# Unit Tests

Checking your code

# Topics

- Unit Tests
- Assert Class
- Running tests
- Debugging tests

# Unit Tests

Checking your code

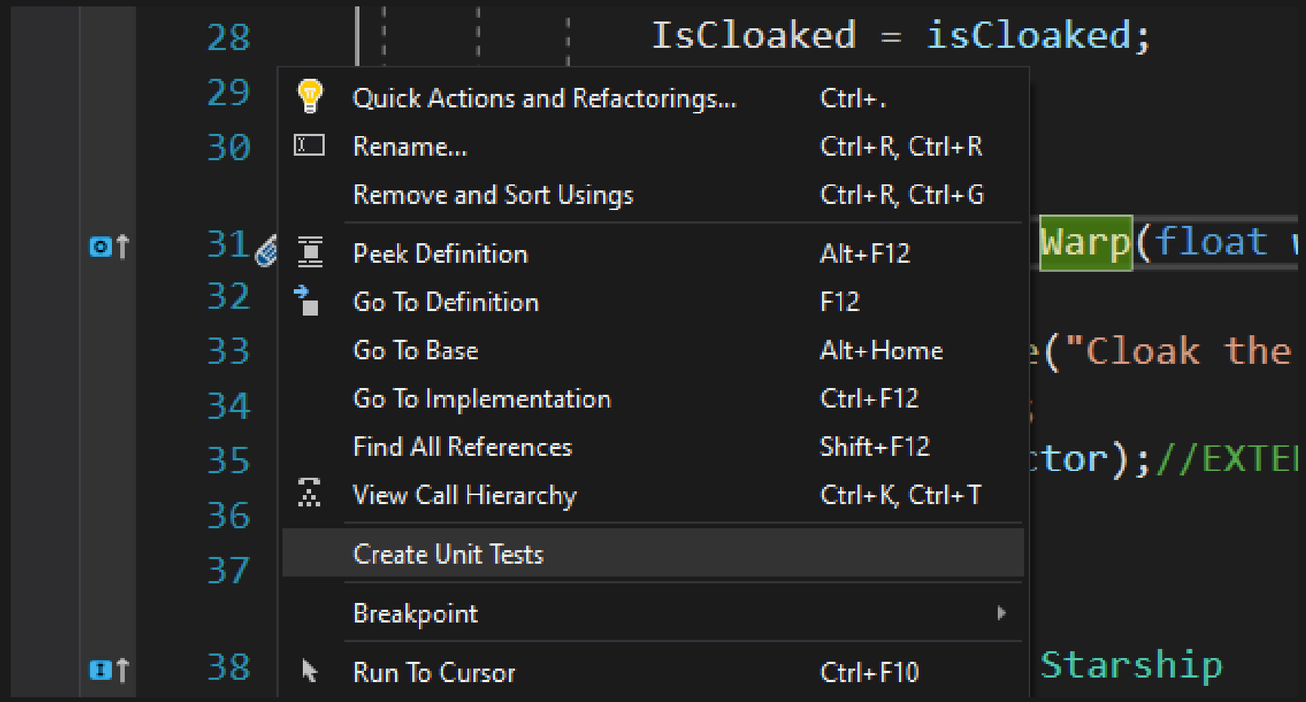
# Unit Tests

- Unit tests are a way for you to verify that your code is working correctly.
- Unit tests are small methods that you write in a separate project.
- They are usually executed in an automated system.

# Unit Test Project

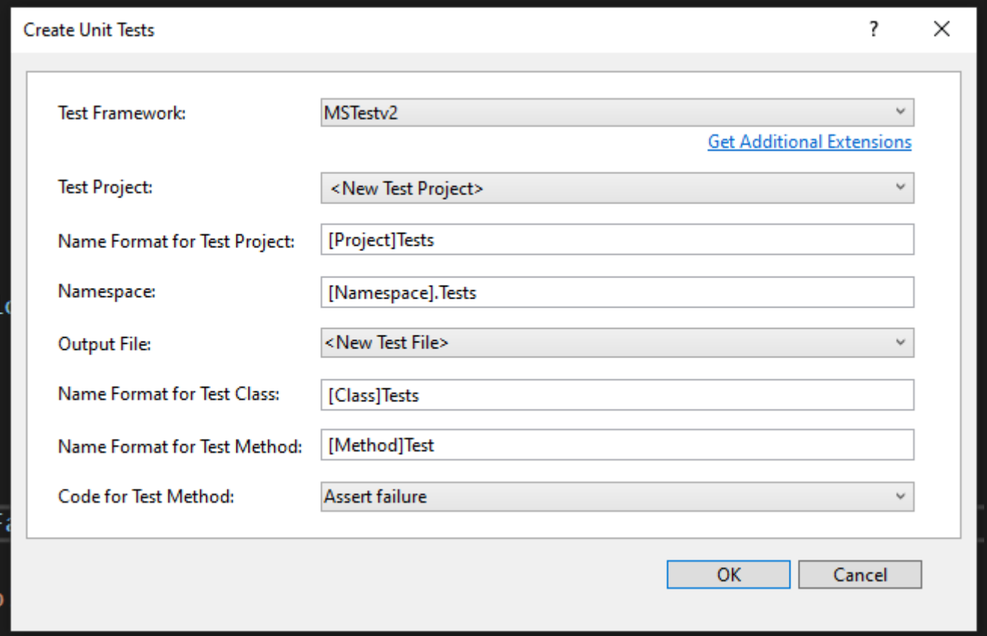
# Add a Unit Test Project

- To add a test for a method, right-click the method and select Create Unit Tests

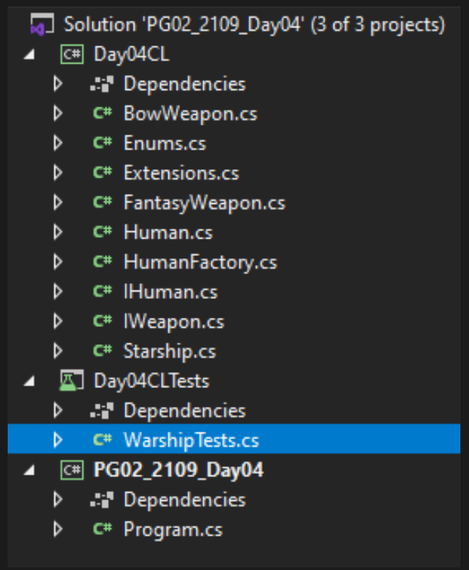


# Add a Unit Test Project

- Click OK on the dialog to add a new project and unit test method.




Your solution after clicking ok



# Assert Class

# Assert Class

- Use the Assert class to test your code.
- There are several methods you can use to test what your code is doing.
- The approach is to do something with your code then use Assert methods to verify that the expected behavior happened.
  - For instance, if your code was supposed to set a property, test if the property was correctly set.

# Testing your code

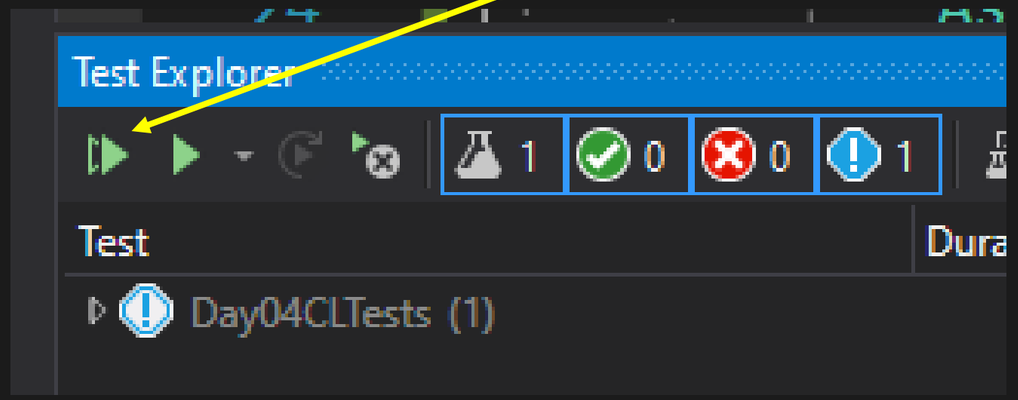
- We want to test that calling the Warp method on a Warship object will set the IsCloaked to true.
  1. Create an instance of Warship.
  2. Call Warp on the instance.
  3. Test using the Assert class that the IsCloaked property is true.

```
public void WarpTest()
{
    //1. Create an instance of Warship
    Warship testShip = new Warship("Test Ship", false);

    //2. Call Warp on the instance
    testShip.Warp(9.6F);

    //3. Test using the Assert class that IsCloaked is true
    Assert.IsTrue(testShip.IsCloaked);
}
```

# Running Tests

# Run your test

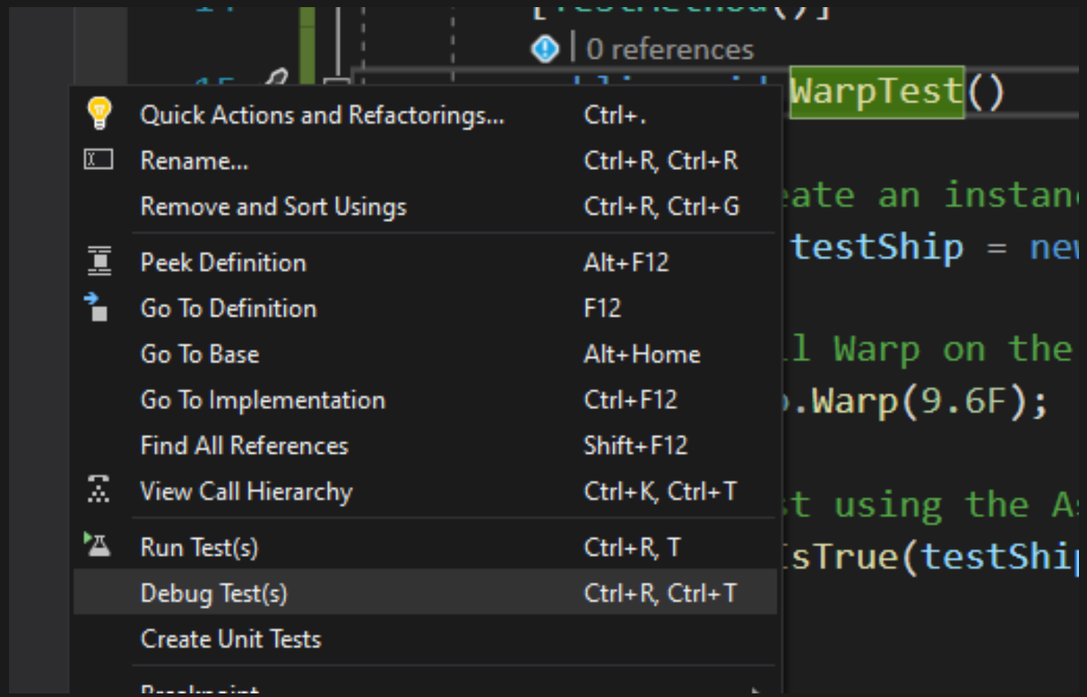
- Open the Test Explorer at the bottom of the Visual Studio window.
  - NOTE: if it is not there, you'll find it in the Test menu.
- To run your test, click the Run All Tests play button.



# Debugging Tests

# Debug your test

- If your code isn't working, you might need to debug the test.
  1. Put a breakpoint in your unit test.
  2. Right-click the test method and select Debug Test(s)



# Challenges

# #1 Unit Test the FantasyWeapon properties

- Add a unit test for the FantasyWeapon constructor.
- Test the following FantasyWeapon properties:
  - Rarity
  - Level
  - MaxDamage
  - Cost

```
public void WarpTest()
{
    //1. Create an instance of Warship
    Warship testShip = new Warship("Test Ship", false);

    //2. Call Warp on the instance
    testShip.Warp(9.6F);

    //3. Test using the Assert class that IsCloaked is true
    Assert.IsTrue(testShip.IsCloaked);
}
```

## LINKS

Add a Unit Test

Test Your Code

## VIDEOS

# #2 Unit Test the BowWeapon properties

- Add a unit test for the BowWeapon constructor.
- Test the following BowWeapon properties:
  - ArrowCapacity
  - ArrowCount

```
public void WarpTest()
{
    //1. Create an instance of Warship
    Warship testShip = new Warship("Test Ship", false);

    //2. Call Warp on the instance
    testShip.Warp(9.6F);

    //3. Test using the Assert class that IsCloaked is true
    Assert.IsTrue(testShip.IsCloaked);
}
```

## LINKS

Add a Unit Test

Test Your Code

## VIDEOS

# #3 Unit Test the DoDamage method

- Add a unit test for the DoDamage method.
- Test that the method returns a value between 0 and MaxDamage on the weapon.

```
public void WarpTest()
{
    //1. Create an instance of Warship
    Warship testShip = new Warship("Test Ship", false);

    //2. Call Warp on the instance
    testShip.Warp(9.6F);

    //3. Test using the Assert class that IsCloaked is true
    Assert.IsTrue(testShip.IsCloaked);
}
```

## LINKS

Add a Unit Test

Test Your Code

## VIDEOS