Unboxing: obj is unboxed back to a value type and moved to the stack
```

the data into a single unit, or class. This approach restricts direct access to some of the object's components, which can prevent the accidental modification of data. In essence, encapsulation helps protect an object’s internal state from unintended or harmful modifications and promotes modularity in programming.

### Inheritance

- A fundamental mechanism in object-oriented programming that allows a new class to take on the properties and methods of an existing class. This new class, often called a subclass or derived class, inherits attributes and behaviors (methods) from the existing class, which is referred to as the superclass or base class. Inheritance facilitates code reusability and can simplify the creation of complex systems by allowing developers to build upon the existing implementations.

### Polymorphism

- Allows methods to do different things based on the object it is acting upon. This means that a single function name can be used for different types. More technically, it enables objects of different classes to be treated as objects of a

1. **Single Responsibility Principle (SRP)**
    - A class should have only one reason to change, meaning it should have only one job or responsibility.
2. **Open/Closed Principle (OCP)**
    - Software entities (classes, modules, functions, etc.) should be open for extension but closed for modification.
3. **Liskov Substitution Principle (LSP)**
    - Objects of a superclass should be replaceable with objects of a subclass without affecting the correctness of the program.
4. **Interface Segregation Principle (ISP)**
    - No client should be forced to depend on methods it does not use. Interfaces should be specific to the client’s needs.
5. **Dependency Inversion Principle (DIP)**

    - Microsoft Learn for C#
- **Codecademy** - Provides an interactive course where you can write C# code in a browser-based environment.
    - Codecademy C# Course

### Books

- **"C# 9.0 in a Nutshell"** by Joseph Albahari - A comprehensive guide to the C# language and its uses.
- **"The C# Player's Guide"** by RB Whitaker - A beginner-friendly guide to learning C#, great for those starting out.

### Online Courses

- **Udemy** - Various courses ranging from beginner to advanced levels, often with comprehensive projects and downloadable resources.
    - Udemy C# Courses
- **Pluralsight** - Known for its deep-dive courses into C# and .NET technologies.

control structures, and loops. Understanding these will help you build robust applications.

- **Console Applications**: You've learned how to create simple console applications in C#. These serve as a great starting point for beginner programmers to understand the flow of a C# program.
- **File I/O**: Handling file input and output is critical for many real-world applications. Today's introduction should help you manage data effectively in your future projects.
- **Practice and Persistence**: Like any programming language, proficiency in C# comes with practice. Utilize the exercises and resources provided to enhance your understanding and skills.
- **Community and Continual Learning**: Engage with the C# community and continue learning through the resources shared. Staying updated and interacting with other developers will greatly aid your growth.