Timestamp: 05/10/2020 10:281

CSC1016S Assignment 4

UML and Arrays

Introduction

This assignment offers practice in OO thinking and in the application of Java technologies.

The first exercise involves OO analysis and writing UML. The second, third and fourth involve constructing OO programs in Java (i.e. writing class declarations that define types of object that contain other objects), expanding the repertoire of technologies to include arrays, and 'static' variables and methods (class members).

The second exercise is the last in our evolving Car Park simulation. The third and fourth concern construction of a simple guessing game called "Cows and Bulls".

Exercise 1 [30 marks]

This question concerns the development of a UML class diagram to represent a given scenario.

Scenario

The Western Province Athletics Association (WPAA), founded in 1965, organises road running races throughout the year. The name, type, date, time and location of each race is published in advance. Sometimes races share a date, but they never occur on the same date and in the same location.

In the world of road racing, multi day ultra-marathons of 4800 kilometres or more are not unknown. The longest race that the WPAA hosts is 100 kilometres. The shortest is 5 kilometres. The most common are 10km races and half marathons (21.097km).

Races are only open to association members. (You must be 14 years or older to join.) Further, entry to a race may be restricted to athletes who, based on the results of previous races, can run the distance within a certain time.

At a WPAA race, timing begins when the starting gun is fired and ends when the race completes: when (a) a predefined 'cut off' time is reached or (b) all athletes have crossed the finish line. Cut off time depends on the type of race (marathon, half-marathon, or whatever).

Times are recorded down to hundredths of seconds. For identification purposes each competitor wears a race bib bearing a unique number. Bib numbers are in the range 99 to 9999 inclusive and are assigned on a per race basis.

Up to two times are recorded to calculate each athlete's race result: their finishing time, naturally, but also the time, according to the race clock, at which they *cross* the start line. Depending on the size of the race, an individual can experience a significant delay. The two timings enable the actual or 'net time' for the distance to be determined.

Though the ranking of finalists is based solely on finishing time, known as 'gun time', net time, as a more accurate measure of performance, is useful to the athletes themselves. It is also used in the context of restricted entry races to identify athletes that qualify.

When the WPAA publishes the results of a race on their website they give the gun time, bib number, athlete name and age. To this end, the WPAA keeps a record of each member's name, address AND date of birth.

Continued

Task

Construct a model of the WPAA universe in the form of a UML class diagram:

- Identify classes of object and their attributes.
- Determine associations between classes; identify where aggregation is appropriate; and include multiplicities.
- Add enough methods to support:
  - An athlete entering/registering for a race and obtaining a bib number
  - Recording an athlete crossing the start line (given bib number, time).
  - Recording an athlete crossing the finish time (given bib number, time).
  - Obtaining a results list i.e. a list of objects that indicate the ranking of finalists, and from which (via methods) name, age, etc can be obtained either directly or indirectly.
  - Obtaining an individual athlete's result; their gun time or net time.
  - Determining that an athlete qualifies for a restricted race based on previous results.
- Method signatures should be enough to document your design i.e. should have meaningful names, parameter names, and return types (if any).

You can assume the existence of an appropriate Duration class of object.

Good design principles to apply:

- Try to make sure that each class of object maps to a single concept and thus has a cohesive collection of related attributes and methods. To give an example, in a shopping application, we have products and we have orders comprising items where each order item identifies a quantity of a product. Item is distinct from product – it is associated with a product but does not contain product attributes or methods.
  Another way to think of this is "separation of concerns".
- Avoid duplicating data – for instance, an athlete's bib number for a race shouldn't be stored in more than one place/object.

NOTE: whereas in the Car Park simulation, we had a clock class of object, don't concern yourself with one here – just assume that times and bib numbers are fed in via method parameters

Submit your UML class diagram in the form of a PDF document called "*WPAA.pdf*".

In addition to the UML diagramming software given in the lecture notes, we can also recommend "Software Ideas Modeler".

To help develop a sense of the task we've provided a similar problem along with its solution in the appendices.

Exercise 2 [20 marks]

Completing the CarParSim solution that you constructed for the previous assignment, you need to:

- Add code to create and populate a TariffTable object.
- Augment the code for the 'depart' command to include printing the tariff that applies to a stay of that duration.
- Add code for an additional 'tariffs' command: When the user enters the 'tariffs' command, a list of parking tariffs will be printed.

Continued

You will find a ZIP file containing all the other simulator components (Ticket, Clock, TariffTable etc) on the Vula page for the assignment.

Here is some more sample I/O:

```
Car Park Simulator
The current time is 00:00:00.
Commands: tariffs, advance {minutes}, arrive, depart, quit.
>advance 10
The current time is 00:10:00.
>arrive
Ticket issued: Ticket[id=80000001, time=00:10:00].
>advance 1
The current time is 00:11:00.
>arrive
Ticket issued: Ticket[id=80000002, time=00:11:00].
>arrive
Ticket issued: Ticket[id=80000003, time=00:11:00].
>depart 80000003
Ticket details: Ticket[id=80000003, time=00:11:00].
Current time: 00:11:00.
Duration of stay: 0 minutes.
Cost of stay : R10.00.
>depart 80000006
Invalid ticket ID.
>depart 80000001
Ticket details: Ticket[id=80000001, time=00:10:00].
Current time: 00:11:00.
Duration of stay: 1 minute.
Cost of stay : R10.00.
>tariffs
[0 minutes .. 30 minutes] : R10.00
[30 minutes .. 1 hour] : R15.00
[1 hour .. 3 hours] : R20.00
[3 hours .. 4 hours] : R30.00
[4 hours .. 5 hours] : R40.00
[5 hours .. 6 hours] : R50.00
[6 hours .. 8 hours] : R60.00
[8 hours .. 10 hours] : R70.00
[10 hours .. 12 hours] : R90.00
[12 hours .. 1 day] : R100.00
>quit
Goodbye.
```

Continued

Exercise 3 [20 marks]

This question concerns the construction of a NumberUtils class declaration that contains a collection of useful routines.

Write a class declaration that satisfies the following specification:

Class NumberUtils

The NumberUtils class contains a collection of routines for working with integers.

Instance variables

None

Constructors

private NumberUtils() {}

// A private, empty-bodied constructor prevents NumberUtil objects from being created.

Methods

public static int[] toArray(int number)

// Given a number that is n *digits in length, maps the digits to an array length* n.
// e.g. given the number 5678, the result is the array {5, 6, 7, 8}.

public static int countMatches(int numberA, int numberB)

// Given two numbers, count the quantity of matching digits – those with the same value and
// position. For example, given 39628 and 79324, there are 2 digits in common: x9xx2x.
// It is assumed that the numbers are the same length and have no repeating digits.

public static int countIntersect(int numberA, int numberB)

// Count the quantity of digits that two numbers have in common, regardless of position.
// For example, given 39628 and 97324, there are 3 digits in common: 3, 7, 2.
// It is assumed that the numbers are the same length and have no repeating digits.

You should make a simple test program (which you do not need to submit) to check your code.

Exercise 4 [30 marks]

This question concerns the construction of a simple game called ‘Cows and Bulls’ that is implemented using the NumberUtils class of question three.

The game involves guessing a mystery 4-digit number. The number contains no repeat digits and no zero.

With each guess, the number of correct digits – bulls – and the number of digits that would be correct if they were in the right position – cows.

For example, say the mystery number is 8657 and the player guesses 4678, there is one bull (6), and two cows (7, 8).

Continued

Class CowsAndBulls

CowsAndBulls implements the logic for a cows and bulls guessing game. The player has

Constants

public final static int NUM_DIGITS = 4;
public final static int MAX_VALUE = 9876;
public final static int MIN_VALUE = 1234;
public final static int MAX_GUESSES = 10;

Constructors

public CowsAndBulls(int seed)
    // Create a CowsAndBulls game using the given randomisation seed *value to generate*
    // a mystery number of NUM_DIGITS *length, and that gives the player* MAX_GUESSES *guesses.*

public int guessesRemaining()
    // Obtain the number of guesses remaining.

public Result guess(int guessNumber)
    // Evaluates a guess that the mystery number is guessNumber*, returning the outcome in the form*
    // of a Result object. Decrements guesses remaining.
    // Assumes that game is not over.

public int giveUp()
    // End the game, returning the secretNumber.

public boolean gameOver()
    // Returns true if (i) the secret number has been guessed, or (ii) there are no more guesses.

On the Vula web page you will find a ZIP file containing resources you can use:

- A NumberPicker class that you should use in the CowsAndBulls constructor to generate a mystery number,
- A Result class that you should use to implement the CowsAndBulls guess method.
- A Game program that completes the game; implementing user interaction.

NOTE:

- the seed value supplied to the CowsAndBulls constructor should be fed to the NumberPicker object that it creates. The purpose of this value is to make the (pseudo) random number generation repeatable i.e. the same seed value causes the same mystery number to be generated.
- To generate a mystery number, create a NumberPicker with the seed value and a range 1-9 (inclusive). Get a digit, multiply by 10 and add a digit, multiply by 10 and add a digit, …

Running the Game program produces the following sorts of interaction.

Sample I/O

```
Your challenge is to guess a secret 4-digit number.
Enter randomisation seed value:
10
Make a guess:
5678
Sorry that's incorrect.
You have 3 cows and 1 bull.
You have 9 guesses remaining
```

Continued

```
Make a guess:
5687
Sorry that's incorrect.
You have 2 cows and 2 bulls.
You have 8 guesses remaining
Make a guess:
8657
Correct!
```

Sample I/O (abbreviated)

```
Your challenge is to guess a secret 4-digit number.
Enter randomisation seed value:
5
Make a guess:
1234
Sorry that's incorrect.
You have 0 cows and 2 bulls.
You have 9 guesses remaining
Make a guess:
…
…
4569
Sorry that's incorrect.
You have 2 cows and 0 bulls.
You have 1 guess remaining
Make a guess:
4537
Sorry, you lose.
```

Submission

Submit the `CarParkSim.java, NumberUtils.java, CowsAndBulls.java` and `WPAA.pdf` files to the automatic marker.

Continued

Appendices

Sample UML modelling problem and solution

Exercise

This exercise concerns the development an OO representation of the employees of the Fynbos Flower Company from the point of view of timekeeping. Consider the following English language description:

Scenario

The Fynbos Flower company is flexible on the hours that employees work, and as a consequence, timekeeping is an important administrative practice. The company records the dates and times of shifts (for want of a better word). Employees sign-in when they arrive and sign-out when they leave (and only work one shift per day).

This information supports a range of enquiries

- whether an employee is present,
- whether an employee was present on a given day,
- the details of the shift worked on a given day,
- the total hours worked during a given week.
- the shifts worked during a given week.

Though, currently, none of the employees bear the same name, the company assigns each a unique ID number that may be used to disambiguate.

The concept of a "week" bears elaboration. A business week starts on a Monday and ends on a Sunday. The weeks are numbered. The first week of the year is the one that contains the first Thursday of the year.

The task is to develop a design for an Employee class along with whatever supporting classes are found necessary (such as a Shift class), documented in the form of a UML class diagram.

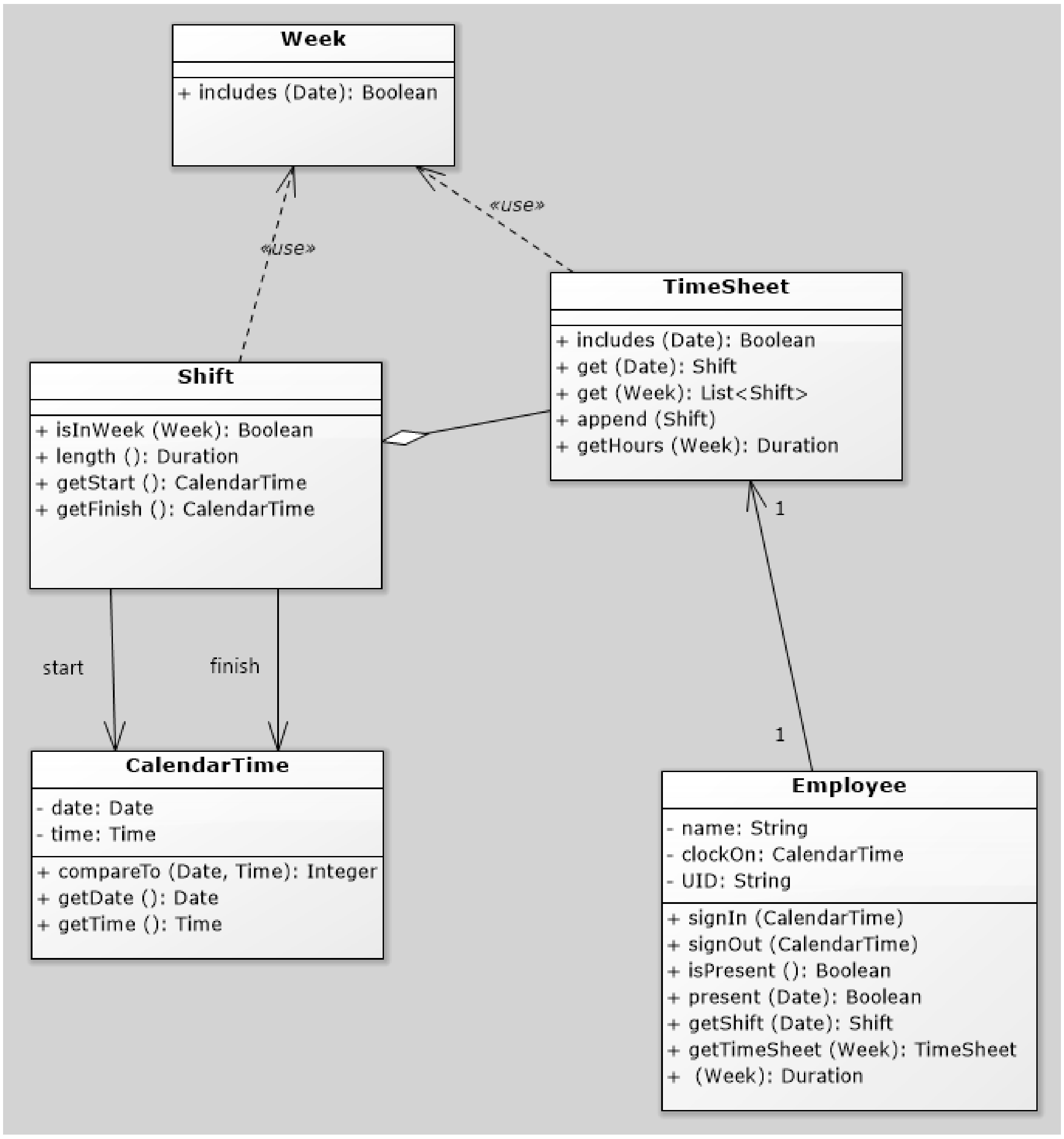
- Give attribute declarations.
- Give method signatures and descriptions of behaviour.
- Show associations, including aggregations where appropriate. Include multiplicities. Navigation optional.
- Aim for a **RICH** representation – think about exercise one, which aims for a richer representation of Money that that provided by integers or double.

Start with the Employee class and think about the variables and methods required to support the enquiries listed.

- What are the responsibilities of an object of this class?
- How are responsibilities fulfilled? *This question will lead to other classes and other responsibilities*

Continued

Solution: UML Class Diagram



END