TP

Le but du TP est de tester la classe ci-dessous PolynomeSecondDegre avec JUnit.

```
package jmf.equation;

public class PolynomeSecondDegre {
	private double a, b, c;
	private double[] racines;

	public PolynomeSecondDegre(double a, double b, double c) {
		super();
		this.a = a;
		this.b = b;
		this.c = c;
	}

	public PolynomeSecondDegre() {
	}

	public double discriminant() {
		return b *b - 4 * a *c;
	}

	public double[] calculRacines() {
		racines = null;
		double discri = discriminant();
		if (discri == 0) {
			racines = new double[1];
			racines[0] = -b / (2 * a);
		} else if (discri > 0) {
			double racineDiscriminant = Math.sqrt(discri);
			racines = new double[2];
			racines[0] = (-b + racineDiscriminant)/ (2 * a);
			racines[1] = (-b - racineDiscriminant)/ (2 * a);
		}
		return racines;
	}
}
```

Construire une classe de test test.TestEquationSecondDegre qui teste la classe PolynomeSecondDegre. On demande d'utiliser la syntaxe JUnit. Cette classe de tests doit avoir trois méthodes :
- la méthode **public void** test2Racines() qui construit le polynome $X^2 + X - 2$ et qui vérifie que laméthode calculRacines()retourne bien les 2 racines 1 et -2 de ce polynome,
- la méthode **public void** test1Racine() qui construit le polynome $X^2 - 2X + 1$ et qui vérifie que laméthode calculRacines()retourne bien la seule racine 1 de ce polynome,
- la méthode **public void** testPasDeRacineReelle() qui construit le polynome $X^2 + X + 1$ et qui vérifie que la méthode calculRacines()retourne aucune racine de ce polynome.

Dans ce TP, une classe (comme la classe test.TestEquationSecondDegre) qui permet de lancer des tests sera appelée une classe de tests.

On peut lancer JUnit dans une fenêtre de commande :
java junit.textui.TestRunner test.TestEquationSecondDegre

Pour simplifier le problème, on se met dans l'hypothèse où on a qu'une seule classe de tests qui possède

éventuellement plusieurs méthodes de tests. C'est le cas dans notre exemple où nous avons une seule classe de tests, la classe test.TestEquationSecondDegre qui possède trois méthodes de tests public void test2Racines(), public void test1Racine() et public void test PasDeRacineReelle().

Le framework JUnit lance les tests de manière récursive (à l'aide du design pattern Composite qu'on ne demande pas d'étudier dans ce problème) en exécutant une méthode spécifique de nom runBare() sur chaque test. Cette méthode n'est d'ailleurs pas banale puisqu'elle a un corps de la forme :
initialisation (la méthode setUp()*)*
lance le test (la méthode de nom runTest()*qui lancera les méthodes de nom de la forme* testXXX()*)fin du test (la méthode tearDown*()*)*

Pour chaque test, JUnit doit donner le résultat du test, c'est à dire si le test a échoué (failure, error, ...) ou non. Pour cela un objet de la classe junit.framework.TestResult est créé au lancement de l'exécution JUnit. Cet objet va collecter les résultats des tests.
Le début du lancement du code de l'environnement JUnit est de la forme :
TestResult result = new TestResult();run(result);
L'environnement construit donc un TestResult qui va contenir les résultats de tous les tests. Pour une exécution, il n'y aura qu'un seul objet référencé result qui va contenir tous les résultats des tests. Pour ne pas passer cette argument result à toutes les méthodes qui seront ensuite appelées, le code de la méthode runBare()est plutôt de la forme :

```
setUp();try
{
        runTest(); //    qui lancera les méthodes de nom de la forme testXXX())
}
catch (AssertionFailedError e) {
        result.addFailure(..., e);
}
catch (Throwable e) { result.addError(...,
        e);
}
tearDown();
```

Comme dans notre exemple (la classe test.TestEquationSecondDegre), JUnit utilise souvent des méthodes de la forme assertXXX(...). Par exemple la méthode assertTrue(boolean condition)est codée :

```
protected void assertTrue(boolean condition) {if (!condition)
                throw new AssertionFailedError();
}
```

où AssertionFailedErrorest une classe de la forme
public class AssertionFailedError extends Error { ... }

Comme une des dernières instructions de l'environnement d'exécution de JUnit consiste à faire afficher les données de l'objet de la classe TestResult référencé par result ci-dessus, indiquez comment, au fur et àmesure, de l'exécution des tests, si le test échoue, l'objet référencé par result est alimenté par des informations de résultats du test.