Agenda

1. Good qualities of unit tests
2. Mocking
   Types of Doubles.
3. How to autowire when youive
   more than one implementation.

What type of inputs will you test?-

1. Edge cases
2. Negative scenarios -> wrong inputs.
3. Normal cases [yes].

```
add (int a, int b)
```

add (2,2)
add (10,20)
add (0,2)
add (10,-2)
add (0,0)

add(-1,-20)
add(MAX_VALUE, 20)
add(—, —)

Qualities of unit tests

1. Fast and Simple.

3A's to follow..

```
test()
    Arrange -> data preparation
    Act     -> call the code that you want to test
    Assert  -> compare 'expected' and 'actual'.
```

```
additionTest()
    int a=20, int b= 30                          -> Arrange
    Calculator calc = new Calculator()

    int res = calc.add(a,b)                      -> Act

    if( res!=50)                                 -> Assert.
        ———
        ———
```

2. isolation

The output expected shouldn't be depending on any other tests that we're running.

Test cases shouldn't be dependent on each other.

Calculator

```
int result = 0, 30 20

add(int a, int b)
    result = result + a + b;
    return result;
```

CalculatorTest

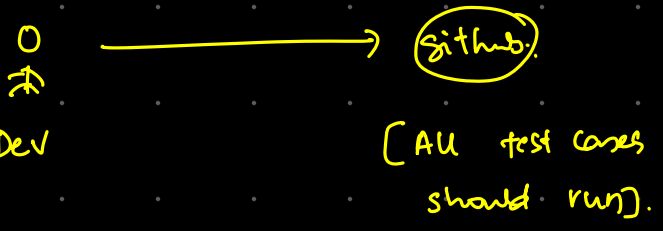
```
Calculator calc = new Calculator()

t1()
    a = 10, b = 20
    result = calc.add(a,b)
    if (result == 30)   ≈ pass
        ==

t2()
    a = 20, b = 30
    result = calc.add(a,b)
    if (result == 50)   ≈ fail.
        ==
```

3. Repeatable

No matter how many times you run the test cases, the test cases should produce the same result for same set of inputs.

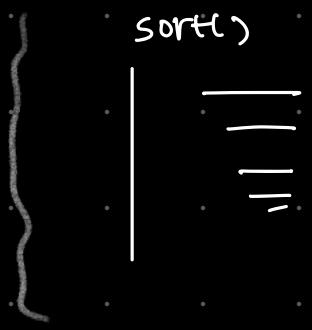
4. Self-checking

* Don't always need human interaction
* They should run automatically.



5. Test behaviour, not implementation

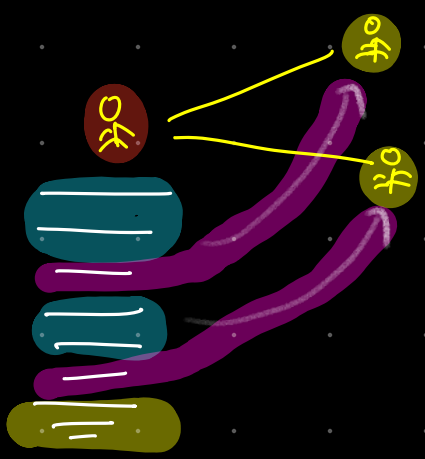
You only check if the function can sort all possible inputs or not, You'll not check the internal logic of sorting.



Implementation can change anytime. Shouldn't result writing test cases again.

Behaviour if it'll it changes, then you'll need to modify the test cases.

Mocking



Assume that external dependencies always work and test the current fn only.

ProductController

ProductService ps;

getProduct(id)

// validate id

// modify id to diff format

Product p = ps.get(modifiedId)

// some changes to product

// return product

This should be mocked.

Product p = new Product("Macbook")

when(ps.get(1)).thenReturn(p)

expectedProduct = controller.getProduct(1);

expectedName.equals("Macbook");

ProductService

ProductServiceTest

An actual object of ProductService will be used. But, a mocked object of repo will be used.

Double

Similar to body double, person who replaces the actor for the stunt.

Types of Double

1. Mock
2. Stub
3. Fake.

When this method is called, you return this or do nothing.

Scenario:

① createProduct → 5

② getProductCount → 5

③ createProduct → 1

④ getProductCount → 6

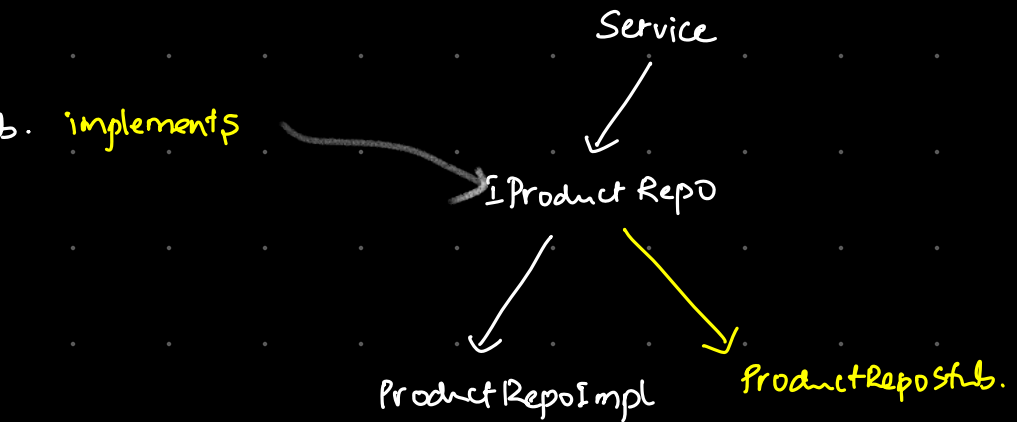
```
Class ProductService.Test
    t1()
        ProductService service = new ProductSerivce();
        ProductRepo repo = Mock(ProductRepo.class)

        when(repo.createProduct()).doNothing()
        when(repo.getProductCount()).thenReturn(5)

        service.createProduct()
        service.createProduct()
        service.createProduct()
        service.createProduct()
        service.createProduct()

        if(service.getProductCount() != 5)
        |
```

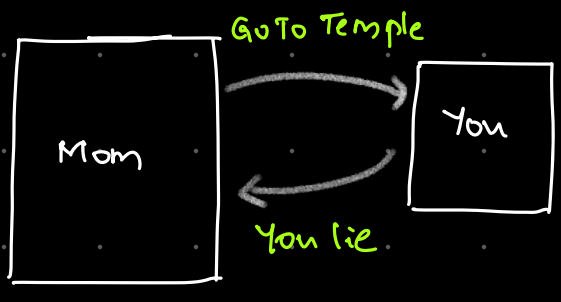
pass

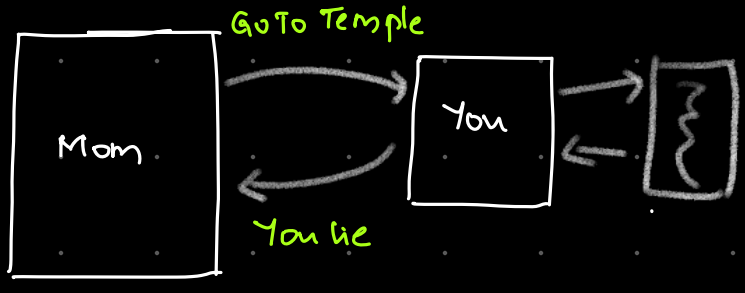
```
service.createProduct()
if (service.getProductCount() != 6)
```
fail.

Stub.

```
class ProductRepoStub. implements
    int count=0;
    createProduct()
        count++;
    getProduct()
        return count;
```

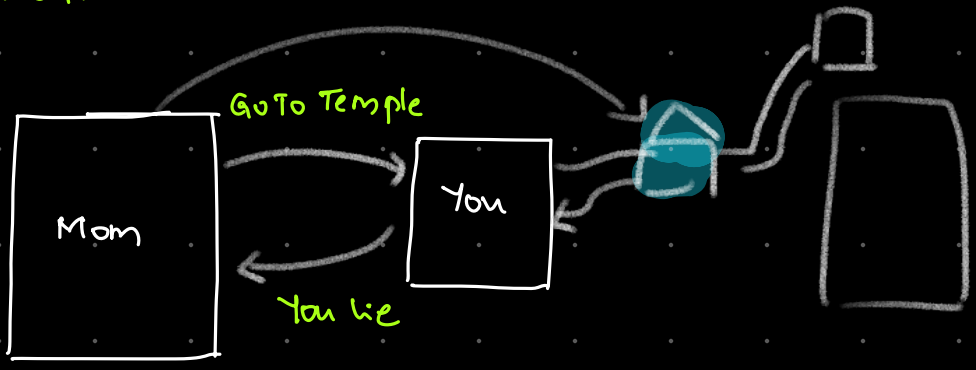






Stub.

Mock.



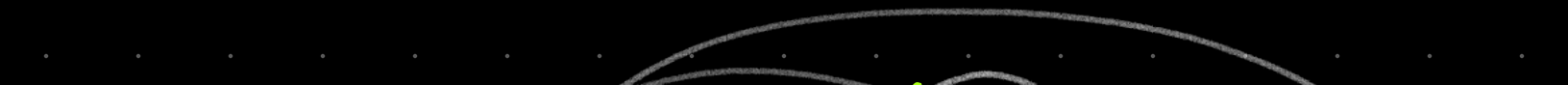
Fake.

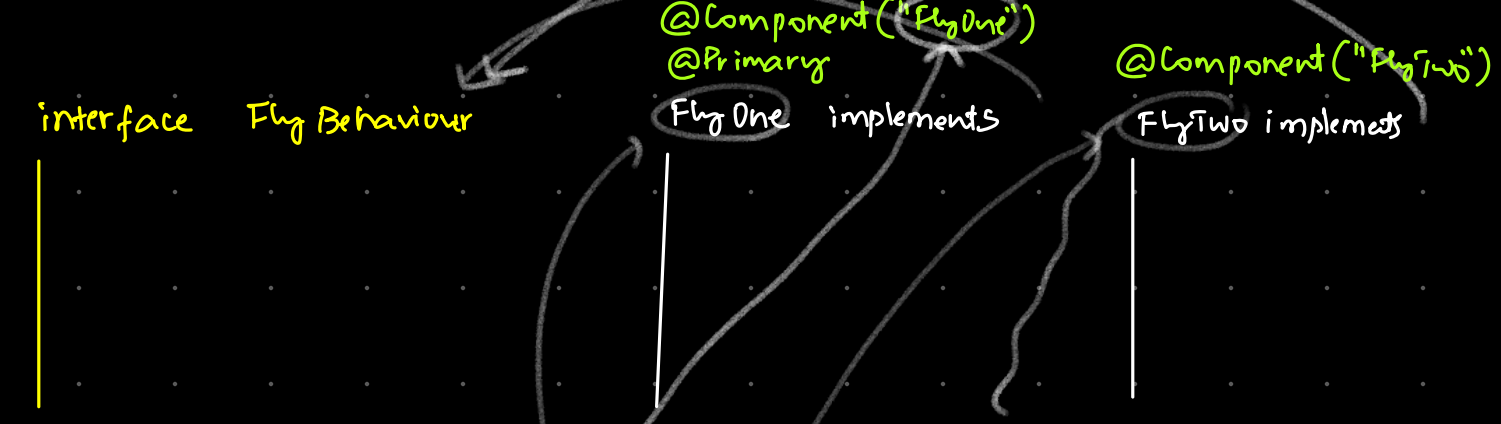
Mock Stub Fake

→ Towards reality.

ProductRepoFake implements IProductRepo.

```
Map<Int, Product> map; int id=1;

createProduct()
    map.put(id, product); id++;

getProduct()
    map.get(id)
```





Bird service

```
@Autowired
@Qualifier("FlyOne")
FlyBehaviour flyBehav;
```

Product

id	name	desc	Cat_id
1	Iphone	—	1
2	Xiome	—	1

Category

id	name	desc
1	Mob	"Phones"

If I delete the category '1', what should happen?

① delete the category, set Cat_id -> null.

② delete the category & products also

③ dont allow the category to delete.

@OneToMany(fetch = ____, cascade = CascadeType.____)