Using Mocks

Anna Filina
TESTING AND LEGACY EXPERT
@afilina afilina.com

Summary

Stubs

Mocks

Dummies

Spies

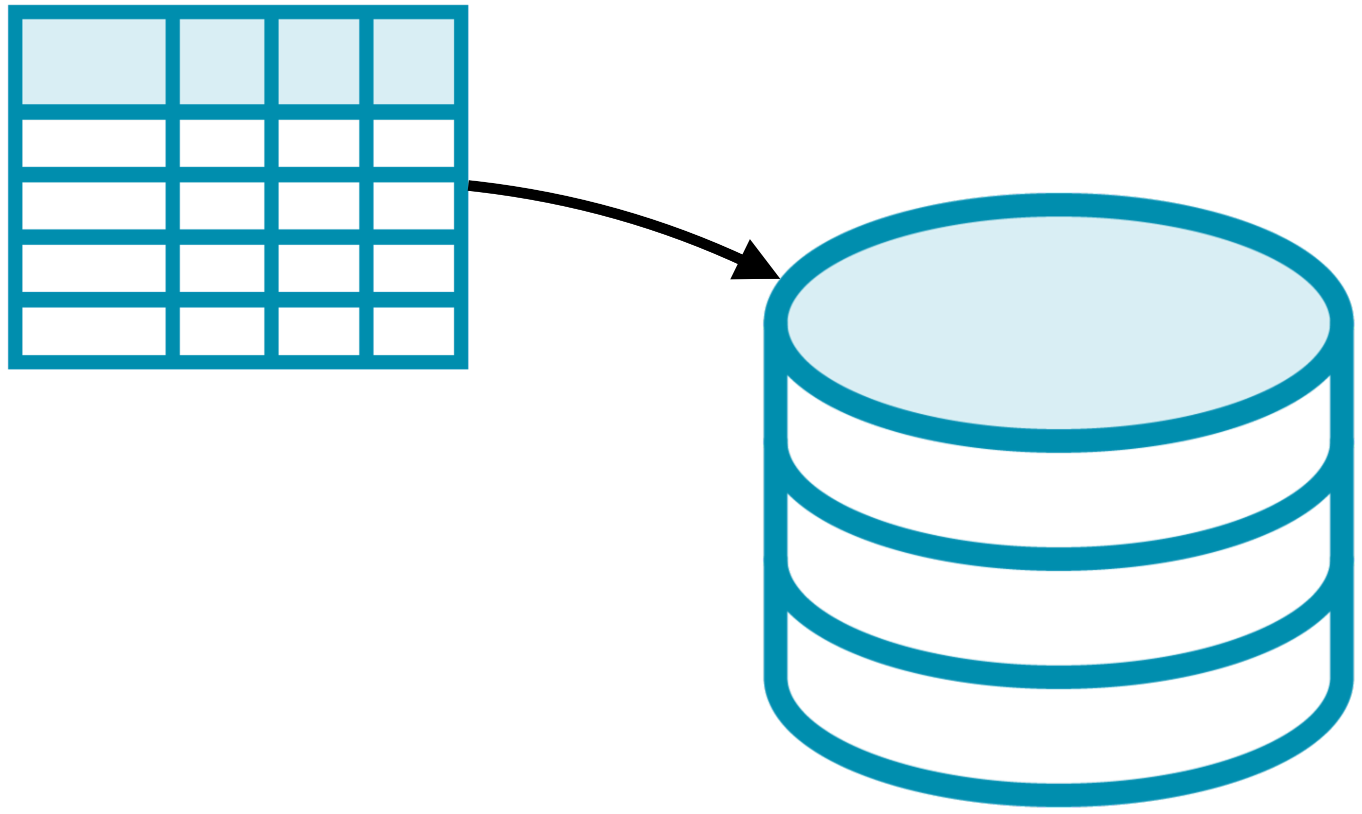
Fakes

Stubs

```php
final class ProductListHandler
{
    public function __construct(
        ProductRepository $productRepository,
        SearchAnalytics $searchAnalytics
    ) {
        $this->productRepository = $productRepository;
        $this->searchAnalytics = $searchAnalytics;
    }
}
```

```php
$connection = new Connection(
    [
        'dbname' => 'pluralsight',
        'user' => 'pluralsight',
        'password' => 'pluralsight',
        'host' => 'mysql.phpunit.local',
        'port' => '3306',
    ],
    new Driver()
);
$config = Setup::createAnnotationMetadataConfiguration(
    [__DIR__ . '/../../src/Entity'],
    true,
    null,
    null,
    false
);
$entityManager = EntityManager::create($connection, $config);
$handler = new ProductListHandler(
    new DoctrineProductRepository($entityManager),
    new FilesystemSearchAnalytics(new Filesystem(), new DateTimeClock())
);
```

```php
public function handle()
{
    $products = $this->productRepository->findProducts();
    $this->searchAnalytics->track(['price' => null, 'name' => null]);

    return $this->createProductsResponse($products);
}
```



```php
$repository = $this->createStub(ProductRepository::class);
$repository
    ->method('findProducts')
    ->willReturn([
        new Product('Concert 1', new Amount(1100)),
        new Product('Concert 2', new Amount(550))
    ]);

$handler = new ProductListHandler($repository, $searchAnalytics);
```

```php
$response = $handler->handle();

self::assertEquals(
    [
        [
            'name' => 'Concert 1',
            'price' => '11.00'
        ],
        [
            'name' => 'Concert 2',
            'price' => '5.50'
        ]
    ],
    json_decode($response->getBody()->getContents(), true)
);
```

Mocks

```php
public function handle()
{
    $products = $this->productRepository->findProducts();
    $this->searchAnalytics->track(['price' => null, 'name' => null]);

    return $this->createProductsResponse($products);
}
```

```php
$searchAnalytics = $this->createMock(SearchAnalytics::class);
$searchAnalytics
    ->expects($this->once())
    ->method('track')
    ->with(['price' => null, 'name' => null]);

$handler = new ProductListHandler($repository, $searchAnalytics);

$response = $handler->handle();
```

```php
public function handle()
{
    $products = $this->productRepository->findProducts();
    // $this->searchAnalytics->track(['price' => null, 'name' => null]);

    return $this->createProductsResponse($products);
}
```

```
Expectation failed for method name is "track" when invoked 1 time(s).
Method was expected to be called 1 times, actually called 0 times.
```

Dummies

```php
final class DoctrineProductRepository implements ProductRepository
{
    public function __construct(EntityManagerInterface $entityManager)
    {
        $this->entityManager = $entityManager;
    }

    //...

    public function getTableName(): string
    {
        return 'product';
    }
}
```

```php
$entityManager = $this->createMock(EntityManagerInterface::class);
$repository = new DoctrineProductRepository($entityManager);

self::assertEquals(
    'product',
    $repository->getTableName()
);
```

```php
public function __construct(
    FtpUploader $ftpUploader,
    CsvParser $csvParser,
    ProductRepository $productRepository,
    ImageResizer $imageResizer,
    Mailer $mailer
) {}
```

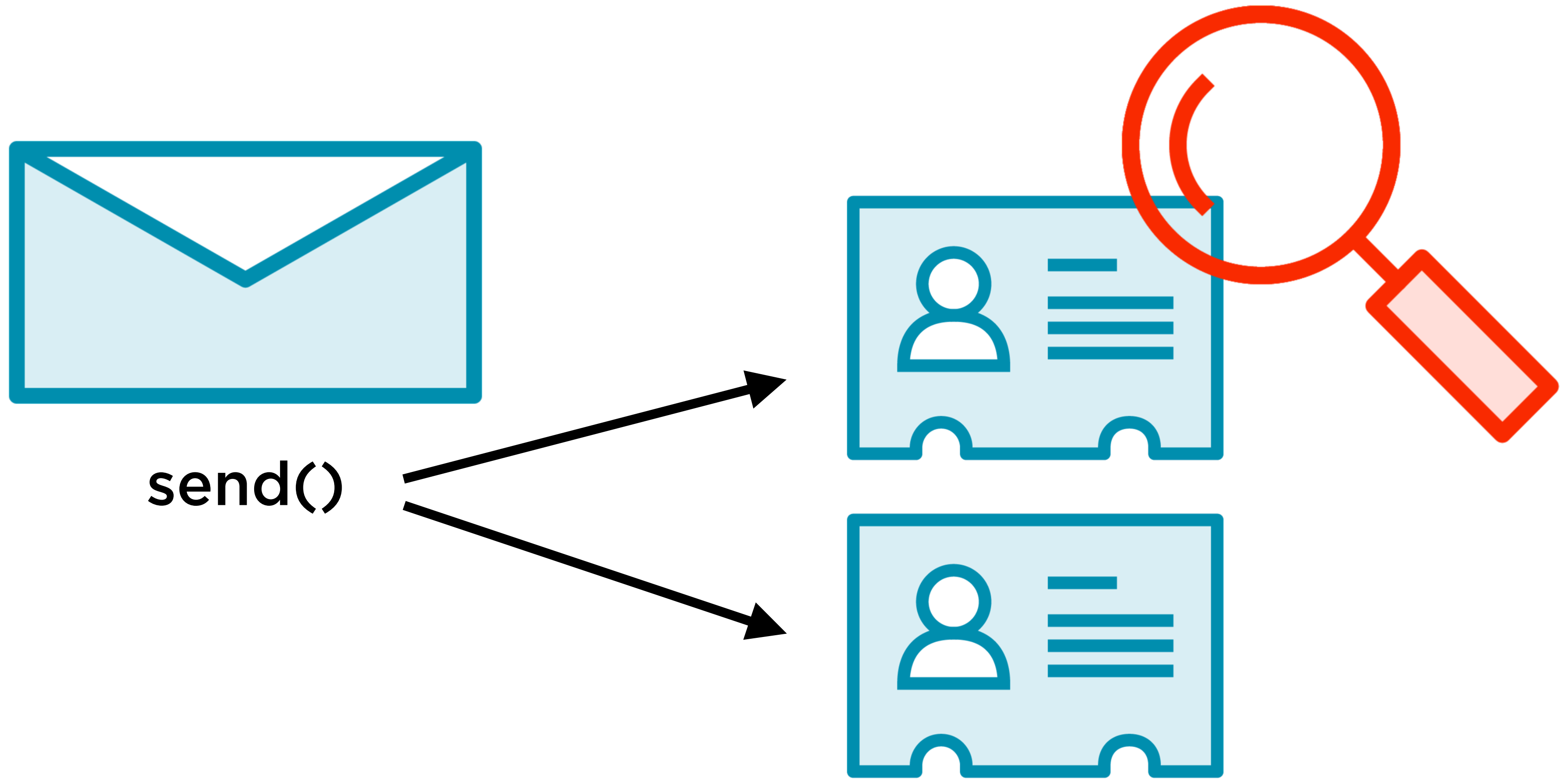
PSR-15

```php
final class ProductListHandler implements RequestHandlerInterface
{
    public function handle(ServerRequestInterface $request): ResponseInterface
    {
    }
}
```

Spies

```php
public function notifyUsers(Product $product): void
{
    $this->notifyUsersInArea($product);
    $this->notifyUsersInterestedInArtist($product);
}
```

```php
$userRepository = $this->createStub(UserRepository::class);
$userRepository
    ->method('findUsersInArea')
    ->willReturn([
        ['email' => 'a@example.com'],
        ['email' => 'b@example.com']
    ]);
$userRepository
    ->method('findUsersInterestedInArtist')
    ->willReturn([
        ['email' => 'a@example.com']
    ]);
```

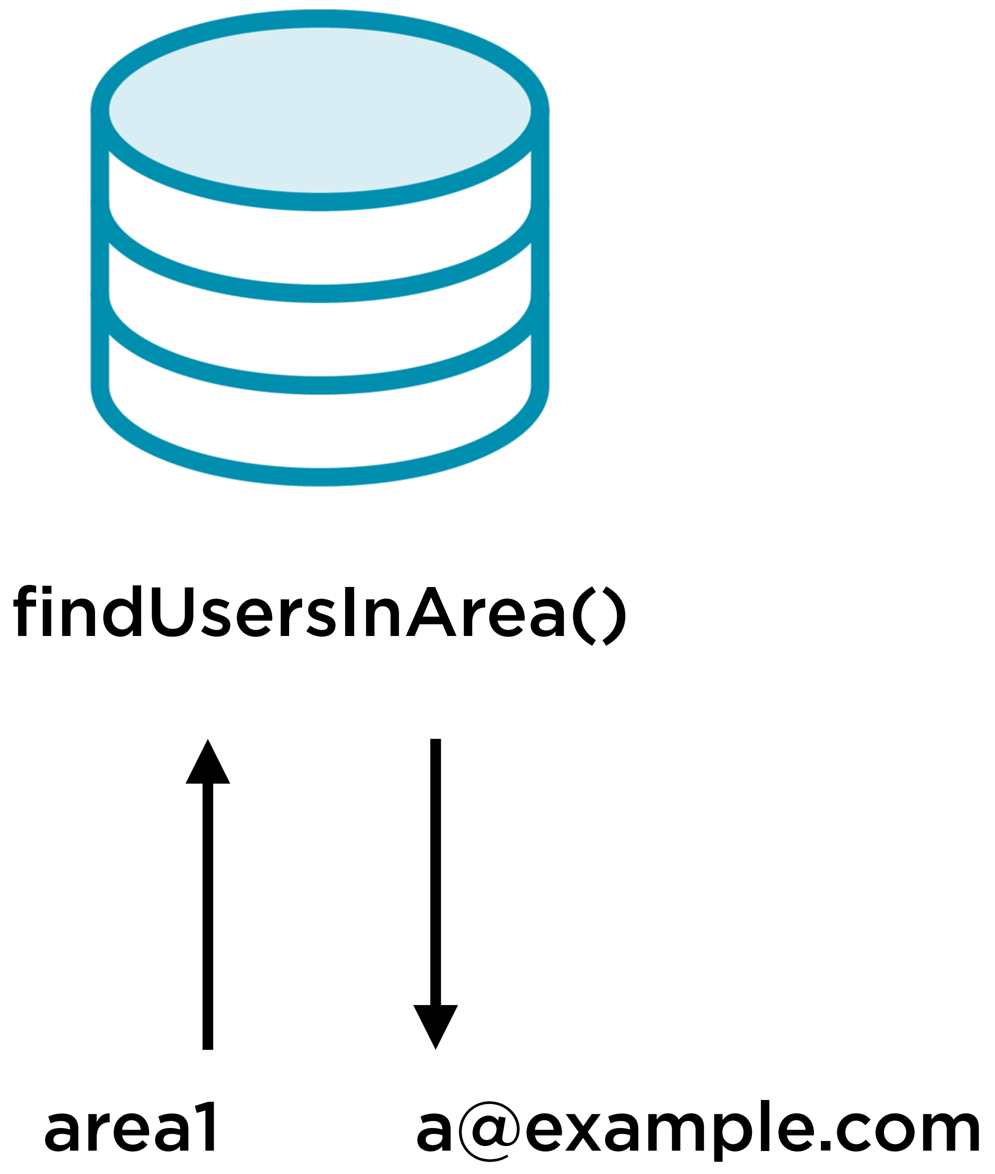

send()

```php
$mailer = $this->createMailerSpy();
$notification = new NewProductNotification($mailer, $userRepository);
```

```php
private function createMailerSpy(): MailerInterface
{
    return new class implements MailerInterface {
        /** @var string[] */
        private array $sendInvocationRecipients;

        public function send(RawMessage $message, Envelope $envelope = null): void
        {
            $this->sendInvocationRecipients[] = array_map(static function (Address $address) {
                return $address->getAddress();
            }, $envelope->getRecipients());
        }

        /** @var string[] */
        public function getSendInvocationRecipients(): array
        {
            return $this->sendInvocationRecipients;
        }
    };
}
```

```php
$notification->notifyUsers(new Product('Concert Name', new Amount(1000)));

$invocationRecipients = $mailer->getSendInvocationRecipients();
$duplicateRecipients = array_intersect(...$invocationRecipients);

self::assertCount(
    0,
    $duplicateRecipients
);
```

Fakes

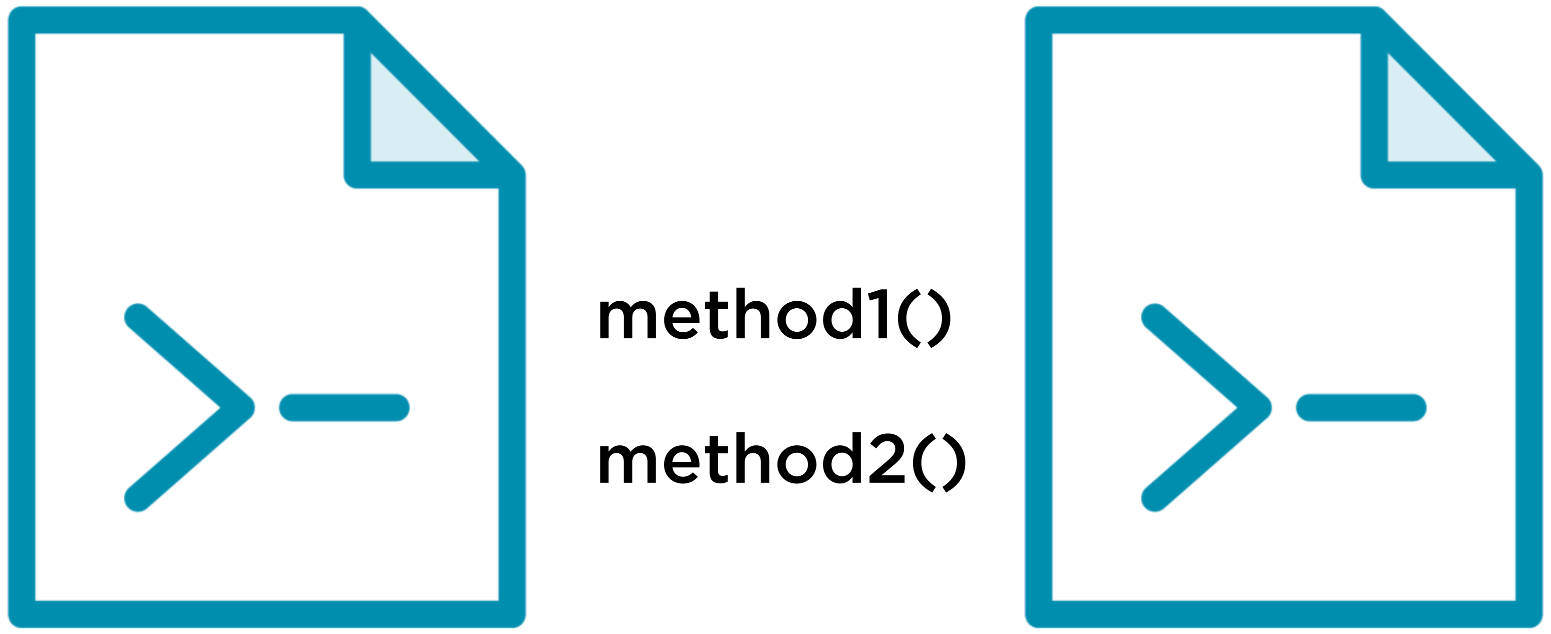

findUsersInArea()
area1
a@example.com

```php
private function createFakeUserRepository(): UserRepository
{
    return new class implements UserRepository {
        public function findUsersInArea(string $area): array
        {
            switch ($area) {
                case 'area1':
                    return [['email' => 'a@example.com']];
            }
            return [];
        }

        public function findUsersInterestedInArtist(string $artistName): array
        {
            return [];
        }
    };
}
```

```php
public function __construct(
    ProductRepository $productRepository,
    SearchAnalytics $searchAnalytics
) { }
```

What Not to Mock

```php
$subject = $this->createPartialMock(MyClass::class, ['method1']);
$subject->method('method1')
    ->willReturn('Xyz');

$subject->method2();
```


method1()
method2()

method1()

method2()

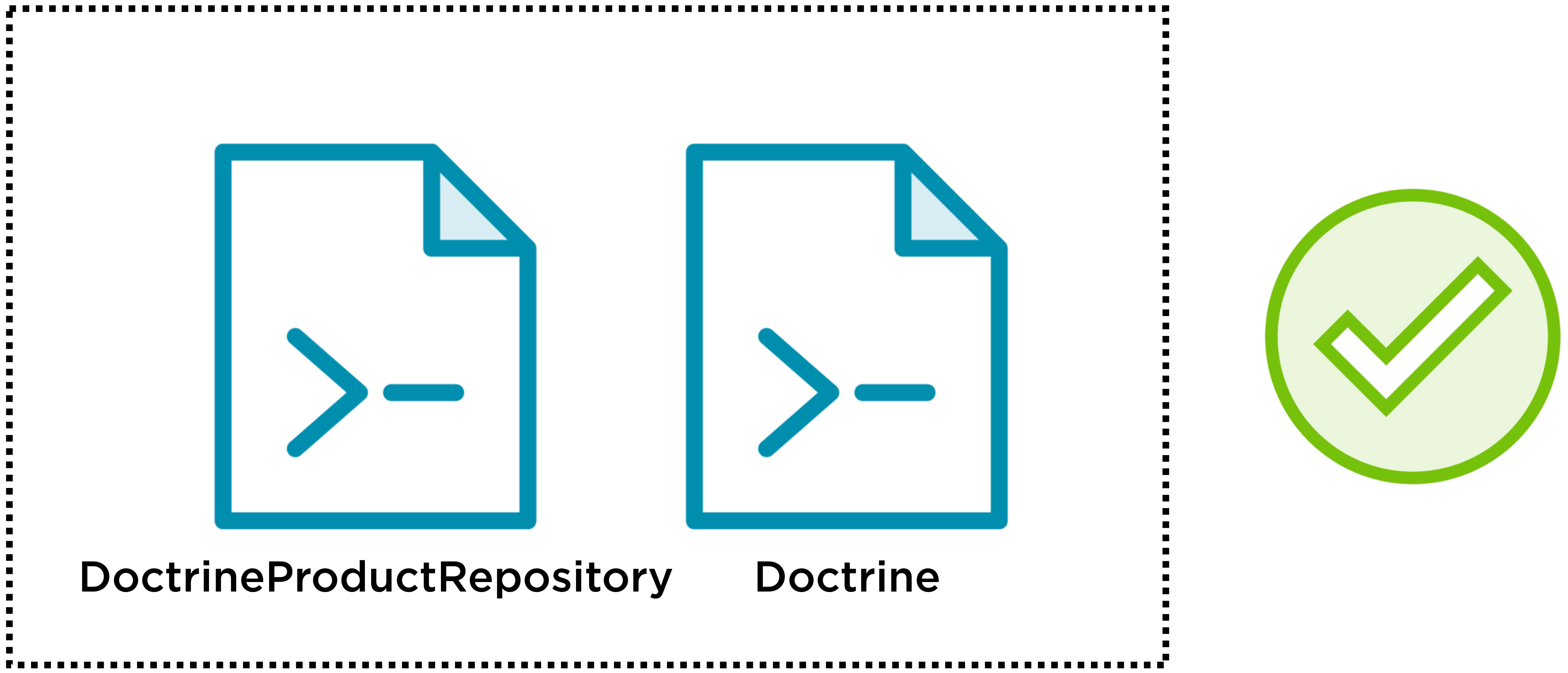
```
public function a()
{
    // do a
    $this->b();
}

private function b()
{
    // do b
}
```

```php
final class ProductListHandler extends AbstractProductHandler
```

```php
final class ProductListHandler
{
    public function __construct(ResponseFormatter $responseFormatter) {}

    public function handle()
    {
        //...
        return $this->responseFormatter->createProductsResponse($products);
    }
}
```

```php
$this->entityManager
    ->createQueryBuilder()
    ->select('product', 'discount')
    ->from(\App\Entity\Product::class, 'product')
    ->getQuery()
    ->execute();
```

```php
public function findProducts(): void
{
    $query = $this->createMock(AbstractQuery::class);
    $query->expects($this->once())
        ->method('execute')
        ->willReturn([$product1, $product2]);

    $queryBuilder = $this->createMock(QueryBuilder::class);
    $queryBuilder->expects($this->once())
        ->method('select')
        ->with('product', 'discount')
        ->willReturnSelf();
    $queryBuilder->expects($this->once())
        ->method('from')
        ->with(Product::class, 'product')
        ->willReturnSelf();
    $queryBuilder->expects($this->once())
        ->method('getQuery')
        ->willReturn($query);
    //...
}
```


DoctrineProductRepository
Doctrine

```php
$transaction = new Transaction(
    new Amount(200),
    Transaction::TYPE_REFUND,
    new AccountNumber('1020AB')
);
```

Up Next:
Measuring Code Coverage