amU Faculté des sciences Aix Marseille Université

Année 2024-25

Domaine Sciences et Technologies
LICENCE INFORMATIQUE : MATH-INFO
Initiation au Génie logiciel : TP 2
Codes UE : SIN5U34

Corrigé TP 2 : Bataille navale

Tâche 1 : Créez la classe `Ship` dans le répertoire `src/main/java` de votre projet qui répond aux spécifications ci-dessus. Votre code devra donner la Javadoc pour toutes les méthodes publiques de la classe.

```java
public class Ship {
  private int lifePoints;
  private final String name;

  /**
   * Construct a ship with the specified length and name.
   *
   * @param length the initial number of life points of the ship
   * @param name the name of the ship
   */

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

  /**
   * Return {@code true} if this ship has zero life points.
   * @return {@code true} if this ship has zero life points
   */
  public boolean hasBeenSunk(){
    return getLifePoints() == 0;
  }

  /**
   * Returns the number of life points of this ship.
   * @return the number of life points of this ship
   */
  public int getLifePoints() {
    return lifePoints;
  }

  /**
   * Decrements by one the number of life points of this ship (minimum 0).
   */
  public void takeAHit(){
    if(lifePoints>0)
      lifePoints--;
  }
  /**
   * Returns a string representation of this ship with its name and
   * its number of life points.
   * @return a string representation of this ship
   */
  @Override
  public String toString() {
    return name + " (" + lifePoints + " LP)";
  }
}
```

Tâche 2 : Créez une classe de test nommée `ShipTest` dans le répertoire `src/test/java` de votre projet qui devra vérifier via des tests unitaires la spécification du comportement de la classe `Ship`.

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

public class ShipTest {
  @Test
  public void testTakeAHit(){
    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);
  }
  @Test
  public void testTakeAHitWithZeroLifePoint(){
    Ship zeroLifeShip = new Ship(0, "Submarine");
    assertThat(zeroLifeShip.getLifePoints()).isEqualTo(0);
    zeroLifeShip.takeAHit();
    assertThat(zeroLifeShip.getLifePoints()).isEqualTo(0);
  }
  @Test
  public void testGetLifePoints(){
    Ship submarine = new Ship(3, "Submarine");
    assertThat(submarine.getLifePoints()).isEqualTo(3);
    Ship carrier = new Ship(5, "Carrier");
    assertThat(carrier.getLifePoints()).isEqualTo(5);
  }
  @Test
  public void testHasBeenSunk(){
    Ship oneLife = new Ship(1, "One life");
    assertThat(oneLife.hasBeenSunk()).isFalse();
    oneLife.takeAHit();
    assertThat(oneLife.hasBeenSunk()).isTrue();
    Ship submarine = new Ship(3, "Submarine");
    for(int index = 0; index < 3; index++) {
      assertThat(submarine.hasBeenSunk()).isFalse();
      submarine.takeAHit();
    }
    assertThat(submarine.hasBeenSunk()).isTrue();
  }
  @Test
  public void testToString(){
    Ship oneLife = new Ship(1, "One life ship");
    assertThat(oneLife.toString()).isEqualTo("One life ship (1 LP)");
    Ship submarine = new Ship(3, "Submarine");
    assertThat(submarine.toString()).isEqualTo("Submarine (3 LP)");
  }
}
```

Tâche 3 : Créez le type `ShotResult` dans le répertoire `src/main/java` de votre projet.

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

Tâche 4 : Créez la classe `Cell` dans le répertoire `src/main/java` de votre projet qui répond aux spécifications ci-dessus. Votre code devra donner la Javadoc pour toutes les méthodes publiques de la classe.

```java
public class Cell {
  private Ship ship = null;
  private boolean hasBeenShot = false;

  /**
   * Constructs a cell with no ship.
   */
  public Cell(){
  }

  /**
   * Returns the ship belonging to this cell or {@code null} if there is
   * no ship in this cell.
   *
```

```
 * @return the ship belonging to this cell
 */

public Ship getShip() {
  return ship;
}

/**
 * Puts a ship in this cell.
 *
 * @param ship the ship to be put in this cell
 */
public void setShip(Ship ship) {
  this.ship = ship;
}

/**
 * Returns {@code true} if this cell has been shot.
 * @return {@code true} if this cell has been shot
 */
public boolean hasBeenShot() {
  return hasBeenShot;
}

/**
 * Shot this cell. The result depends on the state of the cell.
 * If the cell has no ship or has already been shot then it returns
 * {@code ShotResult.MISSED}. Otherwise, the ship contained in the cell
 * takes a hit. If that ship has is sunk then {@code ShotResult.SUNK}
 * is returned and {@code ShotResult.HIT} is returned if that ship
 * is not sunk.
 *
 * @return the result of the shot
 */
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;
}
```

```java
  /**
   * Returns {@code true} if this cell contains a ship.
   * @return {@code true} if this cell contains a ship
   */
  public boolean hasShip() {
    return getShip() != null;
  }
}
```

Tâche 5 : Créez une classe de test nommée `CellTest` dans le répertoire `src/test/java` de votre projet qui devra vérifier via des tests la spécification du comportement de la classe `Cell`.

```java
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.hasShip()).isTrue();
    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.hasShip()).isTrue();
    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);
  }
  @Test
  void testSetShip(){
    Cell cell = new Cell();
    assertThat(cell.getShip()).isNull();
    Ship ship = new Ship(3, "ship");
    cell.setShip(ship);
    assertThat(cell.getShip()).isSameAs(ship);
  }
  @Test
  void testHasShip(){
    Cell cell = new Cell();
    assertThat(cell.hasShip()).isFalse();
    Ship ship = new Ship(3, "ship");
    cell.setShip(ship);
    assertThat(cell.hasShip()).isTrue();
  }

}
```

Tâche 6 : Créez la classe `Coordinates` dans le répertoire `src/main/java` de votre projet qui répond aux spécifications ci-dessus. Votre code devra donner la Javadoc pour toutes les méthodes publiques de la classe.

```java
import java.util.Objects;

public class Coordinates {
  private final int x;
  private final int y;


  /**
   * Construct coordinates in a grid.
   *
   * @param x the X coordinate of this coordinates
   * @param y the Y coordinate of this coordinates
   */
  public Coordinates(int x, int y) {
    this.x = x;
    this.y = y;
  }

  /**
   * Determines whether or not two coordinates are equal. Two instances of
   * {@code Coordinates} are equal if the values of their x and y member
   * fields are the same.
   *
   * @param o an object to be compared with this {@code Coordinates}
   * @return {@code true} if the object to be compared is an instance
   * of {@code Coordinates} and has the same values; false otherwise.
   */
  @Override
  public boolean equals(Object o) {
    if (!(o instanceof Coordinates)) return false;
    Coordinates coordinates = (Coordinates) o;
    return x == coordinates.x && y == coordinates.y;
  }

  /**
   * Returns the X coordinate of this {@code Coordinates}.
   * @return the X coordinate of this {@code Coordinates}
   */
  public int getX() {
    return x;
  }

  /**
   * Returns the Y coordinate of this {@code Coordinates}.
   * @return the Y coordinate of this {@code Coordinates}
   */
  public int getY() {
    return y;
  }

  /**
   * Returns a string representation of this {@code Coordinates}
   * with format (x, y).
   * @return a string representation of this {@code Coordinates}
   */
  @Override
  public String toString() {
    return "(" + x + ", " + y + ')';
  }

  /**
   * Returns a hash code value for this {@code Coordinates}.
   *
   * @return a hash code value for this {@code Coordinates}
   */
  @Override
  public int hashCode() {
    return Objects.hash(x, y);
  }

  /**
   * Returns a new {@code Coordinates} with the specified coordinates
   * added to this coordinates.
   * @param coordinates the coordinates to be added
   * @return the added coordinates
   */
  public Coordinates add(Coordinates coordinates){
    return new Coordinates(this.x + coordinates.x, this.y + coordinates.y);
  }
}
```

Tâche 7 : Créez une classe de test nommée `CoordinatesTest` dans le répertoire `src/test/java` de votre projet qui devra vérifier via des tests la spécification du comportement de la classe `Coordinates`.

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

public class CoordinatesTest {
  @Test
  void testGetX(){
    assertThat(new Coordinates(0,1).getX()).isEqualTo(0);
```

```java
    assertThat(new Coordinates(1,0).getX()).isEqualTo(1);
  }

  @Test
  void testGetY(){
    assertThat(new Coordinates(0,1).getY()).isEqualTo(1);
    assertThat(new Coordinates(1,0).getY()).isEqualTo(0);
  }

  @Test
  void testEquals(){
    assertThat(new Coordinates(0,1)).isEqualTo(new Coordinates(0,1));
    Coordinates coordinates = new Coordinates(0,2);
    assertThat(coordinates).isEqualTo(coordinates);
    assertThat(coordinates).isNotEqualTo(new Coordinates(0,1));
    assertThat(coordinates).isNotEqualTo(new Coordinates(1,2));
  }

  @Test
  void testToString(){
    assertThat(new Coordinates(1,2).toString()).isEqualTo("(1, 2)");
    assertThat(new Coordinates(0,12).toString()).isEqualTo("(0, 12)");
  }

  @Test
  void testAdd(){
    Coordinates coordinates1 = new Coordinates(1,2).add(new Coordinates(2,2));
    assertThat(coordinates1.getX()).isEqualTo(3);
    assertThat(coordinates1.getY()).isEqualTo(4);
    Coordinates coordinates2 = new Coordinates(1,2).add(new Coordinates(0,0));
    assertThat(coordinates2.getX()).isEqualTo(1);
    assertThat(coordinates2.getY()).isEqualTo(2);
  }
}
```

Tâche 8 : Créez le type `Orientation` dans le répertoire `src/main/java` de votre projet.

```java
public enum Orientation {
  VERTICAL, HORIZONTAL
}
```

Tâche 9 : Créez la classe `Position` dans le répertoire `src/main/java` de votre projet qui répond aux spécifications ci-dessus. Votre code devra donner la Javadoc pour toutes les méthodes publiques de la classe.

```java
public class Position {
  private final Ship ship;
  private final Orientation orientation;
  private final Coordinates firstCellCoordinates;

  /**
   * Construct a position for the specified ship, first cell
   * coordinates and orientation.
   * @param ship the ship to be positioned
   * @param firstCellCoordinates the coordinates of the first
   *                             cell of the positioned ship
   * @param orientation the orientation of the positioned ship
   */
  public Position(Ship ship, Coordinates firstCellCoordinates, Orientation orientation) {
    this.ship = ship;
    this.orientation = orientation;
    this.firstCellCoordinates = firstCellCoordinates;
  }

  /**
   * Returns an array with all the coordinates of the cells occupied
   * by the positioned ship.
   * @return the coordinates of the cells occupied by the positioned ship
   */
  public Coordinates[] shipCoordinates(){
    int length = ship.getLifePoints();
    Coordinates increment = (orientation == Orientation.HORIZONTAL) ?
            new Coordinates(1,0) : new Coordinates(0, 1);
    Coordinates coordinates = firstCellCoordinates;
    Coordinates[] shipCoordinates = new Coordinates[length];
    for(int index = 0; index < length; index++){
      shipCoordinates[index] = coordinates;
      coordinates = coordinates.add(increment);
    }
    return shipCoordinates;
  }
}
```

Tâche 10 : Créez une classe de test nommée `PositionTest` dans le répertoire `src/test/java` de votre projet qui devra vérifier via des tests la spécification du comportement de la classe `Position`.

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

public class PositionTest {
  @Test
  void testShipCoordinates_whenShipIsHorizontal(){
    Ship ship = new Ship(4, "Ship1");
    Position position = new Position(ship,new Coordinates(1,2),Orientation.HORIZONTAL);
    assertThat(position.shipCoordinates()).containsExactly(new Coordinates(1,2),
            new Coordinates(2,2), new Coordinates(3,2), new Coordinates(4,2));
  }
  @Test
  void testShipCoordinates_whenShipIsVertical(){
    Ship ship = new Ship(6, "Ship2");
    Position position = new Position(ship,new Coordinates(5,4),Orientation.VERTICAL);
    assertThat(position.shipCoordinates()).containsExactly(new Coordinates(5,4),
            new Coordinates(5,5), new Coordinates(5,6), new Coordinates(5,7),
            new Coordinates(5,8), new Coordinates(5,9));
  }
}
```

Tâche 11 : Créez la classe `Grid` dans le répertoire `src/main/java` de votre projet. Votre code devra donner la Javadoc pour toutes les méthodes publiques de la classe.

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

  /**
   * Construct a grid with the specified dimensions.
   *
   * @param width the width of the grid to be constructed
   * @param height the height of the grid to be constructed
   */
  public Grid(int width, int height){
    this.width = width;
    this.height = height;
    cells = new Cell[width][height];
    for(int column = 0; column < width; column++){
      for(int row = 0; row < height; row++){
        cells[column][row] = new Cell();
      }
    }
  }

  /**
   * Return the width of this grid.
   * @return the width of this grid
   */
  public int getWidth() {
```

```
  return width;
}
/**
 * Return the height of this grid.
 * @return the height of this grid
 */
public int getHeight() {
  return height;
}

private Cell getCell(Coordinates coordinates){
  return cells[coordinates.getX()][coordinates.getY()];
}

/**
 * Returns {@code true} if the grid contains the specified
 * coordinates.
 * @param coordinates to be tested
 * @return {@code true} if the grid contains the specified
 * coordinates
 */
public boolean contains(Coordinates coordinates){
  return 0 <= coordinates.getX()
          && coordinates.getX() < width
          && 0 <= coordinates.getY()
          && coordinates.getY() < height;
}


/**
 * 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}
```

```java
 * 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();
}

/**
 * Returns {@code true} if the cell at the specified
 * coordinates has been shot.
 * @param coordinates the coordinates to be tested
 * @return {@code true} if the cell at the specified
 * coordinates has been shot
 */
public boolean hasBeenShotAt(Coordinates coordinates){
  if(!contains(coordinates))
    return false;
  return getCell(coordinates).hasBeenShot();
}
/**
 * Returns {@code true} if the cell at the specified
 * coordinates contains a ship.
 * @param coordinates the coordinates to be tested
 * @return {@code true} if the cell at the specified
 * coordinates contains a ship
 */
public boolean hasShipAt(Coordinates coordinates){
  if(!contains(coordinates))
    return false;
  return getCell(coordinates).hasShip();
}

/**
 * Put the specified ship in the grid with the specified first
 * cell coordinates and orientation.
 * @param ship the ship to be put in the grid
 * @param firstCell the coordinates of the first cell of the ship
 *                  to be put in the grid
 * @param orientation  the orientation of the ship
 *                     to be put in the grid
 */
public void putShip(Ship ship, Coordinates firstCell, Orientation orientation){
  Position positionedShip = new Position(ship, firstCell, orientation);
  for (Coordinates coordinates : positionedShip.shipCoordinates()){
      getCell(coordinates).setShip(ship);
    }
  }

  /**
   * Returns {@code true} if all the ship contained in this grid have
   * been sunk.
   * @return {@code true} if all the ship contained in this grid have
   * been sunk
   */
  public boolean isCompleted(){
    for(int column = 0; column < width; column++) {
      for (int row = 0; row < height; row++) {
        Coordinates coordinates = new Coordinates(column,row);
        if(hasShipAt(coordinates) && !hasBeenShotAt(coordinates))
          return false;
      }
    }
    return true;
  }
}
```

Tâche 12 : Créez une classe de test nommée `GridTest` dans le répertoire `src/test/java` de votre projet qui devra vérifier via des tests la spécification du comportement de la classe `Grid`.

```java
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);
  }
  @Test
  void testGridIsCompleted() {
    Grid grid = new Grid(3, 3);
    for (int y = 0; y < 2; y++) {
      Ship battleship = new Ship(3, "Battleship");
      Coordinates coordinates = new Coordinates(0, y);
      grid.putShip(battleship, coordinates, Orientation.HORIZONTAL);
    }
    assertThat(grid.isCompleted()).isFalse();
    for (int x = 0; x < 3; x++) {
      for (int y = 0; y < 2; y++) {
        Coordinates coordinates = new Coordinates(x, y);
        grid.shootAt(coordinates);
      }
    }
    assertThat(grid.isCompleted()).isTrue();
  }
}
```

Tâche 13 : Créez la classe `GridPrinter` dans le répertoire `src/main/java` de votre projet. Votre code devra donner la Javadoc pour toutes les méthodes publiques de la classe.

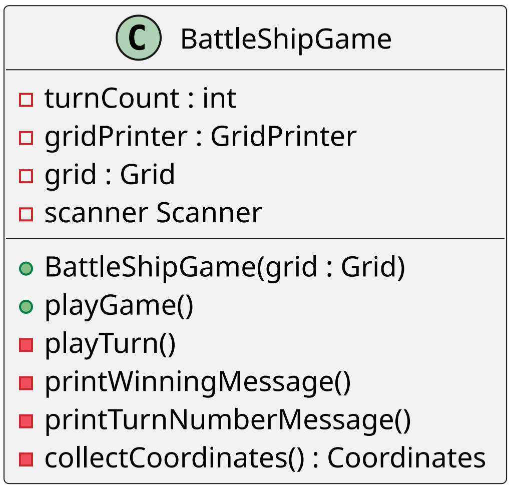
```java
public class GridPrinter {
  private static final char HIT_CHARACTER = 'H';
  private static final char MISSED_CHARACTER = 'M';
  private static final char BLANK_CHARACTER = ' ';
  private final Grid grid;

  /**
   * Construct a {@code GridPrinter} for the specified grid.
   * @param grid the grid to be printed by the {@code GridPrinter}
   */
```

```java
public GridPrinter(Grid grid) {
  this.grid = grid;
}

/**
 * Prints the grid associated to this {@code GridPrinter}.
 * For instance, a 3x3 grid should be printed as follows
 * with H for a hit and M for a miss.
 *
 * +---+---+---+---+
 * |   | A | B | C |
 * +---+---+---+---+
 * | 0 | H |   |   |
 * +---+---+---+---+
 * | 1 |   |   |   |
 * +---+---+---+---+
 * | 2 |   |   | M |
 * +---+---+---+---+
 */
public void printGrid(){
  printLine();
  printCoordinatesRow();
  printLine();
  for(int row = 0; row < grid.getHeight(); row++){
    printRow(row);
    printLine();
  }
}
private void printCoordinatesRow(){
  System.out.print('|');
  printSquare(' ');
  for (int column = 0; column < grid.getWidth(); column++){
    printSquare((char) ('A' + column));
  }
  System.out.println();
}
private void printRow(int row){
  Coordinates increment =  new Coordinates(1,0);
  System.out.print("| " + row + " |");
  for (Coordinates current = new Coordinates(0, row);
       grid.contains(current); current = current.add(increment)){
    printCell(current);
  }
  System.out.println();
}
private void printSquare(char character){
  System.out.print(" " + character + " |");
 }
 private void printCell(Coordinates coordinates){
  if (grid.hasBeenShotAt(coordinates)) {
   if(grid.hasShipAt(coordinates))
    printSquare(HIT_CHARACTER);
   else
    printSquare(MISSED_CHARACTER);
  }
  else
   printSquare(BLANK_CHARACTER);
 }
 private void printLine(){
  System.out.print("+");
  for (int column = 0; column <= grid.getWidth(); column++){
   System.out.print("---+");
  }
  System.out.println();
 }
}
```

La classe `BattleShipGame`

Spécification de la classe `BattleShipGame`

La classe représentant l'état du jeu de la bataille navale s'appelle `BattleShipGame`. C'est cette classe qui gérera les tours de jeu et l'interaction avec l'utilisateur. Ici, on considérera une version très simplifié du jeu dans lequel il n'y a qu'une grille et le but du joueur est de couler tous les bateaux en le minimum de coups.



Description des éléments de la classe (un exemple de partie est disponible à la fin de la planche de TP):

— l'attribut `turnCount` : le nombre de tours de jeu;

— l'attribut `gridPrinter` : l'afficheur de la grille;
— l'attribut `grid` : la grille du jeu;
— l'attribut `scanner` : le lecteur pour l'entrée standard, à initialiser avec `new Scanner(System.in)` (Javadoc de Scanner);
— le constructeur `BattleShipGame(grid : Grid)` qui initialise tous les attributs à partir de la grille spécifiée;
— la méthode `playGame` : fait jouer une partie, c'est-à-dire fait jouer des tours jusqu'à que la grille soit complétée;
— la méthode `playTurn` : fait jouer un tour, affiche le nombre de tours, collecte les coordonnées entrées par le joueur, puis affiche un message

— la méthode `printTurnNumberMessage` affiche le nombre de tours;
— la méthode `printShotResultMessage` affiche le résultat du tir (`You sunk a ship.`, `You hit a ship.` ou `You missed.`);
— la méthode `collectCoordinates` affiche un message demandant les coordonnées à l'utilisateur puis récupère celle-ci. On utilisera le format `B3` (une lettre de `A` à `J` suivi d'un chiffre de `0` à `9`) qu'on pourra récupérer avec `scanner.next("\\p{Upper}\\d")`, cela nous donnera une chaîne de 2 caractère à partir de laquelle on déduira des coordonnées ;
— la méthode `printWinningMessage` affiche le message de victoire.

Tâches

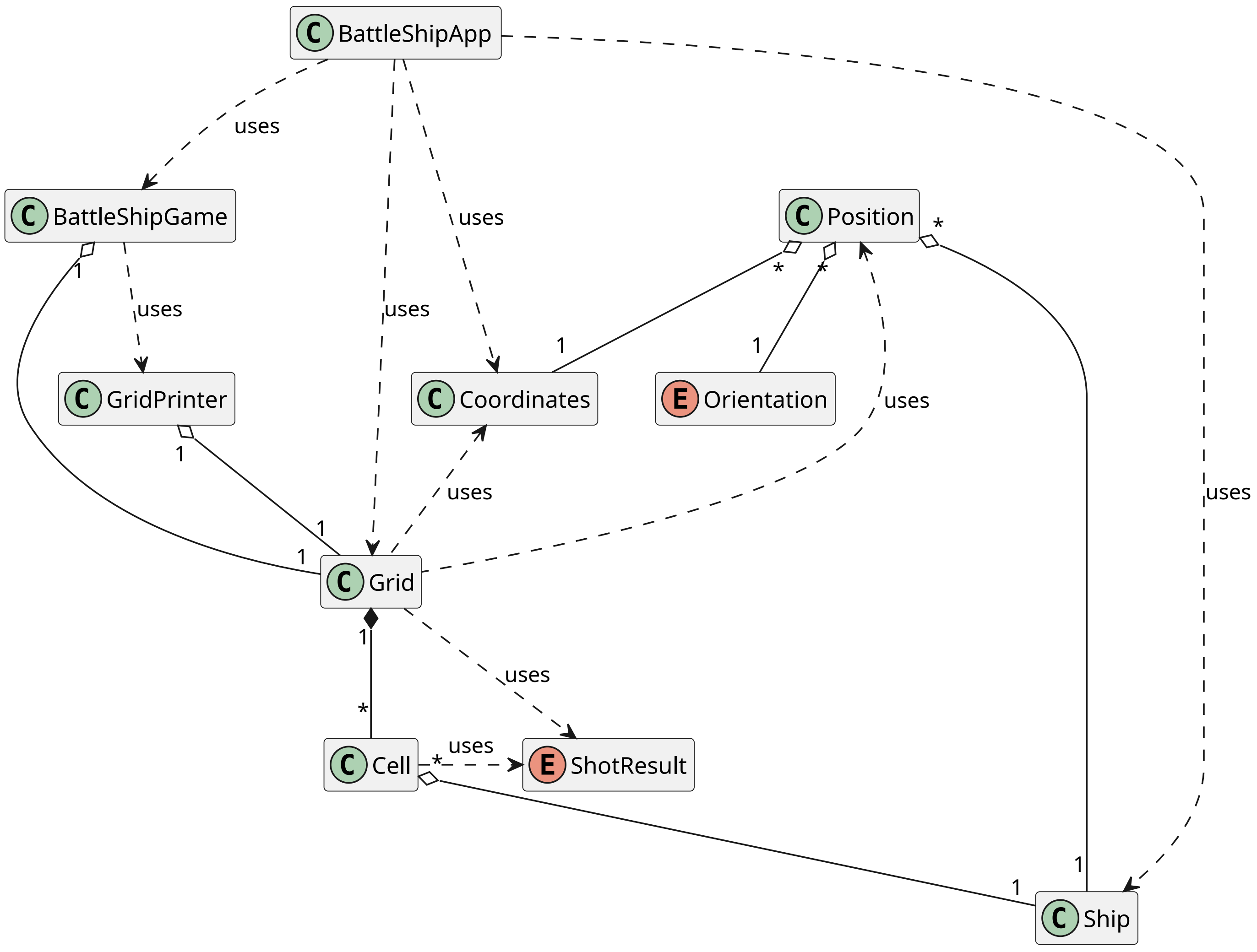
Tâche 14 : Créez la classe `BattleShipGame` dans le répertoire `src/main/java` de votre projet. Votre code devra donner la Javadoc pour toutes les méthodes publiques de la classe.

```java
import java.util.Scanner;
import java.util.regex.Pattern;

public class BattleShipGame {
  private int turnCount = 0;
  private final GridPrinter gridPrinter;
  private final Grid grid;
  private final Scanner scanner = new Scanner(System.in);

  /**
   * Constructs a {@code BattleShipGame} with the specified
   * grid as the board of the game.
   *
   * @param grid the grid used by the constructed
   *             {@code BattleShipGame}
   */
  public BattleShipGame(Grid grid) {
    this.grid = grid;
    this.gridPrinter = new GridPrinter(grid);
  }
  /**
   * Launch this game.
   */
  public void playGame(){
    while (!grid.isCompleted()){
      playTurn();
    }
    printWinningMessage();
  }
  private void printWinningMessage() {
    System.out.println("You have won in " + turnCount + " turns.");
  }
  private void playTurn(){
    gridPrinter.printGrid();
    printTurnNumberMessage();
    Coordinates coordinates = collectCoordinates();
    ShotResult result = grid.shootAt(coordinates);
    printShotResultMessage(result);
    turnCount++;
  }
  private void printTurnNumberMessage() {
    System.out.println("Turn number " + turnCount);
  }
  private static void printShotResultMessage(ShotResult result) {
    switch(result){
      case SUNK -> System.out.println("You sunk a ship.");
      case HIT -> System.out.println("You hit a ship.");
      case MISSED -> System.out.println("You missed.");
    }
  }
  private Coordinates collectCoordinates() {
    System.out.println("Enter coordinates");
    String input = scanner.next("\\p{Upper}\\d");
    int xCoordinates = input.charAt(0) - 'A';
    int yCoordinates = input.charAt(1) - '0';
    return new Coordinates(xCoordinates, yCoordinates);
  }
}
```

Tâche 15 : Créez une classe `BattleShipApp` dans le répertoire `src/main/java` de votre projet. Votre code devra donner la Javadoc pour toutes les méthodes publiques de la classe.

```java
public class BattleShipApp {
  /**
   * Starts a solo Battleship game with a 5x3 grid and a unique ship
   * of length 3 in the first line of the grid.
   *
   * @param args the parameters of the command which are ignored
   */
  public static void main(String[] args){
    Grid grid = new Grid(5,3);
    grid.putShip(new Ship(3, "Battleship"),
            new Coordinates(0,0), Orientation.HORIZONTAL);
    BattleShipGame battleShipGame = new BattleShipGame(grid);
    battleShipGame.playGame();
  }
}
```

Tâche 16 : Donnez le diagramme de toutes les classes du projet en indiquant les types de relations entre les classes et leur multiplicité. Il n'est pas nécessaire de redonner les attributs et les méthodes des classes.



BattleShipApp
BattleShipGame
GridPrinter
Coordinates
Orientation
Position
Grid
Cell
ShotResult
Ship
uses
1
*