ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

5ème année | Nom & prénom : | Site :……

Année 2022-2023 | …… | Groupe :……

Examen de Management de la Qualité

Durée : 1h30min --- Documents non autorisés

QCM : ***Entourer la ou les bonnes réponses***

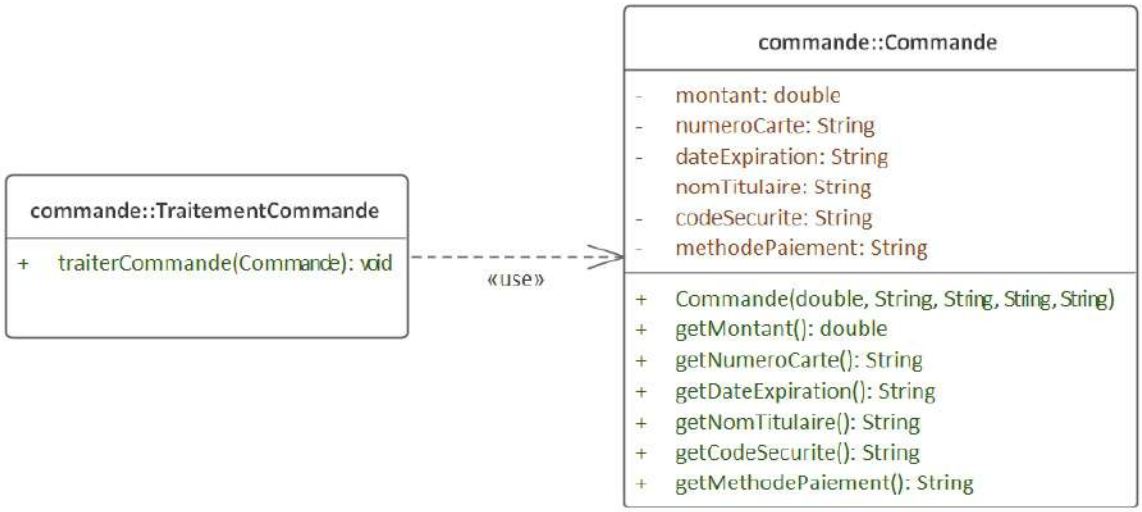
1) *Quel est l'avantage principal de l'utilisation de SonarQube pour une équipe de développement ?*
   - A. *Améliorer la qualité du code*
   - B. *Réduire les temps de développement*
   - C. *Détecter les problèmes de sécurité dans le code plus rapidement*
   - D. *Toutes les réponses sont correctes*
2) SonarQube prend-il en charge plusieurs langages de programmation ?
   - E. *Oui*
   - F. *Non*
3) SonarQube fournit-il un tableau de bord pour visualiser les résultats d'analyse ?
   - G. *Oui*
   - H. *Non*
4) À quoi sert principalement SonarQube?
   - I. *Analyse statique de code*
   - J. *Automatisation de Build*
   - K. *Test de performance*
   - L. *Surveillance réseau*
5) Qu'est-ce que la dette technique dans SonarQube?
   - M. *Une mesure du temps passé à développer un projet*
   - N. *Une mesure de la qualité du code d'un projet*
   - O. *Une mesure de la quantité de code dans un projet*
   - P. *Une mesure de la quantité de travail à faire pour maintenir et améliorer la qualité du code d'un projet.*
6) *À quel principe correspond l'énoncé suivant :* « ***Une instance d'une classe doit pouvoir être substituée sans modification par une instance d'une sous-classe*** »
   - Q. *L'inversion de dépendance*
   - R. *Ouvert / fermé*
   - S. *Séparation des interfaces*
   - T. *Responsabilité unique*
   - U. *Substitution de Liskov*
7) *À quel principe correspond l'énoncé suivant :* « ***Les clients ne devraient pas être forcés de dépendre des méthodes qu'ils n'utilisent pas.*** »
   - V. *L'inversion de dépendance*
   - W. *Ouvert / fermé*
   - X. *Séparation des interfaces*
   - Y. *Responsabilité unique*
   - Z. *Substitution de Liskov*

Exercice 1 :

Considérons la classe Java suivante qui permet le traitement du paiement d'une commande en ligne.

```java
package commande;
public class TraitementCommande {

    public void traiterCommande(Commande commande) {
        if (commande.getMethodePaiement().equals("carte de crédit")) {
            // code pour traiter un paiement par carte de crédit
        } else if (commande.getMethodePaiement().equals("PayPal")) {
            // code pour traiter un paiement PayPal
        } else if (commande.getMethodePaiement().equals("virement bancaire")) {
            // code pour traiter un paiement par virement bancaire
        }
        // ... plus de méthodes de paiement ajoutées au besoin
    }
}
```



Questions :

1) *En analysant le code ci-dessus, est-ce que la classe* ***TraitementCommande*** *proposée respecte-t-elle le principe de « **Ouvert à l'extension et fermé à la modification** » ? justifier votre réponse ?*

……

…non elle ne respecte pas le principe OCP, parce que chque fois on modifier le code source de cette classe

……

……

……

……

……

……

……

2) *Donner le diagramme de classes et le code java de la solution qui respecte le principe « **Ouvert à l'extension et fermé à la modification** » ? (Donner juste le code de la classe* ***TraitementCommande*** *et de l'interface qui sera définie)*

……

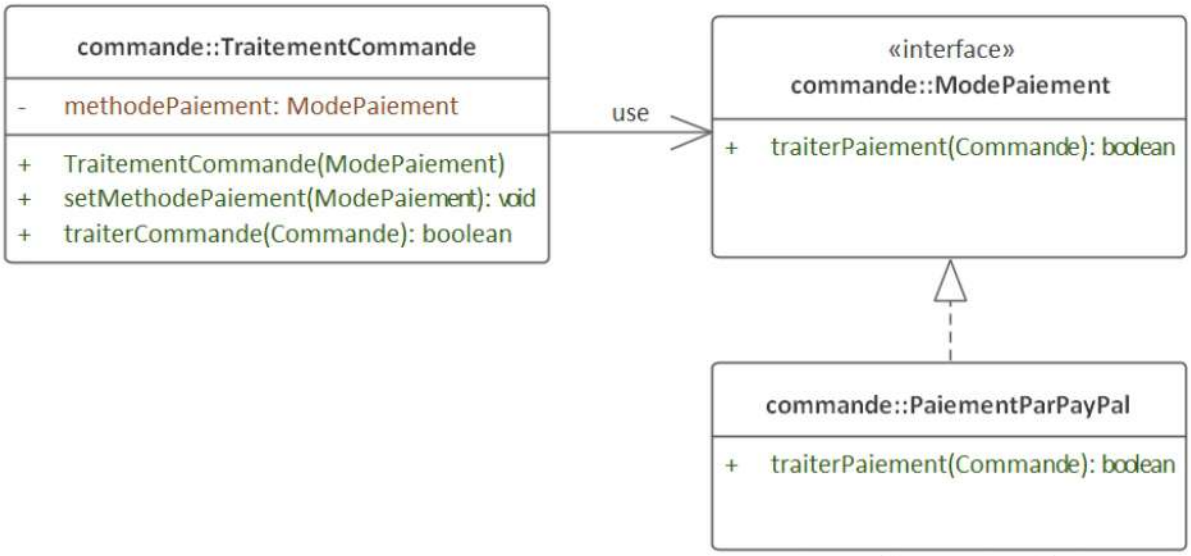
……

```
public class TraitementCommande Implements Imethodpay{
@override
public void cart(){}
@override
public void paypal(){}
@override
public void virement(){}
}

public interface imethodpay{

public void cart();
public void paypal();
public void virement();
```

3) *Considérons maintenant, le diagramme de classes suivant où nous souhaitons tester la méthode* ***traiterCommande(Commande)*** *de la classe* ***TraitementCommande.***



A. *Expliquer comment nous pouvons tester la méthode* ***traiterCommande(commande)*** *sans l'existence de la classe concrète* **PaiementParPayPal** *? (Sans donner du code java)*

… on va utiliser mockito pour tester cette methode sans la creation de la classe concrete. … … … … … …

… …

… …

… …

… …

B. *Dans le code ci-dessous, nous avons testé la méthode* ***traiterCommande(commande)*** *sans l'existence de la classe concrète* **PaiementParCarteBancaire** *en utilisant le Framework Mockito. Compléter les (--------) par les annotations ou les méthodes qui manquent.*

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.BeforeEach;   import org.junit.jupiter.api.Test;
public class TraitementCommandeTest {
  private ModePaiement modePaiement;
  private TraitementCommande traitementCommande;
  private Commande commande;
  @BeforeEach
  public void setUp() {
    modePaiement = mock(ModePaiement.Class) ;
    traitementCommande = new TraitementCommande(modePaiement);
    commande = new Commande();   }

   @Test
  public void testTraiterCommandeAccepte() {
     when (modePaiement.traiterPaiement(commande)). ThenReturn (true);

    boolean estAccepte = traitementCommande.traiterCommande(commande);
    verify(modePaiement).traiterPaiement(commande);
    assertEquals(true, estAccepte);   } }
```

Exercice N° 2 : Tests unitaires avec JUnit 5

Dans cet exercice, nous allons développer des tests unitaires en JUnit 5.0 pour la classe **Compte** *suivante :*

```
public class Compte {
	private String nom; private int numeroCompte; private double solde;
	public Compte(String nom, int numCpt, double soldeInit){
	this.nom = nom;
		this.numeroCompte = numeroCompte;
		this.solde = soldeInitial;}
	public double getSolde() {return solde;	}
	public void setSolde(double nouveauSolde) {
		this.solde = nouveauSolde; }
	public void retirer(double montant) {
		double nouveauSolde = getSolde() - montant;
		if (nouveauSolde < 0) {
			System.out.println("Retrait non autorisé");
			return;}
		else {System.out.println("retrait avec succès");
			setSolde(nouveauSolde);}
	}
	public void deposer(double montant) {
		System.out.println("dépôt avec succès");
		double nouveauSolde = getSolde() + montant;
		setSolde(nouveauSolde);}
}
```

Nous avons écrit la classe de test suivante :

```
1  package comptebancaire;
2  import static org.junit.jupiter.api.Assertions.*;
3  import org.junit.jupiter.api.*;
4  import org.junit.jupiter.params.ParameterizedTest;
5  import org.junit.jupiter.params.provider.CsvSource;
6  @TestMethodOrder(MethodOrderer.OrderAnnotation.class)
7  class CompteTest {
8      Compte moncompte;
9      static int nombreDeTests;
10     @BeforeAll
11     static void setUpAll() {nombreDeTests=0;}
12     @AfterAll
13     private void tearDownAll() {
14         System.out.println(nombreDeTests);}
15     @BeforeEach
16     void setUp() {nombreDeTests++;
17         moncompte=new Compte("Amine", 2023, 5000.0);}
18     @Test
19     @DisplayName("retirer refuse le retrait si le montant dépasse le solde")
20     @Order(1)
21     void retirerDoitRefuserLeRetraitQuandMontantSuperieurSolde() {
22         moncompte.retirer(8000.0);
23         double soldePrevu =5000.0;
24         double soldeCalcule=moncompte.getSolde();
25         assertEquals(soldePrevu, soldeCalcule);
26         }
27     @Test
28     @DisplayName("retirer accepte le retrait si le montant ne dépasse pas solde"
29     @Order(2)
30     void retirerDoitAccepterLeRetraitQuandMontantInferieurSolde() {
31         moncompte.retirer(2000.0);
32         double soldePrevu =3000.0;
33         double soldeCalcule=moncompte.getSolde();
34         assertEquals(soldePrevu, soldeCalcule);
35         }
```

1) *Expliquez brièvement le rôle des instructions écrites de la ligne 15 jusqu'à la ligne 17*

L'annotation BEFOREach indique que cette méthode doit être exécutée avant chaque test dans la classe

la methode setup() pour iniscialise le compte

incrementation de test de compte

creation d'objet compte

2) *Le test présenté par la méthode* ***retirerDoitRefuserLeRetraitQuandMontantSuperieurSolde*** *(lignes 18 à 26) va-t-il passer ou échouer ? justifier votre réponse ?*

il va echouer parce que le solde 5000< 8000 montant retirer

3) *Le test présenté par la méthode* ***retirerDoitAccepterLeRetraitQuandMontantInferieurSolde*** *(lignes 27 à 35) va-t-il passer ou échouer ? justifier votre réponse ?*

il va passee parce que le montant retirer 2000 < aux solde 3000

4) *On souhaite réaliser un test paramétré en utilisant L'annotation* ***@CsvSource*** *pour vérifier les cas de test des retraits suivants :*

Montant de retrait demandé par le client	Solde Qui Doit Rester dans le compte après l'essai de retrait
5 100	5 000
10 000	5 000

*Pour réaliser ce test paramétré, nous avons ajouté la méthode de test suivante (***testRetirer***) dans la classe CompteTest. Complétez cette méthode.*

```
@parametrazedtest
@CsvSource(value={ "5000 : 5100" , "5000 : 10000"
                                                    }, delimiter=' : ')
@DisplayName("retirer plusieurs montants")
@Order(3)
void testRetirer(int soldeInitial, int montantRetrait)
{       Compte compte = new Compte(soldeInitial);
        int soldeAttendu = soldeInitial - montantRetrait;
        compte.retirer(montantRetrait);
        int soldeFinal = compte.getSolde();
        assertEquals(soldeAttendu, soldeFinal);
}
```

Exercice N° 3 : Test fonctionnel avec Selenium WebDriver

Nous souhaitons tester une application web en JEE qui permet d'identifier le type de triangle à partir des longueurs de ses côtes (voir la figure 1 ci-dessous « Types de Triangles »).

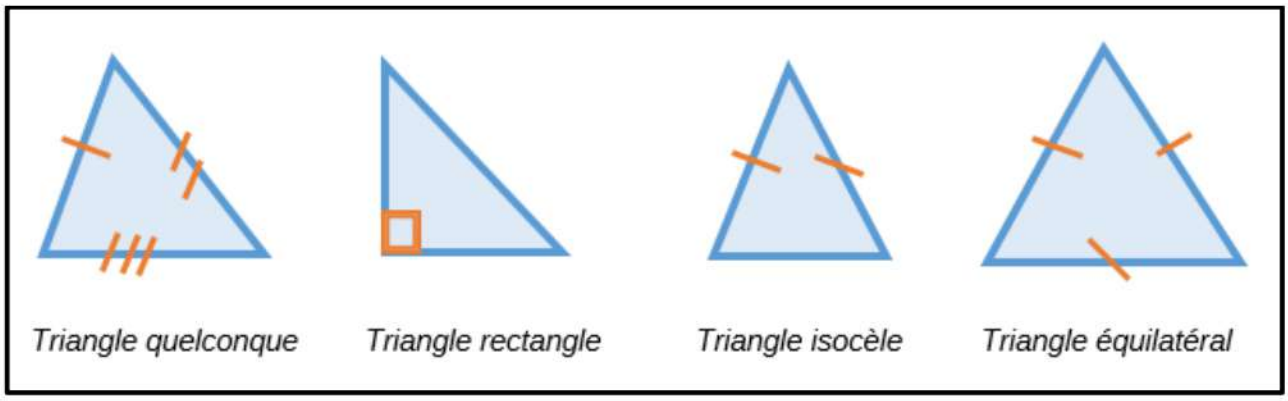


Figure 1 : Types de triangles

Considérons le prototype de la page web de l'application (voir la figure 2) :

Adresse : www.triangle.ma et **titre de la page** : «***types de triangles*** »

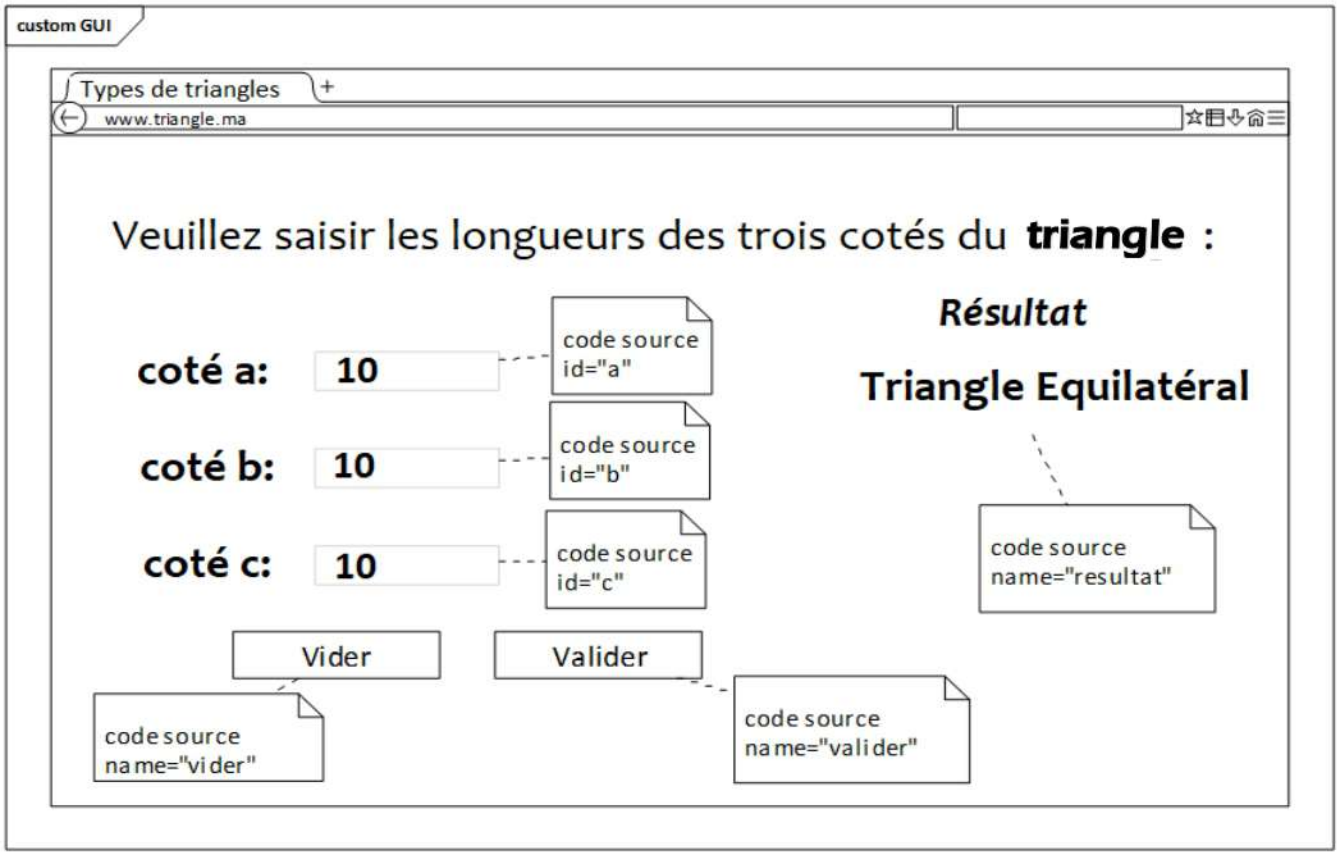


Figure 2 : Prototype de la page web

En utilisant Selenium WebDriver, nous voulons écrire les deux cas de test suivants pour le navigateur google chrome :

- Un cas de test **testTitrePage()** qui permet de vérifier que le titre du site web « www.triangle.ma », est « Types de triangles». Pour ce faire, on doit récupérer le titre de la page www.triangle.ma et le comparer avec la chaine de caractères «Types de triangles».

- Un cas de test **testSaisisValeursEgalesDonneTriangeEquilateral()** qui permet de vérifier que l'application fonctionne bien dans le cas d'un triangle équilatéral de cotés tous égaux à 10. Pour ce faire, on commence par identifier l'élément html qui correspond au côté a et y saisir la valeur 10 puis on fait la même chose pour les côtés b et c. Ensuite on identifie l'élément html qui correspond au bouton « valider » et on fait un clic sur ce bouton et finalement on identifie l'élément html qui correspond au label qui affiche le résultat et on récupère le texte qu'il contient et on le compare avec la chaine de caractères :"Triangle Equilatéral".

Travail à faire : compléter le code suivant qui implémente ces deux cas de test.

```
import static org.junit.Assert.assertEquals;

import static org.junit.Assert.assertEquals;
import org.junit.jupiter.api.*;
import org.openqa.selenium.*;
import org.openqa.selenium.chrome.ChromeDriver;
public class testPageExamen {
        private static WebDriver driver;
        @BeforeAll
        public static void setUpAll() {driver = new ChromeDriver();}
        @AfterAll
        public static void tearDown() {driver.quit();}
        @Test
        public void testTitrePage() {
                String baseUrl ="www.triangle.ma";
                String expectedTitle = "Types de triangles";
                String actualTitle=  driver. get (baseUrl);

                actualTitle = driver. getTitle ();

                assertEquals(actualTitle, expectedTitle);}
        @Test
        public void testSaisisValeursEgalesDonneTriangeEquilateral() {
        String baseUrl ="www.triangle.ma";
        driver.get(baseUrl);
        WebElement cote_a = driver. findElement(By.id("a"));
        cote_a.sendKeys("10");
       WebElement cote_b = driver. findElement(By.id("b"));
       cote_b.sendKeys("10");
      WebElement cote_c = driver. findElement(By.id("c"));
      cote_c.sendKeys("10");
      WebElement  boutton_v= findElement(By.name("valider"));

      boutton_v. click();

      WebElement res = driver.findElement(By.name("resultat"));

      String contenuRes=res.getText();

      assertEquals(contenuRes,"Triangle Equilatéral");

  }}
```

Annexe		
JUnit	**Mockito**	**Selenium WebDriver**
public static void assertEquals(Object expected, Object actual) public static void assertTrue(boolean condition) public static void assertFalse(boolean condition) public static void assertNull(Object actual) public static void assertNotNull(Object actual) public static <T extends Throwable> T assertThrows(Class<T> exceptionType, Executable executable) @ParameterizedTest @BeforeEach @AfterEach @BeforeAll @AfterAll	mock(): permet de créer un objet fictif when(): permet de définir le comportement d'un objet fictif lors d'un appel de méthode thenReturn(): permet de définir la valeur de retour d'un appel de méthode fictif verify(): permet de vérifier que les méthodes ont été appelées avec les arguments appropriés any(): permet de spécifier un argument quelconque pour une vérification de méthode doReturn(), doThrow(), doAnswer(), etc.: permet de définir un comportement personnalisé pour un appel de méthode fictif reset(): permet de réinitialiser un objet fictif pour une utilisation ultérieure.	WebDriver driver = new ChromeDriver(): permet de démarrer un nouveau navigateur Chrome driver.get("url"): permet de charger une page Web spécifique driver.findElement(By.xpath("xpath")): permet de trouver un élément sur la page en utilisant un chemin d'accès XPath driver.click(): permet de cliquer sur un élément driver.sendKeys("input"): permet de saisir du texte dans un champ de formulaire driver.getTitle(): permet de récupérer le titre de la page actuelle driver.quit(): permet de fermer le navigateur et de terminer la session de test driver.close(): permet de fermer la fenêtre du navigateur actuelle. getText() : permet de récupérer le texte d'un élément web driver.findElement(By.id("id")) driver.findElement(By.className("class")) driver.findElement(By.name("name")) driver.findElement(By.tagName("tag")) driver.findElement(By.linkText("text")) driver.findElement(By.partialLinkText("text")) driver.findElement(By.cssSelector("selector"))