Testing code

Learn. To Change.

Module Objectives

- What is Software Testing?
- Why is Software Testing Important?
- What are the benefits of Software Testing?

Unit testing

It is a form of Functional testing

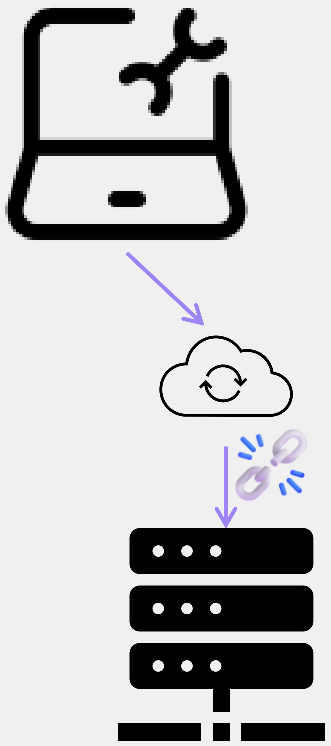
- Tests individual units (Classes, Methods…) in isolation
  Typically by developers, to eliminate defect

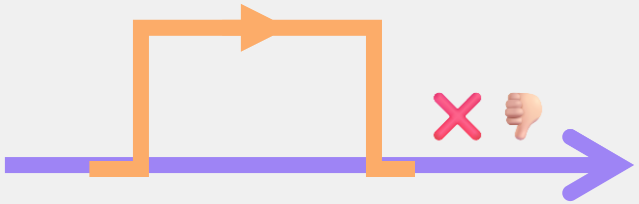
When to write Unit tests

- Write tests while coding
- Before Integration testing
- When a bug is fixed

Common causes of tests failure

- **Bug Fixes Affecting Other Features**
  Example: Fixing a security issue accidentally breaks user authentication
- **New Features Breaking Existing Functionality**
  Example: Adding a new payment method causes old transactions to fail.
- **Code Refactoring Gone Wrong**
  Example: Renaming a function but missing some places where it was used.
- **Dependency Updates**
  Example: Updating a third-party library changes its behavior and breaks integration.
- **Merging Code Changes**
  Example: Two developers working on different parts introduce conflicting logic.





F.I.R.S.T.

Unit tests must be…

- **F**ast
- **I**ndependent
- **R**epeatable
- **S**elf-validating
- **T**imely

- **Robert Martin, *Clean* Code, 2009**

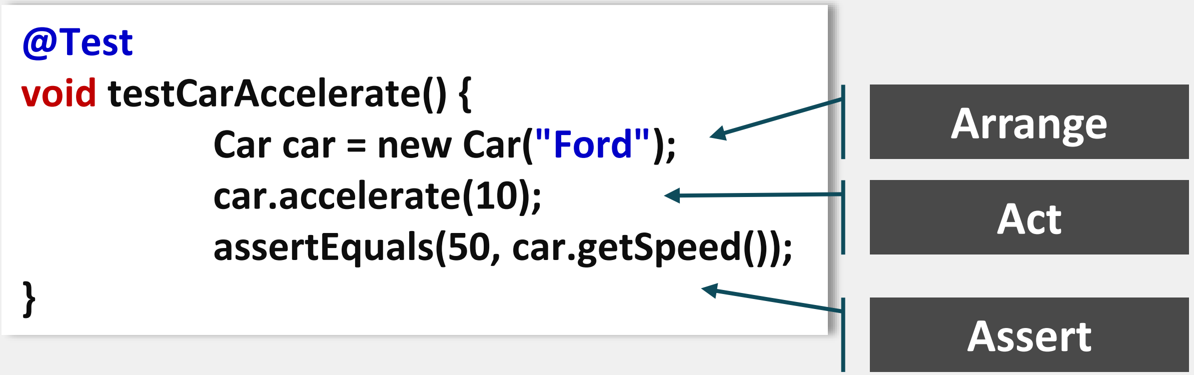
Readable tests: Coding by intention

4 phases:

1. **Setup / Arrange**: set up the initial state for the test.
2. **Exercise / Act:** perform the action under test.
3. **Verify / Assert:** determine and verify the outcome.
4. **Clean-up:** clean up the state created.

Each phase should be:

- Clearly expressed, including your expected outcomes
- Well documented



```
@Test
void testCarAccelerate() {
        Car car = new Car("Ford");
        car.accelerate(10);
        assertEquals(50, car.getSpeed());
}
```

Right B.I.C.E.P

Right: Are the results correct, accurate, expected?

B: Are all the **b**oundary conditions correct?

I: Can you check the **i**nverse relationships?

C: Can you ***c***rosscheck results using other means?

E: Can you force **e**rror conditions to happen?

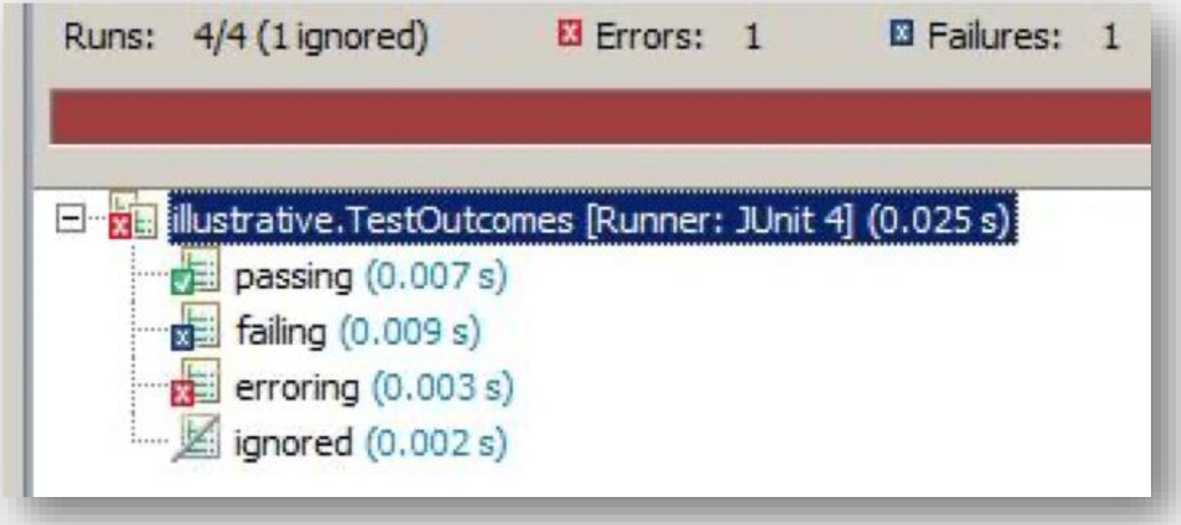
P: Are **p**erformance characteristics within bounds?

Test statuses

Passing: ultimately, all our tests must pass

Failing: in TDD, always start with a test which fails

Erroring: test neither passes nor fails
Something has gone wrong like a run-time error



```
Runs:   4/4 (1 ignored)        Errors:   1        Failures:   1

illustrative.TestOutcomes [Runner: JUnit 4] (0.025 s)
    passing (0.007 s)
    failing (0.009 s)
    erroring (0.003 s)
    ignored (0.002 s)
```

Manual Testing vs Automated Testing

Manual Testing	Automated Testing
Only certain people can execute the tests	Anyone can execute the test
Difficult to consistently repeat tests	Perfect for regression testing
Manual inspections can be error prone and aren’t scalable	Series of contiguous testing, where the results of one test rely on the other
Doesn’t aggregate, indicate how much code was exercised, or integrate with other tools (e.g. build processes)	The build test cycle is increased

Unit vs Component vs Integration

Unit Testing	Component Testing	Integration Testing
Ensures all of the features within the Unit (class) are correct	Similar to unit testing but with a higher level of integration between units - **Tests individual components**	Involves the testing of **two or more integrated components**
Dependent / interfacing units are typically replaced by stubs, simulators or trusted components	Units within a component are tested as together real objects	
Often uses tools that allow component mocking / simulation	Dependent components can be mocked	

LABS

"Unit Testing" lab

+

"Readable Tests" lab

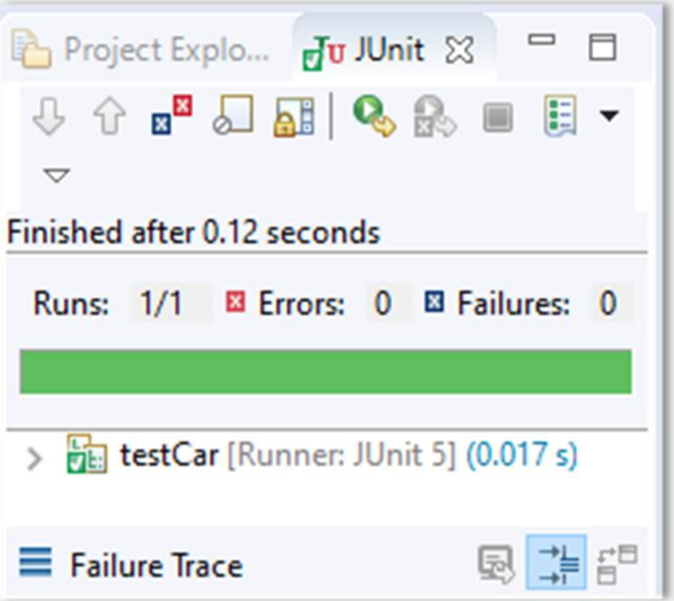
Refer to the following slides for a review of Unit testing methods.

How to create a test?

- New > Other > JUnit > JUnit test case

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

class testCar {
        @Test
        void testCarAccelerate() {
                Car car = new Car("Ford");
                car.accelerate(10);
                assertEquals(50, car.getSpeed());
        }
}
```



```
Finished after 0.12 seconds

Runs:  1/1    Errors:  0    Failures:  0

> testCar [Runner: JUnit 5] (0.017 s)
```

JUnit @Before and @After annotations

method to run *before* each `@Test`

method to run *after* each `@Test`

```java
class testCar {
  Car car;

    @BeforeEach
    public void setUp() {
        car = new Car("Ford");
    }

    @AfterEach
    public void tearDown() {
        car = null;
    }

    @Test
    void testCarAccelerate() {
        System.out.println("@test");
        car.accelerate(10);
        assertEquals(50, car.getSpeed());
    }
}
```

```
assertTrue()
assertNull()
fail()
```

Testing Expected Exceptions with JUnit

Junit 4

```
@Test(expected = IllegalArgumentException.class)
public void testConstrction() {

   new Employee("Fred", -1);

}
```

Junit 5

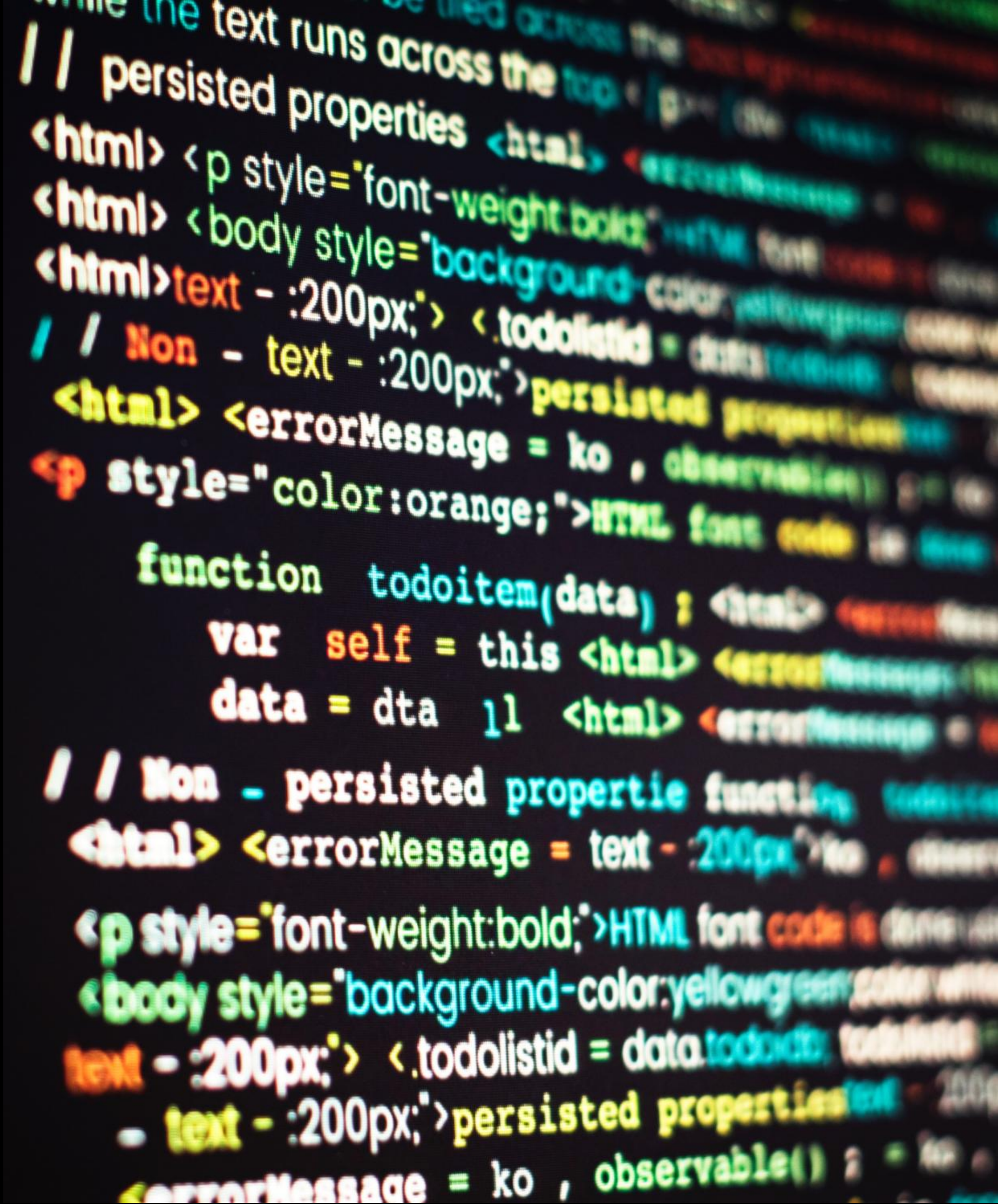
```
@Test
void testWithdrawWithInsufficientFunds() {
   Account acc = new Account("B10", 100, "Bob");

   assertThrows(InsufficientFundsException.class, () -> {
        acc.withdraw(1000);
   });

}
```

Unit testing method for .NET

Learn. To Change.



Write test code

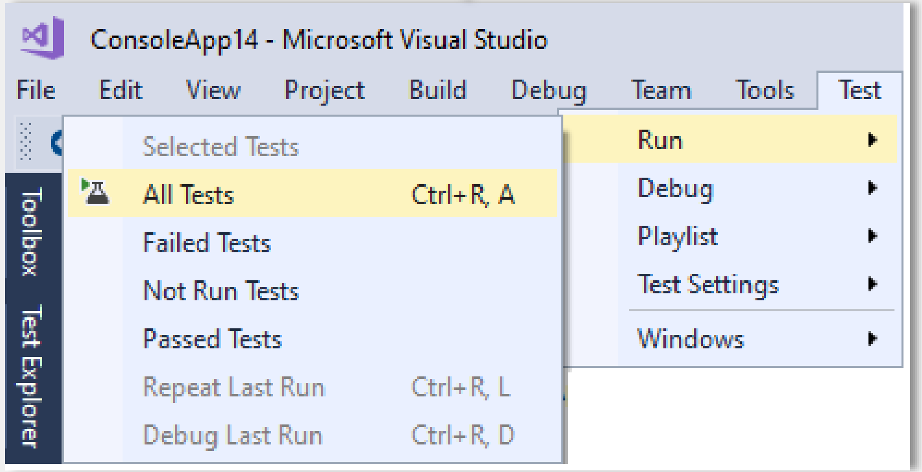
```
[TestClass()]
public class CarTests {
   Car car;

   [TestInitialize]
   public void SetUp() {
       car = new Car("Ford");
   }
   [TestCleanup]
   public void TearDown() {
       car = null;
   }
   [TestMethod()]
   public void accelerateTest() {
       car.accelerate(10);

       Assert.AreEqual(50, car.Speed);
   }
}
```

Create MS-Test project

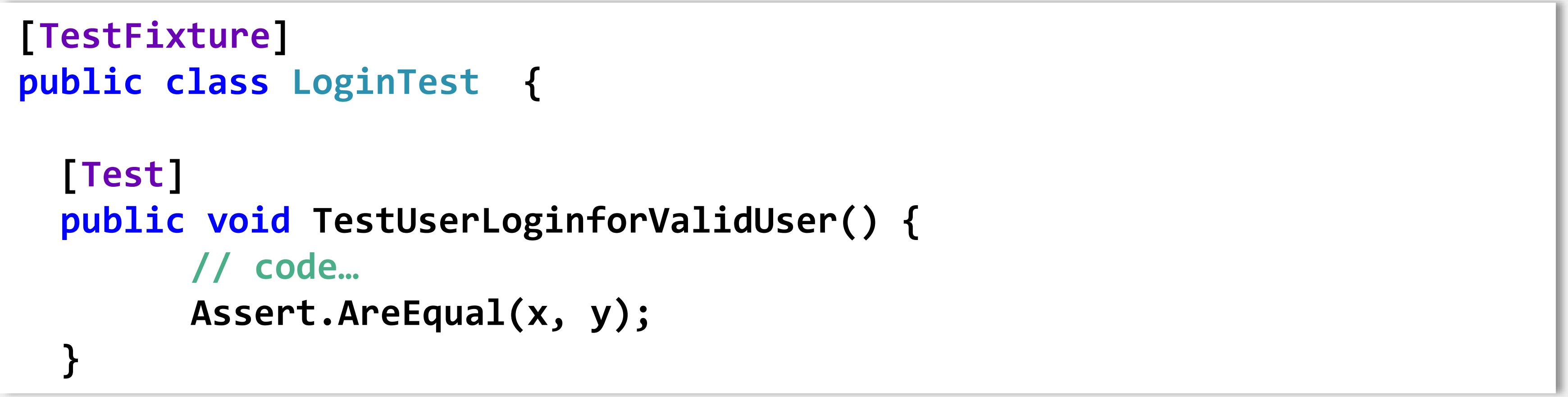
Run



Testing Expected Exceptions with MS-Test

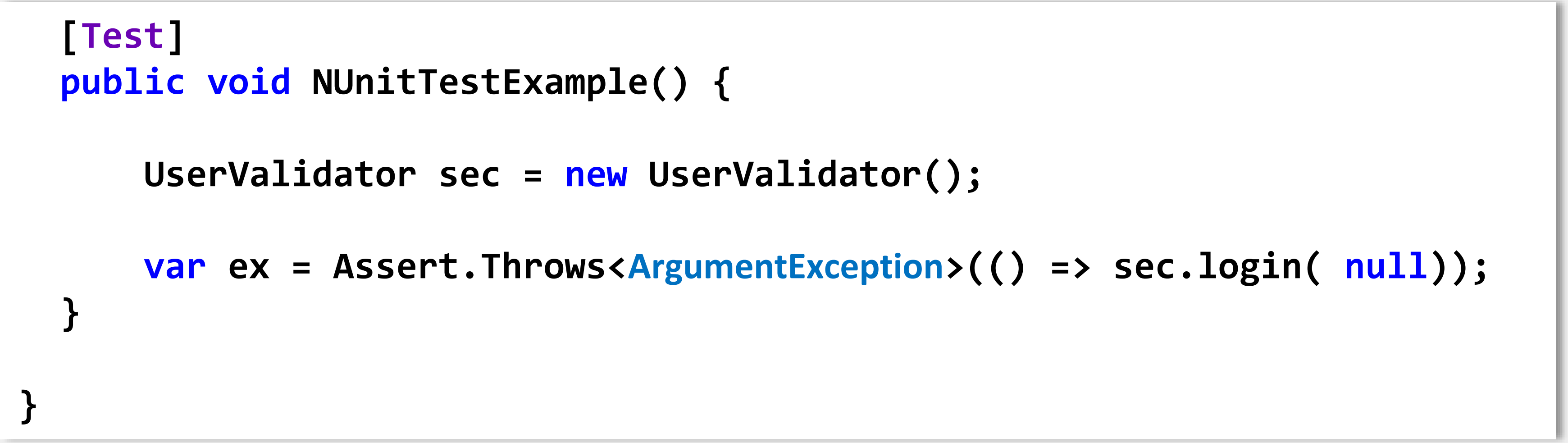
```
[TestMethod]
[ExpectedException(typeof(ArgumentException))]
public void TestNullUsername()
{
        LogonInfo login = new Login(null, "password123");
}
```

NUnit Example – C#



```
[TestFixture]
public class LoginTest  {

  [Test]
  public void TestUserLoginforValidUser() {
        // code…
        Assert.AreEqual(x, y);
  }
```



```
  [Test]
  public void NUnitTestExample() {

      UserValidator sec = new UserValidator();

      var ex = Assert.Throws<ArgumentException>(() => sec.login( null));
  }

}
```

Lab

Please do the exercise for this chapter

Learn. To Change.