Conception orientée objet

Application du cours 8 : Le marché

Cet énoncé vient en appui des diapositives du Cours.

I. Classe à tester

```
public class Poisson implements Comparable <Poisson> {
     private Date datePeche;
     private float poids;

     public Poisson(Date datePeche, float poids) {
          this.datePeche = datePeche;
          this.poids = poids;
     }

     public Date getDatePeche() {
          return datePeche;
     }

     public float getPoids() {
          return poids;
     }

     @Override
     public boolean equals(Object objet) {
          if (objet instanceof Poisson) {
             Poisson poisson = (Poisson) objet;
             return datePeche.equals(poisson.datePeche) && poids == poisson.poids;
          }
          return false;
     }

     @Override
     public int hashCode() {
          return 31 * datePeche.hashCode() + (int) (poids * 100);
     }

     public int compareTo(Poisson poisson) {
          int comparaison = datePeche.compareTo(poisson.datePeche);
          if (comparaison != 0) {
             return comparaison;
          }
          float difference = poids - poisson.poids;
          if (difference < 0) {
             return -1;
          }
          if (difference > 0) {
             return 1;
          } else {
             return 0;
          }
     }
}
```

II. Travail à effectuer

```
class PoissonTest {
   private Date datePeche11Mai = new Date(11, 5, 2017);;
   private static final float POIDS = 1.2f;
   private Poisson poisson;

// Vérifier l'hypothèse que la date de pêche et le poids ont bien été créées.
// Créer l'instance de poisson

      @BeforeEach
      void setUp() {

Assumption.assumeTrue(datePeche11Mai != null, "Date non null");
Assumption.assumeTrue(POIDS != 0, "Poids initialisés");
poisson = new Poisson(datePeche11Mai, POIDS)

      }

// Tester le constructeur de la classe Poisson : poisson est non null

      @Test
      void testPoisson() {

Assertions.assertNotNull(poisson, "Création d'un poisson : ok");

   }

// Tester la méthode hashCode de la classe Poisson

      @Test
      void testHashCode() {

Assertion.assert(
{ 31*datePeche11Mai.hashCode() + (int) POIDS * 100,
poisson.hashCode(), "HashCode de poisson : OK");

      }
```

// Tester le getteur sur la date de pêche du poisson

@Test
void testGetDatePeche() {

Assertions.assertEquals(datePeche11Mai, poisson.getDatePeche(),
"GetDatePeche : Ok");
}

// Tester le getteur sur le poids du poisson

@Test
void testGetPoids() { // on peut utiliser l'assertEquals même sur un float

Assertions.assertEquals(POIDS, poisson.getPoids(),
"GetPoids : Ok");
}

// Tester la méthode equals de la classe Poisson avec :
// Deux poissons ayant la même date de pêche et le même poids
// Deux poissons de date différentes mais de même poids
// Deux poissons de même date de pêche mais de poids différents
// Un poisson et un objet d'un autre type

@Test
void testEqualsPoisson() {

~~Poisson poisson1 = new Poisson~~, poisson,
Assertions.assertTrue(new Poisson(datePeche11Mai, POIDS), "Poisson identique");
Assertions.assertNotEquals(new Poisson(new Date(25, 5, 2017), POIDS),
, poisson, "Poisson de date differentes");
Assertions.assertNotEquals(new Poisson(datePeche11Mai, 1,3f), poisson,
"Poisson de poids differents");
Assertions.assertNotEquals(datePeche11Mai, poisson, "objets differents");

}

 Quand une méthode + 15 lignes a séparer en 2 ses fonctions

// Tester que :
// - un poisson péché le 9 Mai est moins frais que celui péché le 11 Mai,
// - un poisson pesant 500g est plus léger que celui qui pèse 1,2kg (les
// deux poissons étant péchés le 11 Mai)

@Test
void testCompareToInferieur() {

Poisson poissonVieux = new Poisson(new Date(9,5,2017), 1.0f);
Assertions.assertTrue(poissonVieux.compareTo(poisson)<0,
"poisson moins frais");
Poisson poissonSmall = new Poisson(datePeche11Mai, 0.5f);
Assertions.assertTrue(poissonSmall.compareTo(poisson)<0
"poisson plus petit");

}

// Tester que :
// - un poisson péché le 14 Mai est plus frais que celui péché le 11 Mai,
// - un poisson pesant 1,5kg est plus gros que celui qui pèse 1,2kg (les
// deux poissons étant péchés le 11 Mai)

@Test
void testCompareToSuperieur() {

Poisson poissonRecent = new Poisson(new Date(14,5,2017), 1.0f);
Assertions.assertTrue(poissonRecent, compareTo(poisson)>0,
"poisson plus frai");
Poisson poissonBig = new Poisson(datePeche11Mai, 1.5f),
"poisson plus gros");

}

// Tester que 2 poissons ayant été péché le 11 Mai et pesant 1,2kg sont
// Identiques.

```
    @_____________________
    void testCompareToEquals() {

_____________________________________________________________

_____________________________________________________________

_____________________________________________________________

_____________________________________________________________
    }
}
```

III. JUnit 5

1. Assumptions

Modifier and Type	Method and Description
static void	**assumeFalse**(boolean assumption) Validate the given assumption.
static void	**assumeFalse**(boolean assumption, String message) Validate the given assumption.
static void	**assumeFalse**(BooleanSupplier assumptionSupplier) Validate the given assumption.
static void	**assumeFalse**(boolean assumption, Supplier<String> messageSupplier) Validate the given assumption.
static void	**assumeFalse**(BooleanSupplier assumptionSupplier, String message) Validate the given assumption.
static void	**assumeFalse**(BooleanSupplier assumptionSupplier, Supplier<String> messageSupplier) Validate the given assumption.
static void	**assumeTrue**(boolean assumption) Validate the given assumption.
static void	**assumeTrue**(boolean assumption, String message) Validate the given assumption.
static void	**assumeTrue**(BooleanSupplier assumptionSupplier) Validate the given assumption.
static void	**assumeTrue**(boolean assumption, Supplier<String> messageSupplier) Validate the given assumption.
static void	**assumeTrue**(BooleanSupplier assumptionSupplier, String message) Validate the given assumption.
static void	**assumeTrue**(BooleanSupplier assumptionSupplier, Supplier<String> messageSupplier) Validate the given assumption.
static void	**assumingThat**(boolean assumption, Executable executable) Execute the supplied Executable, but only if the supplied assumption is valid.
static void	**assumingThat**(BooleanSupplier assumptionSupplier, Executable executable) Execute the supplied Executable, but only if the supplied assumption is valid.

2. Assertions

JUnit 5.0.2 API

Modifier and Type	Method and Description
static void	**assertAll**(String heading, Executable... executables) *Asserts* that *all* supplied executables do not throw exceptions.
static void	**assertAll**(String heading, Stream<Executable> executables) *Asserts* that *all* supplied executables do not throw exceptions.
static void	**assertArrayEquals**(long[] expected, long[] actual, String message) *Asserts* that expected and actual long arrays are equal.
static void	**assertEquals**(Object expected, Object actual, String message) *Asserts* that expected and actual are equal.
static void	**assertFalse**(boolean condition, String message) *Asserts* that the supplied condition is not true.
static void	**assertIterableEquals**(Iterable<?> expected, Iterable<?> actual, String message) *Asserts* that expected and actual iterables are deeply equal.
static void	**assertLinesMatch**(List<String> expectedLines, List<String> actualLines) *Asserts* that expected list of Strings matches actual list.
static void	**assertNotEquals**(Object unexpected, Object actual, String message) *Asserts* that expected and actual are not equal.
static void	**assertNotNull**(Object actual, String message) *Asserts* that actual is not null.
static void	**assertNotSame**(Object unexpected, Object actual, String message) *Asserts* that expected and actual do not refer to the same object.
static void	**assertNull**(Object actual, String message) *Asserts* that actual is null.
static void	**assertSame**(Object expected, Object actual, String message) *Asserts* that expected and actual refer to the same object.
static <T extends Throwable> T	**assertThrows**(Class<T> expectedType, Executable executable, String message) *Asserts* that execution of the supplied executable throws an exception of the expectedType and returns the exception.
static void	**assertTimeout**(Duration timeout, Executable executable, String message) *Asserts* that execution of the supplied executable completes before the given timeout is exceeded.
static void	**assertTrue**(boolean condition, String message) *Asserts* that the supplied condition is true.
static <V> V	**fail**(String message, Throwable cause) *Fails* a test with the given failure message as well as the underlying cause.