CSC1016S Assignment 4: Arrays and Lists

Assignment Instructions

This assignment involves constructing programs in Java using object composition (i.e. class declarations that define types of object that contain other objects), arrays and lists, and the use of class or 'static' variables and methods.

Question one involves using a class declaration in a way that is similar to the way we construct Python modules – as a means of grouping useful resources. It also involves practicing using arrays in Java.

Question two involves constructing a simple guessing game type of object using the class declaration of question one. It also demonstrates the use of class members to declare things that are related to the type, but no single instance.

Question three involves constructing a class to represent an athlete and their spit times for a race. It provides an introduction to Java lists and iterators.

Exercise One [35 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 Two [30 marks]

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

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).

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 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
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 guesses remaining
Make a guess:
4537
Sorry, you lose.
```

Exercise Three [35 marks]

This question concerns the construction of classes of object that model the results of a running race such as:

Race number	Waypoint				
	2 km	*4 km*	*6 km*	*8 km*	*10 km*
104578	00:06:57	00:13:54	00:20:59	00:28:48	00:36:30
458114	00:11:24	00:22:48	00:34:12	00:55:36	01:05:05
000072	00:09:11	00:19:48	00:29:41	00:36:30	00:45:30
634921	00:09:54	00:18:39	00:27:18	00:39:36	00:49:30

The table depicts the split times for athletes that participated in a 10 km race.

- Each row contains the data for a particular athlete.
  - Their race number appears at the left.
  - Their split times are to the right of their race number.
- A *split time* is the overall time taken to reach a given *waypoint* in the race.
  - There are 5 way points for this race: 2 km, 4 km, 6 km, 8 km, and 10 km.
  - Each athlete has 5 split times: their time at the 2 km point, their time at the 4 km point, and so on.
- Times are recorded as a quantity of hours, minutes and seconds.
- Waypoints are ALWAYS integer quantities of kilometres, and race numbers are always six digits long.

Your task is to develop an Athlete class of object, i.e. a class of object that represents a single row of data in the results table, and along the way, learn about Java lists and iterators.

NOTE: On the Vula page for this test you will find:

- a Time class
- a SplitTime class

Development is broken up into stages. For each, we will describe the Java technology required and give a specification for parts of the Athlete class.

Part One [10 marks] *(Appears as question 3 on automatic marker)*

The initial specification for the Athlete class is as follows:

Class Athlete

An Athlete object represents an athlete from the perspective of their performance in a particular race.
It stores their race number and a list of their split times.

Constructor

```
public Athlete(String raceNumber, ArrayList<SplitTime> splitTimes)
        // Create an Athlete object that represents the athlete with the given race number
        // and list of split times (ordered by distance).
```

Methods

```
public String getRaceNumber()
        // Obtain the race number.
```

The class will use a list instance variable to store split times.

Java list classes may be found in the '`java.util`' package. There are two main classes: 'LinkedList', and 'ArrayList'.

- The LinkedList class, as its name suggests, is implemented as described in lectures - it uses a chain of nodes to store items.
- The ArrayList class, as its name suggests, is implemented using an array.

Though the number of values that can be stored in an array is fixed, the number of values that can be stored in an ArrayList is not. The array in an ArrayList object is dynamically resized as needed.

Generally, a LinkedList performs better than an ArrayList when it comes to adding and removing values. An ArrayList, however, performs better on getting and replacing (setting) values.

LinkedList also proves additional methods that allow it to be used as a queue (covered in future lectures).

Both classes are *generic*. When you declare a variable or formal parameter, or when you create an instance, you must give the type of object that is going to be stored e.g.

```
ArrayList<Money> payments;
payments = new ArrayList<Money>();
LinkedList<String> names = new LinkedList<String>();
```

When creating a list object, you can fill it will the contents of another list (or in fact any Java collection object), for example, assume we've added the names "Jamieson", "Daya" and "Kouassi" to the list referred to by '`names`':

```
ArrayList<String> guests = new ArrayList<String>(names);
```

This gives you enough information to start the Athlete class declaration.

- You should use ArrayList. You'll need an import statement.
- You'll need an instance variable for race number, and one for a list of split times.
- In the constructor, you should create a new list object that contains the contents of that given, assigning it to your split times instance variable.
- You should implement the `getRaceNumber()` method.

Next, we want to add the following methods:

Methods

```
public int getNumTimes()
        // Obtain the number of split times recorded for this Athlete (i.e. length of split times list.)

public String toString()
        // Obtain a String representation of this object in the form
        // "<race number> [<split time>, <split time, …, <split time>]".
        // A split time consist of an integer, followed by a minus sign, followed by a time.
        // e.g. "104578 [00:06:57@2, 00:13:54@4, 00:20:59@6]".
```

For `getNumTimes()`, given a list, you can obtain the size, i.e. the number of items within it, using the `size()` method. For example, the following will output '`3`' to the screen:

```
System.out.println(guests.size());
```

To clarify the behaviour expected of the `toString()` method, say we have an Athlete object, `ath`, that has the split times 00:06:57 at 2 km, and 00:13:54 at 4 km, the statement "`System.out.println(ath.toString());`" should produce the output:

```
[00:06:57@2, 00:13:54@4]
```

Fortunately, this is easy, the text is exactly what you get if you perform `toString()` on an ArrayList of SplitTime objects.

At this point, you have completed part one and will be able to submit to the automatic marker, however, it's good practice to check your work before submission. To do this, you're going to need a test program. Here's a suitable candidate:

```
public static void main(String[] args) {
    ArrayList<SplitTime> t = new ArrayList<SplitTime>();
    t.add(new SplitTime(new Time(0, 6, 57), 2));
    t.add(new SplitTime(new Time(0, 13, 54), 4));
    Athlete athlete = new Athlete("104566", t);
    System.out.println(athlete.getNumTimes());
    System.out.println(athlete);
}
```

The code will output `2` and then `104566 [00:06:57@2, 00:13:54@4]`.

Besides being a test harness, it also serves to show how to create an ArrayList of SplitTimes and fill it with values.

Adding an item to a list is simple, we use the `add()` method.

To give a slightly less cluttered example, in the context of the earlier code fragments, `guests.add("Kabanda")` appends "Kabanda" to the end of the list comprising "Jamieson", "Daya" and "Kouassi".

Part Two [10 marks] *(Appears as question 4 on automatic marker)*

Reinforcement: you don't need any additional list technology for this problem. Modify the Athlete class by adding a constructor that satisfies this specification:

Constructor
public Athlete(String dataLine)
// Create an Athlete object using the data in the given String.
// The String should have the form <race number> [<split time>, ..., <split time>]
// i.e. a race number, followed by a space, followed in square brackets by a list
// of zero or more comma separated split time strings.
// e.g. "131005 [00:06:57@2, 00:10:05@4]"

HINT: consider using resources such as the java.util.Scanner class and the String class `substring()`, `indexof()`, and `trim()` methods.

You should extend the simple test program to check your work before submission – you might want to get it to ask the user to input test data so that you can quickly test a variety of behaviours e.g.

Sample I/O

```
Enter a line of athlete data:
131005 [00:06:57@2, 00:10:05@4]
Creating Athlete object.
```

```
Printing Athlete object:
131005 [00:06:57@2, 00:10:05@4]
Done.
```

Part Three [10 marks] *(Appears as question 5 on automatic marker)*

We've seen how to create and insert data into a list, and how to obtain its size. Now, how do we access the elements?

Here's a challenge that requires such knowledge: modify the Athlete class by adding these methods.

Methods

public Time getTimeAtWaypoint(int distanceA)

// Obtain the time taken by the athlete to reach the waypoint at the given distance.
// Returns null if there is no data for the given distance.

public Time getWaypointInterval(int distanceA, int distanceB)

// Obtain the time taken by the athlete to run from the waypoint at distance A
// to the waypoint at distance B.
// Assume that distanceA≤distanceB.
// Returns null if there is no data for one or other of the given distances.

For clarification, assuming an Athlete object that is referred to by a variable `ath`, and that represents the third competitor in the table on page 4, the following code fragment outputs 16:42.

```
Time t = ath.getWaypointInterval(4, 8);
System.out.println(t);
```

You can probably see that the `getWaypointInterval()` method can be built on the back of the first. For the first, we need to be able to obtain the time for the given distance within the split times list i.e. locate the relevant SplitTime object.

The most straightforward way to access an element of a list is via the `get()` method e.g.

```
String firstGuest = guests.get(0);
String lastGuest = guests.get(guests.size()-1);
```

This fragment assigns "Jamieson" to the variable `firstGuest` and "Kabanda" to the variable `lastGuest`.

If we wish to view the items of a list one-by-one though this isn't the most efficient way. A 'for-each' loop would be better. Say that we want to iterate over our `guests` list, the code looks like this:

```
for(String name : guests)
{
      System.out.println(name);
}
```

The output is:

```
Jamieson
Daya
Kouassi
Kabanda
```

The key part of the statement appears within parentheses following the '`for`' keyword. It consists of a variable declaration then a colon then a list expression.

The body of the loop will execute exactly as many times as there are items in the list. Starting with the first and stopping after the last, for each iteration, the variable `name` will contain the value of the current item.

You should extend the simple test program to check your work before submission.

As a final hint: review the features of the Time class provided.

Part Four [5 marks] *(Appears as question 6 on automatic marker)*

There's a lot going on behind the scenes of a for-each loop. It's actually what is called 'syntactic sugar'. When the Java compiler encounters a for-each loop it converts it to equivalent code that uses an iterator object.

Using our `guests` example for the last time,

```
for(String name : guests)
{
      System.out.println(name);
}
```

Is equivalent to

```
 Iterator<String> iter = guests.iterator();
 while(iter.hasNext()) {
      String name = iter.next();
      System.out.println(name);
 }
```

You can think of a Java Iterator object as a bookmark. It marks a position within the list, and can be moved along from first to last.

- Given a list, an iterator can be obtain for traversing that list by using the '`iterator()`' method.
- The iterator '`next()`' method returns an element of the list and moves along by one.
- The iterator '`hasNext()`' method returns true if there are more elements of the list to see.

Typically, we assign the iterator obtained with '`iterator()`' to a variable. Iterator is a generic type, so we must give the type of element that the '`next()`' method will be returning i.e. the type of element within the list that the iterator is being used to access. Hence in the sample fragment, we have '`Iterator<String> iter`'. The variable name is '`iter`', and it will store a reference to an Iterator of type '`Iterator<String>`'.

Typically, an Iterator is used in a while loop using '`hasNext()`' to control the number of iterations, through there many other possible applications.

Your final task is to add the following method to your Athlete class:

Methods

public Iterator<SplitTime> iterator()

// Obtain an iterator object that can be used to view split times, one after another.
// NOTE: 'Iterator' refers to the 'Iterator' class in 'java.util'.
// (HINT: To preserve data integrity, the method should create a copy of the SplitTimes list and
// return an Iterator for that.)

And complete the following test program (also available on the assignment page on Vula) by implementing the rest of the '`printAthlete`()' method using an Iterator:

```
import java.util.Iterator;
import java.util.Scanner;
public class TestPartFour {

    private TestPartFour() {}

    public static void printAthlete(final Athlete athlete) {
        System.out.printf("Athlete number: %s\n",
           athlete.getRaceNumber());
        /**
        * Your code here.
        */
    }

    public static void main(final String[] args) {
        final Scanner input = new Scanner(System.in);
        System.out.println("Enter a line of athlete data in the form
'<race number> [<split time>, ..., <split time>] ':");
        Athlete athlete = new Athlete(input.nextLine());
        TestPartFour.printAthlete(athlete);
    }
}
```

Sample I/O:

```
Enter a line of athlete data in the form '<race number> [<split
time>, ..., <split time>] ':
131005 [00:06:57@2, 00:10:05@4]
Athlete number: 131005
   2 km : 00:06:57
   4 km : 00:10:05
```

Sample I/O:

```
Enter a line of athlete data in the form '<race number> [<split
time>, ..., <split time>] ':
102346 []
No times recorded.
```

You should submit the completed test program along with your Athlete class.

Marking and Submission

You must submit your answers to the automatic marker, however, question three will be manually marked by a tutor.

Submit the *NumberUtils.java, CowsAndBulls.java, Athlete.java, and TestPartFour.java* files to the automatic marker within a single .ZIP folder. The zipped folder should have the following naming convention:

yourstudentnumber.zip

Appendix: Running Race Time and SplitTime Classes

Class Time

A Time object represents an instant of race time comprising hours, minutes and seconds. The smallest possible value is 00:00:00.

Constructor

```
public Time(int hours, int minutes, int seconds)
        // Create a Time object representing the given time in hours and minutes.

public Time(String string)
        // Create a Time object represented the time given as a string in the form h...h:mm:ss.
        // For example, "10:15:01".
```

Methods

```
public int getHours()
        // Obtain the hours component of this Time.
public int getMinutes()
        // Obtain the minutes component of this Time.
public int getSeconds()
        // Obtain the minutes component of this Time.
public int asDuration()
        // Obtain the value of this Time as a whole number of seconds.
public Time subtract(Time other)
        // Obtain the result of subtracting the other Time from this Time.

public boolean equals(Object o)
        // Determine whether this Time object is equivalent to the given object o.
        // Returns true if o is a Time object and has the same value as this Time object.
public int compareTo(Time other)
        // Compare this Time object to the Other Time object.
        // Returns a negative value if this Time precedes the other Time, 0 if they are equivalent, and
        //  a positive value if other Time precedes this Time.
public String toString()
        // Return a String representation of this Time value in h…h:mm:ss e.g. 0:01:15.
```

Class SplitTime

A SplitTime object represents a split time for an athlete in a particular race. A split time consists of a waypoint, described as a distance from the start line, and the time taken (according to the race clock) to get there.

Constructor

public SplitTime(Time time, int distance)

// Create a Time object representing the given time in hours and minutes.

public SplitTime(String string)

// Create a SplitTime object represented the time given as a string in the form h…h:mm:ss@d…d.
// For example, "10:15:01@15".

Methods

public int getDistance()

// Obtain the distance from the start line.

public Time getTime()

// Obtain the Time taken to reach this distance.

public String toString()

// Return a String representation of this Time value in h…h:mm:ss@d…d.
// For example, " 0:01:15@5".

END