Agenda

1 hr 15 mins

1. Types of Doubles
   - → Mocks
   - → Fakes
   - → Stubs
2. Argument Captors
3. AssertJ → testing json output of APIs becomes more easy
4. Module Ahead Overview
   - → upcoming classes
   - → group project (mentor led)

Types of Doubles

Action Movies ↳ Actors have body double.

↳ instead of actor doing stunts, the body double does those.

```
→ ProductController {
    → ProductService productService;
            ⇑
    getProductById(long id) {
        ps ______ - (id);
    }
}
```

⇒ Instead of real ps being used, a mock of ps gets used while testing.

Doubles: Replacement objects used in place of real objects during testing.

- Mocks
- Stubs
- Fakes.

Mocks

we simulate an object to replicate behaviours of a dependency.

```
Product ControllerTest {
    @MockBean
    ProductService productService;

    when(productService.getById())
        .then(return 123 = );

}
```

```
MockProductService {
        getById() {
            return 123;
        }
}
```

⇒ In mocking, I just hardcode instead of having a concrete double

⇒ independently hardcoding behaviour of every method → there is no state of the obj.

```
@ Test {

    product. set Age( __ )
    product. set Price( __ )
    ProductService. set Config(c)
    ProductService. serve(Product)
}
}
```

2. STUB

⇒ A concrete class created to simulate the behaviour of a real variant.

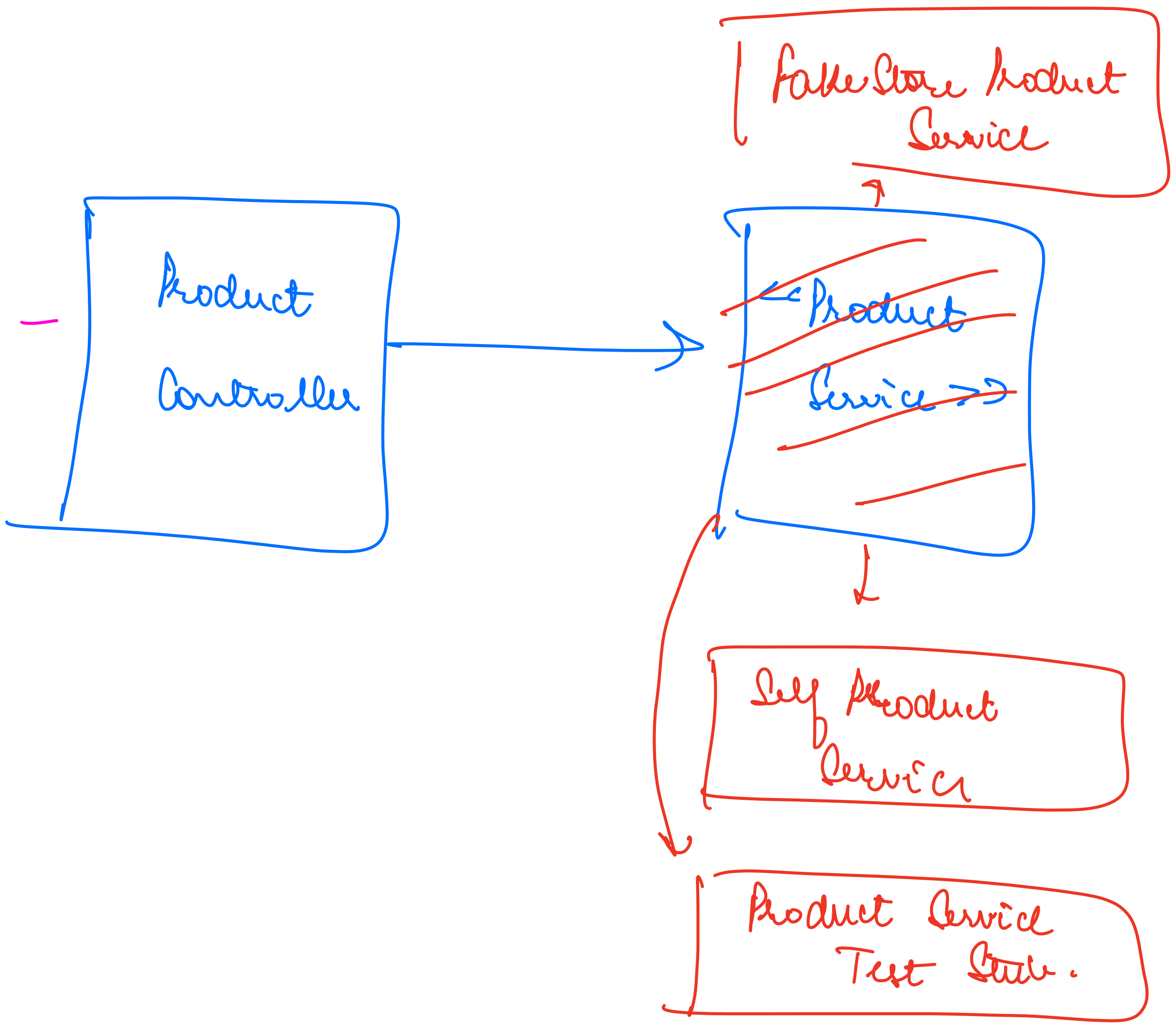
```
class ProductServiceStub implements ProductService {
    List<Product> products;

    getProductById() {
        for (Product p: products)
            if p.id = id:
                return p.
    }
}
```

@ Product ControllerTest {

Product Controller (Product Service Stub)

?



in a mock, you can't maintain a state:

if $\boxed{\text{a.xyz()}}$ → $\boxed{\text{return this}}$

```
a.abc()

a.def()

a.xyz()

assert( __ , __ )
```

```
void createAndFetchSameProductShouldReturn() {
    null Product p = ps.getLatestProduct();
    productService.createProduct( ___ );
    product p = productService.getLatestProduct();
}
```

in mocking ,

```
Product Controller {
        Product Service ps;

}
```

```
Product Controller Test {
PC   pc = new PC();

}
```

when	then
getById(1)	new Product(1, Naman)
Same ( ) .	new Product(1, Naman)

no state b/w multiple calls of product service is maintained.

```
User Service {
    createUser();

}
```

```
UserServiceTest {

    void  shouldNotCreateNewUserIfUserAlreadyExists() {
        ~~userService.createUser(naman@scaler.com)~~
        int countUsers = userService.getUserCount()
        userService.createUser(naman@scaler.)
        int count2 = userService.getCount()
    }
}
```

These should be same

```
when(userService.getCount()).thenReturn(1);
```

User Service can't be mocked → dynamism

User Service can't be same → Not testing in isolation.

→ User Service should be a dummy user Service that I create with basic dynamism

Stub Class

```
Class Dummy User Service  implements  User Service {
        Set < String >  user Emails;

        create User (email) {
                User Emails. add (email).
        }
        get Count () {
                user Emails. count()
        }
}


User Controller Test {
        User Service =  new  Dummy User Service()
        User Controller  =  new UC ( userService )



}
```

```
⇒ User Service {
        cache;

        save() {
            ⇒ cache.save()
              db.save()
        }
}

        getID(long id) {
    ⇐     {  if (cache.hasId(id)) {
                  return cache.get(id) }
             }
             } return db.get(id)
        }
```

Cache may have old data.

```
=  UserController {
           userService.

   }
```

```
DummyCache {
    get ( ) {
    }
    set ( ) {
    }
}
```

Fakes

$$\begin{tikzcd}
 & & \text{hardcode behaviour of an entity} & & \\
\text{Mocks} \arrow[d] & < & \text{Stubs} \arrow[u] & < & \text{Fakes} \arrow[d] \\
\text{hardcode a method} & & & & \text{hardcode a major system}
\end{tikzcd}$$

Break till 10:40

$\boxed{113}$ [all 7 APIs] $\Rightarrow$ Monday

2
1
3

= Group prop + [Sunday EOD]

5 people

4 services