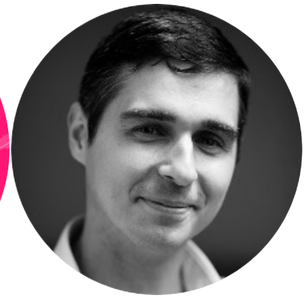
Refactoring Code and Creating a Framework



Andrejs Doronins
Software Developer in Test

Test automation is programming, not testing.

Treat test code seriously.

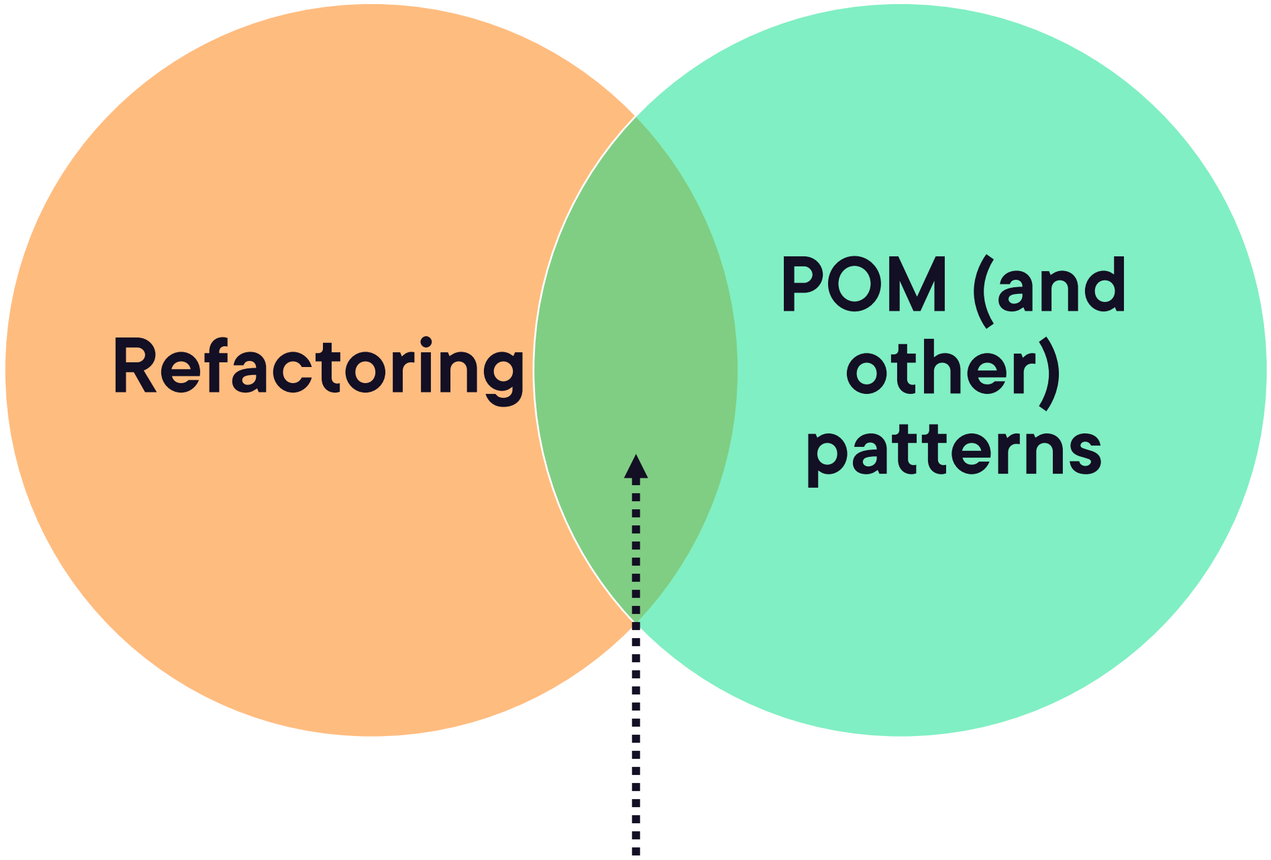
Overview

Refactor:

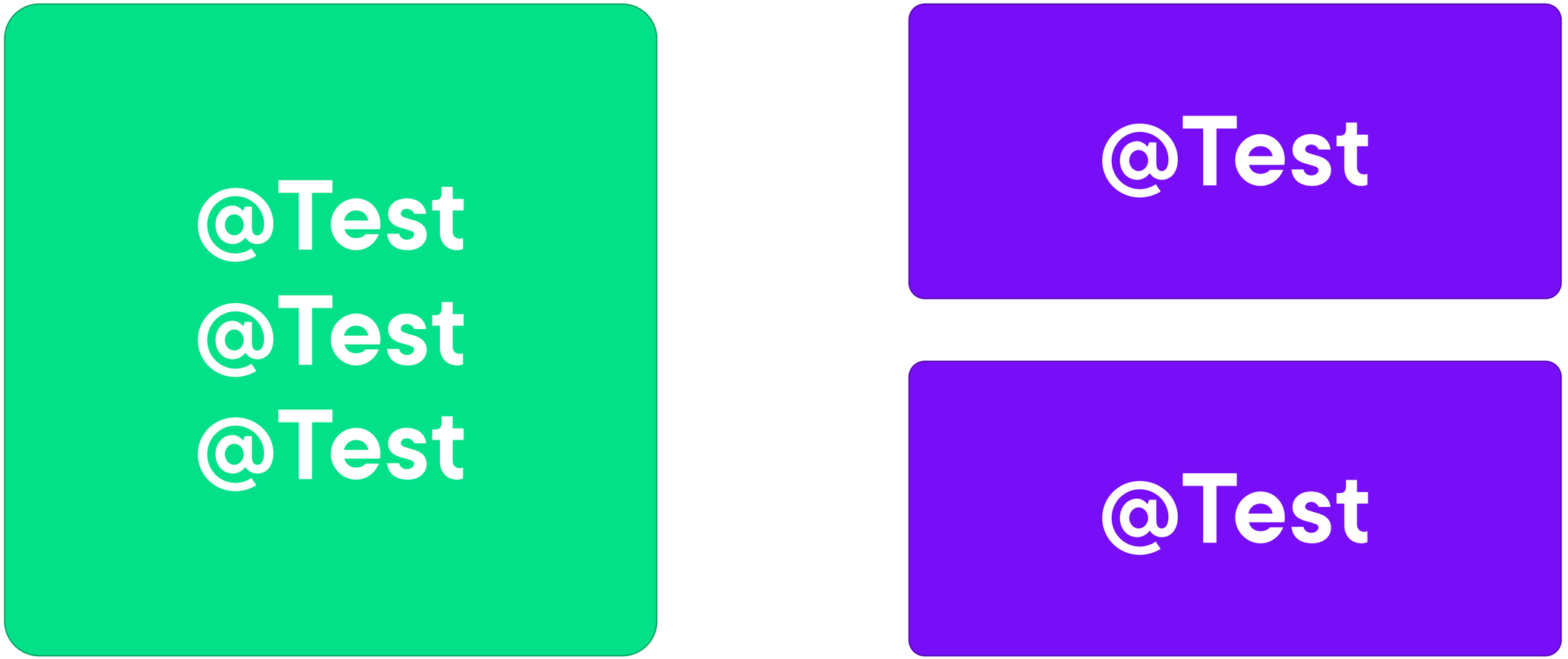
- Object creation
- Setup and cleanup

Framework:

- Page Object Model (POM) pattern



Successful Automation Framework


@Test
@Test
@Test
@Test
@Test

@Setup

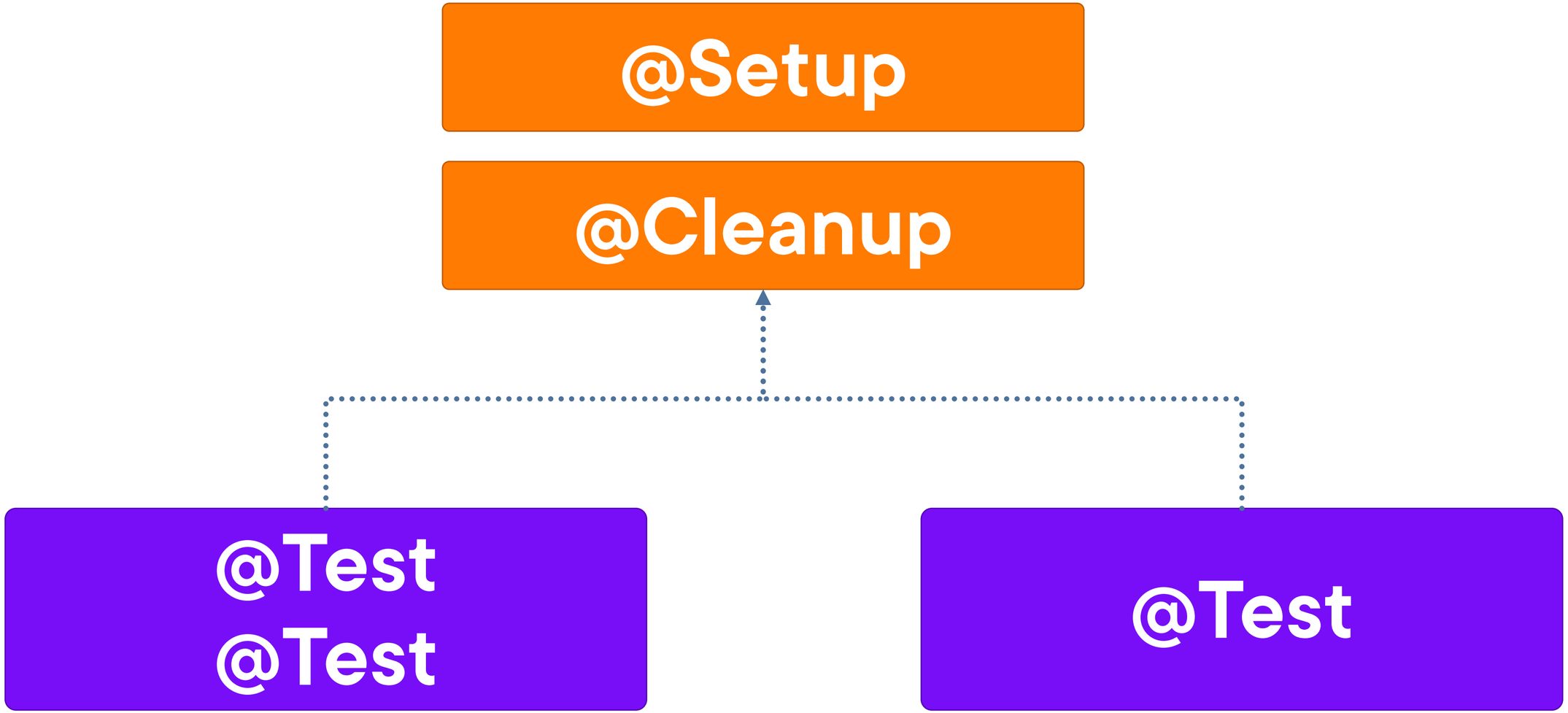
@Test
@Test

@Cleanup

@Setup

@Test

@Cleanup


@Setup
@Cleanup
@Test
@Test
@Test

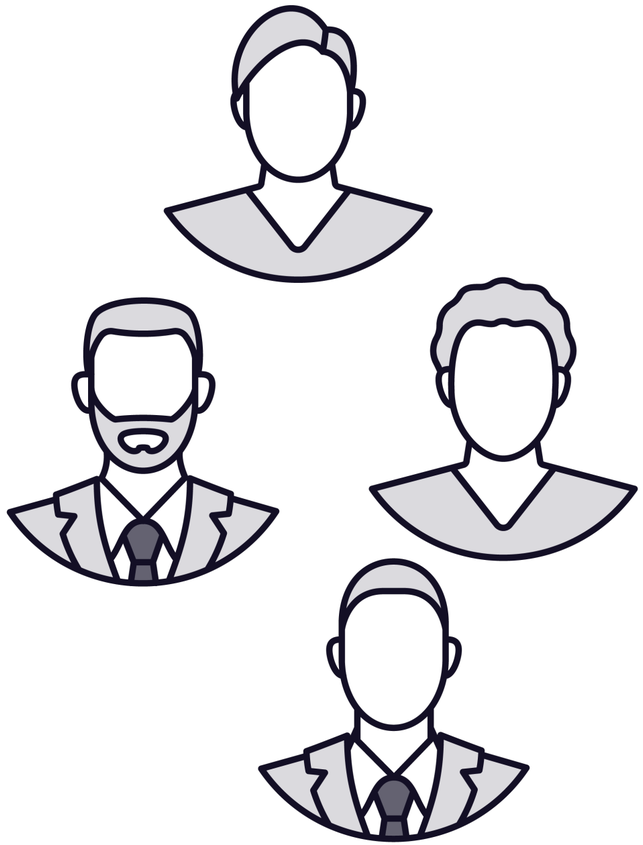
Design Pattern

Repeatable way of organizing code to make it more maintainable.

A blueprint.

Design Pattern

A guideline, not a strict rule that must be followed down to every line



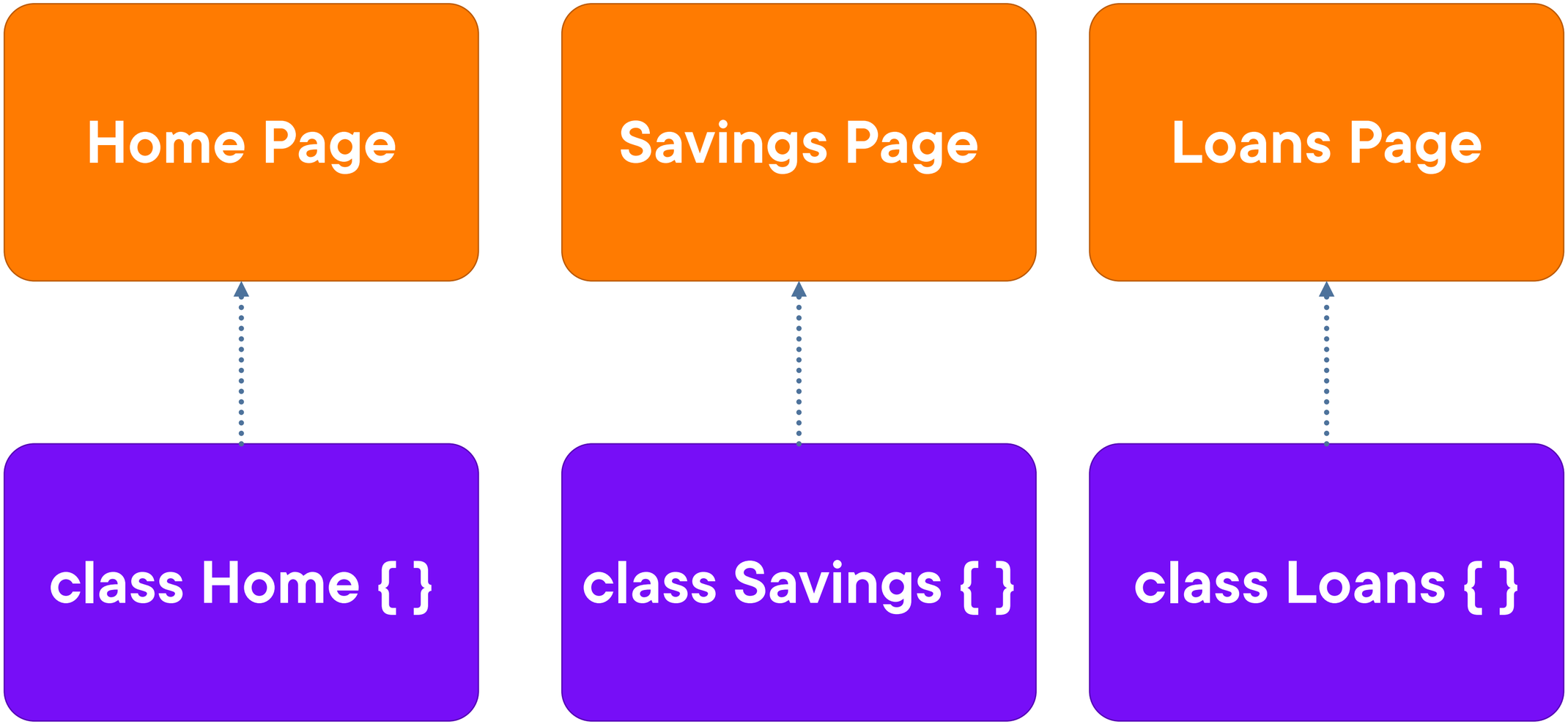
GoF Design Patterns

- Singleton, Strategy, Factory...

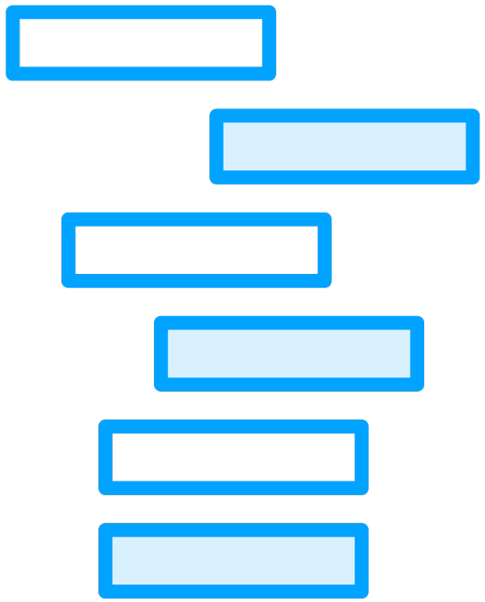
POM - Page Object Model

In an e-shop:

- I found a bug on the shopping cart page
- Testing a new feature on the checkout page
- I haven't tested the personal profile page


Home Page
Savings Page
Loans Page
class Home { }
class Savings { }
class Loans { }

POM Levels of Abstraction



Minimalist

- Hide Selenium locators only

Half-way

- Hide some of Selenium API

Full

- Hide all Selenium API

Exhaustive

- Hide all Selenium and Assertion APIs

Abstraction Level 1

Test

```
@Test
public void test() {
  WebElement email =
        driver.findElement(By.id("email"));

}
```

Page

```
class SomePage {



}
```

Abstraction Level 1

Test

```
@Test
public void test() {
  WebElement email = page.email();

}
```

Page

```
class SomePage {

public WebElement email() {

   return driver.findElement(By.id("email"));

 }

}
```

Abstraction Level 2

Test

```
@Test
public void test() {
  WebElement email = page.email();
  email.sendKeys("my@email.com");

}
```

Page

```
class SomePage {


public WebElement email() {

   return driver.findElement(By.id("email"));

 }

}
```

Abstraction Level 2

Test

```
@Test
public void test() {
  page.fillEmail("my@email.com");
}
```

Page

```
class SomePage {


public void fillEmail(String email) {

   return driver.findElement(By.id("email"))
                .sendKeys(email);

 }

}
```

Abstraction Level 4

Test

```
@Test
public void test() {
  // do things

  assertTrue(box.isDisplayed());
}
```

Page

```
class SomePage {


}
```

Abstraction Level 4

Test

```
@Test
public void test() {
  // do things

  page.verifyBoxIsDisplayed();
}
```

Page

```
class SomePage {


public void verifyBoxIsDisplayed() {

   assertTrue(box.isDisplayed());
 }

}
```

DRY - Don’t Repeat Yourself

Refactor duplicate code

Refactoring Duplicate Code

Home

```
class Home {

        clickHomeNavLink();
        clickSavingsNavLink();
        clickLoansNavLink();

        // other methods relevant
        // to Home page only

}
```

Header

```
class Navigation {









}
```

Refactoring Duplicate Code

Home

```
class Home {




        // other methods relevant
        // to Home page only

}
```

Header

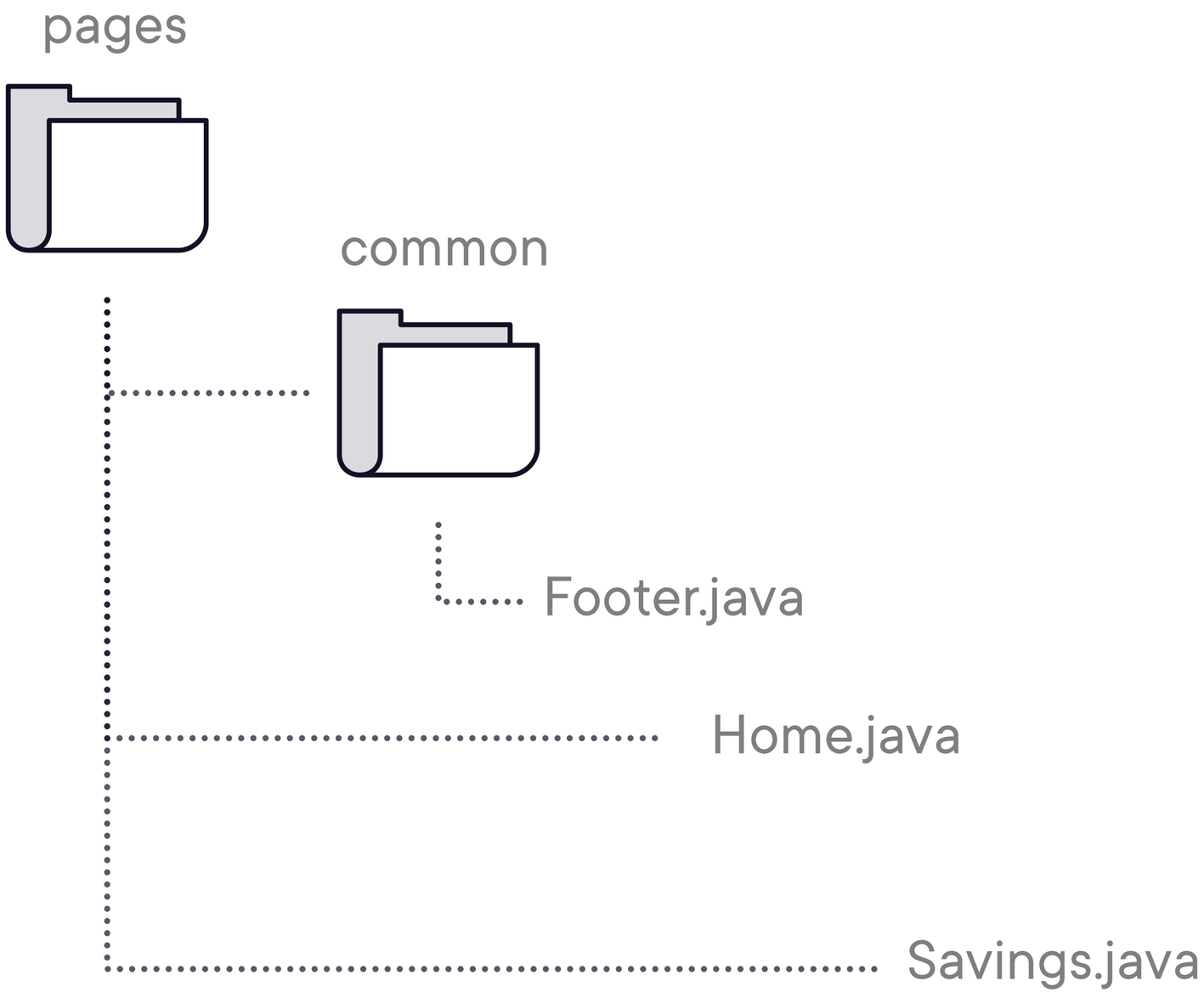
```
class Navigation {


        clickHomeNavLink();
        clickSavingsNavLink();
        clickLoansNavLink();


}
```

Component Object Model?

Widget Object Model?


pages
common
Footer.java
Home.java
Savings.java

Summary

Refactored duplication:

- Object creation to factories
- Util methods to util classes
- Setup, cleanup
- Test class hierarchy

POM pattern and its levels of abstraction