```
#1 Mark for including all classes given by the UML class diagram, with correct inheritance structure (ETO)
#1 Mark if all class attributes are made private/public/protected as specified by the UML class diagram
#1 Mark if all class methods are made private/public/protected as specified by the UML class diagram
~~from abc import ABCMeta, abstractmethod~~
from random import randint

class Board():
    #1 Mark for defining a constructor for the class Board with appropriate attributes
    def __init__(self, width, height, number):
        self.__columns = width
        self.__rows = height
        self.__board = [["~"]*self.__columns for i in range(self.__rows)]
        self.__playerNumber = number

    def display(self, number):
        firstLine = "-"
        for c in range(self.__columns):
            if c < 9:
                firstLine += ("| " + str(c+1) + " ")
            else:
                firstLine += ("|" + str(c+1) + " ")
        firstLine += "|"
        print(firstLine)

        #1 Mark for displaying hits, misses, ships and unshot locations on the board (ETO)
        for r in range(self.__rows):
            print(str(chr(r+65)), end='')
            for x in self.__board[r]:
                #1 Mark for hiding the locations of the opponent's ships (ETO)
                if self.__playerNumber != number and x == "S":
                    y = "~"
                else:
                    y = x
                print("| " + y + " ", end="")
            print("|")

    #1 Mark for creating relevant accessor methods to access Board's private attributes
    def getWidth(self):
        return self.__columns
```

- Displays board
- Hides relevant ships
- Adjusts for diff sizes

```
    def getHeight(self):
        return self.__rows

    #1 Mark for implementing the takeShot method as described
    def takeShot(self, row, column):
        if self.__board[row][column] == "." or self.__board[row][column] == "X":
            return "Invalid"
        elif self.__board[row][column] == "S":
            self.__board[row][column] = "X"
            return "Hit"
        else:
            self.__board[row][column] = "."
            return "Miss"
```

-Takes shot at opponent
- Returns outcome

```
    #1 Mark for overriding placeShip to work with either a human player or CPU player (ETO)
    def placeShip(self, size, number, player="CPU"):
        #1 Mark for looping until valid input is given (ETO)
        while True:
            columnSet = False
            rowSet = False
            orientationSet = False

            if player == "Human":
                self.display(number)

            #1 Mark for getting a valid location on the board (ETO)
            while not columnSet:
                if player == "Human":
                    try:
                        column = int(input("Enter the column where you would like to position the ship (1-"
+ str(self.__columns) + "):"))
                        print()
                        if column >= 1 and column <= self.__columns:
                            column = column - 1
                            columnSet = True
                        else:
                            print("That column doesn't exist. Please try again.")
                    except:
                        print("That column doesn't exist. Please try again.")
                else:
                    column = randint(0, self.__columns -1)
                    columnSet = True

            while not rowSet:
                if player == "Human":
                    try:
                        #1 Mark for accepting letter input and converting into appropriate row (A is row
1/board[0], C is row 3/board[2] etc.) (ETO)
                        row = ord(input("Enter the row where you would like to position the ship (A-" +
str(chr(self.__rows+65)) + "):").upper())
                        print()
                        if row >= 65 and row <= self.__rows+65:
                            row = row-65
                            rowSet = True
                        else:
                            print("That row doesn't exist. Please try again.")
                    except:
                        print("That row doesn't exist. Please try again.")
                else:
                    row = randint(0, self.__rows -1)
                    rowSet = True

            validPos = True

            #1 Mark for getting the orientation of the ship (ETO)
            while not orientationSet:
                if player == "Human":
                    orientation = input("Do you want to place your ship vertically down or horizontally to
the right(v/h)?:")
                    print()
                else:
                    if randint(0,1) == 0:
                        orientation = "v"
                    else:
                        orientation = "h"

                if orientation.lower() == "v" or orientation.lower() == "vertical":
                    orientationSet = True
                    try:
                        for r in range(row, row + size):
                            if self.__board[r][column] == "S":
                                validPos = False
                        if validPos == True:
                            for r in range(row, row + size):
                                self.__board[r][column] = "S"
                            return
                    except:
                        pass

                elif orientation.lower() == "h" or orientation.lower() == "horizontal":
                    orientationSet = True
                    try:
                        for c in range(column, column + size):
                            if self.__board[row][c] == "S":
                                validPos = False
                        if validPos == True:
                            for c in range(column, column + size):
                                self.__board[row][c] = "S"
                            return
                    except:
                        pass

                else:
                    print("You can only position your ship vertically down (v) or horizontally to the 
right(h)!")
            if player == "Human":
                print("You can't position the ship like that! Try again (The ship is " + size + "tiles 
long):")

    #1 Mark for implementing the checkWinner method as described
    def checkWinner(self):
        for r in range(self.__rows):
            for c in range(self.__columns):
                if self.__board[r][c] == "S":
                    return False
        return True
```

-Checks if all ships have been sunk by assuming theres a winner unless it finds an "S" in the 2D array.

```
class Player(~~metaclass=ABCMeta~~):
    #1 Mark for defining a constructor for the class Player with appropriate attributes
    def __init__(self, number, width, height):
        self._playerNumber = number
        self._playerBoard = Board(width, height, number)
        self._placeShips()

    #1 Mark for creating relevant accessor methods to access Player's private attributes
    def getNumber(self):
        return self._playerNumber

    def getBoard(self):
        return self._playerBoard

    #1 Mark for defining appropriate abstract methods
    ~~@abstractmethod~~
    def _placeShips(self):
        pass

    ~~@abstractmethod~~
    def takeShot(self, board):
        pass

    ~~@abstractmethod~~
    def _getColumn(self):
        pass

    ~~@abstractmethod~~
    def _getRow(self):
        pass
```

setup for sub classes to override

Inheritance

```
class HumanPlayer(Player):
    #1 Mark for implementing the placeShips method as described (ETO)
    def _placeShips(self):
        print("Position your carrier (5 tiles long):")
        self._playerBoard.placeShip(5, self._playerNumber, "Human")
        print("Your carrier is in position!")
        print("Position your battleship (4 tiles long):")
```

Polymophism

```
        self._playerBoard.placeShip(4, self._playerNumber, "Human")
        print("Your battleship is in position!")
        print("Position your cruiser (3 tiles long):")
        self._playerBoard.placeShip(3, self._playerNumber, "Human")
        print("Your cruiser is in position!")
        print("Position your submarine (3 tiles long):")
        self._playerBoard.placeShip(3, self._playerNumber, "Human")
        print("Your submarine is in position!")
        print("Position your destroyer (2 tiles long):")
        self._playerBoard.placeShip(2, self._playerNumber, "Human")
        print("Your destroyer is in position!")
        print()
```

Polymorphism

```
    #1 Mark for implementing the takeShot method as described
    def takeShot(self, board):
        shotMade = False
        while not shotMade:
            column = self._getColumn(board)
            row = self._getRow(board)
            result = board.takeShot(row, column)
            if result == "Invalid":
                print("You've already shot that target, aim somewhere else!")
            else:
                shotMade = True
                print(result)
```

Polymorphism

```
    #1 Mark for implementing the getColumn method as described
    def _getColumn(self, board):
        while True:
            try:
                column = int(input("Enter the column you would like to target (1-" + str(board.getWidth())
+ "):"))
                if column >= 1 and column <= board.getWidth():
                    return column - 1
                else:
                    print("That column doesn't exist. Please try again.")
            except:
                print("That column doesn't exist. Please try again.")
```

Polymorphism

```
  #1 Mark for implementing the getRow method as described
  def _getRow(self, board):
    while True:
      try:
        row = ord(input("Enter the row you would like to target (A-" +
str(chr(board.getHeight()+64)) + "):").upper())
        print()
        if row >= 65 and row < board.getHeight() + 65:
          return row - 65
        else:
          print("That row doesn't exist. Please try again.")
      except:
        print("That row doesn't exist. Please try again.")
```

Inhertance

```
class ComputerPlayer(Player):
```

Polymorphism

```
  #1 Mark for implementing the placeShips method as described (ETO)
  def _placeShips(self):
    print("The computer is positioning its ships...")
    self._playerBoard.placeShip(5, self._playerNumber)
    self._playerBoard.placeShip(4, self._playerNumber)
    self._playerBoard.placeShip(3, self._playerNumber)
    self._playerBoard.placeShip(3, self._playerNumber)
    self._playerBoard.placeShip(2, self._playerNumber)
    print("The computer has positioned its ships!")
```

Polymorphism

```
  #1 Mark for implementing the takeShot method as described
  def takeShot(self, board):
    shotMade = False
    while not shotMade:
      column = self._getColumn(board)
      row = self._getRow(board)
      result = board.takeShot(row, column)
      if result != "Invalid":
        shotMade = True
        print(result)
```

Polymorphism

```
  #1 Mark for implementing the getColumn method as described
  def _getColumn(self, board):
    return randint(0, board.getWidth()-1)
```

Polymorphism

```
    #1 Mark for implementing the getRow method as described
    def _getRow(self, board):
        return randint(0, board.getHeight()-1)

def main():
    widthSet = False
    heightSet = False

    while not widthSet:
        try:
            width = int(input("Enter the width of your game board (10-26):"))
            print()
            if width >= 10 and width <= 26:
                widthSet = True
            else:
                print("The width must be an integer from 10-26. Please try again.")
        except:
            print("The width must be an integer from 10-26. Please try again.")

    while not heightSet:
        try:
            height = int(input("Enter the height of your game board (10-26):"))
            print()
            if height >= 10 and height <= 26:
                heightSet = True
            else:
                print("The height must be an integer from 10-26. Please try again.")
        except:
            print("The height must be an integer from 10-26. Please try again.")

    player1 = HumanPlayer(1, width, height)
    player2 = ComputerPlayer(2, width, height)
    board1 = player1.getBoard()
    board2 = player2.getBoard()

    while True:
        print()
        print("It's your turn:")
        makeShot = False

        while not makeShot:
            result = input("Would you like to take a shot(1), look at the computer's board(2), or look at 
your board(3)?:")
            print()
            if result == "1":
                makeShot = True
            elif result == "2":
                board2.display(player1.getNumber())
            elif result == "3":
                board1.display(player1.getNumber())
            else:
                print("That is not a valid option!")

        board2.display(player1.getNumber())
        player1.takeShot(board2)

        if board2.checkWinner():
            print()
            board2.display(player1.getNumber())
            input("You have won!")
            return

        print()
        print("It's the computer's turn:")
        player2.takeShot(board1)
        board1.display(player1.getNumber())
        if board1.checkWinner():
            print()
            board1.display(player1.getNumber())
            input("You have lost!")
            return

if __name__ == '__main__':
    main()
```