**ShipType**

- relativePositions : Position
- captainsQuarters : Position
- partOfShip(p : Position) : boolean

**Ship**

- position : Position
- hits : Position
- attack(position: Position) : ShipAttackResult
- sonar(position: Position) : Position

**Fleet**

- sonarPulseAttacks : Int
- sunkSomeEnemyShip : boolean
- attack(position: Position) : FleetAttackResult
- sonar(position: Position) : Position
- turn() : boolean

**FleetAttackResult**

- Hit
- Miss
- Surrender

**SimpleFleetAttackResult**

- Hit
- Miss
- Surrender

**Position**

- x: Int
- y: Int

**ShipAttackResult**

- Hit
- Miss
- Sunk

assume any link without cardinality is 1..1

```
player1: Fleet
player2: Fleet
: Ship
: ShipType

alt

turn()

attack(position)

loop (for each ship)

attack(position)

partOfShip(relativePos)

isCaptainsQuarters(relativePos)

the attack result

sonar(position)

loop (for each ship)

sonar(position)

partOfShip(relativePos)

revealed positions

is winner
```

The sequence diagram shows the turn() method for each player, followed by the attack(position) method for each ship. The sonar(position) method reveals positions, and the partOfShip(relativePos) method determines if the relative position is part of the ship. The loop (for each ship) iterates through each ship, and the isCaptainsQuarters(relativePos) method checks if the relative position is the captain's quarters. The result of the attack is determined, and the is winner method is called to determine the winner.