Option D — Object-oriented programming

An international school organizes a regional swimming competition for students from 10 different schools. Each school will send a team of 5 to 15 swimmers.

Each swimmer can enter up to 5 events (such as the “50 m freestyle” or “100 m butterfly”).

Each event consists of one or more races. A race can be a qualifying heat, or a final. The final has the best 8 swimmers from all the qualifying heats in the event.

Each race has a maximum of 8 swimmers.

The UML diagrams for the classes `Swimmer` and `Race` are provided below

`Swimmer`

`- String name`
`- String school`
`- String[5] eventID`
`-` **`double`**`[5] time`

`+ constructor`
`+ accessor and mutator methods`
`+ addTimes()`

`Race`

`- Swimmer[8] swimmer`
`-` **`double`**`[8] time`

`+ constructor`
`+ accessor and mutator methods`
`+ addSwimmers()`
`+ addTimes()`

14. (a) Define the term *mutator method*. [1]

(b) State **one** additional instance variable of type `boolean` which could be added to the class `Race` as indicated above. [1]

(c) With reference to both class UMLs provided above, distinguish between a class and an instantiation. [3]

In this scenario, `Swimmer` objects are aggregated in a `Race` object.

(d) (i) Outline **one** advantage of using aggregation in this context. [2]

(ii) Outline **one** disadvantage of using aggregation in this context. [2]

(e) Construct code for the constructor of the class `Swimmer` that instantiates an object with parameters `name` and `school`. The event IDs should be set to “empty” and the times to 0.0 [4]

Many swimmers in the event have names that cannot be represented using basic character sets such as ASCII.

(f) Describe **one** feature of modern programming languages that allows the wide range of students’ names to be represented correctly. [3]

(Option D continues on the following page)

(Option D continued)

15. A generic `Event` class is defined as follows:

```
class Event
{
  private String eventID;
  private int numberOfRaces;
  private Race[] races;
  private Race finals;

  public Event(String ID, int numberOfRaces)
  {
    eventID = ID;
    races = new Race[numberOfRaces];
    for(int i = 0; i < numberOfRaces; i++)
    {
      races[i] = new Race();
    }
    finals = new Race();
  }

  public void addSwimmers()
  {
    // fills the qualifying heats with swimmers
  }

  public void fillFinals()
  {
    // fills the finals race with the best 8 from the qualifying heats
  }

  // more methods()
}
```

(a) The same method identifier `addSwimmers` is used in both classes `Race` and `Event`.

Explain why this does not cause a conflict. [3]

The `Event` class above assumes that the event has more than 8 swimmers and requires qualifying heats. However, an event with less than 9 swimmers has no qualifying heats, so the original `Event` class was inherited by a new class `FinalsOnlyEvent`.

(b) Outline **two** advantages of the OOP feature “inheritance”. [4]

(c) Outline how method overriding can help to create the new class `FinalsOnlyEvent`. [2]

(Option D continues on the following page)

Turn over

(Option D continued)

16. An `Event` has been instantiated with 2 qualifying heats for a total of 11 swimmers.

```
Event free100 = new Event("100 m free style",2);
```

The swimmers were added to the two `Race` arrays and after the races, their times were recorded as shown in the table.

(For the purpose of this question, the name represents the full swimmer object.)

`races[0]`

`swimmer`	Andy	Bella	Chris	Duc	Eric	null	null	null
`time`	34.2	33.8	40.9	36.3	34.6	0	0	0
	[0]	*[1]*	*[2]*	*[3]*	*[4]*	*[5]*	*[6]*	*[7]*

`races[1]`

`swimmer`	Fiona	George	Hetty	Idan	Jo	Karl	null	null
`time`	41.2	36.6	37.6	35.2	48.8	37.2	0	0
	[0]	*[1]*	*[2]*	*[3]*	*[4]*	*[5]*	*[6]*	*[7]*

The method `fillFinals()` will select the 8 fastest swimmers, in ascending order of time, from both `swimmer` arrays and copy them to the `swimmer` array in the `finals` race.

(a) Sketch the resulting `swimmer` array in `finals`. [3]

To help with this selection, all entries from `races[0]` and `races[1]` will be copied into two new parallel arrays of size 16, one array for swimmers and one array for their times.

(b) Construct the code fragment for the given situation that will copy swimmers and times into two parallel arrays named `tempSwimmer` and `tempTime`. [6]

(Option D continues on the following page)

(Option D, question 16 continued)

The two temporary arrays will be sorted using the following code.

```
int i,j;
Swimmer swapSwimmer;
double swapTime;
for(i = 0; i < 15; i++)
{
  for(j = 0; j < 15; j++)
  {
    if(tempTime[j] > tempTime[j + 1])         // if wrong order then…
    {
      swapSwimmer = tempSwimmer[j];           // swap the swimmer and…
      tempSwimmer[j] = tempSwimmer[j + 1];
      tempSwimmer[j + 1] = swapSwimmer;
      swapTime = tempTime[j];                 // swap the time
      tempTime[j] = tempTime[j + 1];
      tempTime[j + 1] = swapTime;
    }
  }
}
```

(c) (i) State the name of this sorting algorithm. [1]

(ii) Outline **two** improvements to this code that would make the algorithm more efficient. [4]

(d) Construct the code fragment that will copy the names of the 8 fastest swimmers in ascending order of time from the array `tempSwimmer` to the array `swimmers` in the race `finals`. [6]

(Option D continues on the following page)

(Option D continued)

17. The organizing school arranges for visiting students to stay with a host family. The information about each of the visiting students is stored in a file. The student records are sorted by name. Some of the other variables included are school, gender, age and host family name, as shown in the UML diagram below.

```
Visitor
-----------------------------------
- String name
- String school
- char gender
- int age
- String hostFamily
      … more variables
-----------------------------------
+ constructors
+ accessor and mutator methods
```

A program needs to be written to match visiting students with host families. The matching process requires data to be manipulated extensively (adding, editing, deleting). The file will be read into RAM.

This program will be used for different events with different numbers of visitors. Therefore, it will be implemented using a dynamic data structure.

(a) Define the term *object reference*. [1]

(b) Outline **one** reason why a linked list may be more suitable than a binary tree in this particular situation. [2]

It has been decided to use a single linked list named `guests` to store and manipulate the `Visitor` objects.

(c) (i) Construct the code needed to instantiate an object `guests` of the `LinkedList` class. [1]

(ii) Construct the code for the method `penultimate()` that returns the second to last element in the linked list `guests`. You may assume that `guests` is locally accessible. [4]

(Option D continues on the following page)