The University of Melbourne

School of Computing and Information Systems

SWEN20003 Object Oriented Software Development Semester 2 Mid-semester Test

Length: This paper has 8 pages including this cover page.

Authorised materials: None

Time: 45 minutes, with 5 minutes reading time

Instructions to students: This exam is worth a total of 40 marks and counts for 10% of your final grade. Please answer all questions in the provided spaces on the test page; you may use the additional space provided for rough work. Please write your student ID in the space below. The test may not be removed from the test venue.

Advice: You do not need to write comments, but your answers must be **legible**; if we can't read it, we can't mark it. All worded answers must be written in English, and all code questions answered in Java. Make sure you read the **entire** test before starting.

Student ID:

Examiner's use only:

Q1	*Q2*	*Q3*	*Q4*

Define

Q1. Give **brief** answers for the following questions. [7 MARKS]

(a) What is the difference between a primitive and a class? [1 MARK]

Primitives are just data/values, objects are collections of data that can perform actions/computations.

(b) What is *boxing* and *unboxing*? [1 MARK]

Boxing is when a primitive is converted to its equivalent wrapper class (i.e. int to Integer), and unboxing is the reverse.

(c) Why should mutable instance variables be *copied* when returned? [1 MARK]

Copying prevents external objects from accessing an object's reference, preventing privacy leaks.

(d) When methods and variables have **no** privacy modifier, who can access them? [1 MARK]

Methods and variables with no modifiers have package visibility, meaning only classes from the same package can access them by name.

(e) What is an *abstract* class? [1 MARK]

An abstract class is one that is used to represent common information between subclasses, but itself doesn't have information to be instantiated.

(f) Write one line of code that defines a constant with value 10, to represent the maximum size of an array. [1 MARK]

```
public static final int MAX_ELEMENTS = 10;
```

(g) Write one line of code that sorts `students`, an array of `Student` objects. [1 MARK]

```
Arrays.sort(students);
```

Design

Q2. Answer the following questions by providing a *class design* (classes, attributes, and methods) that models the important **data** in the scenario.

UML notation is *preferred*, but all answers that appropriately represent relationships, data types, privacy, and other important information will be accepted. You **do not** need to model a "main" class. [10 MARKS]

(a) Your team is building a new system to record details of artwork across Australia, and you are in charge of modelling the search systems.

A piece of art is defined by its title, the name of its creator/artist, its value, and the museum where it resides.

A museum is defined by its name, all the artwork that it holds, and the value of all its artwork.

Artwork can be displayed in *at most* one museum, while a museum can own and store any number of works of art. [4 MARKS]

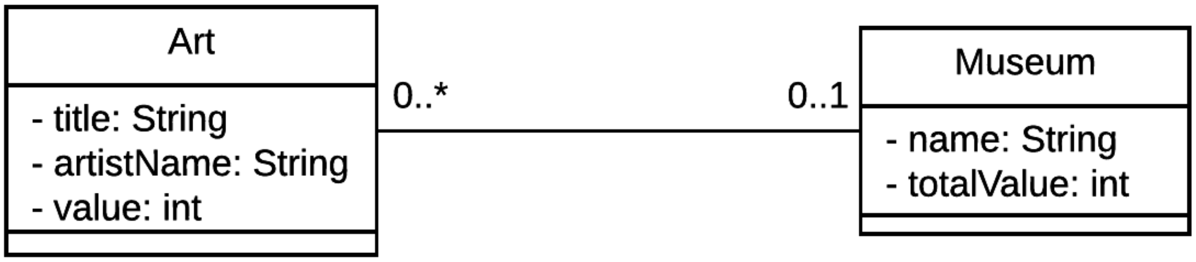

Figure 1: UML model of *Australian Art* system

(b) The Australian Football League (AFL) are rebuilding their fantasy football software, and you are tasked with its design.

A football game has up to 50 people involved, who each have a name, and a number.

These people may be: players, who also record the number of goals scored; umpires, who record the number of whistles blown; and coaches, who record the number of injuries. Some players are also captains, and record the number of players on their team. [6 MARKS]

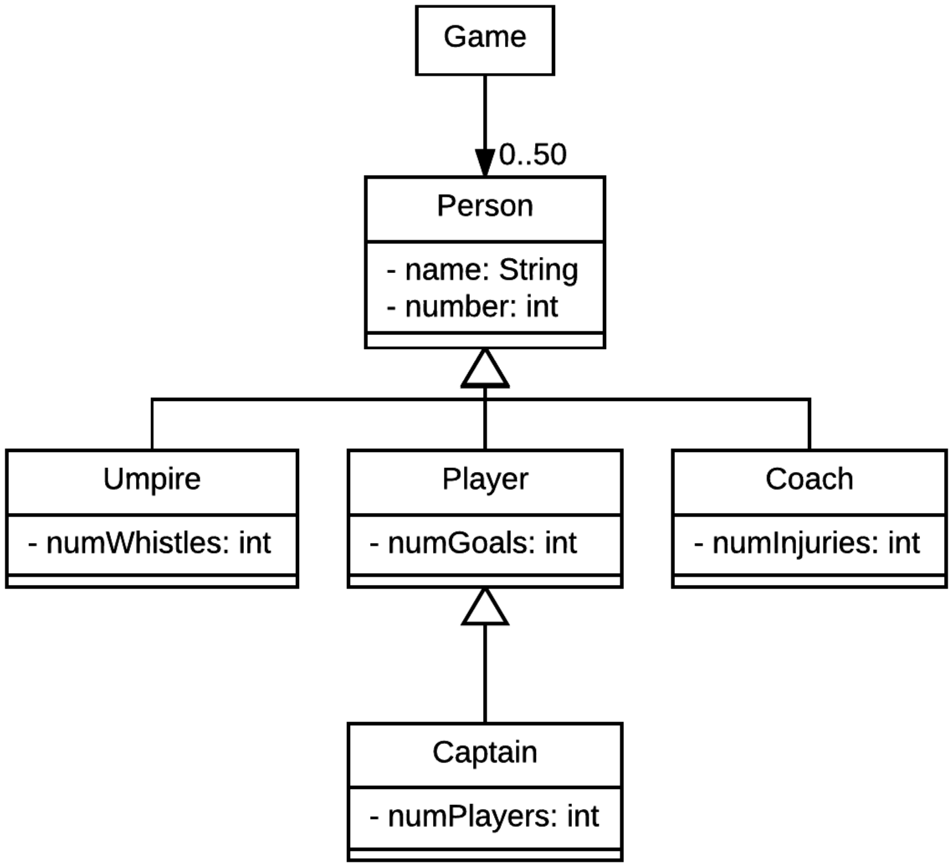

Figure 2: UML model of *AFL* system

Continued overleaf ...

Develop

Q3. In this question we will step you through implementing *classes* and *methods* in Java for an Object Oriented game called *ConnectN*, where two players try to make lines of length N on a 2D board.

You can assume that the classes or methods in earlier questions "exist" in later questions, even if you haven't answered the question. You must follow proper Object Oriented Design principles, and Java programming conventions. [23 MARKS]

(a) Implement a `Row` class, including constructor. A row contains a line of `n` characters, as well as a counter for how many cells in the row are filled. Each character should be initialised to a space (i.e. it is "empty") when the object is created. [3 MARKS]

```
public class Row {

    private static final char EMPTY = ' ';

    private char characters[];

    private int numFilled = 0;

    public Row(int n) {
        characters = new char[n];

        for (int i = 0; i < n; i++) {
            characters[i] = EMPTY;
        }
    }

}
```

(b) Implement a `Board` class, including constructor. A board contains `n` rows and a number `winLength`, the number of characters needed to win the game. [3 MARKS]

```
public class Board {

    private Row rows[];

    private int winLength;

    public Board(int n, int winLength) {
        this.winLength = winLength;

        rows = new Row[n];

        for (int i = 0; i < n; i++) {
            rows[i] = new Row(n);
        }
    }

}
```

(c) Implement a `Player` class, including constructor. A player is defined by whether they are red or yellow. [1 MARKS]

```java
public class Player {

    private boolean isRed;

    public Player (boolean isRed) {
        this.isRed = isRed;
    }

}
```

- Instance variable name and type (0.5 marks)
- Constructor defined and initialises instance variables (0.5 marks)

(d) Implement an **immutable** `Move` class, including constructor. It should contain the *index* of the row where the player is making a move. [2 MARKS]

```java
public class Move {

    public final int index;

    public Move(int index) {
        this.index = index;
    }

}
```

(e) Implement the following method for the `Row` class:

`public void makeMove(Player player)`, which represents a player placing a piece in the row. After making a move, the *leftmost unfilled* cell in the row should now contain the player's character: 'R' if the player is red, 'Y' if they are yellow. [2 MARKS]

```java
public void makeMove(Player player) {

    this.characters[numFilled] = player.isRed() ? 'R' : 'Y';

    numFilled += 1;

}
```

(f) Implement the following method for the `Board` class:

`public void makeMove(Player player, Move move)`, which represents a player placing a piece on the board. After making a move, the selected row should now contain the player's character in the leftmost unfilled cell. [1 MARK]

```java
public void makeMove(Player player, Move move) {

    this.rows[move.index].makeMove(player);

}
```

(g) Implement the following method for the `Board` class:

`public boolean isValidMove(Move move)`, which returns `true` if the move is valid. A move is valid if it is on the board, and if the row selected still has unfilled cells. [2 MARKS]

```
public boolean isValidMove(Move move) {

    return move.row >= 0 && move.row < this.rows.length
            && this.rows[move.row].getNumFilled() < this.rows.length;

}
```

(h) Finally, implement the following method for the `Board` class:

`public boolean gameOver(Player current, Move move)`, which returns true if the move just played results in a line of `current` player's characters of length `winLength` on the board.

You may assume that the Board class has the following methods:

`private boolean columnComplete(Player current, Move move)`, which returns true if there is a `winLength` sequence of identical characters in a single **column**.

`private boolean diagonalComplete(Player current, Move move)`, which returns true if there is a `winLength` sequence of identical characters in a single **diagonal**.

Note that there is no method to check whether a sequence of characters was completed in a **row**; you must write this functionality yourself. [3 MARKS]

```
public boolean gameOver(Player current, Move move) {

    return columnComplete(current, move) ||
        rowComplete(current, move) ||
        diagonalComplete(current, move);

}

public boolean rowComplete(Player current, Move move) {

    if (rows[move.index].getNumFilled() < winLength) {
        return false;
    }

    char characters[] = rows[move.index].getCharacters();

    char colour = current.isRed() ? 'R' : 'Y';

    for (int i = numFilled - winLength; i > 0 && i < numFilled; i++) {
        if (characters[i] != colour) {
            return false;
        }
    }

    return true;

}
```

(i) Using what you have implemented in the previous questions, write a main method such that a single game of ConnectN can be played between two players, one red, and one yellow.

You may assume that the Player class has the following method:

`public Move makeMove(Board board)`, returns a move where the player will place a piece.

Continued overleaf ...

You may also assume that the Board class has the following method:

`public boolean isFull()`, returns true if none of the rows have empty spaces left.

The main method should create a board and two players. Players should alternate making moves until either `gameOver` or `isFull` returns true. There are no prompts in this game. When the game is over, if there is a winner, the winning player's colour gets printed (i.e. "Red player wins!"). Otherwise the game should say "Tie!". [6 MARKS]

```
public static void main(String args[]) {

    Board board = new Board(10, 4);

    Player red = new Player(true);
    Player yellow = new Player(false);

    boolean redTurn = true;

    while (true) {

        Player current = redTurn ? red : yellow;

        Move move = current.makeMove(board);

        if (board.isValidMove(move)) {
            board.makeMove(current, move);
        }

        if (board.gameOver(current, move)) {
            String colour = current.isRed() ? "Red" : "Yellow";
            System.out.format("%s player wins!", colour);
            break;
        } else if (board.isFull()) {
            System.out.print("Tie!");
            break;
        }

        redTurn = !redTurn;

    }

}
```

— End of Test —