TDD

Test Driven Development

Tomasz Sadza 2022-10-03, 2022-10-04, **2022-10-05**

TDD

Dzień 3

:(

```
class ShoppingCard {
  public function any() {
    return $receipt =
      (new Paypal())->getReceipt();
   }
}

tests/

test('payment call', function() {

  $receipt = (new ShoppingCard())->any();
  …

})
```

```
src/

class PayPal
```

Dummy

Dummy class as a Substitute for External Service

```
class ShoppingCard {
  public function __construct(
    private Gateway $paymentGateway
  ) { }
}

tests/

test('payment call', function() {

  $s = new ShoppingCard(new FakePayments);
})
```

```
src/

interface Gateway {
  public function getReceipt();
}

class PayPal implements Gateway

tests/

class FakePayments implements Gateway
```

Dummy

PHPUnit can create a Dummy (Test Double)

```
class ShoppingCard {
  public function __construct(
    private Gateway $paymentGateway
  ) { }
}

tests/

test('payment call', function() {

  $s = new ShoppingCard(
    $this->createMock(Gateway::class)
  ); …
})
```

```
src/

interface Gateway {
  public function getReceipt();
}

class PayPal implements Gateway
```

Stub

When Substitute has to only answer the question

```
test('payment call', function() {

  $gateway = $this->createMock(Gateway::class)
    ->method('getReceipt')
    ->willReturn('fake receipt id');

  $s = new ShoppingCard($gateway);
  …
})
```

```
src/

interface Gateway {
  public function getReceipt();
}

class PayPal implements Gateway
```

Stub

When Substitute has to only answer the question

```
test('payment call', function() {

  $gateway = new GatewayStub();

  $s = new ShoppingCard($gateway);
  …
})
```

```
src/

interface Gateway {
  public function getReceipt();
}

tests/

class GatewayStub implements Gateway {

  public function getReceipt() { … }
}
```

Mock

When Substitute can be a reason to fail (test has expectations)

```
test('payment call', function() {

  $gateway = new GatewayStub();
  $mailer = $this->createMock(Mailer::class)
    ->expects($this->once())
    ->method('deliver')
    ->with('ok, done');

  $s = new ShoppingCard($gateway, $mailer);
  …
})
```

```
class ShoppingCard {
  public function __construct(
    private Gateway $paymentGateway,
    private Mailer $mailer
  ) { }
  public function submit() {
    …
    $this->mailer->deliver('ok, done');
    …
  }
}

class Mailer { … }
```