**The Ambulance class is described as follows:**

```
public class Ambulance{

      Private Patient[] patients;
      private int capacity; //how many Patients it can hold
      private int bedNum; //the index of the next available bed

      public Ambulance(int capacity){

            this.patients = new Patient[capacity];
            this.bedNum = 0;
      }

      public void addPatient(Patient p){
            this.patients[index] = p;
            this.bedNum = this.bedNum + 1;
      }

      public void removePatient(){
            this.patients[bedNum] = null;
            this.bedNum = this.bedNum - 1;
      }

      public String getPatientName(int bed){
            if(this.patients[bed]!=null){
                  return patients[bed].getName();
            }
      }
}
```

a) How many attributes does the Ambulance class have? [1]

b) How many accessor methods does the Ambulance class have? [1]

c) How many mutator methods does the Ambulance class have? [1]

d) In another class an Ambulance with a capacity of 10 will be constructed. Write a line of code that will construct an instance of an Ambulance with 10 beds. The instance variable will be *ambulance*.

[1]

e) A Patient has a name eg “Joan Rivers”, an age eg 56 and an ID eg “pat-01”. All of this data is sent as parameters when a Patient object is first constructed.

Using encapsulation (including data hiding), design the entire Patient class (you can limit your design to include 1 accessor and 1 mutator method):

[4]

f) A computer program in another class is as follows:

```
Patient p1 = new Patient("Jack", 34, "pat-07");
Patient p2 = new Patient("Jane", 81, "pat-05");
Patient p3 = new Patient("Jess", 39, "pat-04");
Patient p4 = new Patient("Xin", 23, "pat-08");
Patient p5 = new Patient("Sola", 75, "pat-03");
Patient p6 = new Patient("Ken", 17, "pat-12");
Ambulance a = new Ambulance(10);
a.addPatient(p1);
a.addPatient(p2);
a.addPatient(p3);
a.addPatient(p4);
a.addPatient(p5);
a.removePatient();
a.removePatient();
a.addPatient(p5);
a.addPatient(p6);
a.addPatient(p4);
a.removePatient();
```

Write down, including the index, the names of each Patient in the Ambulance object's **patients** array.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |

[1]

g) Outline the advantages of encapsulation. [2]

h) Draw UML diagrams showing the relationship between the Patient and Ambulance classes. Make sure each diagram has 3 sections. Include all properties and methods, including constructor methods for the Ambulance class, and only properties for the Patient class.

i) A new Ambulance class method needs to be written, countAge.

This method takes an integer as a parameter and counts all patients in the ambulance who are equal to or older than the parameter. It returns the count.

Assume the Patient class has an accessor method getAge().

Write the Ambulance class method, countAge

[6]

**HL extension questions**

j) What data structure could be used to help simulate the behaviour of the Ambulance class?

[1]

k) Explain your choice of data structure from the previous question.

[2]

l) Outline the purpose of the stack access method isEmpty()

[2]

m) The stack TOWNS holds several town names, and the name “Cardiff” is on the top of the TOWNS stack

An algorithm is needed that will reverse the contents of the TOWNS stack. The name “Geneva” should be on top of the TOWNS stack after reversing its contents:

**a.** The content in the `TOWNS` stack **before** it is reversed

| Cardiff |
|---|
| Washington DC |
| The Hague |
| Singapore |
| Geneva |

**b.** The content in the `TOWNS` stack **after** it is reversed

| Geneva |
|---|
| Singapore |
| The Hague |
| Washington DC |
| Cardiff |

Construct an algorithm that will reverse the `TOWNS` stack **using an empty queue**. You may assume that the `TOWNS` stack is inputted and a new empty queue named `TEMP` is initialized.

You must use stack access methods **and** queue access methods in your response. **note the following methods are usually not given in an exam question - this is for revision purposes only.*

| **Stack access methods** | **Queue access methods** |
|---|---|
| `isEmpty()` | `isEmpty()` |
| `push()` | `enqueue()` |
| `pop()` | `dequeue()` |

[5]