Paper 2—Option D: Object Orientated Programming (OOP)

Scenario

A bus company operates in a local city. The bus company operates along fixed routes where there are marked stops and sometimes bus shelters for people to wait in out of the weather. People (passengers) pay the driver a specified fare for travel when they enter a bus.

There are many objects in this company, here are some of them:

Object	Description
Bus	A physical vehicle that carries *passengers* on a specific *route* and has a *driver*.
Passenger	A person that travels on a *bus*.
Route	A series of roads/streets the *bus* travels over from its start to its destination.
Bus Stop	A named place on a *route* where people wait for a *bus*. May be a simple marker or may have a shelter and seats.
Driver	A person qualified to drive a *bus* and trained to drive it over a given *route*.

These two objects have already been defined for the Bus Company:

BusRoute
Integer: route String: start
setRoute(Integer: route) setStart(String start) Integer getRoute() String getStart() String toString()

Bus
Integer: id String: driver BusRoute: route
setId(Integer: id) setDriver(String driver) setBusRoute(BusRoute: route) Integer getId() String getDriver() BusRoute getBusRoute() String toString()

The `toString()` method returns a String implementation of an object.

These are implemented in code as follows:

```
public class BusRoute
{
  private int route;
  private String start;
  public BusRoute(int r, String s)
  {
    setRoute(r);
    setStart(s);
  }
  public void setRoute(int r){ route = r; }
  public void setStart(String s){ start = s; }
```

```
  public int getRoute(){ return route; }
  public String getStart(){ return start; }
  public String toString()
  {
    return "Route: " + route + " start: " + start;
  }
}



public class Bus
{
  private int id;
  private String driver;
  private BusRoute route;
  public Bus(int i, String d, BusRoute r)
  {
    setId(i);
    setDriver(d);
    setRoute(r);
  }
  public void setId(int i){ id = i; }
  public void setDriver(String d){ driver = d; }
  public void setRoute(BusRoute r){ route = r; }
  public int getId(){ return id; }
  public String getDriver(){ return driver; }
  public BusRoute getRoute(){ return route; }
  public String toString()
  {
    return "Bus id:" + id + " - " + driver + ": " + route.toString();
  }
}
```

SL/HL Core

D1 (a) Explain the term **parameter variable**, using an example from the code. [2 marks]

A parameter variable is one that is passed to a method, an example being *setDriver(**String Driver**)* within the Bus object, with the String Driver being the variable.

(b) Describe **one** additional field that might have been included in the BusRoute class. Include data types and sample data. [2 marks]

An additional 'finish' (destination) field could be added. This could either be in String form (e.g. "Chelsea"), or as an Integer, with the number representing the corresponding location. Altering the system would be made easier if an Integer system were to be used, as many routes could terminate at the same location (and so only one entry would need amending).

(c) Identify the output produced by the following code fragment: [2 marks]

```
Bus bus = new Bus(1001, "N Prakesh", new
BusRoute(431, "Klang"));
System.out.println(bus.toString());
```

The code fragment would produce the output below:

```
Bus id: 1001 - N Prakesh: Route: 431 start: Klang
```

Consider the code fragment below:

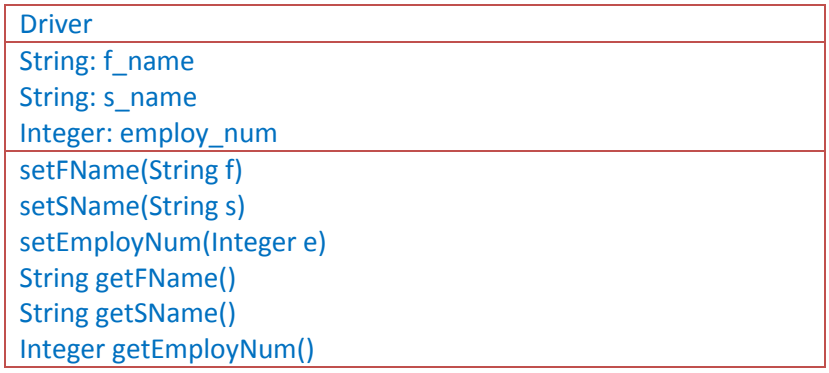
```
private static final int MAX_BUSES = 12;
private Bus[] buses = new Bus[MAX_BUSES];
buses[0] = new Bus(1001,
                   "N Prakesh",
                   new BusRoute(431, "Klang"));
buses[1] = new Bus(1010,
                   "J Carey",
                   new BusRoute(342, "Tanglin"));
buses[2] = new Bus(1014,
                   "H Lee",
                   new BusRoute(411, "Queenstown"));
buses[3] = new Bus(1015,
                   "K Peters",
                   new BusRoute(319, "Jamaica Street"));

showBusDrivers(buses, 1010);
```

(d) Construct the method `showBusDrivers(Bus[] b, int n)` which lists the drivers for all buses with a route number less than or equal to the parameter variable (n). [6 marks]

```
Public String showBusDrivers(Bus[] b, int n) {
   for(int x = 0; x < MAX_BUSES; x ++) {
      if(b[x].getRoute().getRoute() < n) {
         System.out.println(b[x].getDriver() + "\n"
      }
   }
}
```

The company wishes to keep track of its drivers in more detail, including first and last name and employee number – this is a 4-digit whole number

(e) Construct a suitable diagram for this Driver object. [3 marks]



D2 In relation to the Bus example:

(a) Outline how encapsulation is used.. [2 marks]

Encapsulation is where an Object uses methods to restrict access from outside its class and to prevent methods from outside the class accidentally altering data within the class. As an example, the Bus object contains the private variables int id, String driver, and BusRoute route which cannot be directly accessed by other classes/methods.

(b) Outline a disadvantage of using Object Oriented Design. [2 marks]

To construct an Object Oriented program is difficult and for many smaller tasks potentially unnecessary.

(c) Explain how a programming team could benefit from an Object Oriented Design approach. [4 marks]

An Object Oriented design approach allows a team to divide a job up into a number of smaller tasks. For example, each member of the team could work on an individual object (e.g. one could work on the Bus object, whilst the other works on BusRoute).
So long as the basic fundamentals of each Object are known, ie. Its classes (their variables, and what they return), the other members of the team can get on with their work.

Recall that a *Bus Stop* is one of many **named** places on a *route* where *buses* stop to pick up or drop off *passengers*. It may or may not have a **shelter** to protect *passengers* from the weather. The **distance** in Km from the start point of the *Bus Route* is important information for planning.

(d) Design the Bus Stop Object using a simple object diagram. [3 marks]

BusStop
String name Boolean shelter Double distance
setName(String n) setShelter(Boolean s) setDistance(Double distance) String getName() Boolean getShelter() Double getDistance()

(e) Suggest how Bus Stop information for a given Bus Route instance could be stored, giving both sample data and sample code fragments to show how it could be implemented. [4 marks]

The BusRoute class could be stored as [??] an array of Bus stops e.g.

```
BusStop[] stops_arr = new BusStop[x];
stops_arr[0] = new BusStop("Name", false, 3.50);
```

The example parameter variables are as follows:
"Name" represents the Bus Stop name
false provides the Boolean value for whether a shelter exists
3.50 is a double figure to indicate the distance from the start point.

Firstly a BusStop array is created, of size x. Each element would then be entered into the array (the element at key 0 is shown entered above).

D3 The company grows, offers more routes of different types and decides to use three different types of bus:

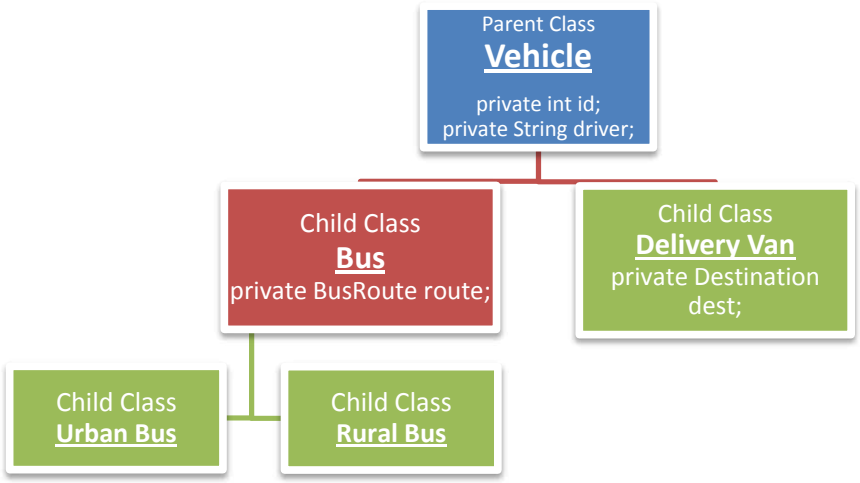
A bus that operates on busy city routes – the Urban Bus – has only a driver.

A smaller bus that operates on longer country routes – it carries an additional person to collect the fares.

A van that can be used for heavy equipment deliveries rather than passengers – the Delivery Van – it carries a co-driver and a helper.

These vehicles will have some things in common, such as a *driver*, and other elements that are different, for example both Urban and Rural buses will operate on a fixed *route* whereas the Delivery Van will take equipment to specified *destinations* (such as factories or other businesses).

(a) Construct diagrams to show how you would re-design the Bus class to implement inheritance. [8 marks]



(b) Explain **the** advantage of inheritance for this situation [4 marks]

Repetition could be avoided: Structuring the design in this way would allow each different vehicle to inherit the same fundamental qualities (such as an ID and a driver) from the parent class whilst giving it individual properties through separate child classes. Both the Rural and Urban bus may share additional properties and so perhaps a further class 'Bus' could be created to contain these. Not having duplicate code is particularly helpful when mistakes within a program are found.

A method is required in the subclasses that returns the number of employees per vehicle.

(c) Outline how polymorphism might apply in this design. [3 marks]

Polymorphism is a technique for allowing the same name to be used for methods that function slightly differently from each other. Each class should contain a countEmployees method. The vehicle class can contain a countEmployees method which returns one. The rural bus class must override this method and return two.

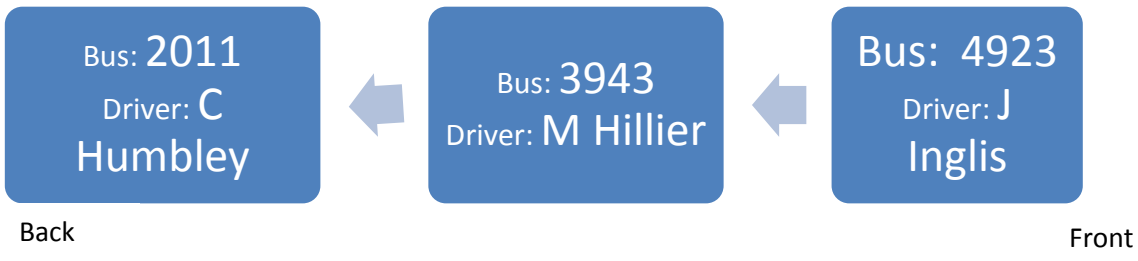
HL Extension

D4 The bus company decides to run a simulation over a particular route to see what happens when several buses are started on the route a set time apart. A queue will be used to hold the individual Bus instances.

(a) Identify three features of a queue that make it suitable for this purpose. [3 marks]
A queue uses a First in First Out (FIFO) structure which is relevant for the task.

(b) Construct a diagram of the queue after the following code has been executed: [3 marks]

```
public class BusSim
{
  private LinkedList<Bus> busQueue;

  public static void main(String[] args){ new BusSim(); }
  public BusSim()
  {
    // Create new LinkedList for Bus instances
    busQueue  = new LinkedList<Bus>();
    BusRoute route = new BusRoute(903, "Nerang Creek Road");
    Bus bus1 = new Bus(2011, "C Humbley", route);
    Bus bus2 = new Bus(3943, "M Hillier", route);
    Bus bus3 = new Bus(4923, "J Inglis", route);
    busQueue.addFirst(bus1);
    busQueue.addFirst(bus2);
    busQueue.addFirst(bus3);

  }
}
```



Recall that the method of the `LinkedList` class `remove(int index)` removes the element at the specified position in the list while the method `size()` returns the number of elements in the list.

(c) Construct the method `removeBus(int pos)` which attempts to remove a bus from the queue at the specified position, and returns true if successful and false if it fails. [4 marks]

```
Private boolean removeBus(int pos) {
   if(pos < busQueue.size() && pos > -1) {
      busQueue.remove(pos);
      return true;
   }
   else { return false; }
}
```

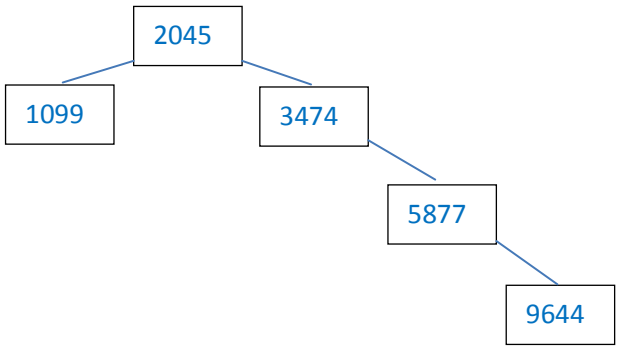
A large company might have several hundred buses running. Each one has a unique id stored with the Bus instance.

(d) Explain how a binary tree could be used to store these ids such that they can be quickly retrieved (if they exist) by a search. [3 marks]

A binary tree is a method used to reduce the time it takes to search a list for a given value. The tree orders particular values to the left and the right given an initial element (e.g. values to the left are lower, values to the right are higher). This makes searching more efficient, particularly when a large amount of data exists.

The tree stores the ids 2045, 3474, 5877, 1099, 9644.

(e) Draw a diagram of an ordered binary tree containing these keys assuming they were inserted in the order given. [5 marks]



A binary tree node may be inserted iteratively or recursively.

(f) Identify **two** disadvantages of the recursive algorithm. [2 marks]

- A recursive algorithm is harder to write
- Takes up more stack space and overflow error may occur