Conception des Tests de conformité

Le test logiciel est la méthode la plus populaire pour vérifier un logiciel. Cette méthode représente environ **40-60% du prix final**. Tester, c'est exécuter le programme dans l'intention d'y trouver des anomalies ou des défauts

Plus précisément : une **spécification** est un ensemble explicite d'exigences à satisfaire par un logiciel, API, module, ….
Le **test** de conformité est l'exécution ou l'évaluation d'un système ou d'une composant par des moyens automatiques ou manuels, pour vérifier qu'il répond à ses spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus.

Deux familles de tests utilisées simultanément dans l'industrie, car elles sont complémentaires :
Tests structurels (boîte de verre) : les données de tests sont produites après une analyse du code ; ces tests détectent principalement les erreurs commises.
Tests fonctionnels (boîte noire) : test de conformité par rapport à la spécification. Ces tests détectent principalement les erreurs d'omission par rapport à la spécification.

Le test est une activité intellectuelle :
- déterminer des scenarios plausibles pouvant mettre un logiciel en défaut ;
- concevoir et construire des bancs de tests permettant de vérifier les fonctionnalités et le respect des contraintes.

Quelques principes de base pour réaliser des tests.
- Un.e programmeur.e ne doit pas concevoir/écrire les bancs de tests ses propres programmes (tout simplement pour ne pas être juge et partie).
- Ne pas effectuer des tests avec l'hypothèse qu'aucune erreur ne va être trouvée.
- La définition des sorties ou résultats attendus doit être effectuée avant l'exécution d'un test ; car il y a des chances non négligeables de prendre un résultat erroné mais semblant cohérent pour un résultat correct.
- Inspecter minutieusement les résultats (les traces) de chaque test.
- Les jeux de tests doivent être écrits pour des entrées invalides ou incohérentes.

Conception des jeux de tests d'une méthode

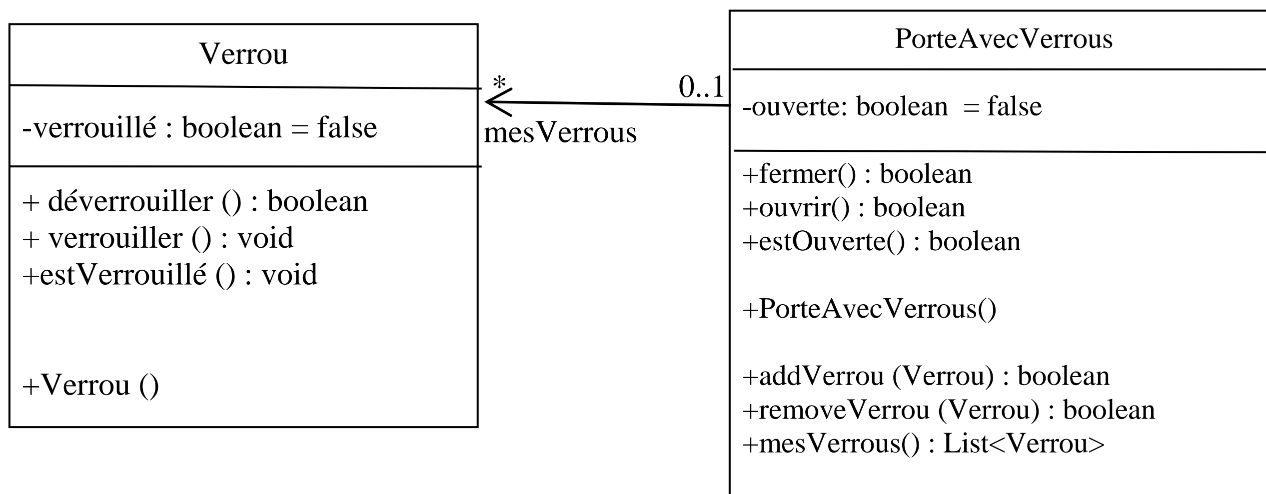
De manière générale, les actions que **doit réaliser** une méthode sont de trois types :
- changement de l'état du système : mise à jour des attributs et des associations,
- changement des données pérennes (fichiers, base de données),
- effet de bord (affichage sur la sortie d'erreur, la sortie standard)
- valeur retournée ou création d'une exception.

Les actions **réalisées par** une méthode dépendent de 3 choses :
- les valeurs de ses paramètres,
- l'état de l'objet (valeurs des attributs, des associations) voir l'état des objets voisins,
- les données pérennes (fichiers, base de données).

La conception de jeu de tests pour une **méthode** consiste à élaborer un jeu de tests en vue de vérifier que la méthode réalise bien toutes les actions qu'elle doit réaliser et ceci dans les divers cas d'usage (valeur de ses paramètres, état du système et des données pérennes, …).

Dans le cadre de la conception d'un jeu d'aventure dans le monde réel, Il a été demandé à un concepteur débutant de réaliser les tests de conformité des objets du monde « réel » : verrous, et portes. A vous de l'aider.

Verrou
Un verrou peut être verrouillé ou déverrouillé



Porte avec Verrous
Une porte peut être ouverte ou fermée.
La méthode « fermer » retourne *true* si la porte est fermée à la fin de l'exécution de la méthode ; de manière similaire, la méthode « ouvrir » retourne *true* si la porte est ouverte à la fin de l'exécution.

Une porte peut avoir un ou plusieurs verrous.
Les verrous d'une porte doivent être déverrouillés (verrouillé == false) pour l'ouvrir.
Tous les verrous d'une porte doivent être déverrouillés (verrouillé == false) pour la fermer. Sinon, le pêne ou loquet des verrous en position verrouillés empêcheront la fermeture de la porte.

Questions

1. Listez les cas d'usage de la méthode fermer de la classe PorteAvecVerrous et pour chaque cas d'usage, listez les valeurs à tester.
2. Déterminer la signature de la méthode fermerTest et implémentez cette méthode.
3. Proposez les scénarios de tests permettant de tester tous les cas d'usage de la méthode fermer.

Les cases d'usage de la méthode fermer de la classe PorteAvecVerrou :

1. La porte n'a pas de verrou
   a. La porte est ouverte
   b. La porte est fermée
2. La porte a un seul verrou qui est déverrouillé
   a. La porte est ouverte
   b. La porte est fermée
3. La porte a un seul verrou qui est verrouillé
   a. La porte est ouverte
   b. La porte est fermée
4. La porte a plusieurs verrous qui sont tous déverrouillés
   a. La porte est ouverte
   b. La porte est fermée
5. La porte a plusieurs verrous, un seul verrou est verrouillé
   a. La porte est ouverte
   b. La porte est fermée
6. La porte a plusieurs verrous, plusieurs sont verrouillé
   a. La porte est ouverte
   b. La porte est fermée

Le code de la méthode fermerTest et les scénarios en Java

```
public class PorteAvecVerrousTests {

	public static void fermerTest(PorteAvecVerrous porte) {

		boolean peutFermer = peutFermer(porte) ;
		boolean porteOuverteAvant = porte.estOuverte() ;

		boolean res = porte.fermer() ;

		assertEquals( !res, porte.estOuverte() ) ;

		if (porteOuverteAvant == false) {
			assertEquals(true, res) ;
			return ;		}
		if (peutFermer == false)	{
			assertEquals( !porteOuverteAvant, res) ;
			return ;		}
		assertEquals(true, res) ;
	}

	private static boolean peutFermer(PorteAvecVerrous porte) {
		List<Verrou> verrous = porte.mesVerrous() ;
		if (verrous == null) return true ;
		for (Verrou verrou : verrous) {
			if (verrou.estVerrouille()) return false ;
		}
		return true ;
    }
```

```
	@Test
	public void scenariosFermer() {
		PorteAvecVerrous porteSansVerrou = new PorteAvecVerrous();
		fermerTest(porteSansVerrou); // cas 1.b
		porteSansVerrou.ouvrir() ;
		fermerTest(porteSansVerrou); // cas 1.a

		Verrou verrou1 = new Verrou ();
		PorteAvecVerrous porteAvecUnVerrou = new PorteAvecVerrous();

		porteAvecUnVerrou.addVerrou(verrou1) ;
		fermerTest(porteAvecUnVerrou); // cas 2.b
		verrou1.verrouiller() ;
		fermerTest(porteAvecUnVerrou); // cas 3.b
		verrou1.deverrouiller() ;
		porteAvecUnVerrou.ouvrir() ;
		verrou1.verrouiller() ;
		fermerTest(porteAvecUnVerrou); // cas 3.a
		verrou1.deverrouiller() ;
		fermerTest(porteAvecUnVerrou); // cas 2.a

		Verrou verrou2 = new Verrou ();
		Verrou verrou3 = new Verrou ();
		Verrou verrou4 = new Verrou ();
		PorteAvecVerrous porteAvecDesVerrous = new PorteAvecVerrous();
		porteAvecDesVerrous.addVerrou(verrou2) ;
		porteAvecDesVerrous.addVerrou(verrou3) ;
		porteAvecDesVerrous.addVerrou(verrou4) ;

		fermerTest(porteAvecDesVerrous) ; // cas 4.b
		porteAvecDesVerrous.ouvrir() ;
		fermerTest(porteAvecDesVerrous) ; // cas 4.a

		verrou2.verrouiller() ;
		fermerTest(porteAvecDesVerrous) ; // cas 5.b
		verrou2.deverrouiller() ;
		porteAvecDesVerrous.ouvrir() ;
		verrou2.verrouiller() ;
		fermerTest(porteAvecDesVerrous); // cas 5.a
		verrou2.deverrouiller() ;
		verrou3.verrouiller() ;
		fermerTest(porteAvecDesVerrous) ; // cas 5.b
		verrou3.deverrouiller() ;
		porteAvecDesVerrous.ouvrir() ;
		verrou3.verrouiller() ;
		fermerTest(porteAvecDesVerrous); // cas 5.a

		verrou4.verrouiller() ;
		fermerTest(porteAvecDesVerrous); // cas 6.b
		verrou4.deverrouiller() ; verrou3.deverrouiller() ;
		porteAvecDesVerrous.ouvrir() ;
		verrou3.verrouiller() ; verrou4.verrouiller() ;
		fermerTest(porteAvecDesVerrous); // cas 6.a
	}

}
```