Meta

Wednesday, June 18, 2025 10:48 AM

GitHub URL: https://github.com/vivekduttamishra/202506-lnw-api

This Notebook

[202506-lnw-api](202506-lnw-api)

What is API?

Wednesday, June 18, 2025 10:55 AM

- It is an application package used to create a generic design
    - Through remote
        - Like REST
    - Or offline access

- It is an interface
    - To get data

- To fetch the data for your application.

- To establish communication between two independent application or entities like libraries or service provider.

- Means to interact with other application.

- Application
- Library
- Framework
- API

Application
- ~~Collection of libraries~~
    - ~~Using the libraries~~.

- Is a complete solution for performing specific task

Library
- ~~Collection of Framework~~
- Set of related functionalities
- A reusable collection of functionalities.
    - Not a complete solution.
    - They are helpers
- Smallest unit of reuse.
- Any function/class you write is a library.
- Typically a user defined code invokes the library.

Framework
- **Structure to develop an application**
- **Contains generic elements of the solution**
- **Specialized using user defined codes.**
- **Framework invokes user defined code**
    - **Reverse of how you use**

- **Framework invokes user defined code**
    - **Reverse of how you use library.**

Quiz

Wednesday, June 18, 2025 11:06 AM

You are driving on ITPL road on a busy morning and someone asks

What are you doing?

1. I am burning the fuel
    - When we press accclerator it creates cumbtion

2. I am driving. Following the traffic rules and maps.
    - Learning to drive

3. I am going to the office for work

4. I am going to office for work.
    - Work from Home.

4. You didn't stop, just moved on.

What is a program?

- A set of instructions to perform a task.

OO Fundamentals

Wednesday, June 18, 2025 11:35 AM

Abstraction?

- Hiding Data?
- Conceptual representation of an idea.
- Showing only the needful things.
- Giving the output without showing the process.

```
interface Shape{
        double
        Perimeter();
}


class Triangle :
Shape{
  //data is hidden

  public double
Perimeter(){
        //logic is also
        hidden.
 }

}
```

Encapsulation?

It is meaningless unless we know why we are binding

- Binding data and methods together.
- Applies cohession
  - Incidental/illogical
  - Temporal
    - Things work at same time
    - May still be unrealated
  - Hierarchial
    - Of same class hirarchy
    - Often mutually exclusive
  - Functional
    - The real Why

- Forms a unit of Responsibility
  - Only a responsible design can be effectively reusable.
  - It is a unit of reuse

Assignment

Wednesday, June 18, 2025 12:26 PM

- What is the relationship between a Car and a Taxi.
  - Create necessary classes and programming element (class, function, interface) to depict the relationship

```
interface Vehicle
{
        void Move();
}

class Car : Vehicle
{
      public void Move(){…}
}
```

InheritianceW

- Represents "Is A"
- Reusability?

Solution A

```
class Taxi : Car
{

}


Car hondaCity= new Car();

Taxi taxi = new Taxi();
```

Problem:

- No company manufactures Taxi
- Taxi may not always be a Car. It may be
  - Bike
  - Halicoptor
- When I talk about Car, which car?

Solution B

```
class Veichle{
        public string Type; //private or taxi
}

class Car : Veichle{


}
class Bike: Veichle{

}


var car1 = new Car("private");
var  taxi1 = new Car("taxi");

var bike=new Bike("private");
var taxi2 = new Bike("taxi");
```

Advantage

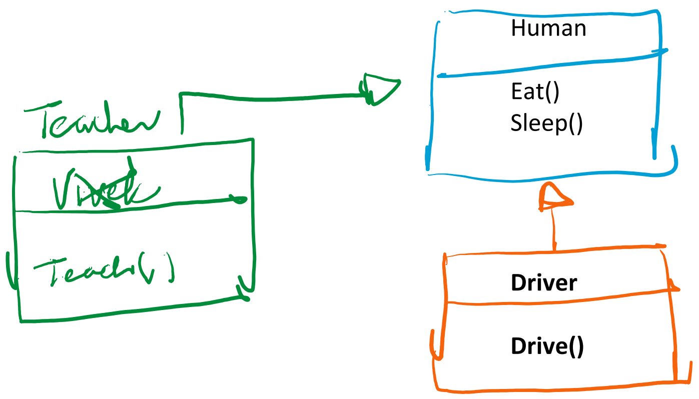
- Has A model
- More Realistic.

Disadvantage

- Enum/string merges multiple codes in the same functionality
- Nested if-else

Use Case

Wednesday, June 18, 2025 1:00 PM



```
class Teacher  :  Human{

        public void Teach(){…}
}

class Driver : Human{
        public void Drive(){…}
}
```

```
var vivek= new Teacher();
var vivek = new Vivek();

vivek.Eat();
vivek.Sleep();
```

vivek.Teach();

vivek.Drive();

```
var prabhat= new Human()
prabhat.Eat();
prabhat.Sleep();
```

var vivek = new Teacher() or new Driver() ???

Parent-Child Relationship

Wednesday, June 18, 2025 1:16 PM

```
class Son : Father{

}
```

Should be is a type of

~~var santa = new Father();~~

~~var banta = new Son();~~

How will banta inherit Santa's bank balance?

```
var  santa = new Person();

var banta = new Person();

bank.Transfer(santa.BankAccount, banta.BankAccount, amount, password);
banta.Dna.Inherit(santa.Dna)
```

Inheritance is NOT Parent-Child Relationship.

- Realworld inheritance is between two Objects
  - These objects are often of the same type.
  - It is more give and take.

- In OO, inheritance is between two classes
  - They do not represent parent-child
  - They represent is-a-type-of

Inheritance And Reuse

Wednesday, June 18, 2025 2:32 PM

```
class Crow{

        public Egg LayEgg(){return new CrowEgg(); }
        public void Fly(){…}
        public Color Color{get{return Color.black; } }
}

class  Parrot : Crow
{
   public override Color Color { get{return Color.green;}}

}


[Test]
public void CrowsAreBlack()
{
    Crow crow = new Parrot();  //parrot is a type of Crow

    Assert.Equal( Color.black, crow.Color); //crow is green

}

[Test]
public void ParrotBabiesAreParrot()
{
    Parrot parrot=new Parrot();
    object baby = parrot.LayEgg().Hatch();

    Assert.That( baby is Parrot );

}
```

```
class CrowEgg
{
    object Hatch(){
        return Crow();
    }
}
```

Solution

```
public override Egg LayEgg(){
    return new ParrotEgg();
}
```

- This works.
- But what is point of inheritance if we need to override
  - We lost reuse?

- We have inherited LayEgg but not overridden it
  - It will return a Crow's egg (implementation)
    - Will hatch to a Crow Object

Employee

Programmer ProjectManager Accountant

```
var  vivek = new Employee();
```

Inheritance Summary

Thursday, June 19, 2025 10:51 AM

- Inheritance is a class-to-class relationship
    - Class-to-class relationship is static and at design time
    - It doesn't model parent-child relationship
- It shouldn't be used for reusability
    - Static and non-scalable design.

- It shouldn't model a parent-child relationship
    - .In the real world inheritance is between two objects
        - In programming it is between to classes
            - Class and objects are not same.
- INHERIT ONLY IF THERE EXISTS A RELATIONSHIP
    - IS A TYPE OF
    - AN HIERARCHY
- DO NOT USE INHERITANCE FOR
    - REUSE
    - FOR RELATIONSHIPS LIKE
        - PARENT-CHILD RELATIONSHIP
        - HAS A
        - IS LIKE A
        - IS SIMILAR TO
        - WORKS TOGETHER

Law of OO Design

Wednesday, June 18, 2025 2:58 PM

Prefer "Has A" (composition) over "Is A" (inheritance)

- "Is a" static relationship; "Has a" is a dynamic relationship
    - You can change whatever you have
        - You can choose not to have
            - If I have a cellphone
                - I can change it
                - I can decide not to have it
    - You can't change who you are "Is A"
        - You can't change the fact that you are a human
- "Is a" is non-scalable; "Has A" is scalable
    - If I have a cellphone, I can have two or three
    - I can't be two times human being.

Try to change "Is A" relationship in your design to "Has A" relationship.

- "vivek" is an Employee
    - Attempt #1
        - vivek "has an" Employee
            - It is not same.
            - Meaning changes.
    - Attemp #2
        - Vivek "has an" Employement
            - You may need to change the design.
        - Advantage?
            - Vivek can choose to
                - Change the job: vivek.Employement = new SelfEmployment();
                - Leave the job: vivek.Employement=null;

- If vivek is an Employee
    - He can be a one time employee
- If Vivek has an employement
    - He can have multiple Employements

```
Class ________{

    //Employement Employeent;

  List<Employement> Employements;
}
```

Reuse Using Inheritance

- We can reuse using inheritance
  - It works!
    - But slum houses also work.

- What is the problem?
  - Bad Relationship
    - Game is not a List
  - What if I need more than one list
    - List of Scores
    - List of Players
    - List of Moves

```
Class Game : List<Score>{



	Score FindScore( int gameId){

		return (score from this
			where score.gameId==gameId
			select score).FirstOrDefault();
	}
}

var game = new Game();

game.Add( new Score(…));
```

Has A for Resuse

- Encapsulate to reuse.
- Think Reuse, think encapsulation
- Dynamic, Scalable

```
Class Game {
	List<Score> scores;
	List<Player> players;
	List<Move> moves;

	void AddScore(Score score){
		scores.Add(score);
	}

	Score FindScore( int gameId){

		return (score from scores
			where score.gameId==gameId
			select score).FirstOrDefault();
	}
}

var game = new Game();

game.Add( new Score(…));
```

DESIGN PRINCIPLES

Wednesday, June 18, 2025 3:27 PM

- They are fundamental best practices in a software design.

1. Open-Close Principle

- Ultimate goal of a software design!
- Your design should be
  - Open
    - Extension
      - New Feature (Additional new feature)
      - Modify Existing Feature
      - Delete Existing Feature
    - Why?
      - Requirement changes over a period of time
      - Our design should be ready to accommodate future changes
        - Future Proof!
  - Close
    - Modification
      - At the source code level!
      - DON'T MEND IT IF NOT BROKEN
    - Why?
      - Change triggers a cycle of
        - Test
        - Deploy
        - Distribute
      - Change may induce new Bugs
      - A change may not be acceptable to all stakeholders.
        - It may be a matter of choice.
- How?
  - All changes should be additive.
    - To add new feature write new code
    - To modify existing feature write new code
    - To delete existing feature write new code

  - 100% OCP is not feasible
    - Often not desirable.
    - Idea is to reduce the surface area of change
      - Changes should be minimal
      - Shouldn't have ripple effects.

2. Single Responsibility Principle

- You code (component, function, object) should have a single responsibility
  - One Reason to Exist
  - One Reason to change.
    - Theoretically closed for all except one reason!
      - Practially most of the time such codes are completely closed. (follows 100% OCP)

- One Responsible <> One Function per class.
  - It certainly means few related functions
    - A Printer
      - should have
        - Print()
        - Eject()
        - Cancel()
      - Shouldn't have
        - Scan()
          - Printers! Don't scan!
    - A Car
      - should have
        - Start
        - Turn
        - Move
        - Stop
      - Shouldn't have
        - Drive
          - Cars don't drive themselves.

How do we achieve SRP?

1. Meaningful Names

- Your class, methods, objects should have meaningful name
- Name=>responsibility
  - If name is not clear we can't enforce SRP?
- Can
  - Bird Fly?
    - Yes
  - Fish Fly?
    - No
  - Foo do Bar?
    - We don't know what is foo and bar
    - Can't know if it is a right model

2. Avoid composite names joined by And/Or

- IncomeAndSalesTaxCalculator
  - Responsible for
    - Income Tax
    - Sales Tax
- InsertOrUpdate
  - Responsible for
    - Insert
    - Update
- CreateAndAdd

        - Creates and adds
        - Can't create for future additional
        - Can't add existing object

3. Avoid Abstract name for a concrete class

- TaxCalculator
    - Still calculating Income and Sales Tax
- Save
    - Still using insert / delete

4. Most methods should use most field most of the time

- Shouldn't have mutually exclusive/option fields
- Shouldn't have fields/parameters are always null in a given scenario.

3. DRY (Don't Repeat Yourself)

- Redundant code MUST be avoided
- It suggests same responsibility (task) is listed at multiple places
    - No clear responsibility definition.
    - When that code need to change, it must be changed at multiple places.
- More problematic in case of partial redundancy.

Solution: Two Step Solution

1. Encapsulate Whatever Repeats
    - Create class and function to encapsulate redundant code (steps)
        - It may have some gaps where code is specific to the scenario.
            - Parameterize if change is just data
2. Abstract whatever Changes
    - Implementation would be a specific use case
    - Parameterize the repeating code with an abstract element

Assignment #2

Wednesday, June 18, 2025 4:21 PM

- Create a simple Calculator

Use Case:

Simple Test for Calcualtor

```
var calculator = new Calculator();

calculator.Calculate( 20, "plus", 30);
calculator.Calculate(20, "minus", 4);
calculator.Calculator(20, "foo", 5);
```

- You shouldn't change the method signature

```
//expected output

20 plus 30 = 50
20 minus 4 = 16
Invalid Operator: foo
```

- Create a Application to take user input and provide result
- Should work in two modes

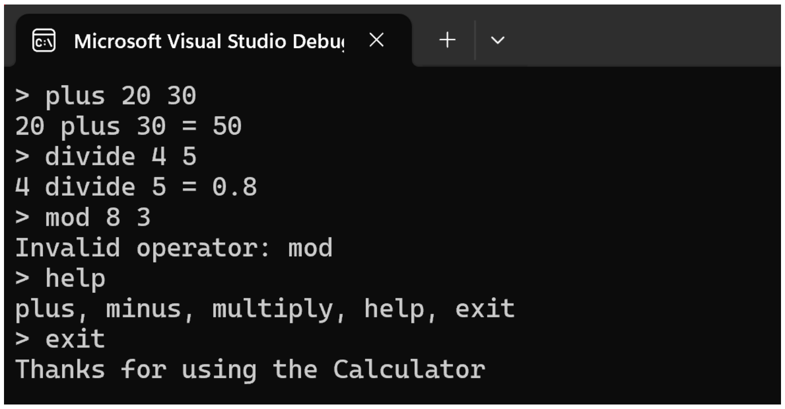
Command Line Argument Driven

```
c:\project>calculator plus 20 30
20 plus 30 = 50
```

Shell Input (User Input) Driven

```
c:\project> calculator
> help
Plus, minus, multiply, divide, exit
> plus 20 30
20 plus 30 = 50
>minus 4 4
4 minus 4 = 0
>exit

c:\>
```

Expected Result



Microsoft Visual Studio Debu

```
> plus 20 30
20 plus 30 = 50
> divide 4 5
4 divide 5 = 0.8
> mod 8 3
Invalid operator: mod
> help
plus, minus, multiply, help, exit
> exit
Thanks for using the Calculator
```

Version #1

```
0 references
public class CalculatorV1
{
    0 references
    public void Calculate(double number1, string oper, double number2)
    {
        switch (oper.ToLower())
        {
            case "plus":
                Console.WriteLine($"{number1} {oper} {number2} = {number1+number2}");
                break;
            case "minus":
                Console.WriteLine($"{number1} {oper} {number2} = {number1 - number2}");
                break;
            case "multiply":
                Console.WriteLine($"{number1} {oper} {number2} = {number1 * number2}");
                break;
            default:
                Console.WriteLine($"Invalid Operator: {oper}");
                break;
        }
    }
}
```

Redundant Code

- What if tomorrow we need a different formatting for the result?
- We need to change at n Places.

Solution: DRY

- Check out in [Design Principles]

DRY Resolved

Version #2

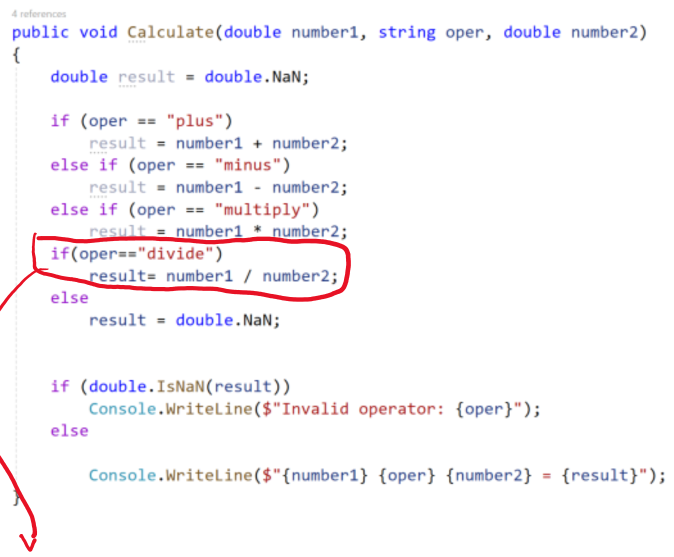
```
double result = double.NaN;
switch (oper.ToLower())
{
    case "plus":
        result = number1 + number2;
        break;
    case "minus":
        result = number1 - number2;
        break;
    case "multiply":
        result = number1 * number2;
        break;
    default:
        break;
}

if (result == double.NaN)
    Console.WriteLine($"Invalid operator: {oper}");
else

    Console.WriteLine($"{number1} {oper} {number2} = {result}");
```

But it still violates SRP

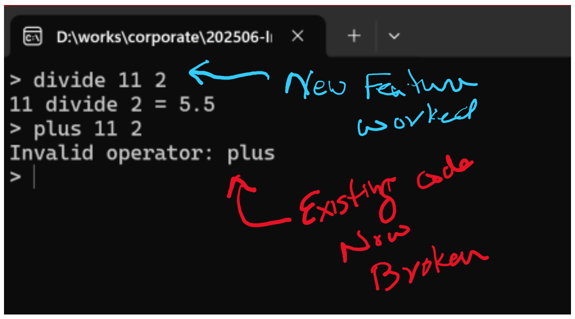
- Typically, all switch-case and if-else ladder violates SRP (and in turn OCP)
- Avoid them

Why OCP violation is BAD?

- Check out Design Principles nodes for more details
- In this case a new code may introduce breaking changes



```
4 references
public void Calculate(double number1, string oper, double number2)
{
    double result = double.NaN;

    if (oper == "plus")
        result = number1 + number2;
    else if (oper == "minus")
        result = number1 - number2;
    else if (oper == "multiply")
        result = number1 * number2;
    if(oper=="divide")
        result= number1 / number2;
    else
        result = double.NaN;


    if (double.IsNaN(result))
        Console.WriteLine($"Invalid operator: {oper}");
    else

        Console.WriteLine($"{number1} {oper} {number2} = {result}");
}
```



```
> divide 11 2
11 divide 2 = 5.5
> plus 11 2
Invalid operator: plus
>
```

Switch Case

Wednesday, June 18, 2025 5:55 PM

```
void  DoJob ( Context context) {

    switch(context){

        case context.A:
            DoJobA();
            break;
        case context.B:
            DoJobB();
            break;
        default:
            DoDefaultJob();
            break;

    }

}
```

- Three different responsibilities for three different context
- They are merged together
- Violates SRP
    - Mutually exclusive code blocks

- There is a Job (doJob)
    - That has multiple implementation (Forms)
        - JobA
        - JobB
        - JobC
    - Form varies based on context

- Explanation 2
    - One Name (DoJob)
    - Many Forms (DoJobA, DoJobB, DoJobC)
    - Varies on context

Is this polymorphism?

- NO
- It is avoiding polymorphism

Switch case to Polymorphism

```
void  DoJob ( Context context) {

    switch(context){

        case context.A:
            new JobA().DoJob();
            break;
        case context.B:
            new JobB().DoJob();
            break;
        default:
            new DefaultJob().DoJob();
            break;
```

- Interface IJob
    - Implemented by
        - JobA
        - JobB
        - DefaultJob

```
        }

}
```

POLYMORPHISM REPLACES CASE BODY NOT case selection

- Polymorphism is not a complete replacement switch case
- It replaces what we do in different cases
- It can't decide which case to execute
    - It doesn't include switching (selection) logic

What can really replace switch or nested if-else

- There are two patterns to use
    - Chain Of Responsibility
        - Use for-if loop
    - Use a Dictionary to replace switch case
        - Key —> case

Polymorphism?

Thursday, June 19, 2025 11:11 AM

What is polymorphism

- One name many forms

How to achieve Polymorphism?

- Overloading
- Overriding

```
class  ParkerPen
{

    public  void Use( Hand hand){

        Console.WriteLine("writing")
    }

    public void Use(Pocket pocket){

        Console.WriteLine("status")
    }

}

public void TestPen(){

    var p= new ParkerPen();
    var context= GetHandOrPocket(); //assume it returns pocket
    if( context is Hand)
        p.Use(context as Hand)
    else if(context is Pocket)
        p.Use(context as Pocket);
}
```

Creating API

Thursday, June 19, 2025 3:14 PM

API Design

Two essential Elements

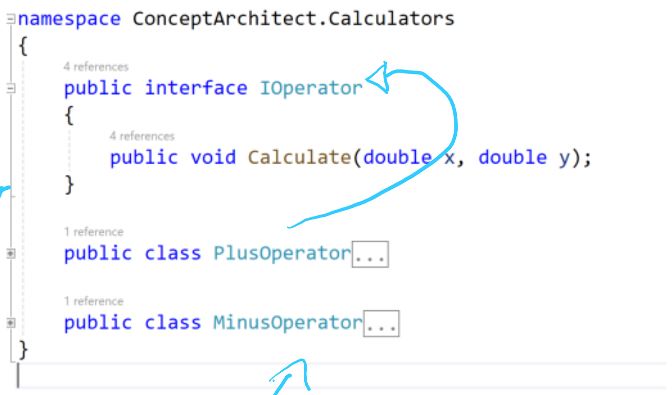
1. Framework

- Includes common/generic functionalities that defines the core of the system.
- Exposes public interfaces (or contracts) to create new functionality that can work with the framework
    - This enables OCP
- Should have ability to **plugin** new capabilities dynamically without modifying the framework.

2. API Implementation (Extension/Plugin)

- Implements the API (interface) for the enhanced capabilities.
- Can be done internally (same organisation) or at the community level
    - Community contribution is NOT core or MUST for API
        - It may be for internal use cases only

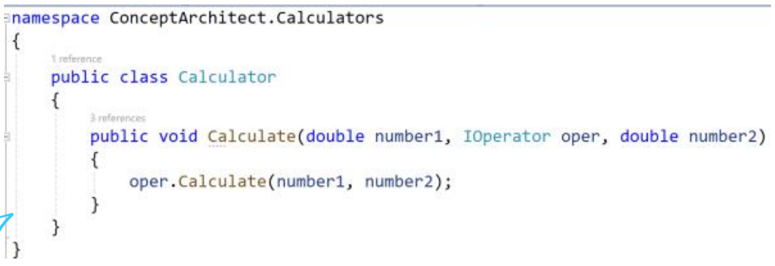
- The functionalities are expected to work under the control of framework and not stand alone
- We don't call framework
    - The framework calls my logic
        - This is also known as Inversion of Control



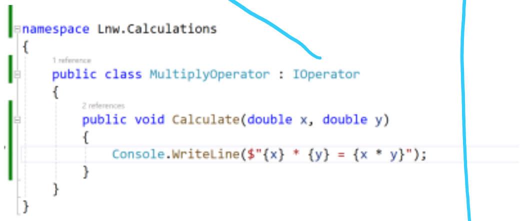
```
namespace ConceptArchitect.Calculators
{
    4 references
    public interface IOperator
    {
        4 references
        public void Calculate(double x, double y);
    }

    1 reference
    public class PlusOperator...

    1 reference
    public class MinusOperator...
}
```

Extended by 3rd party plugin

Built-in operators

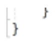


```
namespace ConceptArchitect.Calculators
{
    1 reference
    public class Calculator
    {
        3 references
        public void Calculate(double number1, IOperator oper, double number2)
        {
            oper.Calculate(number1, number2);
        }
    }
}
```

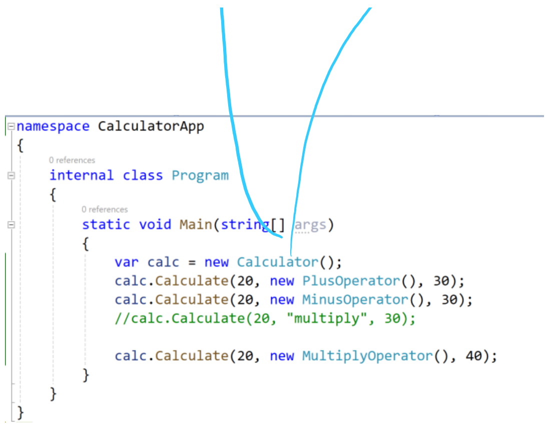


```
namespace Lnw.Calculations
{
    1 reference
    public class MultiplyOperator : IOperator
    {
        2 references
        public void Calculate(double x, double y)
        {
            Console.WriteLine($"{x} * {y} = {x * y}");
        }
    }
}
```

Use calculator(framework)



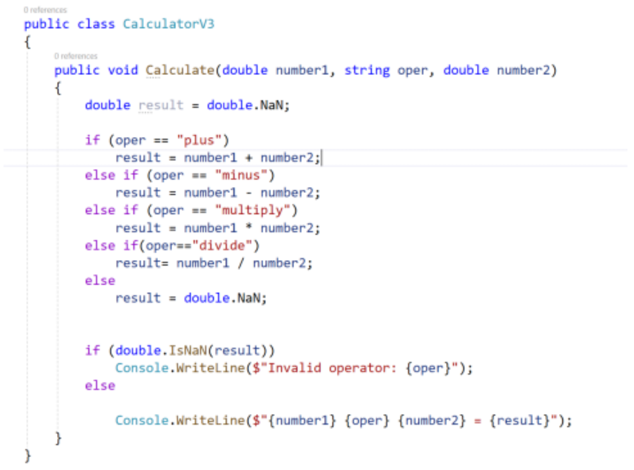
```
    }
}
```



```
namespace CalculatorApp
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            var calc = new Calculator();
            calc.Calculate(20, new PlusOperator(), 30);
            calc.Calculate(20, new MinusOperator(), 30);
            //calc.Calculate(20, "multiply", 30);

            calc.Calculate(20, new MultiplyOperator(), 40);
        }
    }
}
```

Key design elements

1. Calculator exposes an interface (Dependency Inversion Principle)
    - It doesn't know the exact operation.
2. Each operation is independent and Singly Responsible
    - They are not directly related or connected
    - This allows us to add new features without changing existing design.
3. This is different from original design



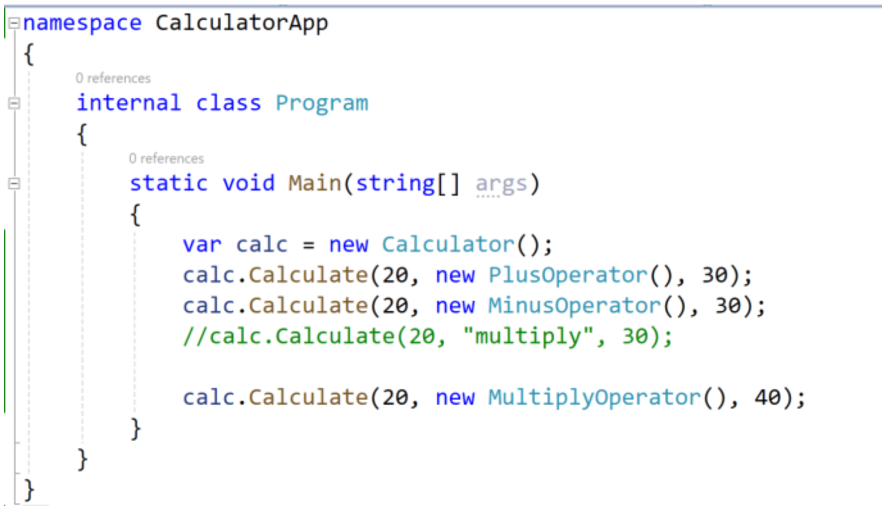
```
0 references
public class CalculatorV3
{
    0 references
    public void Calculate(double number1, string oper, double number2)
    {
        double result = double.NaN;

        if (oper == "plus")
            result = number1 + number2;
        else if (oper == "minus")
            result = number1 - number2;
        else if (oper == "multiply")
            result = number1 * number2;
        else if(oper=="divide")
            result= number1 / number2;
        else
            result = double.NaN;


        if (double.IsNaN(result))
            Console.WriteLine($"Invalid operator: {oper}");
        else

            Console.WriteLine($"{number1} {oper} {number2} = {result}");
    }
}
```

User Input

Thursday, June 19, 2025 3:20 PM

How do I use the calculator with Console Application (command line / Shell)

- When we using Console.ReadLine() we read a String
  - String can't be converted to a new Object creation easily

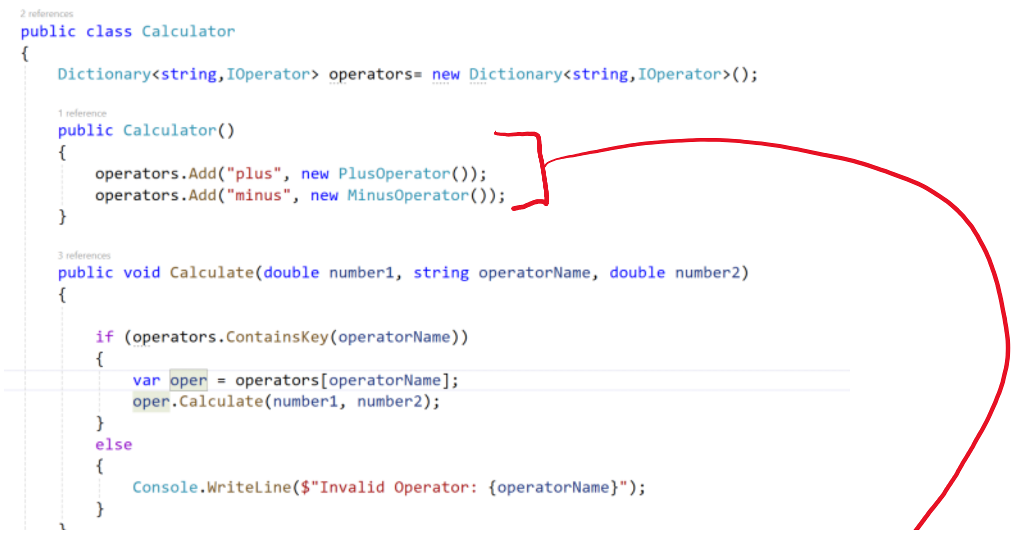


```
namespace CalculatorApp
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            var calc = new Calculator();
            calc.Calculate(20, new PlusOperator(), 30);
            calc.Calculate(20, new MinusOperator(), 30);
            //calc.Calculate(20, "multiply", 30);

            calc.Calculate(20, new MultiplyOperator(), 40);
        }
    }
}
```

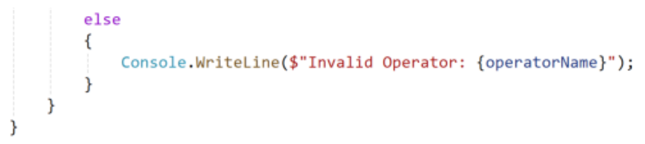
How do we input the operation (string)

Dictionary of Operations

- It's a generic pattern to replace switch case in your code.



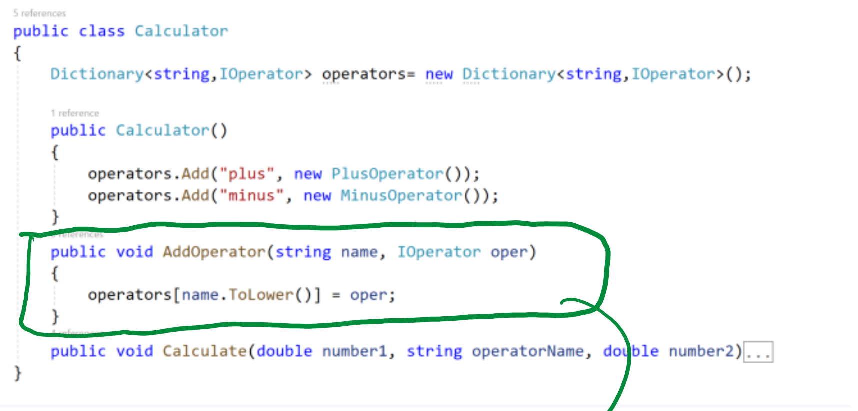
```
2 references
public class Calculator
{
    Dictionary<string,IOperator> operators= new Dictionary<string,IOperator>();

    1 reference
    public Calculator()
    {
        operators.Add("plus", new PlusOperator());
        operators.Add("minus", new MinusOperator());
    }

    3 references
    public void Calculate(double number1, string operatorName, double number2)
    {

        if (operators.ContainsKey(operatorName))
        {
            var oper = operators[operatorName];
            oper.Calculate(number1, number2);
        }
        else
        {
            Console.WriteLine($"Invalid Operator: {operatorName}");
        }
```



```
            else
            {
                Console.WriteLine($"Invalid Operator: {operatorName}");
            }
        }
    }
```

- We have 2 hard coded operators
- Framework doesn't give provision to expand and introduce new plugins
    - No Dependency Injection option provided.

Solution



```
5 references
public class Calculator
{
    Dictionary<string,IOperator> operators= new Dictionary<string,IOperator>();

    1 reference
    public Calculator()
    {
        operators.Add("plus", new PlusOperator());
        operators.Add("minus", new MinusOperator());
    }
    public void AddOperator(string name, IOperator oper)
    {
        operators[name.ToLower()] = oper;
    }
    public void Calculate(double number1, string operatorName, double number2)...
}
```

Assignment 2.1

Thursday, June 19, 2025 4:06 PM

- Take the code for GIT Hub
- Introduce the formatting and display option
- Make sure I should be able to use
  - different styles of formatting
  - Different architecture
    - Console
    - GUI
  - Display Error and Results differently

Phase 2

- Currently for every operator we need to
  - Create a class
  - Implement interface
  - Write the logic
- But the interface has a single method
- Can we replace this interface with a delegate.
- The Calculator should continue to work with current operators
  - Calculator should work with operators that are either interface or delegate

Assignment 2.2

Thursday, June 19, 2025 5:38 PM

Create a CLI Framework

It should support

1. Command Line Argument Mode

```
d:\>  cli book-list
d:\>  cli find-book author=vivek
d:\>  cli export-books books.json
```

2. Shell Mode (Interactive mode

```
d:\> cli
> help
book-list  find-book export-books add-book remove-book
> help export-books
Exports books in a given format based on file extension
export-book  books.json

> remove-book rashmirathi
> exit
```

The Exact commands should be added as a plugin model

- CLI should be able to run different commands that are added to the CLI
- Each command will have
  - One or more names (like alias)
    - book-list get-all-books books
  - Help Text to explain what this command does
  - The command may take as many parameter as it needs
  - If command returns something it should be displayed
  - Command may be async or synchronous

```
void Main(string []args){

    Cli app = new Cli();

    app.AddCommand(new BookListCommand())
    app.AddCommand(new BookDeleteCommand());
    …

    app.Run(args);
}
```

Assignment 2.3

Thursday, June 19, 2025 5:50 PM

Create a switch-case API that can be used in place of traditional switch case statement.

- It should eliminate the basic problems of switch case like
  - OCP
- Should by dynamic.

Animal Hierarchy

Friday, June 20, 2025 10:44 AM

```
public abstract class  Animal
{
    public abstract void Eat();
    …
}

class Horse : Animal
{
    public override void Eat(){
         Console.WriteLine("Grass");
    }
}

class Tiger : Animal
{
    public void Hunt(){
        Console.WriteLine("Hunts its
        prey");
    }

    public override void Eat(){
        Hunt();
        Console.WriteLine("Eats Flesh");
    }

}
```

```
Horse h =new Horse();
h.Eat();  //Grass

Animal a = h;
a.Eat();

Tiger t = new Tiger();
t.Hunt();  //Hunts it prey

t.Eat();  // Hunt its prey. Eats Flesh

Animal a2 = t;

a2.Hunt(); //compile time error. Animal doesn't have hunt
```

Why is it printing Grass?

- "a" is Animal Reference (not Horse reference)
- Animal class Eat doesn't print Grass
- Why is this code printing grass?
  - Because it calls the method from "object" class and not from "reference" class
    - This is polymorphism

Why should this line give error

- "a" refers to a tiger object
- We already know methods are invoked from object class and not from reference class
- Tiger object has hunt.
- So why should this line give error
- Here Hunt is (indirectly) called using Animal reference
  - Technically it is possible to call hunt using Animal reference

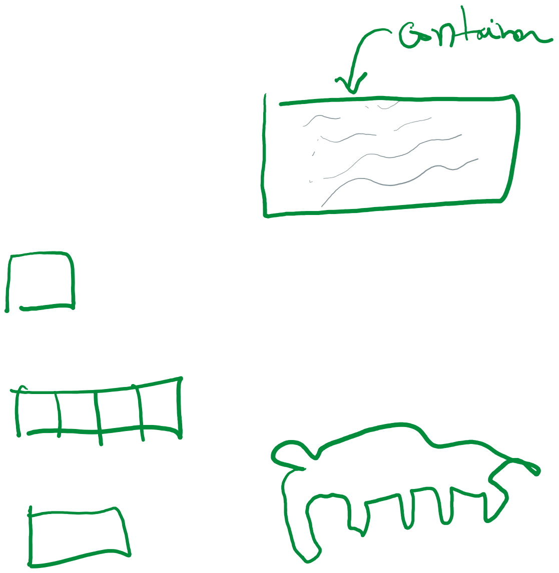
```
a2.Eat(); //Hunts its prey. Eats Flesh
```

Delegate?

Friday, June 20, 2025 11:14 AM

- Delegates are Objects
- The are containers for a method
- Once they wrap a method they can be used just like mehthod.
- But now you can also
  - Refer (Store) this object using a variable
  - Can add this object to an array/list/dictionary

- For each different method signature there is a differnent delegate type



How Function and Delegate Signature Match

- To refer to a function like one below

```
void Welcome(string name){

}
```

- We create a delegate type

```
delegate void Greeter(string name)
```

Important Note

- Here Greeter is not a function reference or pointer or variable.
- Internally Greeter is a **class**
- This is a shortcut for autogenerating a class like one below

```
public  class Greeter : MulticastDelegate{

     public void Invoke( string name ){
         //some logic that will call your function here
     }
     //more code here
}
```

To Refer to the function we need a Object of type Greeter

Greeter g1= new Greeter(Welcome);

Now we can use "g1" that contains Welcome function as Welcome

```
g1.Invoke("LNW");  //internally calls Welcome("LNW");
```

Greeter object doesn't work only with Welcome

- It can work with any function that matches the same signature like

```
void GoodBye(string name){}
void Wish(string xyz) { }
void PerformComplexOperation(string operationName) {}
```

- But it can't work with functions that don't match signature

```
void Greet( string name1, string name2) {…}  //two parameter
string GetGreetingMessage(string name) {} //non void return type
```

Simplified Delegate Signature

Auto Boxing of a method to Delegate

Greeter g2 = GoodBye ; //same as **new Greeter(GoodBye)**

Implicit Invoke call

- **You can invoke a delegate as if it is a function**

```
g2.Invoke("Lnw") ;   // same as g2.Invoke("lnw") =>   GoodBye("Lnw")
```

What does Delegate Signature include (and not include)

```
delegate void Greeter( string name );
```

What does this signature include?

- The function returns void
- The function takes a string argument
- Extra Information
  - The name of this argument may be "name"

        - Not compulsary

What does this signature doesn't include?

- Implementation details (body)
    - Function can do anything under given constraint of parameter type and return type.
- Method Name
    - The function can have any name
        - Welcome
        - GoodBye
        - DeleteUserById
- Class Name
    - This function can belong to any class
- Scope
    - This function can any scope
        - Public
        - Private
        - Static
        - Non-static

Untitled

Friday, June 20, 2025 12:21 PM

What does this Code Mean?

```
interface IBinaryOperator{

    double Calculate( double x, double y)
}
```

- It will represent an Object
  - That can be of any class
    - Name doesn't matter
      - PlusOperator
      - MinusOperator
      - ...
  - But it must
    - Implement IBinaryOperator
    - Must have a function
      - called Calculate
        - You can't change this name
      - Should be
        - public
        - Non-static
      - Concern
        - There can only one function matching this signature in a class
        - For each different implementation we need a different class

```
delegate double BinaryOperator(double x, double)
```

- `It represents an Object`
  - `That contains a function`
    - `That MUST match the same signature`

- What doesn't matter
  - Class Name
    - PlusOperator
    - MinusOperator
  - Function Name
    - It can be any name need not be Calculate
      - Plus
      - Minus
      - Multiply
      - ...
  - Scope of the funciton
    - Private
    - Public
    - Static
    - Non-static
- Advantage
  - Multiple matching functions can be present in same class
  - No interface implementation required.
    - You can use existing codes if signature matches.
      - Math.Power

Built-in Generic Delegate Types

Friday, June 20, 2025 12:54 PM

- C# provides some built-in generic delgate types that we can use instead of creating our own
- There are two families of such delegates

Action Delegates

- They are delegates that take 0 or more parameters and returns nothing

What we need	User defined Example	Action version

A function that takes nothing and returns nothing	delegate void Job();
	Job j= DoSomething;
		delegate void Action()
		Action a = DoSomething;

A void function that takes an int parameter	delegate void OneIntArg(int x); OneIntArg x = PrintTable	Action<T> Action<int> x = PrintTable
A void function that takes and int and string	delegate void TwoArg(int x, string y); TwoArg x = SomeFunction	Action<A,B> Action<int,string> x = SomeFunction

- There are 18 overloaded types that can take from 0 to 17 parameter

Func Delegate

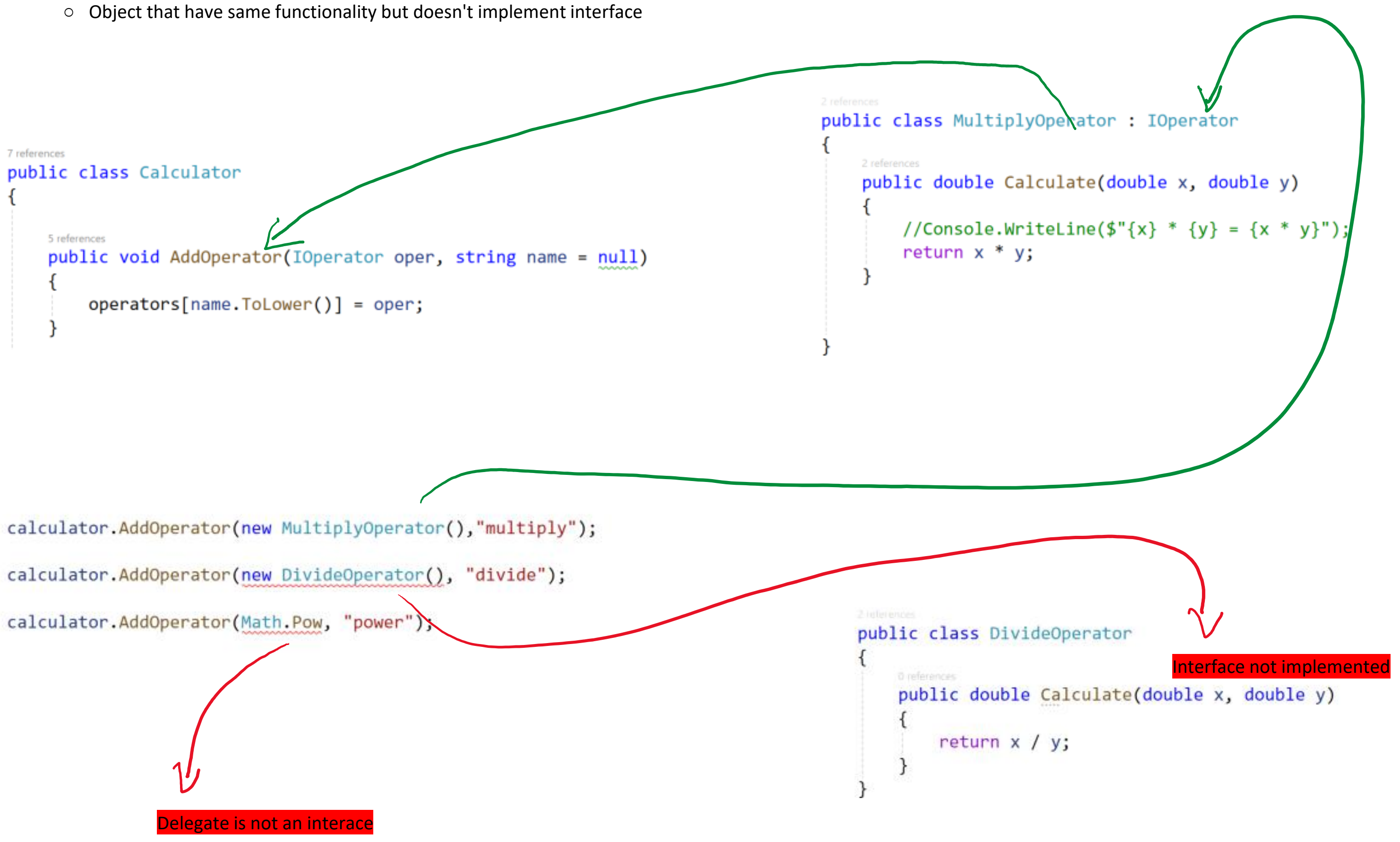
- They are functions that can take 0 or parameter and return a value.
- Note return type is always mentioned at the end
- There are again 18 overloads.

What we need	User Define Example	Func version
A function that takes nothing and returns an int	delegate int GetValue() GetValue v= GetRandomValue	Func<int> v = GetRandomValue;
A function that takes an int and returns a bool	delegate bool IntChecker() IntChecker c = IsPrime	Func<int,bool> v = IsPrime
A function that takes two int and returns a string	delegate string Del(int a, int b) Del d = CallSomeFunc	Func<int,int,string> v =CallSumFun

Calculator supporting both inerface and delegate operators

Friday, June 20, 2025 1:10 PM

- Currently calculator takes only interface
- It can' work with
  - Delegate
  - Object that have same functionality but doesn't implement interface



```
7 references
public class Calculator
{
    5 references
    public void AddOperator(IOperator oper, string name = null)
    {
        operators[name.ToLower()] = oper;
    }
```

```
2 references
public class MultiplyOperator : IOperator
{
    2 references
    public double Calculate(double x, double y)
    {
        //Console.WriteLine($"{x} * {y} = {x * y}");
        return x * y;
    }

}
```

```
calculator.AddOperator(new MultiplyOperator(),"multiply");

calculator.AddOperator(new DivideOperator(), "divide");

calculator.AddOperator(Math.Pow, "power");
```

```
2 references
public class DivideOperator
{
    0 references
    public double Calculate(double x, double y)
    {
        return x / y;
    }
}
```

Interface not implemented

Delegate is not an interace

Adapter Design Pattern

- It translates the contract (interface)
- Useful an object has functionality but doesn't match the interface requirement

Problem

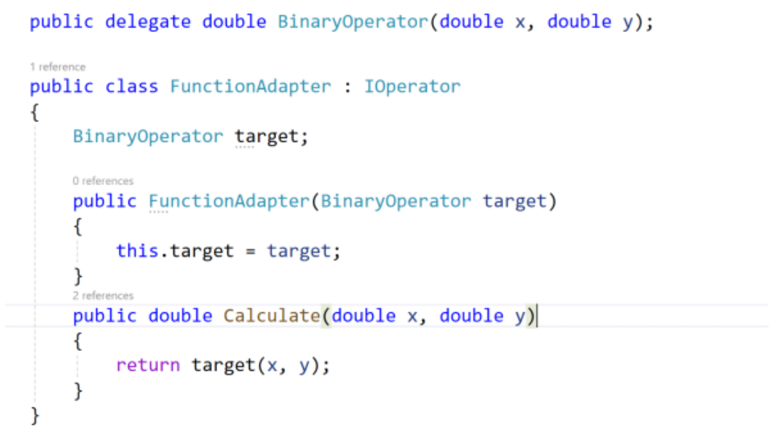
- My current design works with IOperator interface
  - It can work with only those objects that implements IOperator

- I have many built-in functions like Math.pow that can't support Ioperator
  - They are operations that can't work for my framework

Solution: Adapter

- Create a single Object that implements Ioperator
  - This object can be easily used with my Calcualtor
    - Calculator or application need not change

- It should wrap delegate as a parameter
- When user calls Calculate it internally calls delegate



```
public delegate double BinaryOperator(double x, double y);

public class FunctionAdapter : IOperator
{
    BinaryOperator target;

    public FunctionAdapter(BinaryOperator target)
    {
        this.target = target;
    }
    public double Calculate(double x, double y)
    {
        return target(x, y);
    }
}
```

Reflection

Friday, June 20, 2025 3:17 PM

Object	Property	Behavior	Code

employee	• Id
	• Name
	• Email
	• Phone
	• Salary
		• Work	class Employee{
			    int id;
			   string name;
			   double salary;
			   short workingHours;
			
			   public void Work(){}
			
			}

tiger	• Age
	• Weight
	• Name
		• Eat()
		• Hunt()
		• Move()
			class Tiger : Animal{
			    public void Hunt(){
			
			    }
			}

class	• Name
	• Fields
	• Methods
	• Properties
	• Constructors
	• Namespace
	• BaseClass
	• Interfaces
	• Scope
		• createObject	class Type
			{
			    string name;
			    string namespace;
			    MethodInfo[] methods;
			    ...
			    public object CreateObject();
			}

method	• Name
	• Return type
	• Parameters
	• Scope
	• Defining class
		• Invoke	Class MethodInfo
			{
			    string name;
			    ...
			    public void Invoke();
			}

Reflection

- Reflection is a set of classes that helps us identify (and use) programming elements at runtime without knowing their exact names
- Programming element may mean
  - Types like class, interface, struct, enum
  - Methods
  - Fields
  - Parameters
  - Constructors

- Reflection will allow to
  - Access information related to those elements like
    - Type
      - Name
      - Namespace
      - Methods
      - Fields
      - Constructors

    - Method
        - Name
        - Return type
        - Parameters
        - Scope
    - Field
        - Name
        - Type
        - Defining class

- Use that particular type programmetically
    - Type
        - You can create an object
    - Method
        - You can invoke the method
    - Field/Property
        - You can get or set values

Plugin Discovery

Friday, June 20, 2025 4:36 PM

What we are currently doing.

- Various different operators may be present in different assemblies
    - ConceptArchitect.Calculators (core library)
    - CalculatorApp (My main application)
    - Lnw.Calculations (third party library)
    - Lnt.Operations (another third party library)

- In the application project I need to add the project reference
    - If tomorrow we have more operators from another source
        - We will need to add them as references
        - We will need to
            - Recompile
            - Redistribute full application

-
- In (Main)
    - we need to manually add different operators to the calculator
        - calc.AddOperator(new MultiplyOperator(), "multiply");
    - If we need to add a new operator
        - We need to change main function
        - We will need to
            - Recompile
            - redistribute

What we want

- Various different Operators can come from different assemblies (dll files)
- Main Application will be compiled only with core library
    - No third party library reference added

- Any new feature (plugin) will be compiled as a separate dll
    - We don't need to compile main application

- We will copy the new feature.dll into a folder called plugins

- Calculator app while starting will automatically
    - Search all the available dll
    - Load them in memory
    - Search them for any IOperator present
    - Automatically add them to the calculator

[CalculatorBinary]

- Calculator.exe
- ConceptArchitect.Calculators.dll
- [plugins] <———— files are just dropped here. They will be read automatically
  - Lnw.Calculations
  - Lnt.TrigonometryLibrary

Attributes

Friday, June 20, 2025 5:33 PM

- Attributes are like meta Information (additional information) that we can attach to our programming elements like
  - Type
  - Method
  - Field
  - Parameter

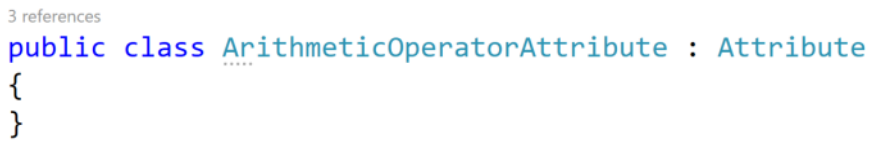
- They are actually Objects that can have
  - Fields
  - Properties
  - Behaviors

- They can be attached to an element like example below

```
[Hunter]
class Tiger{


}

class  Cow{

    [SpecialBehavior]
    public string ProvideMilk(){

    }

}


[Operator( Name="Permutation", Alias="per,p", Help="Calculates Permutation of
two numbers")]
class PermutationOperator : IOperator{

}
```

How do we create these attributes

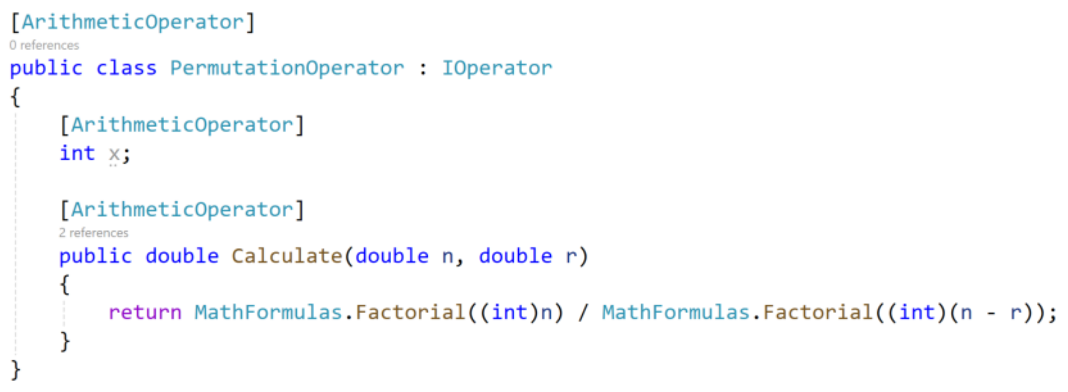
- We create a class that
  - Must inherit Attribute class
  - Should have an Attribute suffix



```
3 references
public class ArithmeticOperatorAttribute : Attribute
{
}
```

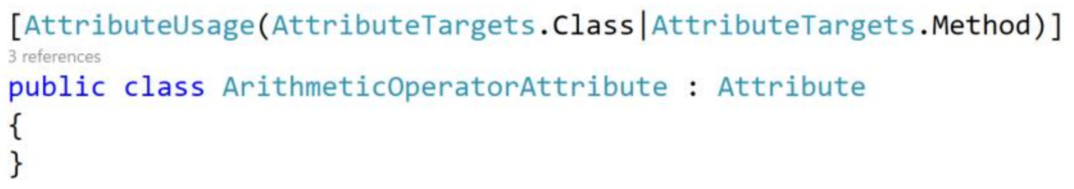
- Now we can apply it to any programming element

- Note we don't need to write Attribute suffix when adding attribute



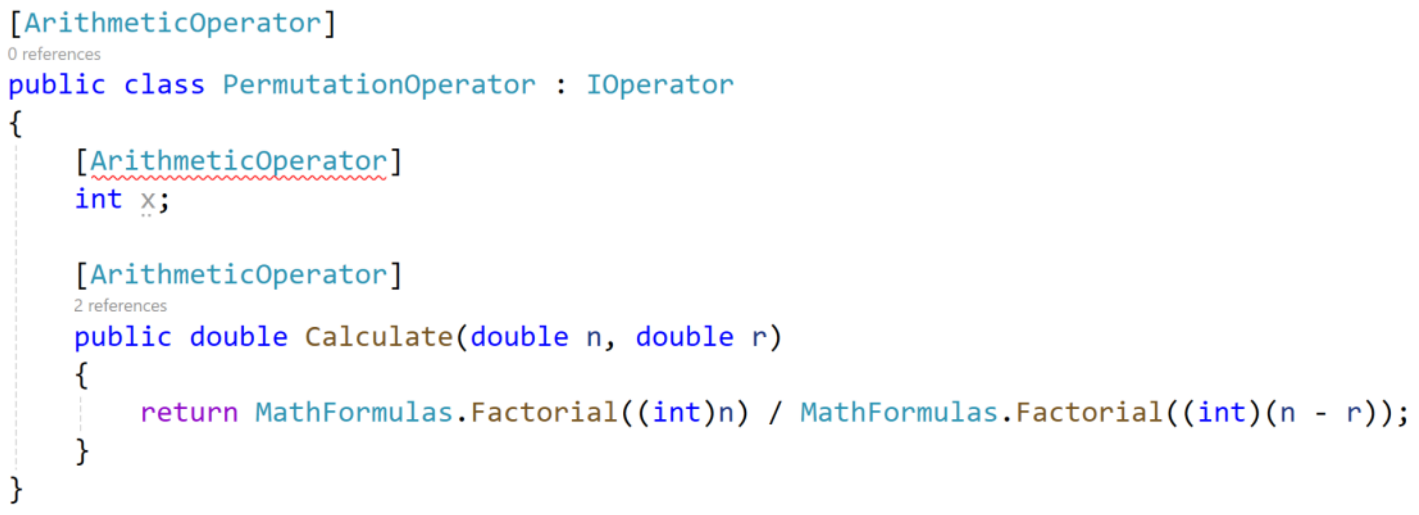
```
[ArithmeticOperator]
0 references
public class PermutationOperator : IOperator
{
    [ArithmeticOperator]
    int x;

    [ArithmeticOperator]
    2 references
    public double Calculate(double n, double r)
    {
        return MathFormulas.Factorial((int)n) / MathFormulas.Factorial((int)(n - r));
    }
}
```

- Problem
  - We can apply it to anything
  - We want to apply it only to classes or methods but not to fields



```
[AttributeUsage(AttributeTargets.Class|AttributeTargets.Method)]
3 references
public class ArithmeticOperatorAttribute : Attribute
{
}
```

- Now I can't apply it to a field



```
[ArithmeticOperator]
0 references
public class PermutationOperator : IOperator
{
    [ArithmeticOperator]
    int x;

    [ArithmeticOperator]
    2 references
    public double Calculate(double n, double r)
    {
        return MathFormulas.Factorial((int)n) / MathFormulas.Factorial((int)(n - r));
    }
}
```

What happens when we apply an attribute

- NOTHING
  - They are like dead objects

    - They sit and do nothing

- We can access and use the attribute objects only using reflection

var arithemticOperator = type.GetCustomAttribute()

Logger

Wednesday, October 8, 2025 10:19 PM

```
using Microsoft.Extensions.Logging;

class Program
{
    static void Main()
    {
        using var loggerFactory = LoggerFactory.Create(builder =>
        {
            builder
                .AddConsole()
                .AddDebug()
                .SetMinimumLevel(LogLevel.Information);
        });

        ILogger logger = loggerFactory.CreateLogger<Program>();

        logger.LogInformation("Application started at {time}", DateTime.Now);
        logger.LogWarning("Low disk space warning at {drive}", "C:");
        logger.LogError("An exception occurred: {message}", "Something bad happened");
    }
}
```

What is Unit Testing?

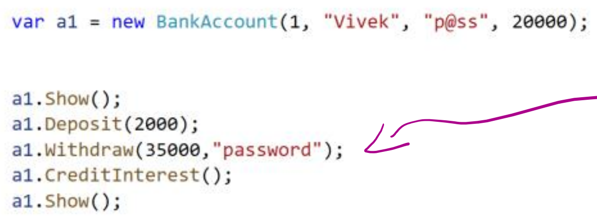
Wednesday, October 8, 2025 3:08 PM

- Testing the smallest unit of code to ensure it is working as expected.
- It is NOT a functional test that the QA Team performs
- It is a code-level test that the developer must perform
- The goal of the test is to uncover bugs related to
    - Logic
    - Data Integrity
    - Application Flow
- We traditionally do these tests based on a console based application

Problems with console based Application

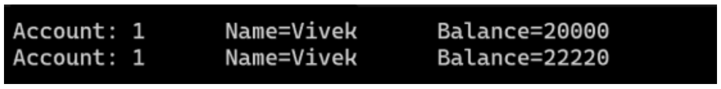
Wednesday, October 8, 2025 3:28 PM

1. Output is for human eyes.



```
var a1 = new BankAccount(1, "Vivek", "p@ss", 20000);


a1.Show();
a1.Deposit(2000);
a1.Withdraw(35000,"password");
a1.CreditInterest();
a1.Show();
```

- Perhaps withdraw didn't happen
  - But why?
    - Wrong password?
    - Wrong amount?



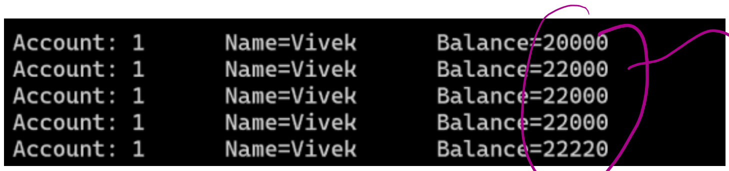
```
Account: 1      Name=Vivek      Balance=20000
Account: 1      Name=Vivek      Balance=22220
```

- Note:
  - In the output, there is absolutely no clue as to what the ideal result his ere.
  - The computer can't check this result.
  - We must manually calculate it the result is correct.

2. Final Output is composite of multiple Operation

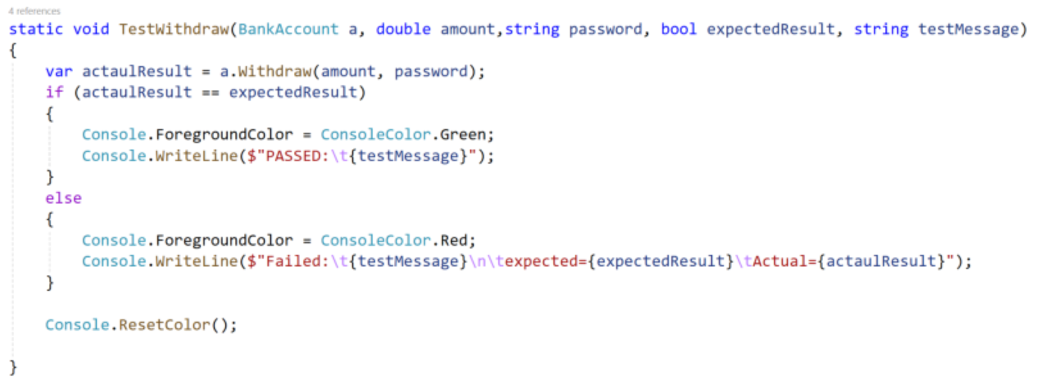
- Each operation may influence each other.
- If result is unexpected we may not know who is the culprit

How do I know what was supposed to be the ideal value here?

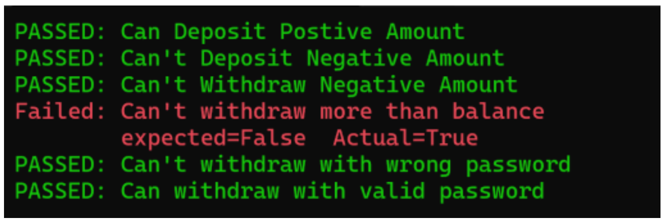


```
Account: 1      Name=Vivek      Balance=20000
Account: 1      Name=Vivek      Balance=22000
Account: 1      Name=Vivek      Balance=22000
Account: 1      Name=Vivek      Balance=22000
Account: 1      Name=Vivek      Balance=22220
```

3. Let's try to create proper Test Helper to organize the code.

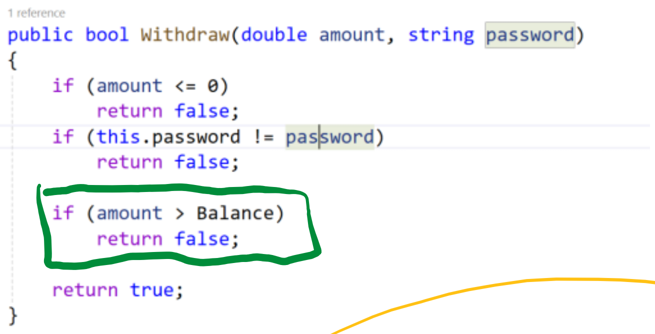


```
4 references
static void TestWithdraw(BankAccount a, double amount,string password, bool expectedResult, string testMessage)
{
    var actaulResult = a.Withdraw(amount, password);
    if (actaulResult == expectedResult)
    {
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine($"PASSED:\t{testMessage}");
    }
    else
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine($"Failed:\t{testMessage}\n\texpected={expectedResult}\tActual={actaulResult}");
    }

    Console.ResetColor();

}
```



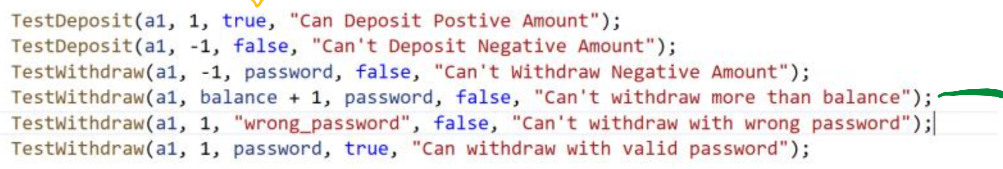
```
PASSED: Can Deposit Postive Amount
PASSED: Can't Deposit Negative Amount
PASSED: Can't Withdraw Negative Amount
Failed: Can't withdraw more than balance
        expected=False  Actual=True
PASSED: Can't withdraw with wrong password
PASSED: Can withdraw with valid password
```

4. Let's fix the Bug we found

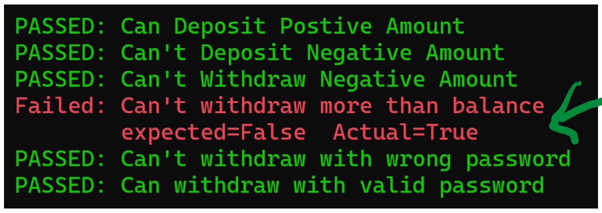


```
1 reference
public bool Withdraw(double amount, string password)
{
    if (amount <= 0)
        return false;
    if (this.password != password)
        return false;

    if (amount > Balance)
        return false;

    return true;
}
```

A \



```
TestDeposit(a1, 1, true, "Can Deposit Postive Amount");
TestDeposit(a1, -1, false, "Can't Deposit Negative Amount");
TestWithdraw(a1, -1, password, false, "Can't Withdraw Negative Amount");
TestWithdraw(a1, balance + 1, password, false, "Can't withdraw more than balance");
TestWithdraw(a1, 1, "wrong_password", false, "Can't withdraw with wrong password");
TestWithdraw(a1, 1, password, true, "Can withdraw with valid password");
```



```
PASSED: Can Deposit Postive Amount
PASSED: Can't Deposit Negative Amount
PASSED: Can't Withdraw Negative Amount
Failed: Can't withdraw more than balance
        expected=False  Actual=True
PASSED: Can't withdraw with wrong password
PASSED: Can withdraw with valid password
```

- This withdrawal shouldn't be allowed as per new logic which is correct
- But the test is still failing (although the logic is fixed)

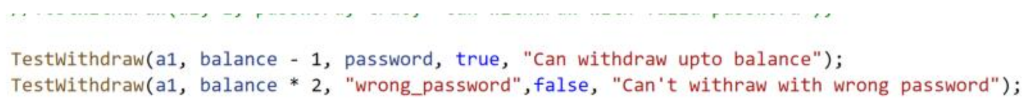
- A previous test had added an additional Rs into BankAccount
    - Current balance is now balance+1
    - This is actual amount we want to withdraw
        - Which is allowed.

False Negative

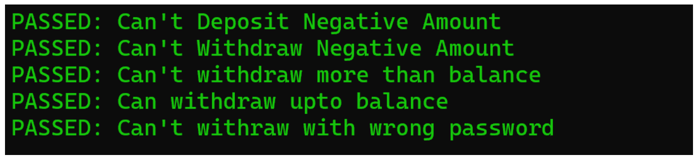
- A Test fails although the logic is correct because one test may be influencing another
- The data is inconsistent
- The test assumes a balance of 20000 actual balance is 20001
    - Thus test indicates failure when it was actually a success

False Positive

- A test passes even if the programming logic written is wrong
- This may be due to the incorrect data we have.



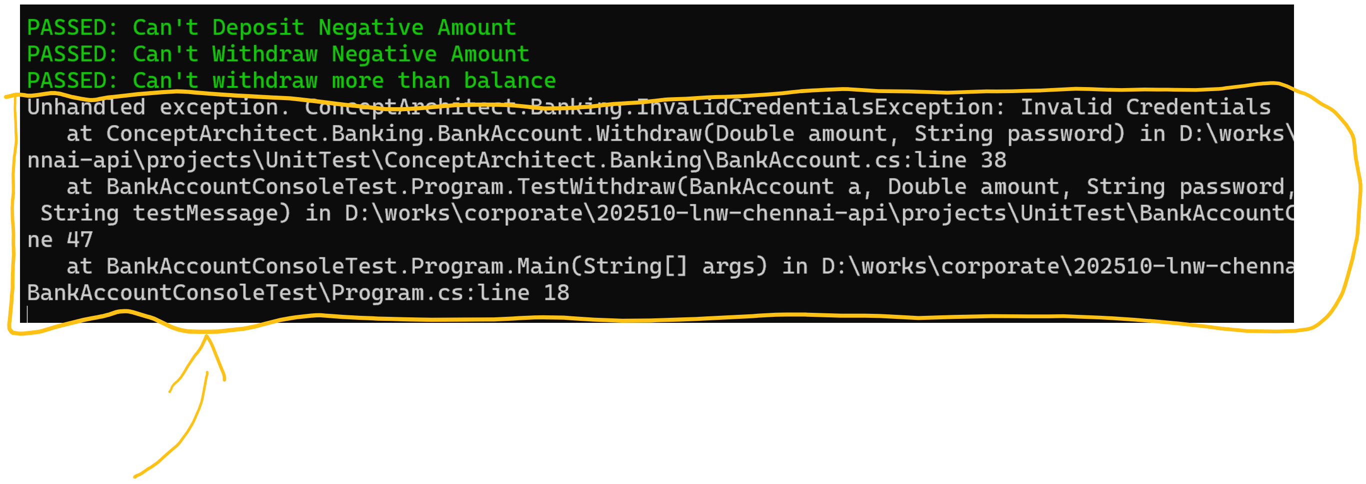
```
TestWithdraw(a1, balance - 1, password, true, "Can withdraw upto balance");
TestWithdraw(a1, balance * 2, "wrong_password",false, "Can't withraw with wrong password");
```



```
PASSED: Can't Deposit Negative Amount
PASSED: Can't Withdraw Negative Amount
PASSED: Can't withdraw more than balance
PASSED: Can withdraw upto balance
PASSED: Can't withraw with wrong password
```

- Withdrawal was rejected because of balance issue not because of the wrong password

What if some situation throws exception



```
PASSED: Can't Deposit Negative Amount
PASSED: Can't Withdraw Negative Amount
PASSED: Can't withdraw more than balance
Unhandled exception. ConceptArchitect.Banking.InvalidCredentialsException: Invalid Credentials
   at ConceptArchitect.Banking.BankAccount.Withdraw(Double amount, String password) in D:\works\
nnai-api\projects\UnitTest\ConceptArchitect.Banking\BankAccount.cs:line 38
   at BankAccountConsoleTest.Program.TestWithdraw(BankAccount a, Double amount, String password,
 String testMessage) in D:\works\corporate\202510-lnw-chennai-api\projects\UnitTest\BankAccountC
ne 47
   at BankAccountConsoleTest.Program.Main(String[] args) in D:\works\corporate\202510-lnw-chenna
BankAccountConsoleTest\Program.cs:line 18
```

Test Passed or Failed?

- We expected an exception to be thrown for the wrong password
- It threw an exception
  - That's the expected use case
- Thus Test Passed
- But Human psychology —> Exception = Failure.

What Happened to the Rest of the Tests?

- They are not executed because application crashed.

Summarize

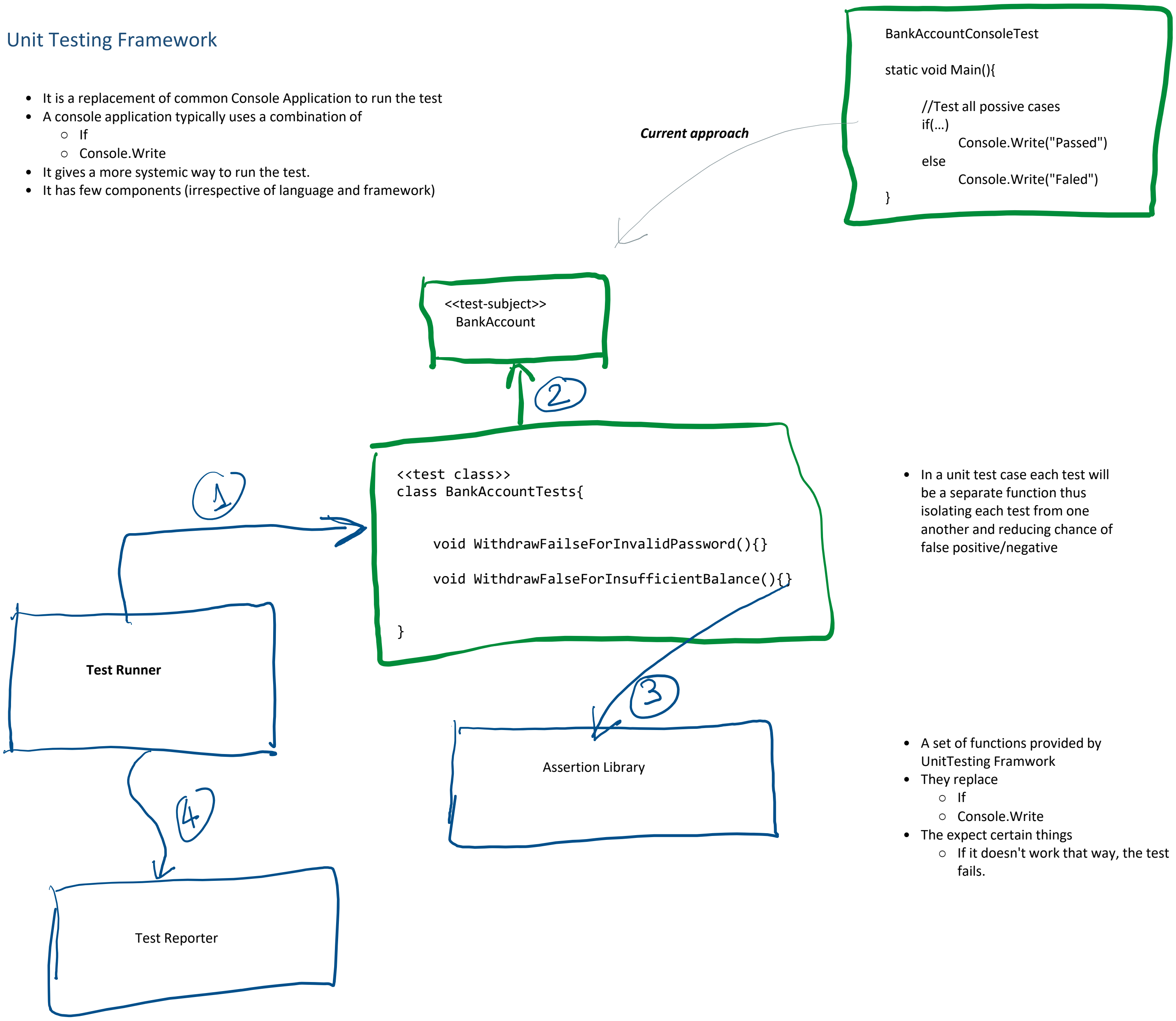
- Output are for human eyes only
  - We need to decide if current data is expected
    - (we may solve this problem by creating an organized test)
- Tests are not isolated
  - A test result may change the object state
  - Tests are written based on initial state assumption
- May lead to
  - False positive
    - Code is wrong but passing the rest
  - False negative
    - Code is not wrong but test is failing
- Exception ejects the program and all test may not execute.

How Unit Testing Works

Wednesday, October 8, 2025 4:05 PM

Unit Testing Framework

- It is a replacement of common Console Application to run the test
- A console application typically uses a combination of
  - If
  - Console.Write
- It gives a more systemic way to run the test.
- It has few components (irrespective of language and framework)



```
BankAccountConsoleTest

static void Main(){

        //Test all possive cases
        if(...)
                Console.Write("Passed")
        else
                Console.Write("Faled")
}
```

```
<<test class>>
class BankAccountTests{


    void WithdrawFailseForInvalidPassword(){}

    void WithdrawFalseForInsufficientBalance(){}


}
```

- In a unit test case each test will be a separate function thus isolating each test from one another and reducing chance of false positive/negative

- A set of functions provided by UnitTesting Framwork
- They replace
  - If
  - Console.Write
- The expect certain things
  - If it doesn't work that way, the test fails.

- Test reporter can present the test results to the user
- It may be
  - Simple Console Output
  - A GUI reporting the test results.

Test Frameworks

Wednesday, October 8, 2025 4:17 PM

There are several Test Framework Available to Unit Test .Net codes

1. MS Test

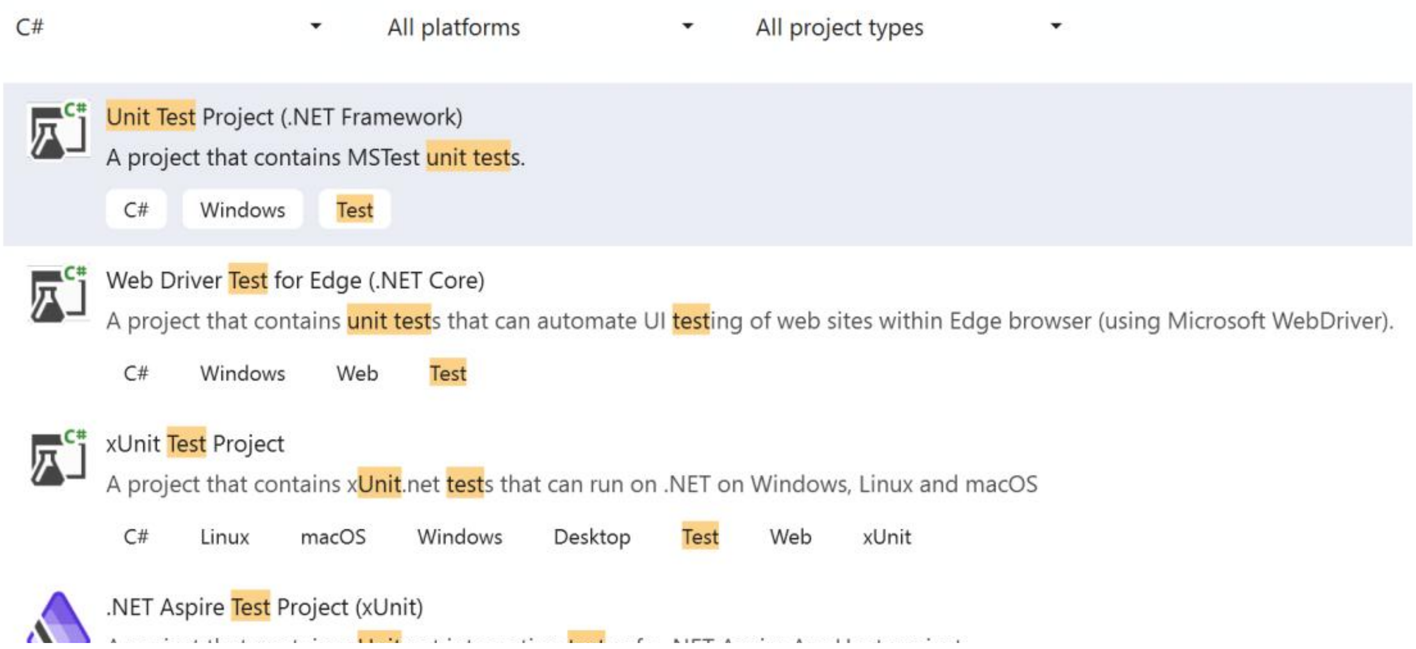
- Oldest Microsoft Test Framework
- Works only with
    - Visual Studio
    - .NET Framework application

2. Nunit

- Third-party framework
- Created by the same team that created Java's Junit framework
    - JUnit is the first unit testing framework in any language.

3. Xunit

- A modern cross-platform testing framework created by Microsoft
- Works for both .NET framework and .NET core
- Works across OS platforms
- Can work even outside Visual Studio

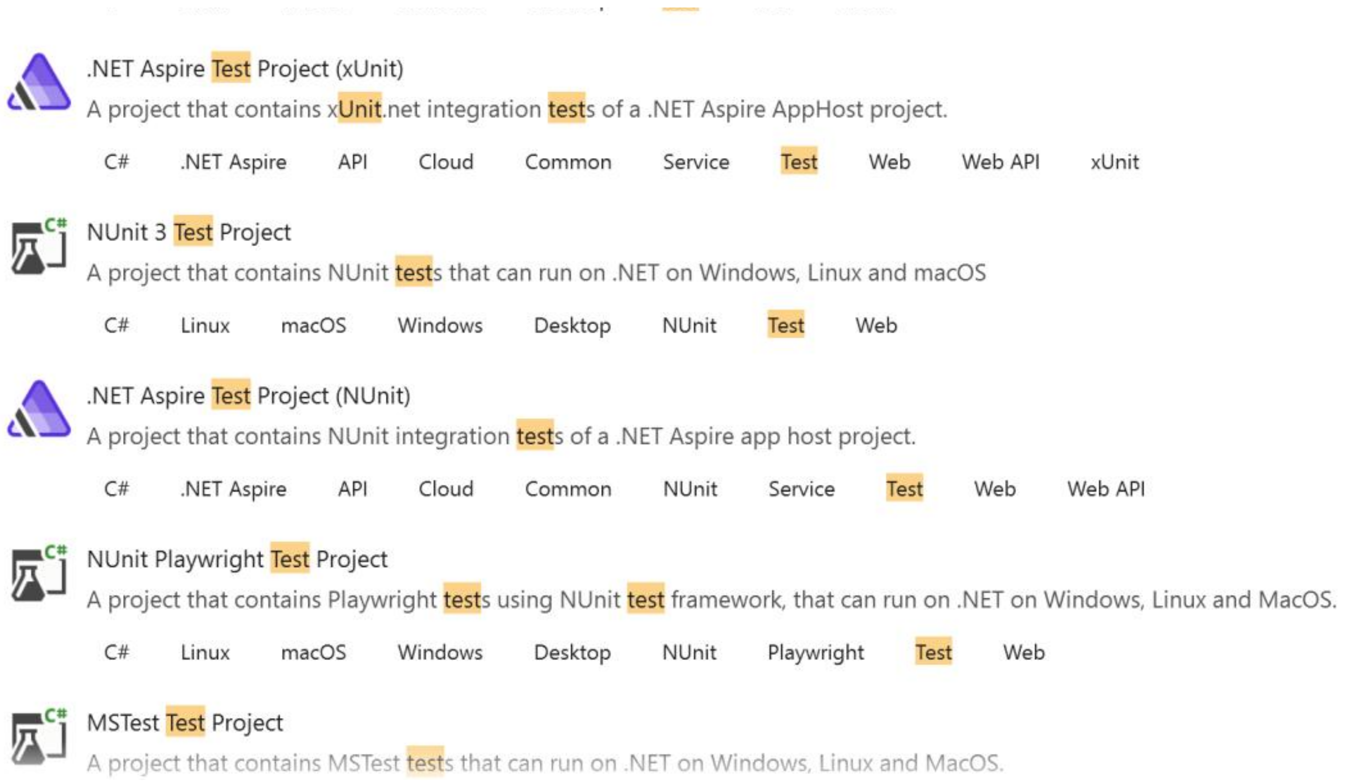


Unit Test Project (.NET Framework)
A project that contains MSTest unit tests.
C# Windows Test

Web Driver Test for Edge (.NET Core)
A project that contains unit tests that can automate UI testing of web sites within Edge browser (using Microsoft WebDriver).
C# Windows Web Test

xUnit Test Project
A project that contains xUnit.net tests that can run on .NET on Windows, Linux and macOS
C# Linux macOS Windows Desktop Test Web xUnit

.NET Aspire Test Project (xUnit)


.NET Aspire Test Project (xUnit)
A project that contains xUnit.net integration tests of a .NET Aspire AppHost project.
C# .NET Aspire API Cloud Common Service Test Web Web API xUnit

NUnit 3 Test Project
A project that contains NUnit tests that can run on .NET on Windows, Linux and macOS
C# Linux macOS Windows Desktop NUnit Test Web

.NET Aspire Test Project (NUnit)
A project that contains NUnit integration tests of a .NET Aspire app host project.
C# .NET Aspire API Cloud Common NUnit Service Test Web Web API

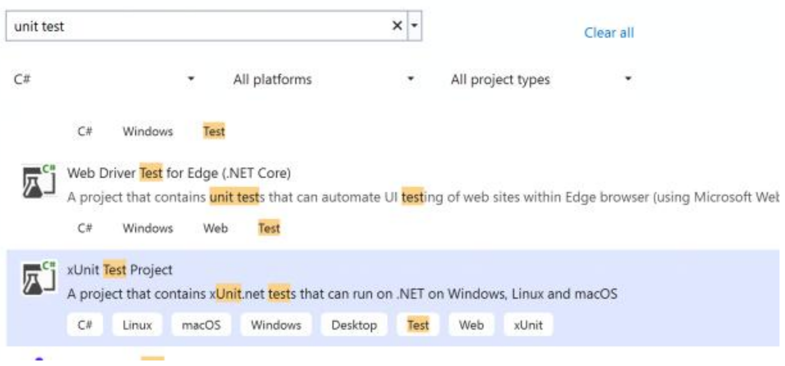
NUnit Playwright Test Project
A project that contains Playwright tests using NUnit test framework, that can run on .NET on Windows, Linux and MacOS.
C# Linux macOS Windows Desktop NUnit Playwright Test Web

MSTest Test Project
A project that contains MSTest tests that can run on .NET on Windows, Linux and MacOS.

Xunit Getting Started

Wednesday, October 8, 2025 4:21 PM

Step #1 Create an XUnit Project



unit test

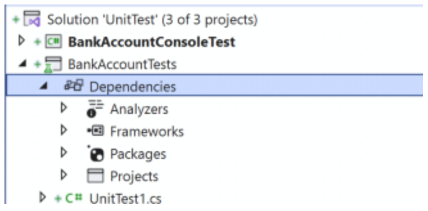
Clear all

C# | All platforms | All project types

C# Windows Test

Web Driver Test for Edge (.NET Core)
A project that contains unit tests that can automate UI testing of web sites within Edge browser (using Microsoft Wel
C# Windows Web Test

xUnit Test Project
A project that contains xUnit.net tests that can run on .NET on Windows, Linux and macOS
C# Linux macOS Windows Desktop Test Web xUnit



```
Solution 'UnitTest' (3 of 3 projects)
  BankAccountConsoleTest
  BankAccountTests
    Dependencies
      Analyzers
      Frameworks
      Packages
      Projects
    UnitTest1.cs
```

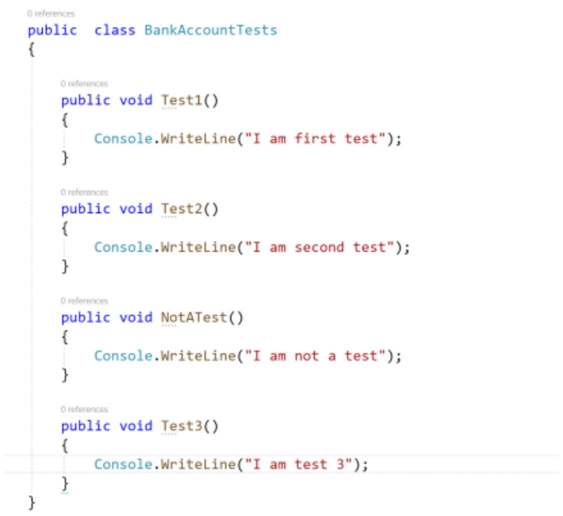
Test Organization

1. We need to create a Test Class

- A Test class is an ordinary C# class that can have one or more test methods
- The class is just a group of methods.
- Generally we name it after the class/method we are testing with TestSuffix
    - Example
        - BankAccountTests
            - To test all aspect of BankAccount
        - BankAccountWithdrawTests
            - To test all aspects of Withdraw function only
                - We use this if there are too many tests needed for a single function
- It can have one or more test methods

2. Test Methods

- Test Methods are standard public non static methods
- They should be attributed with [Fact]

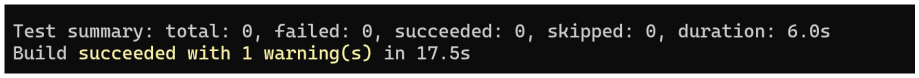


```
public  class BankAccountTests
{
    public void Test1()
    {
        Console.WriteLine("I am first test");
    }

    public void Test2()
    {
        Console.WriteLine("I am second test");
    }

    public void NotATest()
    {
        Console.WriteLine("I am not a test");
    }

    public void Test3()
    {
        Console.WriteLine("I am test 3");
    }
}
```

3. Running a Test

1. From the command Line (Only for .NET Core)

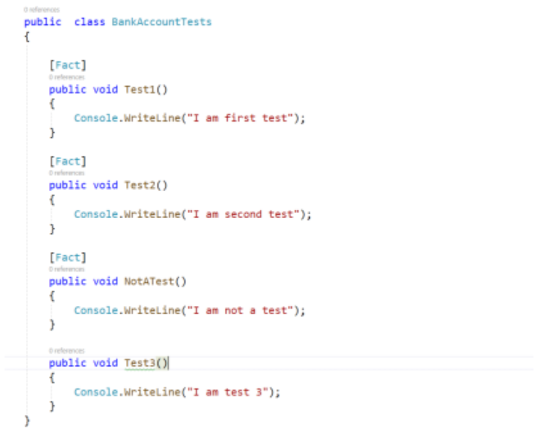
c:\project> dotnet test



```
Test summary: total: 0, failed: 0, succeeded: 0, skipped: 0, duration: 6.0s
Build succeeded with 1 warning(s) in 17.5s
```

Why?

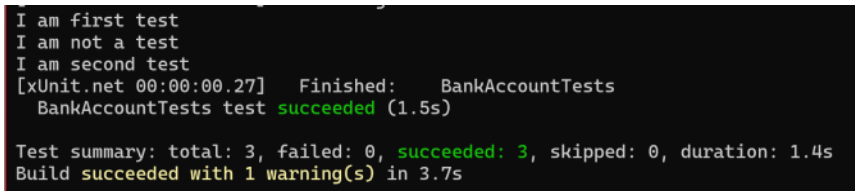
- Only those methods that are marked with [Fact] qualifies as test



```
public  class BankAccountTests
{
    [Fact]
    public void Test1()
    {
        Console.WriteLine("I am first test");
    }

    [Fact]
    public void Test2()
    {
        Console.WriteLine("I am second test");
    }

    [Fact]
    public void NotATest()
    {
        Console.WriteLine("I am not a test");
    }

    public void Test3()
    {
        Console.WriteLine("I am test 3");
    }
}
```

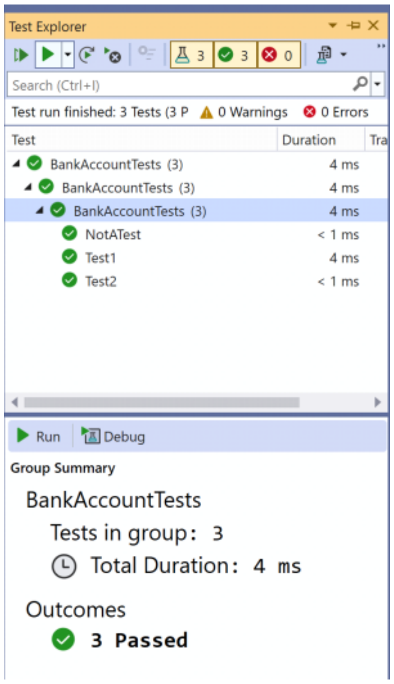
IMPORTANT NOTE

- Test names don't matter
  - NotATest is also executed
  - Test3 is not executed
- There is no specific order to run test
  - They may run in any order
  -



```
I am first test
I am not a test
I am second test
[xUnit.net 00:00:00.27]    Finished:    BankAccountTests
  BankAccountTests test succeeded (1.5s)

Test summary: total: 3, failed: 0, succeeded: 3, skipped: 0, duration: 1.4s
Build succeeded with 1 warning(s) in 3.7s
```

Running the Test with Visual Studio



```
Test Explorer
3  3  0
Search (Ctrl+I)
Test run finished: 3 Tests (3 P   0 Warnings   0 Errors
Test                              Duration   Tra
◢ BankAccountTests (3)            4 ms
  ◢ BankAccountTests (3)          4 ms
    ◢ BankAccountTests (3)        4 ms
        NotATest                  < 1 ms
        Test1                     4 ms
        Test2                     < 1 ms
```

Run Debug

Group Summary

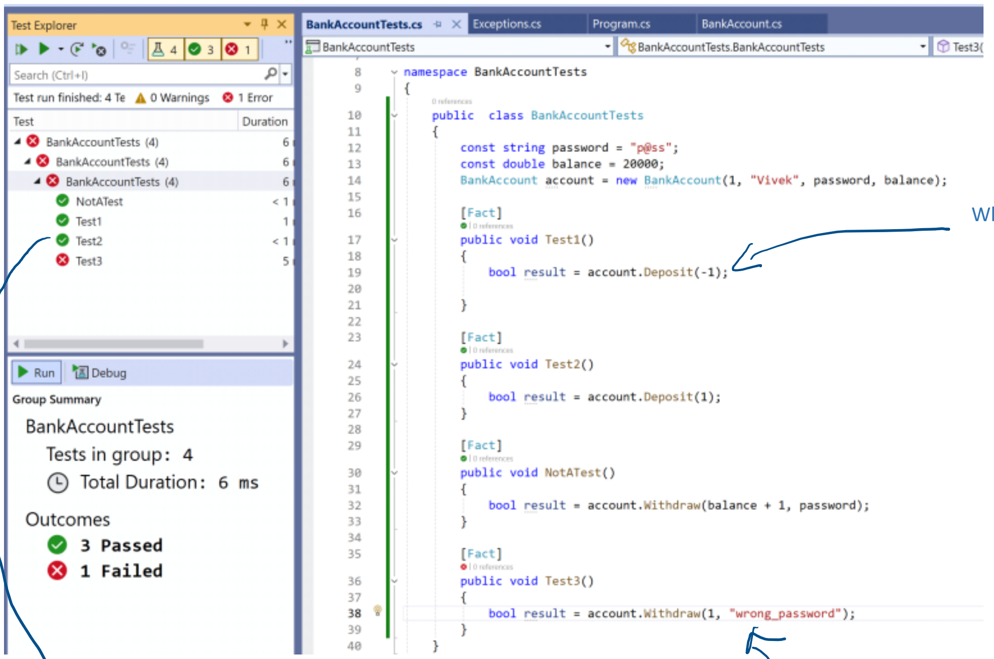
BankAccountTests
Tests in group: 3
Total Duration: 4 ms

Outcomes
3 Passed

Why did the test pass?

- Because it had no reason to fail.
- If a function completes without error, test is passed.

Consider the current Test Example



```
Test Explorer
4  3  1
Search (Ctrl+I)
Test run finished: 4 Te   0 Warnings   1 Error
Test                              Duration
◢ BankAccountTests (4)            6
  ◢ BankAccountTests (4)          6
    ◢ BankAccountTests (4)        6
        NotATest                  < 1
        Test1                     1
        Test2                     < 1
        Test3                     5
```

Run Debug

Group Summary

BankAccountTests
Tests in group: 4
Total Duration: 6 ms

Outcomes
3 Passed
1 Failed

BankAccountTests.cs   Exceptions.cs   Program.cs   BankAccount.cs

BankAccountTests   BankAccountTests.BankAccountTests   Test3(

```csharp
namespace BankAccountTests
{
    public class BankAccountTests
    {
        const string password = "p@ss";
        const double balance = 20000;
        BankAccount account = new BankAccount(1, "Vivek", password, balance);

        [Fact]
        public void Test1()
        {
            bool result = account.Deposit(-1);

        }

        [Fact]
        public void Test2()
        {
            bool result = account.Deposit(1);
        }

        [Fact]
        public void NotATest()
        {
            bool result = account.Withdraw(balance + 1, password);
        }

        [Fact]
        public void Test3()
        {
            bool result = account.Withdraw(1, "wrong_password");
        }
    }
```

Why did this test pass?

- Because this function finished without any error
- It may or may not have desired result but it didn't crash

What is Test2 or Test3 testing?

- Names like test2 or test3 are not helpful in identifying the current test

Why did this test fail?

- Because any unexpected exception is considered as a

What is Test2 or Test3 testing?

- Names like test2 or test3 are not helpful in identifying the current test

Why did this test fail?

- Because any unexpected exception is considered as a failure

Test Structuring Best Practices

1. DAMP Principle

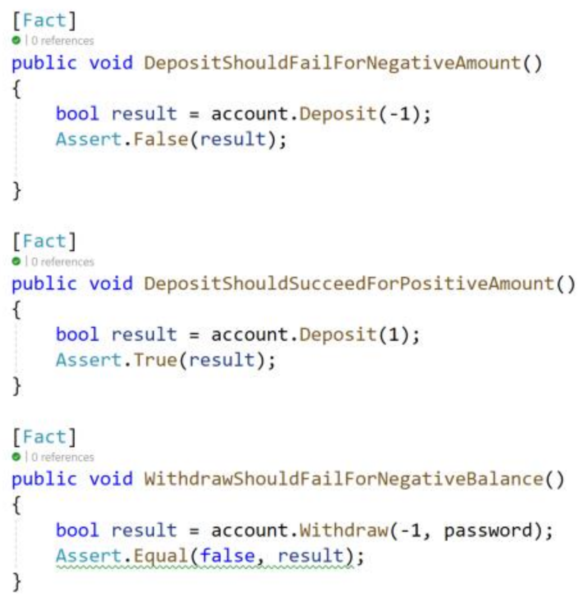
- Descriptive and Meaningful Phrase
- Name shouldn't just be a meaningful word but a phrase describing the intent
- Avoid
  - Test1()
  - TestWithdraw()
- Use
  - WithdrawFailsForNegativeAmount()

2. Tests run on A-A-A

1. Arrange
   - Arrange your code with well-defined initial data.
2. Act
   - Perform the job you want to test
3. Assert
   - Call assert function to assert your expectations

3. Test follows ACID

- Atomic
  - Test should be atomic
  - It should test one use case or test path
  - Avoid
    - TestWithcraw
  - Test
    - Withdraw Fails for
      - Negative amount
      - Insufficient balance
      - Invalid credentials
    - Passes of
      - Happy case
- Consistency
  - Test should always give same result for same data set
- Isolation
  - Test must be isolated
  - One test shouldn't interfere with other
- Durable
  - Test can run again and again returning same expected information



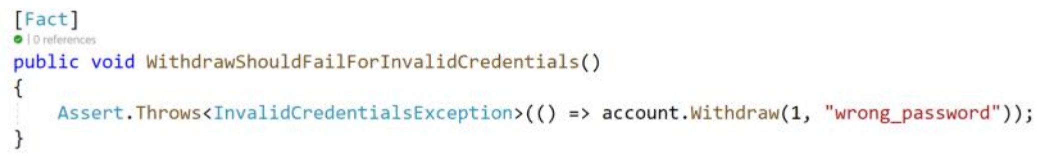
```
[Fact]
| 0 references
public void DepositShouldFailForNegativeAmount()
{
    bool result = account.Deposit(-1);
    Assert.False(result);

}

[Fact]
| 0 references
public void DepositShouldSucceedForPositiveAmount()
{
    bool result = account.Deposit(1);
    Assert.True(result);
}

[Fact]
| 0 references
public void WithdrawShouldFailForNegativeBalance()
{
    bool result = account.Withdraw(-1, password);
    Assert.Equal(false, result);
}
```

Tes

How do I Test Exception

- By default any exception will cause a failure
- But the good news, it will not stop other tests from running

Testing for Exception



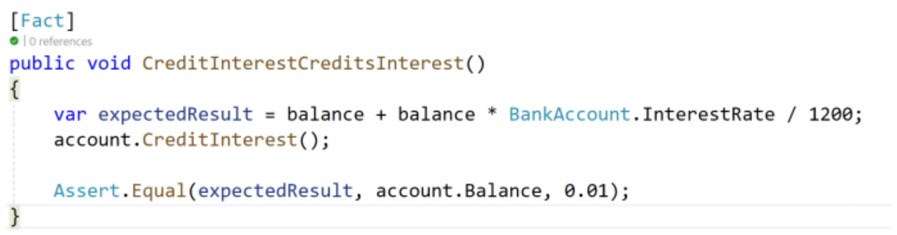
```
[Fact]
| 0 references
public void WithdrawShouldFailForInvalidCredentials()
{
    Assert.Throws<InvalidCredentialsException>(() => account.Withdraw(1, "wrong_password"));
}
```

If we don't test right we may get false positives

- Out withdraw tests are passing
- But that is all false positive
- Actually withdraw has a logical error
- But since we have wrong assert, we are unable to identify the bug

How do I test credit Interest (A function that doesn’t return anything) — what do I assert on?

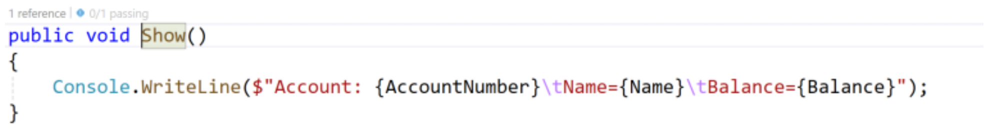
- We can compare the value of states that gets effected by current function call.
- Here Credit interest may not return a value but it does update the balance



```
[Fact]
| 0 references
public void CreditInterestCreditsInterest()
{
    var expectedResult = balance + balance * BankAccount.InterestRate / 1200;
    account.CreditInterest();

    Assert.Equal(expectedResult, account.Balance, 0.01);
}
```

How do I test account.Show?

- It returns nothing
- It doesn't affect any state of the object



```
1 reference | 0/1 passing
public void Show()
{
    Console.WriteLine($"Account: {AccountNumber}\tName={Name}\tBalance={Balance}");
}
```

- This code is untestable.

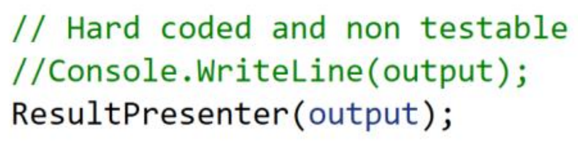
Revisiting Calculator

Wednesday, October 8, 2025 5:48 PM

Problem

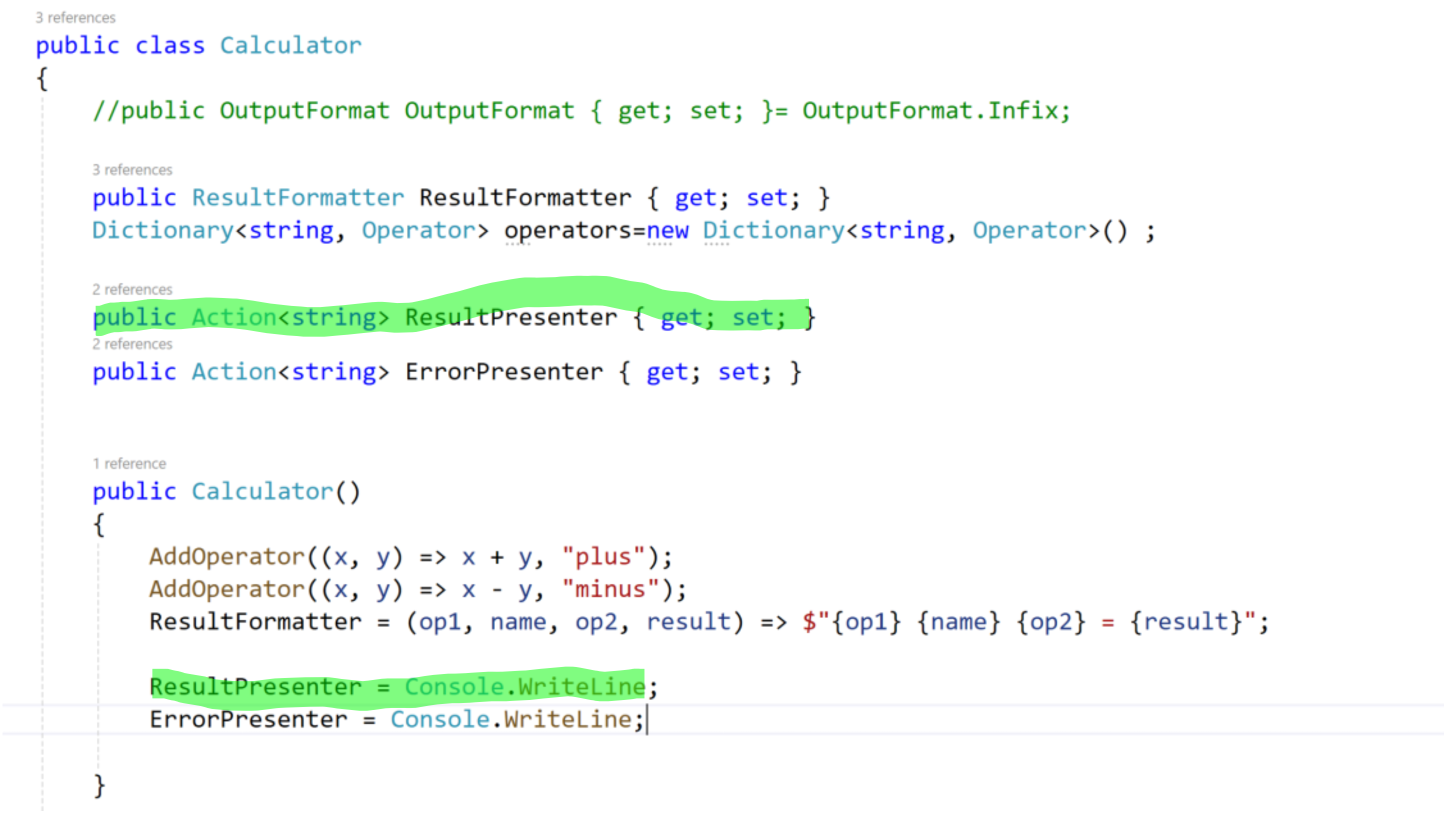
- Our calculator always emits the result to Console
- Console.WriteLine is not testable
  - Anything not testable is, in general, not a good code
- We need to find an alternative
- An alternative is no just for making it testable
  - It will often make the code more usable
- Actual Enhancement
  - We may replace Console with a Real GUI

Step #1 replace hard-coded Console.WriteLine with a Result Presenter



```
// Hard coded and non testable
//Console.WriteLine(output);
ResultPresenter(output);
```

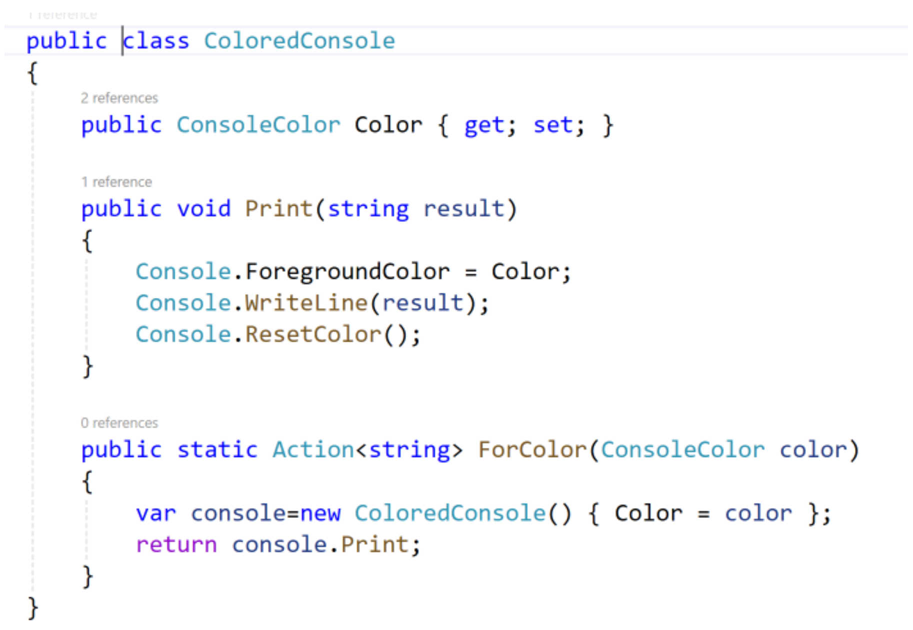
Step #2 Supply Console.WriteLine as default Result Presenter



```
3 references
public class Calculator
{
    //public OutputFormat OutputFormat { get; set; }= OutputFormat.Infix;

    3 references
    public ResultFormatter ResultFormatter { get; set; }
    Dictionary<string, Operator> operators=new Dictionary<string, Operator>() ;

    2 references
    public Action<string> ResultPresenter { get; set; }
    2 references
    public Action<string> ErrorPresenter { get; set; }


    1 reference
    public Calculator()
    {
        AddOperator((x, y) => x + y, "plus");
        AddOperator((x, y) => x - y, "minus");
        ResultFormatter = (op1, name, op2, result) => $"{op1} {name} {op2} = {result}";

        ResultPresenter = Console.WriteLine;
        ErrorPresenter = Console.WriteLine;

    }
```

How is the code advantageous?

- Console.WriteLine is not replaceable
- ResultPresenter is replaceable



```
public class ColoredConsole
{
    public ConsoleColor Color { get; set; }

    public void Print(string result)
    {
        Console.ForegroundColor = Color;
        Console.WriteLine(result);
        Console.ResetColor();
    }

    public static Action<string> ForColor(ConsoleColor color)
    {
        var console=new ColoredConsole() { Color = color };
        return console.Print;
    }
}
```

Assingment 4.3

Wednesday, October 8, 2025 6:00 PM

- Write Unit Testing to Test following Case

Test for InfixFormatter

- Infix formatter formats the result correctly

Test For calculator

- Calculator performs correct calculation for valid operator
- Calculator "Invalid Operator:'foo'" if we pass invalid operator foo
- Calculator doesn't call formatter if operator is invalid
- Calculator uses the value returned by formatter for printing
- Calculator invokes ResultPresenter with formatted data
- Calculator invokes ErrorPresenter with error data

Interface

Thursday, October 9, 2025 11:23 AM

```
abstract class  Animal{
    public abstract string Eat();
    public abstract string Move();
}

public class Horse : Animal{
    public  string Eat() {returns "Grass"; }
    public string Move() {return "Runs";}

}

public class Tiger : Animal{
    public string Move() {return "Walks";}

    public  string Eat() {returns Hunt()+" Eats"; }

    //new method
    public string Hunt(){ returns "Hunts" ;}

}


Tiger tiger = new Tiger();

tiger.Move();  //walks
tiger.Hunt() ;  //Hunts


Animal animal = tiger;

animal.Move();  //walks
animal.Hunt();  //compilation error: Hunt is not defined.
animal.Eat();  // Hunts Eats <—— calls Hunt inside eat
```

Why Compiler gives a compile time error on Hunt?

- Objects are created at runtime
- Compiler can't be sure what Animal reference will be refering to —> tiger or horse or something else
- It can't be 100% sure that object being referred has a Hunt method
- To be safe in all situation, it allows only those method calls it is 100% sure about
  - Methods already defined and known to Animal reference

How to solve the Hunt Problem

1. We need to define an interface Ihunter

```
interface IHunter{

    void Hunt();

}
```

2. Let all animals who hunt implment this interface

```
class Tiger : Animal, IHunter{…}
class Eagle : Bird, IHunter{…}
class Snake : Animal, IHunter{…}
…
```

3. Now check for interface, type cast and use

This is why the interface exists. Interface=>promise

- It is a promise that a given method(s) will be present in the current object
- Same reason we use delegate
  - Operator delegate promises that current method will
    - Take 2 int

- It is a promise that a given method(s) will be present in the current object
- Same reason we use delegate
  - Operator delegate promises that current method will
    - Take 2 int
    - Return an int
  - Action<string> promises that it will
    - Take string
    - Return void
  - ResultFormatter promises that it will
    - Take —> op1, operatorName, op2, result
    - Return —> string

3. Now check for interface, type cast and use

```
Animal [] animals = {….}

foreach(var animal in animals){

    if( animal is IHunter){
        var hunter = animal as IHunter;
        hunter.Hunt();
    }

}
```

Why should the compiler give a compile-time error on Hunt?

- Modern languages like C# (and Java), unlike c++, do not produce Stand Alone Executables
- They have a run-time environment
- Since a runtime engine is present, it can certainly check if current object has Hunt method or not
- Why should this decision be taken eagerly at compile time?

Reflection

Thursday, October 9, 2025 2:09 PM

What is Reflection?

- It is a library that enables us to explore the programming elements at runtime
- Programming elements
  - Class
  - Fields
  - Properties
  - Methods
  - Assembly
- We can find out their details
  - What methods are present in a class
  - What are parameters of a method
- We can use them dynamically without making direct calls
  - Create object of a class
  - Get/set values of properties/fields
  - Invoke a method

How it works?

- Employee
  - employeeId
  - Name
  - Salary
  - Work()
- Snake
  - Poison
  - Length
  - Age
  - Colour

```
class Employee{
        int employeeId;
        string name;
        double salary;

        public void Work(){}
}

class Snake{
        int poison;
        int length;
        int age;
        Color color;
}
```

How about creating a class called "Class"

- What properties or behavior a class object can have?
- What do we know about a class/Type
  - Property
    - Name
    - List of
      - Methods
      - Properties
      - Fields
      - Constructor
    - Base Class
    - List of
      - Interfaces
    - Scope [public/abstract/static]
    - Namespace
  - Behavior
    - Create an object

```
class ~~Class~~ Type  //represents class, struct, interface,  enum
{
        string name;
        string namespace;
        Type baseType;
        Type[] interfaces;
        MethodInfo[] methods;

}
```

```
class MethodInfo
{
        string name;
        Type definingType;
        Type returnType;
        ParameterInfo[] parameters;
```

- What do we know about a Method?

```
        Type definingType;
        Type returnType;
        ParameterInfo[] parameters;

}

class ParameterInfo
{
        Type type;
        String name;

}
```

- What do we know about a Method?
  - Property
    - Name
    - List of Parameters
    - Return type
    - Scope
    - Defining class
  - Behavior
    - Invoke

Available classes

- System
  - Type
- System.Reflect
  - MethodInfo
  - ParameterInfo
  - FieldInfo
  - ConstructorInfo
  - PropertyInfo

How to create a Type Object?

How not to create it

Type type = new Type();

Why?

- Type represents class/struct/enum/interface.
- To create them, you need to declare them.
  class Tiger{ }

- Not
  - Type Tiger =new Type()

How to get the type object

- Type object always refers to an existing type already created
- We access them using various factory methods provided by the framework

Approach #1. If you have an object, you can find its type

var tiger = new Tiger();

var type = tiger.GetType(); //object class method

Approach #2. If you have an class, you can find its type object

var type2 = typeof(Tiger); //C# keyword

Approach #3. If you know full class name, you can find its type object

```
var type3 = Type.GetType("Tiger");  //note it can even be a user input.
```

Approach #3. __________

Note

- The type1, type2 and type3 are exactly same object and not different copies
    - There will be a single type object per type
    - It doesn't create a new one every time.

Attributes

Thursday, October 9, 2025 3:26 PM

Conceptually

- They are meta information that can be attached to any programing element like
  - Type
  - Method
  - Field
  - Constructor
  - Parameter
- Think of them as special comments
- They exists at runtime
- They can be accessed using Reflection (Only)
- They are non-obstrusive
  - If you don't know or care for them, they will not disturb you.

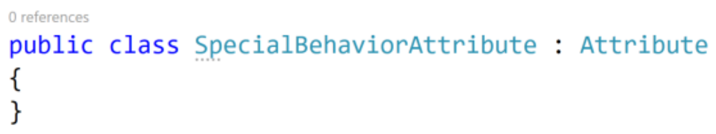
Programmatically

- An attribute is a class that
  - Extends Attribute class
  - Conventionally has Attribute Suffix in their name

- [Fact] —-> class FactAttribute:Attribute{}

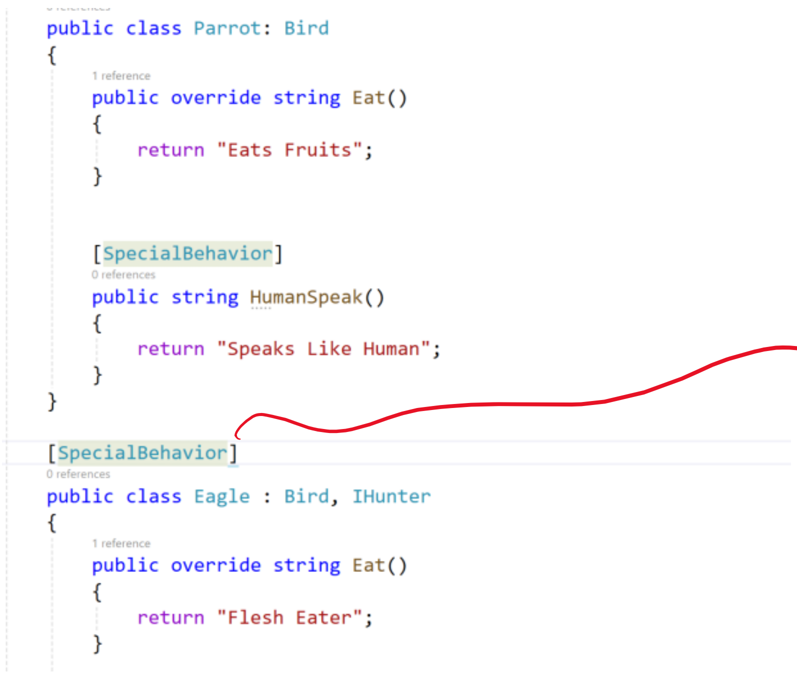
- [Serializable] —> class SerializableAttribute:Attribute{}

Let's create a special behavior attribute



```
public class SpecialBehaviorAttribute : Attribute
{
}
```

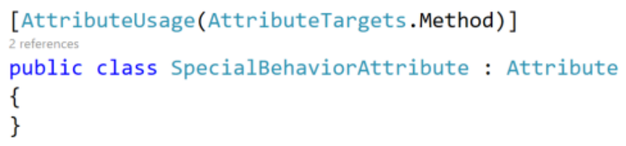
- Now you can use this attribute with any of the programming element



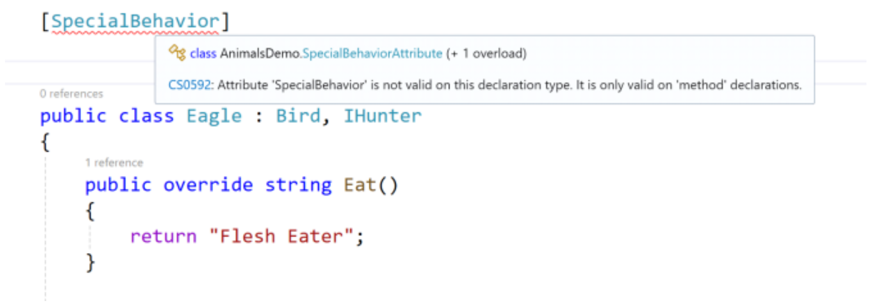
```
public class Parrot: Bird
{
    1 reference
    public override string Eat()
    {
        return "Eats Fruits";
    }


    [SpecialBehavior]
    0 references
    public string HumanSpeak()
    {
        return "Speaks Like Human";
    }
}

[SpecialBehavior]
0 references
public class Eagle : Bird, IHunter
{
    1 reference
    public override string Eat()
    {
        return "Flesh Eater";
    }
```

- Doesn't make any sense here!
- It is a class
- This attribute makes sense only for methods

We can Limit Attribute usage to certain type only using another Attribute



```
[AttributeUsage(AttributeTargets.Method)]
2 references
public class SpecialBehaviorAttribute : Attribute
{
}
```



```
[SpecialBehavior]
    class AnimalsDemo.SpecialBehaviorAttribute (+ 1 overload)
    CS0592: Attribute 'SpecialBehavior' is not valid on this declaration type. It is only valid on 'method' declarations.
0 references
public class Eagle : Bird, IHunter
{
    1 reference
    public override string Eat()
    {
        return "Flesh Eater";
    }
```

Attributes still does nothing

- Attributes are like dead objects (e.g. chairs)
- They do nothing on their own
- If you need them you need to go and use them using Reflection

Discoverable Plugin Options

Thursday, October 9, 2025 3:46 PM

Assumptions

- Operators May be present in different assemblies created by different companies
- They may be present in one or more classes
- They will always be static methods that will take 2 int and return an int.
- One operation may have multiple names
    - Eg. Plus, sum, add
- These plugins will not be referred during compile time
- They will be read from a plugins folder sitting next to executable file.

How will it work?

When application loads, it should do the following

1. Search for 'plugins' folder sitting next to executable
2. Search all dll files in the plugins folder
3. Load all dll files as assembly
4. Iterate over all types defined in assembly
5. Iterate through all the static methods in the assembly and search for methods that meets my requirement
6. Add the matching methods directly into my calculator by calling Add Operator