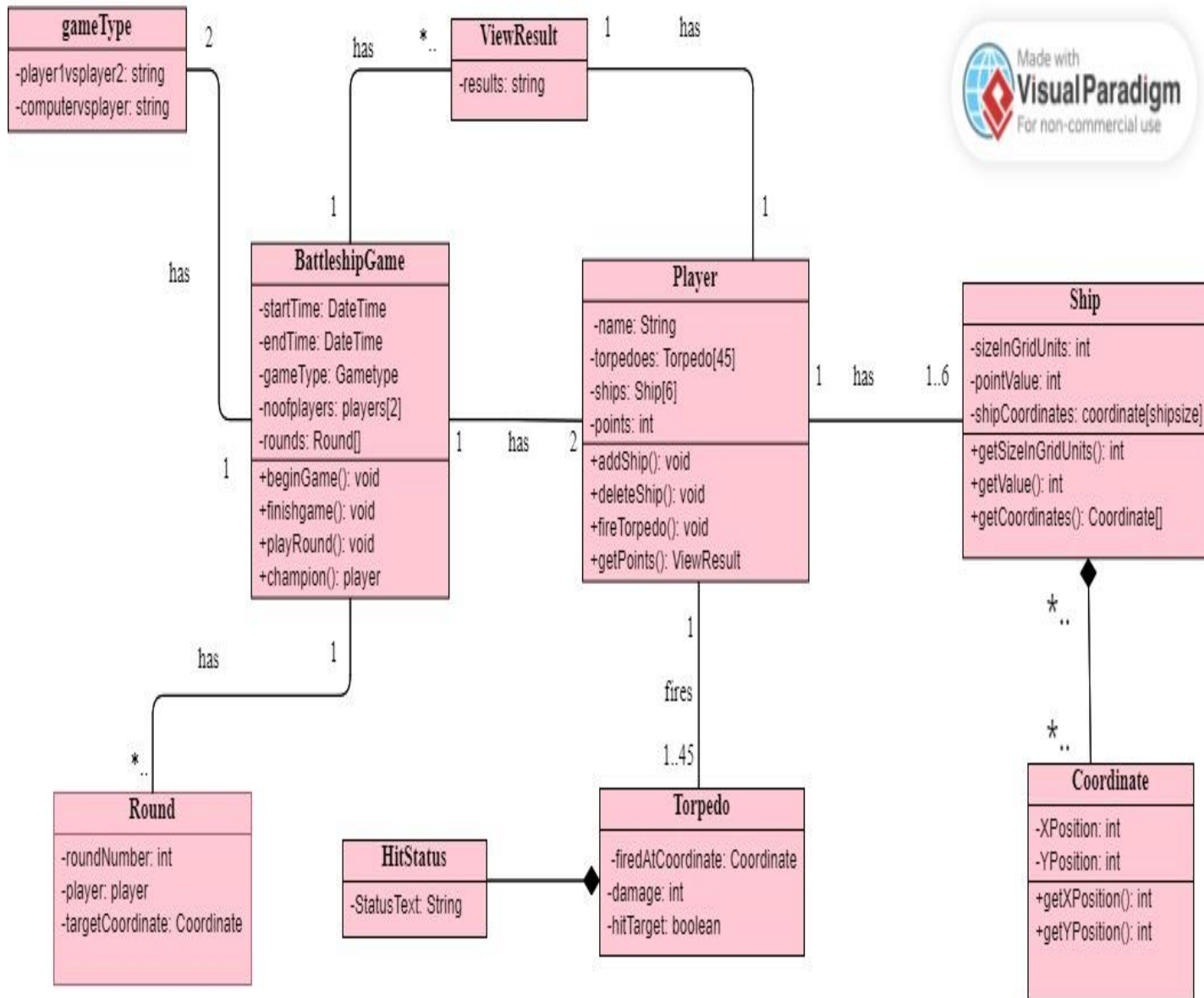


ASSIGNMENT -2

Rashmitha Ashwathappa

Neu-id: 002837410

PART-1: UML Class Diagram



PART-2 CODE:

```
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author Rashmitha Ashwathappa
 */

// Enumeration for GameType
enum GameType {
    PLAYER_VS_PLAYER,
    COMPUTER_VS_PLAYER
}

class Coordinate {
    private int x;
    private int y;

    public Coordinate(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }
}
```

```
class ViewResult {
    private String result;

    public ViewResult(String result) {
        this.result = result;
    }

    public String getResult() {
        return result;
    }
}

class Ship {
    private int sizeInGridUnits;
    private int pointValue;
    private List<Coordinate> shipCoordinates;

    public Ship(int sizeInGridUnits, int pointValue) {
        this.sizeInGridUnits = sizeInGridUnits;
        this.pointValue = pointValue;
        this.shipCoordinates = new ArrayList<>();
    }

    public int getSizeInGridUnits() {
        return sizeInGridUnits;
    }

    public int getPointValue() {
        return pointValue;
    }
}
```

```
public List<Coordinate> getShipCoordinates() {  
    return shipCoordinates;  
}  
  
public void addCoordinate(Coordinate coordinate) {  
    shipCoordinates.add(coordinate);  
}  
  
public void removeCoordinate(Coordinate coordinate) {  
    shipCoordinates.remove(coordinate);  
}  
}
```

```
class Player {  
    private String playerName;  
    private Torpedo[] torpedoes;  
    private List<Ship> ships;  
    private int points;  
  
    public Player(String playerName) {  
        this.playerName = playerName;  
        this.torpedoes = new Torpedo[45];  
        this.ships = new ArrayList<>();  
        this.points = 0;  
    }  
  
    public String getPlayerName() {  
        return playerName;  
    }  
}
```

```
public Torpedo[] getTorpedoes() {  
    return torpedoes;  
}
```

```
public void addShip(Ship ship) {  
    ships.add(ship);  
}
```

```
public void removeShip(Ship ship) {  
    ships.remove(ship);  
}
```

```
public int getPoints() {  
    return points;  
}
```

```
public HitStatus throwTorpedo(Coordinate coordinate) {  
    // Logic for throwing a torpedo and determining hit status  
    return new HitStatus("Missed"); // Placeholder  
}
```

```
private static class HitStatus {  
  
    public HitStatus(String missed) {  
    }  
}  
}
```

```
class Torpedo {  
    private Coordinate firedAtCoordinate;
```

```

    public Torpedo(Coordinate coordinate) {
        this.firedAtCoordinate = coordinate;
    }

    public Coordinate getFiredAtCoordinate() {
        return firedAtCoordinate;
    }
}

class Round {
    private int roundNumber;
    private Player player;
    private Torpedo torpedo;
    private HitStatus hitStatus;

    public Round(int roundNumber, Player player, Torpedo torpedo, HitStatus hitStatus) {
        this.roundNumber = roundNumber;
        this.player = player;
        this.torpedo = torpedo;
        this.hitStatus = hitStatus;
    }

    private static class HitStatus {

        public HitStatus() {
        }
    }
}

class BattleShipGame {
    private DateTime startDateTime;

```

```
private DateTime endDateTime;
private GameType gameType;
private Player[] players;
private List<Round> rounds;
private ViewResult viewResult;

public BattleShipGame(GameType gameType, Player[] players) {
    this.gameType = gameType;
    this.players = players;
    this.rounds = new ArrayList<>();
    this.viewResult = null; // Initialize as null
}

public void startGame() {
    // Logic for starting a game
}

public void endGame() {
    // Logic for ending a game
}

public void playRound() {
    // Logic for playing a round
}

public Player getWinner() {
    // Logic for determining the winner
    return null; // Placeholder
}
```

```
public void setViewResult(ViewResult result) {
    this.viewResult = result;
}

public ViewResult getViewResult() {
    return viewResult;
}

private static class DateTime {

    public DateTime() {
    }
}

public class Main {
    public static void main(String[] args) {
        // Create players
        Player player1 = new Player("Player 1");
        Player player2 = new Player("Player 2");

        // Create a BattleShipGame with two players
        Player[] players = { player1, player2 };
        BattleShipGame game = new BattleShipGame(GameType.PLAYER_VS_PLAYER, players)

        // Start the game
        game.startGame();

        // Add ships to players
        Ship ship1 = new Ship(3, 100);
```



```
Ship ship2 = new Ship(4, 200);
player1.addShip(ship1);
player2.addShip(ship2);
// Play a round
game.playRound()
// End the game
game.endGame();
// Determine the winner
Player winner = game.getWinner();
if (winner != null) {
    System.out.println("Winner: " + winner.getPlayerName());
} else {
    System.out.println("It's a tie!");
}
// Set and get view result
ViewResult result = new ViewResult("Game Over");
game.setViewResult(result);
System.out.println("View Result: " + game.getViewResult().getResult());
}
}
```

SNAPSHOT: Code running successfully

The screenshot displays an IDE interface with the following components:

- Projects Panel (Left):** Shows a project named 'Umlclass' with sub-packages 'Source Packages' and 'Test Packages'. The 'Main.java' file is selected under 'Source Packages'.
- main - Navigator (Bottom Left):** Displays the class hierarchy. The 'Main' class is selected, showing its methods: 'Main()' and 'main(String[] args)'. The 'GameType' class is also visible, showing its methods: 'GameType()', 'valueOf(String name): GameType', and 'values(): GameType[]'.
- Source Editor (Center):** Displays the code for 'Main.java'. The code is as follows:

```
201 Player player2 = new Player(playerName: "Player 2");
202
203 // Create a BattleShipGame with two players
204 Player[] players = { player1, player2 };
205 BattleShipGame game = new BattleShipGame(gameType: GameType.PLAYER_VS_PLAYER, players);
206
207 // Start the game
208 game.startGame();
209
210 // Add ships to players
211 Ship ship1 = new Ship(sizeInGridUnits: 3, pointValue: 100);
212 Ship ship2 = new Ship(sizeInGridUnits: 4, pointValue: 200);
213 player1.addShip(ship: ship1);
214 player2.addShip(ship: ship2);
215
216 // Play a round
217 game.playRound();
218
219 // End the game
220 game.endGame();
221
222 // Determine the winner
```
- Output - Umlclass (run) (Bottom Right):** Displays the output of the program. The output is as follows:

```
run:
It's a tie!
View Result: Game Over
BUILD SUCCESSFUL (total time: 0 seconds)
```