Conférence

15ème édition de la

Journée Française des Tests Logiciels

13 juin 2023

Beffroi de Montrouge

Automatisation:

Récolter le fruit de votre design pattern

Yassine, IBN EL HASSAN

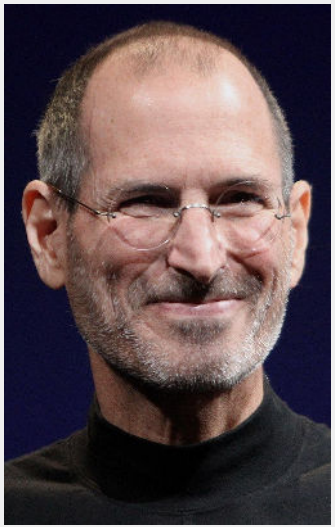
Ismail, Ktami

CFTL
Comité Français des Tests Logiciels

WerinGroup
QUALITY ENGINEERING

Simplifier la conception

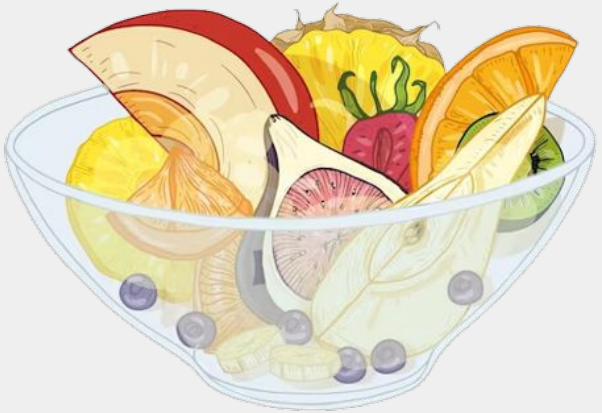
“Lorsque vous essayez de résoudre un problème, les premières solutions que vous trouvez sont très complexes et la plupart des gens s'en arrêtent là. Mais si vous continuez à chercher, restant concentré sur le problème à résoudre, continuant d'enlever la peau de l'oignon, vous arriverez souvent à des solutions élégantes et simples.”



Steve Jobs

Que préférez-vous ?

OU



Salade de POM(mes)

Salade de fruits

Quelques chiffres pour commencer

23

Design Patterns

- ➔ Création
- ➔ Structuration
- ➔ Comportement

2

Automatisation

- ➔ Page Object Model (POM)
- ➔ Data Driven Testing (DDT)

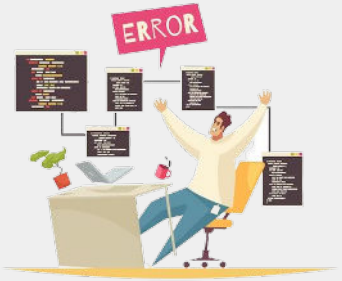
5

Exemples concrets

- ➔ POM
- ➔ Factory
- ➔ Singleton
- ➔ Builder
- ➔ Facade

C’est quoi un Design Pattern ?

stackoverflow



- Solution a un problème connu
- Se base sur la capitalisation d’expérience
- Décrit les grandes lignes et doit être implémenté / ce n’est pas un algorithme

Les design patterns des tests autos

Une activité de développement

Suit les mêmes principes
Économiser du temps et des efforts de développement

Maintenabilité

Pour un code réutilisable, fiable, stable

Organisation du code

- Structurer pour une clarté de code
- Meilleur utilisation

Des exemples comme guide

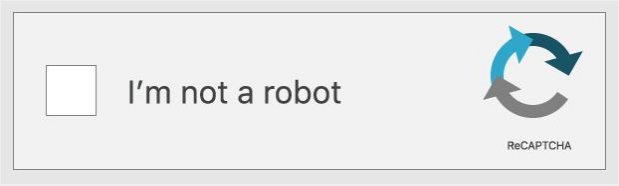
Login → Recherche article → Ajout au panier → Choix mode livraison → Choix mode Paiement

Multi-navigateurs

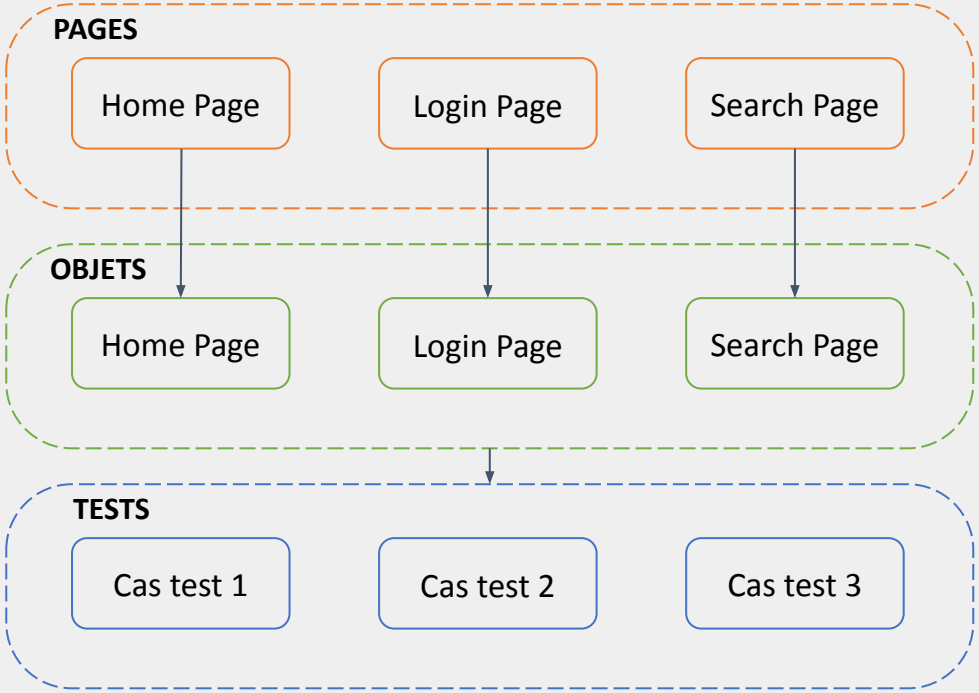
Les scripts avant POM : Le Login

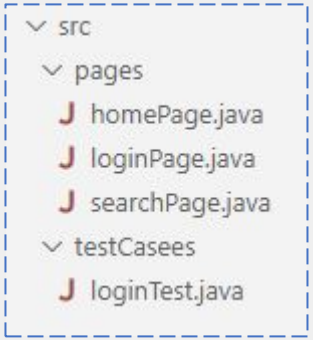
```
public class Login_Test_Without_POM{
    public static void main(string[] args) {

        //Instantiating chrome driver
        webDriver driver = new ChromeDriver();

        //Opening web application
        driver.get("https://localhost/loginpage.html");

        //Locating the Login button and clicking on it
        driver.findE1ement(By."login")).click();

        //Locating the username & password and passing the credentials
        driver.findE1ement(By.id("userName")).sendKeys("myusername");
        driver.findElement(By.id("password")).sendKeys("mypassword");

        //Click on the login button
        driver.findE1ement (By."login")).click();

        //Click on Logout button
        driver.findE1ement (By.id("logout")).click();

        //Close driver instance
        driver.quit();
    }
}
```

- Tout le code est dans la même classe main
- Copier coller de l'action de login dans tous les scripts
- Que se passe-t-il si on ajoute un captcha ?



POM : comment ça marche ?





Le Login avec POM

```
public class Login Page{
    private WebDriver driver ;
    @FindBy(id="usernarne")
    private WebElement usernarnelnput;
    @FindBy(id="password")
    private WebElement passwordlnput;
    @FindBy(css="input[type='submit']")
    private WebElement loginButton;

    public LoginPage(WebDriver driver){
        this.driver = driver;
        PageFactory.initEIements(driver, this);
    }

    public void enterusername(string username){
        usernameInput.sendKeys(username);
    }

    public void enterpas (String password){
        passwordlnput.sendKeys(password);
    }

    public void clickLoginButton(){
        loginButton.click();
    }
}
```

Login Object

```
public class LoginTest {
    public static void main(String[] args){
        // set the path of the ChromeOriver executable
        System.setProperty( "webdriver.chrome.driver", "path/to/chromedriver.exe");

        // create a new instance of the ChroneOriver
        WehDriver driver - new ChromeDriver();

        // navigate to the login page
        driver.get("https://localhost/loginpage.htm");

        // create a new instance 0+ the LocinPaee class
        LoginPage loginPage - new LoginPage(driver);

        // enter a usernane and passu,ord, and click the login button
        loginPage.enterUsername("myusernarne");
        loginPage.enterPassword("mypassword");
        loginPage.clickLoginButton();

        // close the browser
        driver. quit();
    }
}
```

Test case Login

Le Page Object Model

- Le design pattern le plus populaire
- Réduit la duplication de code et améliore la maintenance des tests
- Décrit les éléments avec lesquels un utilisateur peut interagir
- Décrit les tâches qu'il peut effectuer sur la page
- Utilise les méthodes d'extraction de données

PAGE OBJECT PATTERN

Pourquoi utiliser d'autres Design Patterns ?

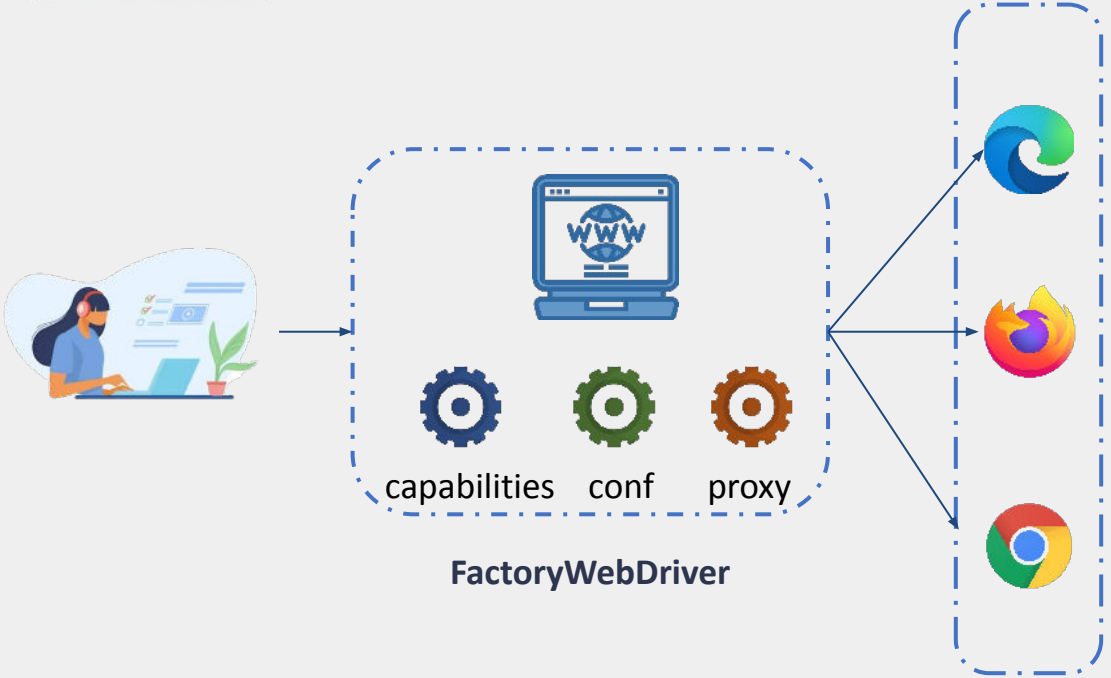
- Ne résout pas tous les problèmes de conceptions
- D'autres Patterns en complément
- Pas adapté aux tests d'API

Les tests multi-navigateurs

- Comment faciliter l'instanciation et masquer le détail de l'implémentation ?
- Comment centraliser la configuration des webdrivers ?
- Comment variabiliser le choix du navigateur ?

→ **Factory Pattern**

Les tests multi-navigateurs



- Accède au pilote et à la version
- Appel le navigateur souhaité
- avec les configurations souhaitées

Factory Pattern : le code

```java
public class WebDriverFactory {

    public static WebDriver createWebDriver(String browser) {
        WebDriver driver = null;

        switch (browser.toLowerCase()) {
            case "chrome":
                System.setProperty("webdriver.chrome.driver", "C:\\your_path\\chromedriver.exe");
                DesiredCapabilities capabilities = DesiredCapabilities.chrome();
                capabilities.setCapability("chrome.binary", "C:\\Program Files\\Google\\Chrome\\Application");
                ChromeOptions options = new ChromeOptions();
                options.merge(capabilities);
                options.addArguments("--headless");
                options.addArguments("--no-default-browser-check");
                options.setExperimentalOption("prefs", chromePreferences);
                driver = new ChromeDriver(options);
                break;

            case "firefox":
                System.setProperty("webdriver.firefox.driver", "C:\\your_path\\firefox.exe");
                DesiredCapabilities capabilities = DesiredCapabilities.firefox();
                FirefoxOptions options = new FirefoxOptions();
                options.merge(capabilities);
                options.addArguments("--headless");
                options.addPreference("browser.link.open_newwindow", 3);
                options.addPreference("browser.link.open_newwindow.restriction", 0);
                options.setLogLevel(Level.FINEST);
                driver = new FirefoxDriver(options);
                break;
            default:
                throw new IllegalArgumentException("Unsupported browser: " + browser);
        }
        return driver;
    }
```

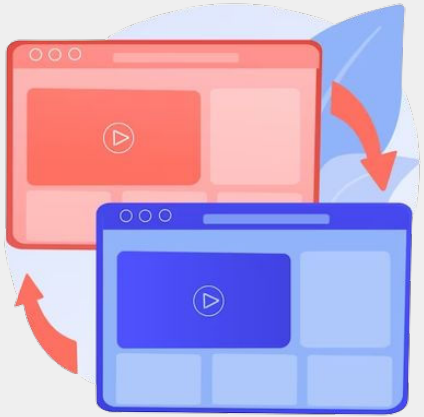
Sans Factory

```java
@Test
public void GoogleChromeTest() {
    System.setProperty("webdriver.chrome.driver", "C:\\your_path\\chromedriver.exe");
    DesiredCapabilities capabilities = DesiredCapabilities.chrome();
    capabilities.setCapability("chrome.binary", "C:\\Program Files\\Google\\Chrome\\Application");
    ChromeOptions options = new ChromeOptions();
    options.merge(capabilities);
    options.addArguments("--headless");
    options.addArguments("--no-default-browser-check");
    options.setExperimentalOption("prefs", chromePreferences);
    WebDriver driver = new ChromeDriver(options);

    driver.get("https://ecommerce/loginpage.html");
}
```

Avec Factory

```java
@Test
public void GoogleChromeTest() {
    WebDriver driver = WebDriverFactory.createWebDriver("Chrome");
    driver.get("https://ecommerce/loginpage.html");
}
```

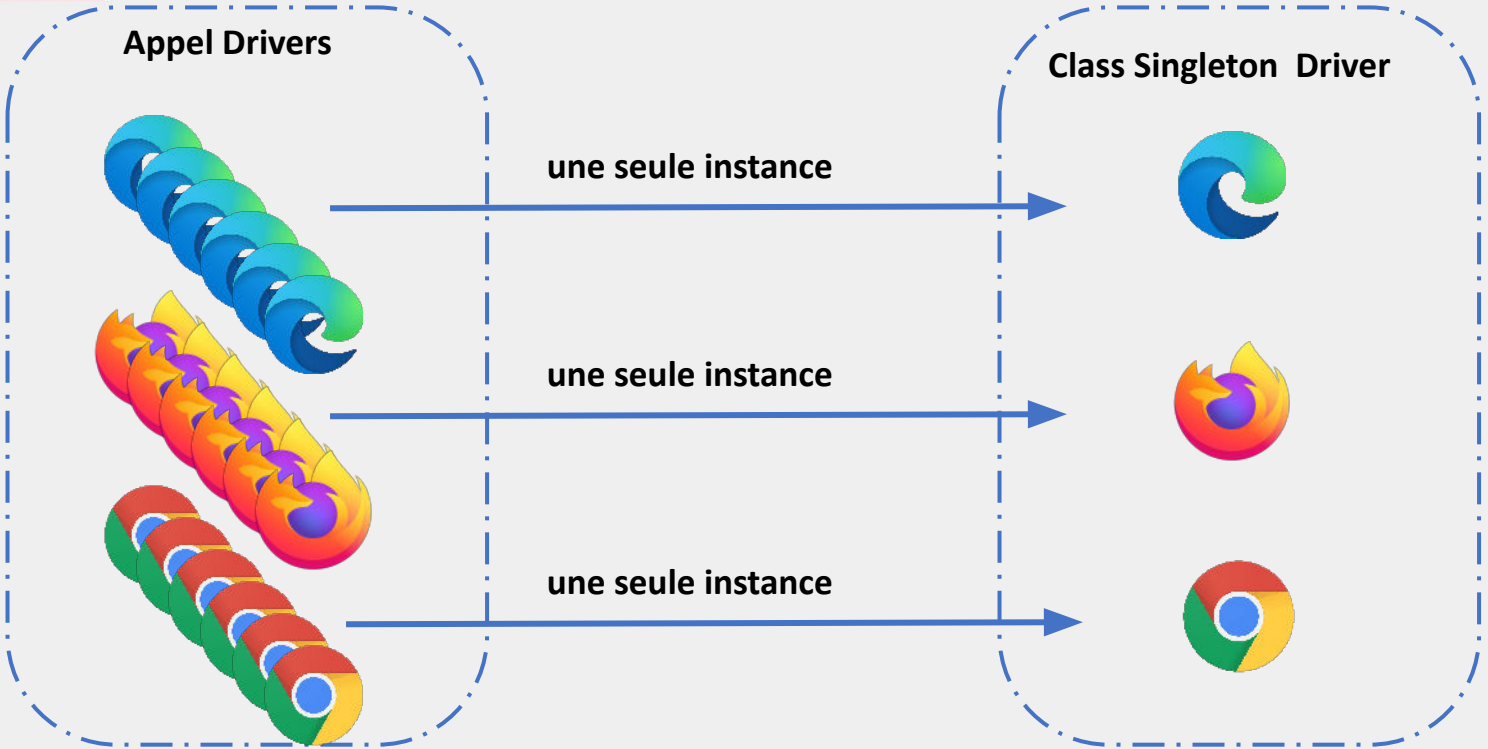
Comment éviter de créer de multiples objets ?



Singleton Pattern : assure qu'une seule instance d'un objet (WebDriver) est utilisée.

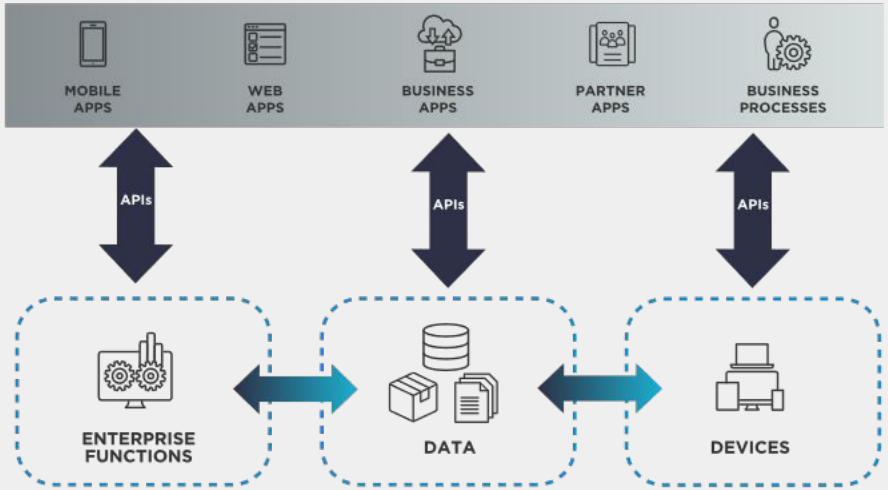
- Une seule instance à la fois
- Utilise le même objet d'une classe dans toutes les classes
- Renvoie la même instance si elle est à nouveau instanciée

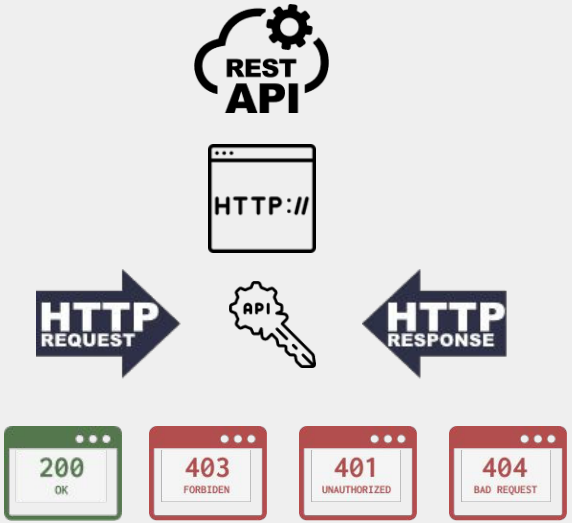
Singleton : comment ça fonctionne ?



Les tests APIs

Les API sont des mécanismes qui permettent aux composants logiciels de communiquer entre eux à l'aide d'un ensemble de définitions et de protocoles.



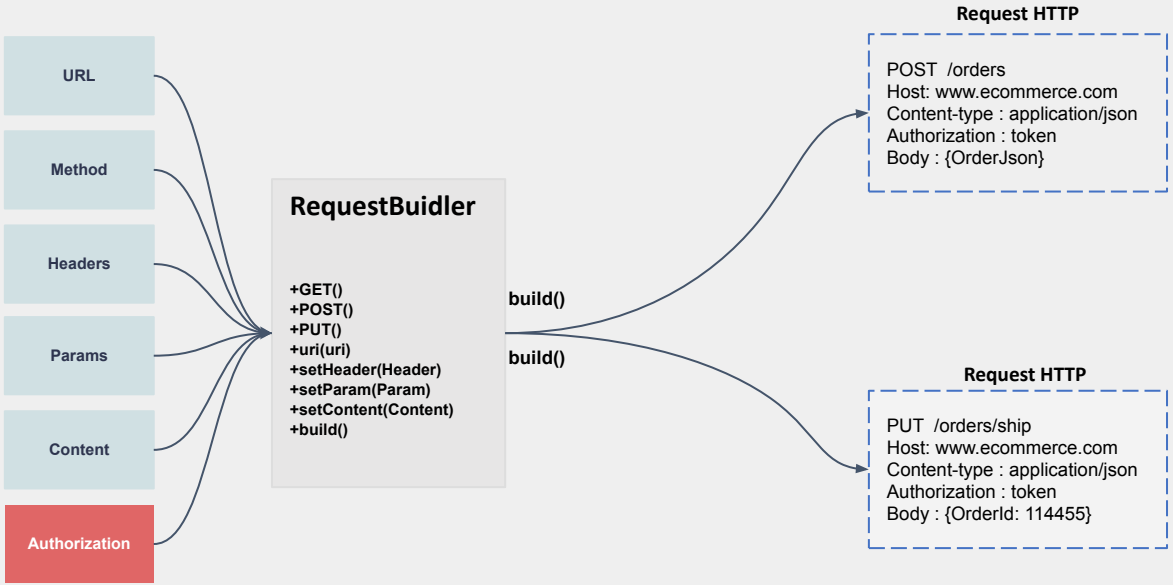


Comment construire des requêtes complexes ?

Builder Pattern : Facilite et personnalise la création des objets complexes.

- Réduire le nombre de paramètres
- Multi représentation d'un objet
- Séparation complexité construction VS représentation

Comment le Builder facilite la création d'une requête HTTP ?



Implémentation Builder : le passage de commande

RequestBuidler

+GET()
+POST()
+PUT()
+uri(uri)
+setHeader(Header)
+setParam(Param)
+setContent(Content)
+build()

Sans Builder

```
String apiUrl = "https://www.ecommerce.com/orders";
HttpClient httpClient = HttpClients.createDefault();
HttpPost httpPost = new HttpPost(apiUrl);

httpPost.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
httpPost.setHeader(HttpHeaders.AUTHORIZATION, "Bearer "+token);

String jsonBody = "{\"param1\":\"value1\",\"param2\":\"value2\"}";
StringEntity requestEntity = new StringEntity(jsonBody, ContentType.APPLICATION_JSON);
httpPost.setEntity(requestEntity);
```

Avec Builder

```
RequestBuilder.POST()
.uri("https://www.ecommerce.com/orders")
.setHeader("Content-Type", "application/json")
.setHeader("Authorization", "Bearer "+ token)
.setContent({\"param1\":\"value1\",\"param2\":\"value2\"})
.build();
```

Workflow complexe / trop de page objects

Facade Pattern : simplifie la gestion de scénarios longs et complexes

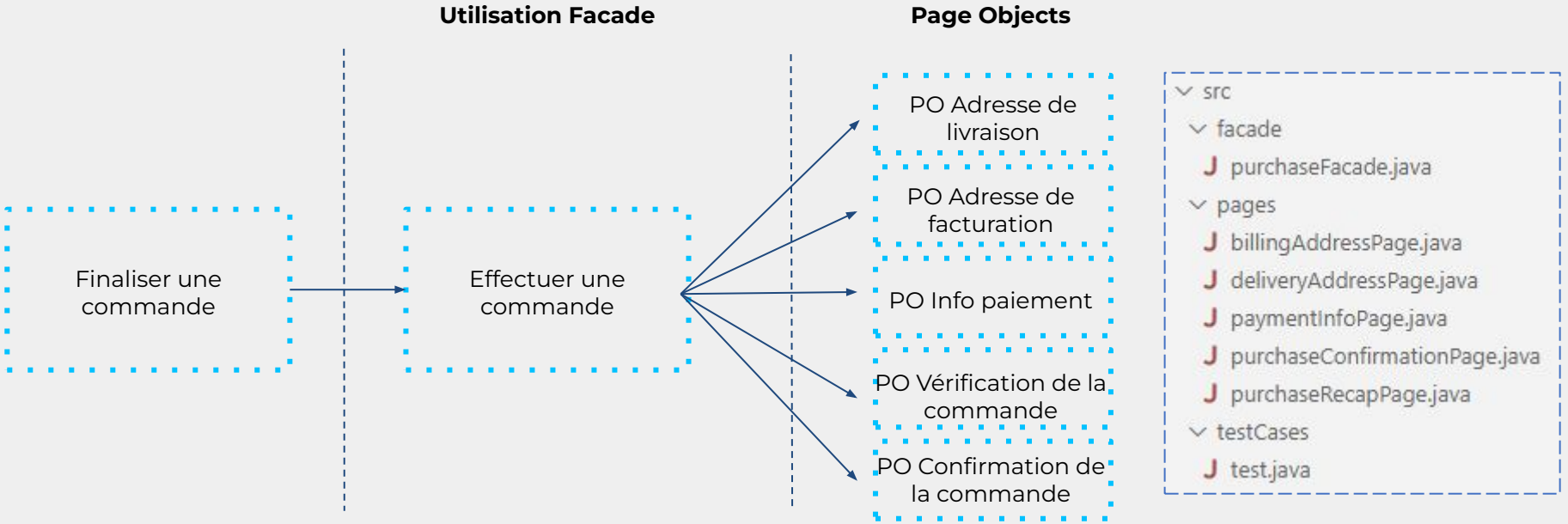
- Comment gérer un nombre important de Page Objects ?
- Comment gérer les changements de workflow ?
- Comment rendre le code plus lisible et simplifier les tests ?

Les scripts avant Façade : le tunnel d'achat

```
public class Purchase_Test_Without_Facade{
    @Test
    public void testl() {
        billingAddressPage bittingPO = new billingAddressPage();
        billing.enterAddress(address1, address2, zipcode. city, country);
        billingPO.clickAddAddressButton();

        deliveryAddressPage detiveryPO = new deliveryAddressPage();
        deliveryPO.enterAddress(address1, address2, zipcode, city, country);
        deliveryPO.clickAddAddressButton();

        paymentInfoPage paymentPO = new paymentInfoPage();
        paymentPO.selectProverType(provider);
        paymentPO.enterPaymentInfo(ccNumber, expirationDate, ccv);
        paymentPO.clickAddCreditCardButton();

        purchaseConfimationPage confirmationPO = new purchaseConfirmationPage();
        confirmationPO.checkRecapInfo();
        confirmationPO.ctickPurchaseButton();

        purchaseConfirmationPage recapPO = new purchaseConfimationPage();
        recapPO.checkConfimatiortqessage();
    }
}
```

- Beaucoup de page objets à faire instancier
- Copier coller des actions du tunnel d'achat dans tous les scripts
- Que se passe-t-il si on veut acheter plusieurs types de produits ?

Workflow complexe / trop de page objects



Le tunnel d'achat avec Facade Pattern

```
PostOrderFeedbackWithoutFacadeTest ( ) {
    billingAddressPage billingPO = new billingAddressPage();
    billingPO.enterAddress(address1, address2, zipcode, city, country);
    billingPO.clickAddAddressButton();

    deliveryAddressPage deliveryPO new deliveryAddressPage() ;
    deliveryPO.enterAddress(address1, address2, zipcode, city, country) ;
    detiveryPO.clickAddAddressButton();

    paymentInfoPage paymentPO = new paymentInfoPage();
    paymentPO.selectProviderType(provider);
    paymentPO.enterPaymentInfo(ccNumber, expirationDate, ccv) ;
    paymentPO.clickAddCreditCardButton();

    purchaseConfirmationPage confirmationPO = new purchaseConfirmationPage();
    confirmationPO.checkRecapInfo();
    confirmationPO.clickPurchaseButton();

    purchaseConfirmationPage recapPO = new purchaseConfirmationPage();
    recapPO.checkConfirnationMessage();

    orderHistoryPage orderPO new orderHistoryPage();
    orderPO.openFeedbackSection();
    orderPO.enterFeedback("Good quality product!");
    orderPO.SubmitFeedback();
    orderPO.verifyFeedbackSuccessMessage();
}
```

Sans Facade

```
PostOrderFeedbackWithFacadeTest(){
    purchaseJourneyFacade.placeOrder();
    orderFeedbackJourneyFacade.submitFeedback();
    orderPO.verifyFeedbackSuccessMessage();
}
```

Avec Facade

Pour terminer

- Les design patterns sont votre **boîte à outils**
- **Head First design pattern** de Eric et Elisabeth Freeman
- **refactoring.guru/fr**

Conférence

15ème
édition de la

Journée Française des Tests Logiciels

13 juin 2023

Beffroi de
Montrouge

Merci de votre écoute !

CFTL

Comité Français
des Tests Logiciels