

Question 1 : Donner le code définissant les attributs de `Ship` ainsi que le constructeur de la classe.

```
private int lifePoints;
private final String name;

public Ship(int length, String name) {
    this.lifePoints = length;
    this.name = name;
}
```

Question 2 : Donnez la méthode de tests qui permet de vérifier la spécification du comportement de la méthode `takeAHit()`.

```
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.*;

public class ShipTest {
    @Test
    public void testHit(){
        Ship submarine = new Ship(3, "Submarine");
        assertThat(submarine.getLifePoints()).isEqualTo(3);
        submarine.takeAHit();
        assertThat(submarine.getLifePoints()).isEqualTo(2);
        Ship carrier = new Ship(5, "Carrier");
        assertThat(carrier.getLifePoints()).isEqualTo(5);
        carrier.takeAHit();
        carrier.takeAHit();
        assertThat(carrier.getLifePoints()).isEqualTo(3);
    }
}
```

Question 3 : Comment peut-on définir le type `ShotResult`.

Puisque `ShotResult` n'a que trois valeurs possible, la meilleure façon de le définir est d'utiliser un `enum` en java.

```
public enum ShotResult {
    MISSED, HIT, SUNK
}
```

Question 4 : Comment peut-on gérer la présence ou l'absence d'un bateau sur une case ?

Pour indiquer qu'un bateau n'est pas présent dans la case, on peut laisser la valeur `null` (valeur par défaut des références vers objets en Java) pour l'attribut `ship` de `Cell`.

Question 5 : Donnez la ou les méthodes de tests qui permettent de vérifier cette spécification du comportement de la méthode `takeAShot()`.

```
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.*;

public class CellTest {
    @Test
    void testFirstShotWithNoShipIsMissed(){
        Cell cell = new Cell();
        assertThat(cell.hasShip()).isFalse();
        assertThat(cell.hasBeenShot()).isFalse();
        assertThat(cell.takeAShot()).isEqualTo(ShotResult.MISSED);
        assertThat(cell.hasBeenShot()).isTrue();
    }
    @Test
    void testFirstShotAtShipWithTwoLifeIsHit(){
        Cell cell = new Cell();
        Ship battleship = new Ship(2, "Battleship");
        cell.setShip(battleship);
        assertThat(cell.takeAShot()).isEqualTo(ShotResult.HIT);
        assertThat(cell.hasBeenShot()).isTrue();
    }
    @Test
    void testFirstShotAtShipWithOneLifeIsSunk(){
        Cell cell = new Cell();
        Ship oneLifeShip = new Ship(1, "OneLifeShip");
        cell.setShip(oneLifeShip);
        assertThat(cell.takeAShot()).isEqualTo(ShotResult.SUNK);
        assertThat(cell.hasBeenShot()).isTrue();
    }
    @Test
    void testTwoShotOnTwoCellsAtShipWithTwoLifeIsSunk(){
        Cell cell1 = new Cell();
```

```

Cell cell2 = new Cell();
Ship battleship = new Ship(2, "Battleship");
cell1.setShip(battleship);
cell2.setShip(battleship);
assertThat(cell1.takeAShot()).isEqualTo(ShotResult.HIT);
assertThat(cell2.takeAShot()).isEqualTo(ShotResult.SUNK);
}
@Test
void testSecondShotWithNoShipIsMissed(){
    Cell cell = new Cell();
    assertThat(cell.hasShip()).isFalse();
    cell.takeAShot();
    assertThat(cell.hasBeenShot()).isTrue();
    assertThat(cell.takeAShot()).isEqualTo(ShotResult.MISSED);
}
@Test
void testSecondShotAtShipIsMissed(){
    Cell cell = new Cell();
    Ship oneLifeShip = new Ship(1, "OneLifeShip");
    cell.setShip(oneLifeShip);
    cell.takeAShot();
    assertThat(cell.hasBeenShot()).isTrue();
    assertThat(cell.takeAShot()).isEqualTo(ShotResult.MISSED);
}
}
}

```

Question 6 : Donnez le code de la méthode `takeAShot()`.

```

public ShotResult takeAShot(){
    if(hasBeenShot())
        return ShotResult.MISSED;
    hasBeenShot = true;
    if(!hasShip())
        return ShotResult.MISSED;
    ship.takeAHit();
    if(ship.hasBeenSunk())
        return ShotResult.SUNK;
    return ShotResult.HIT;
}

```

Question 7 : Comment peut-on rendre les instances de position immuables ?

Pour rendre les instances de `Coordinates` immuable (et donc rendre impossible la modification des valeurs de

leurs attributs), il suffit de définir les deux attributs x et y avec le mot-clé **final** qui interdit l'affectation en dehors du constructeur ou de la valeur par défaut à la déclaration de l'attribut.

Question 8 : Donnez le code de la méthode `equals` de `Coordinates`.

```
public boolean equals(Object o) {
    if (!(o instanceof Coordinates)) return false;
    Coordinates coordinates = (Coordinates) o;
    return x == coordinates.x && y == coordinates.y;
}
```

Question 9 : Donnez le code définissant les attributs de `Grid` ainsi que le constructeur de la classe sachant qu'initialement toutes les cases sont vides (pas de bateau) et que les dimensions du plateau de jeu sont fixées à la construction de l'objet.

```
public class Grid {
    private final int width;
    private final int height;
    private final Cell[][] cells;

    public Grid(int width, int height){
        this.width = width;
        this.height = height;
        cells = new Cell[width][height];
        for(int x = 0; x < width; x++){
            for(int y = 0; y < height; y++){
                cells[x][y] = new Cell();
            }
        }
    }
}
```

Question 10 : Dessinez le diagramme de classe de `Grid`.

Afin de pouvoir placer les bateau dans la grille, on va supposer qu'il existe une classe `Position` représentant la position d'un bateau dans la grille.

C Grid

- ❑ width : int
 - ❑ height : int
 - ❑ cells : Cell[][]
-
- Grid(width : int, height : int)
 - getWidth() : int
 - getHeight() : int
 - getCell(coordinates : Coordinates) : Cell
 - contains(coordinates : Coordinates) : boolean
 - shootAt(coordinates : Coordinates) : ShotResult
 - hasShipAt(coordinates : Coordinates) : boolean
 - hasBeenShotAt(coordinates : Coordinates) : boolean
 - display()
 - put(ship : Ship, position : Position)

Question 11 : Donnez la signature, la javadoc, les méthodes de test et le code de la méthode `shootAt` de `Grid`.

La signature et la javadoc de la méthode `shootAt` pourraient être les suivants :

```
/**
 * Shoot at the {@code Cell} of the {@code Grid} at the specified {@code coordinates} and
 * return the result of the shot.
 *
 * If the specified coordinates are outside the range of the grid, the call of the method
 * only returns {@code ShotResult.MISSED}.
 *
 * For the first shot at some specific coordinates included in the {@code Grid}, the result
 * depends on the presence or not of a ship at the corresponding {@code Cell}. If there are
 * no ship, the result of the shot is always {@code ShotResult.MISSED}. If there is a ship
 * the result of the shot is {@code ShotResult.SUNK} if it was the last life point of the
 * ship and {@code ShotResult.SUNK} otherwise. For any subsequent shot, the result of the
 * shot is always {@code ShotResult.MISSED}.
 *
 * @param coordinates the coordinates of the target {@code Cell}
 * @return {@code ShotResult.MISSED}, {@code ShotResult.HIT} or {@code ShotResult.SUNK}
 *         depending on the result of the shot.
 */
public ShotResult shootAt(Coordinates coordinates){
    if(!contains(coordinates))
        return ShotResult.MISSED;
    return getCell(coordinates).takeAShot();
}
```

Le code des méthodes de tests pourrait être le suivant :

```
import org.junit.jupiter.api.Test;

import static org.assertj.core.api.Assertions.assertThat;

public class GridTest {
    @Test
    void testShotOutsideRangeIsMissed(){
        Grid grid = new Grid(3,3);
        for(int y = 0; y < 3; y++){
            Ship battleship = new Ship(3, "Battleship");
            Coordinates coordinates = new Coordinates(0,y);
            grid.putShip(battleship, coordinates, Orientation.HORIZONTAL);
        }
        for(int x = 0; x < 3; x++) {
            for (int y = 0; y < 3; y++) {
                assertThat(grid.hasShipAt(new Coordinates(x, y))).isTrue();
            }
        }
        assertThat(grid.shootAt(new Coordinates(-1,-1))).isEqualTo(ShotResult.MISSED);
        assertThat(grid.shootAt(new Coordinates(-1,0))).isEqualTo(ShotResult.MISSED);
        assertThat(grid.shootAt(new Coordinates(0,-1))).isEqualTo(ShotResult.MISSED);
        assertThat(grid.shootAt(new Coordinates(3,3))).isEqualTo(ShotResult.MISSED);
        assertThat(grid.shootAt(new Coordinates(3,1))).isEqualTo(ShotResult.MISSED);
        assertThat(grid.shootAt(new Coordinates(1,3))).isEqualTo(ShotResult.MISSED);
    }
    @Test
    void testFirstShotWithNoShipIsMissed(){
        Grid grid = new Grid(3,3);
        Coordinates coordinates = new Coordinates(0,0);
        assertThat(grid.hasShipAt(coordinates)).isFalse();
        assertThat(grid.hasBeenShotAt(coordinates)).isFalse();
        assertThat(grid.shootAt(coordinates)).isEqualTo(ShotResult.MISSED);
        assertThat(grid.hasBeenShotAt(coordinates)).isTrue();
    }
    @Test
    void testFirstShotAtShipWithTwoLifeIsHit(){
        Grid grid = new Grid(3,3);
        Coordinates coordinates = new Coordinates(0,0);
        Ship battleship = new Ship(2, "Battleship");
        grid.putShip(battleship, coordinates, Orientation.HORIZONTAL);
        assertThat(grid.shootAt(coordinates)).isEqualTo(ShotResult.HIT);
        assertThat(grid.hasBeenShotAt(coordinates)).isTrue();
    }
    @Test
    void testFirstShotAtShipWithOneLifeIsSunk(){
        Grid grid = new Grid(3,3);
```

```

Coordinates coordinates = new Coordinates(0,0);
Ship oneLifeShip = new Ship(1, "OneLifeShip");
grid.putShip(oneLifeShip, coordinates, Orientation.HORIZONTAL);
assertThat(grid.shootAt(coordinates)).isEqualTo(ShotResult.SUNK);
assertThat(grid.hasBeenShotAt(coordinates)).isTrue();
}

@Test
void testTwoShotOnTwoCellsAtShipWithTwoLifeIsSunk(){
    Grid grid = new Grid(3,3);
    Coordinates coordinates = new Coordinates(0,0);
    Ship battleship = new Ship(2, "Battleship");
    grid.putShip(battleship, coordinates, Orientation.HORIZONTAL);
    assertThat(grid.shootAt(coordinates)).isEqualTo(ShotResult.HIT);
    assertThat(grid.shootAt(new Coordinates(1,0))).isEqualTo(ShotResult.SUNK);
}

@Test
void testSecondShotWithNoShipIsMissed(){
    Grid grid = new Grid(3,3);
    Coordinates coordinates = new Coordinates(0,0);
    assertThat(grid.hasShipAt(coordinates)).isFalse();
    grid.shootAt(coordinates);
    assertThat(grid.hasBeenShotAt(coordinates)).isTrue();
    assertThat(grid.shootAt(coordinates)).isEqualTo(ShotResult.MISSED);
}

@Test
void testSecondShotAtShipIsMissed(){
    Grid grid = new Grid(3,3);
    Coordinates coordinates = new Coordinates(0,0);
    Ship battleship = new Ship(2, "Battleship");
    grid.putShip(battleship, coordinates, Orientation.HORIZONTAL);
    assertThat(grid.hasShipAt(coordinates)).isTrue();
    grid.shootAt(coordinates);
    assertThat(grid.hasBeenShotAt(coordinates)).isTrue();
    assertThat(grid.shootAt(coordinates)).isEqualTo(ShotResult.MISSED);
}
}

```