

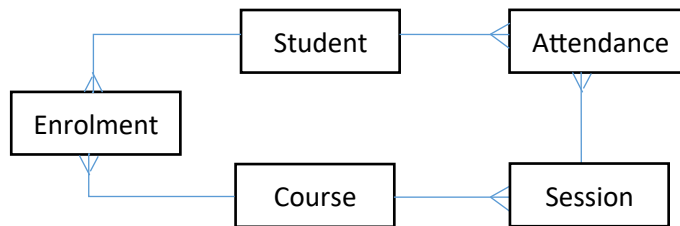
Jurong Pioneer Junior College
2026 H2 Computing Paper 1 (Syllabus 9569)
Suggested Solutions & Marks Scheme (v13)

1	(ai) <pre>ClassParticipation(<u>studentID</u>, studentName, courseID, courseName, <u>sessionID</u>, sessionDate, startTime, deliveryMode, enrolDate, status, joinTime, punctuality)</pre> 2m – all attributes from the scenario listed and composite primary key (studentID, sessionID) identified. Also accept (studentID, courseID, sessionID) as the key. SR: minus 1m for each missing or wrong key attribute, minus 1m if attributes from the scenario are omitted (minimum 0m).
	(aii) Using a single 1NF table causes data redundancy. For example (any one of the following): • The same student details (name) are repeated for every course and session they attend. • The same course details (name) are repeated for every session and for every student enrolled in that course. • The same session details (date, start time, delivery mode) are repeated for each student attending that session. • The same enrolment details (enrolment date, status) are repeated for every session of that course the student attends. OR Using a single 1NF table leads to anomaly. For example (any one of the following): • Update anomalies Updating a student's name or a course's name requires changes in many rows. Missing one update leads to inconsistent data. • Insertion anomalies A new course or session cannot be added to the table before any student has attended it, without creating a dummy participation row. • Deletion anomalies Deleting the last row for a course or session could unintentionally remove all information about that course or session.

1m – identification of one limitation

1m – explain/expand on the limitation phrased in question's context

(aiii)



1m – all entities identified correctly

2m – all relationships between 2 entities are correct

SR: minus 1m for each relationship or entity shown incorrectly (minimum 0m).

Expected: five one-to-many relationships, Student to Enrolment, Course to Enrolment, Course to Session, Student to Attendance, Session to Attendance.

(aiv)

STUDENT(studentID, studentName)

COURSE(courseID, courseName)

SESSION(sessionID, courseID, sessionDate, startTime, deliveryMode)

ENROLMENT(studentID, courseID, enrolDate, status)

ATTENDANCE(studentID, sessionID, joinTime, punctuality)

1m – all tables & PKs identified

1m – all FKs identified: courseID in SESSION; studentID and courseID in ENROLMENT; studentID and sessionID in ATTENDANCE (shown above with double underline, meaning both PK and FK; candidates should show solid plus dashed underline)

1m – composite primary keys for ENROLMENT and ATTENDANCE tables (SESSION has the single-attribute key sessionID)

(bi)

```

SELECT
    s.studentID,
    COUNT(a.sessionID) AS totalSessionsAttended
FROM
    Student AS s,
  
```

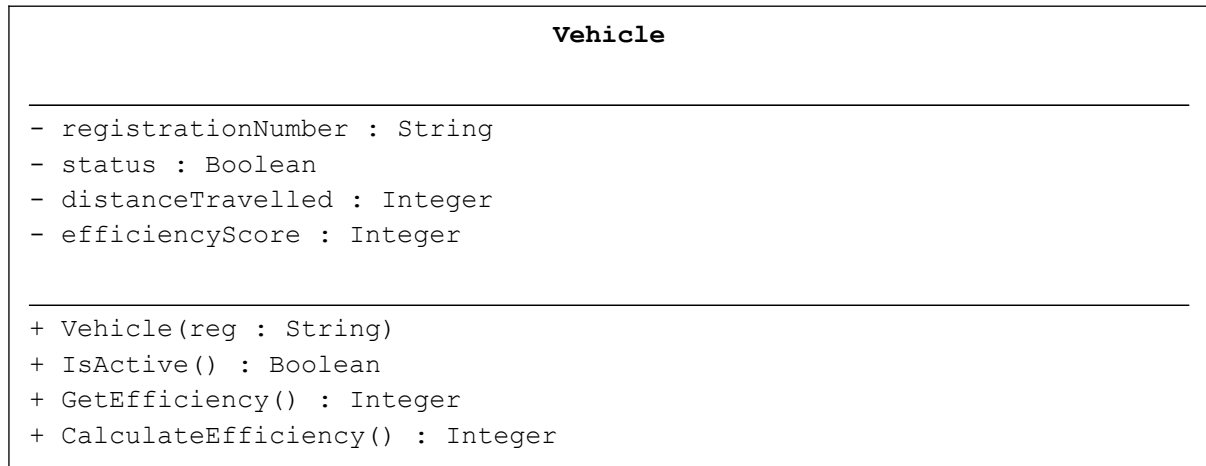
	<pre> Attendance AS a WHERE s.studentID = a.studentID GROUP BY s.studentID; </pre> 1m – SELECT studentID and COUNT(...) of sessions 1m – FROM Attendance, with or without a join to Student (a correct answer using only the Attendance table, e.g. SELECT studentID, COUNT(*) FROM Attendance GROUP BY studentID, earns full marks) 1m – GROUP BY studentID
	(bii) <pre> SELECT s.studentID, s.studentName, COUNT(a.sessionID) AS attendanceCount FROM Student AS s, Attendance AS a WHERE s.studentID = a.studentID GROUP BY s.studentID, s.studentName ORDER BY attendanceCount DESC LIMIT 3; </pre> 1m – SELECT studentID, studentName and COUNT(...) with Student joined to Attendance on studentID 1m – GROUP BY studentID, studentName 1m – ORDER BY the count in descending order and LIMIT 3 (accept TOP 3, FETCH FIRST 3 ROWS ONLY; accept ORDER BY COUNT(*) DESC)
	(ci) Either of the following: • Unlike relational databases, NoSQL systems (e.g., MongoDB) can scale horizontally. The huge write loads, as more users stream classes, can be distributed across many low-cost, lower-end servers. • The flexible schema allows new fields such as additional device or network metrics to be added without altering a rigid relational schema or performing costly migrations.

	1m – for identification of advantage 1m – for explaining the advantage
	(cii) A relational database is still preferable for JPStream when storing structured, business-critical data such as student enrolments, course information, and official attendance records. These require strict referential integrity to ensure correctness. 1m – type of data 1m – why the type of data requires RDBMS

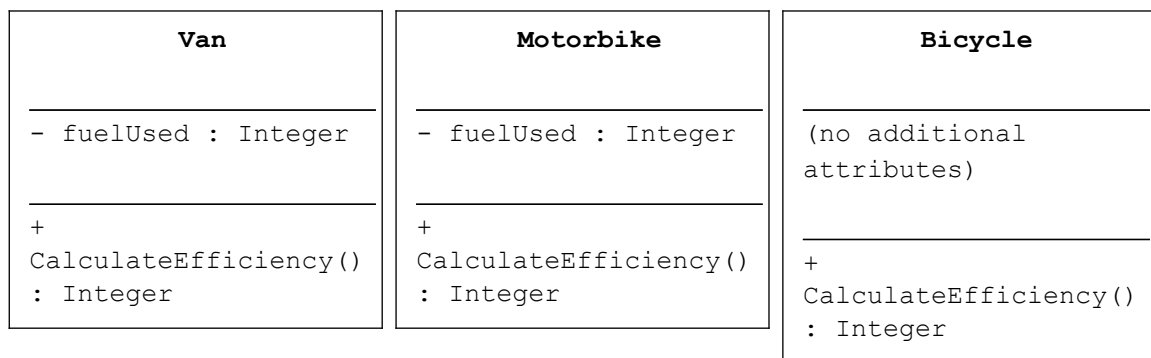
2

(a) Class diagram [8m]

Superclass:



Subclasses (each inherits from Vehicle — shown with a hollow-headed arrow pointing up to Vehicle):

**Notes on the diagram:**

- Van and Motorbike inherit registrationNumber, status, distanceTravelled and efficiencyScore from Vehicle — these are not repeated in the subclasses.
- Each subclass overrides CalculateEfficiency() with its own formula (this is the polymorphism referred to in part (c)(iii)).
- Bicycle has no additional attributes but still overrides CalculateEfficiency() since its efficiency score changes differently (+1).

Any equivalent, sensible naming and layout showing the same structure is acceptable.

Mark Scheme (8m):

1m – Vehicle superclass named and drawn as a class box

1m – Van, Motorbike and Bicycle subclasses

1m – inheritance shown with arrows from each subclass to Vehicle

1m – four superclass attributes (registrationNumber, status, distanceTravelled, efficiencyScore)

	with sensible data types; accept status as String or Boolean 1m – fuelUsed attribute in Van and Motorbike only, none in Bicycle 1m – constructor and/or getter/setter methods in Vehicle 1m – CalculateEfficiency() declared in Vehicle and overridden in each subclass 1m – method that sets status to inactive when efficiencyScore is 0 or less (e.g. UpdateStatus() or IsActive()) Note: the motorbike rule in the paper can be read as distance - fuel x 3 or as (distance - fuel) x 3. Accept either reading here and in (c)(iii).
	(b) Instantiation [1m] Instantiation is the process of creating an actual object from a class — that is, using the class as a template/blueprint to create a specific object with its own values for the attributes defined in the class (e.g. creating a specific Van object with registration number “SBA1234A”).

(c)(i) Encapsulation [2m: 1m definition, 1m example from the simulation]

Encapsulation is the bundling of data (attributes) and the methods that operate on that data within a single class, and restricting direct access to that data from outside the class (data hiding).

Example: in the simulation, efficiencyScore is declared as a private attribute. It cannot be changed directly by code outside the class; it can only be updated by calling the class's own CalculateEfficiency() method, which ensures the value is always changed in a controlled, valid way.

(c)(ii) Inheritance [2m: 1m definition, 1m example from the simulation]

Inheritance is a mechanism where a subclass automatically acquires (inherits) the attributes and methods of its superclass, so they do not need to be redefined in each subclass.

Example: Van, Motorbike and Bicycle all inherit registrationNumber, status, distanceTravelled, efficiencyScore and methods such as IsActive() from the Vehicle superclass, so this shared code only has to be written once.

(c)(iii) Polymorphism [2m: 1m definition, 1m example from the simulation]

Polymorphism is the ability for a method with the same name to behave differently depending on which class (object) it is called on — each subclass provides its own implementation of an inherited method (method overriding).

Example: CalculateEfficiency() is called in the same way for a Van, a Motorbike or a Bicycle object, but it produces a different result each time because each subclass has its own version of the method (Van subtracts fuelUsed $\times$ 2, Motorbike adds distance and subtracts fuelUsed $\times$ 3, Bicycle simply adds 1).

(d) Test data [3m]

Type of test	Test data
Normal	25 (any value 5–50 inclusive)
Abnormal	51 or -5 or "abc" (any out-of-range value or wrong data type)
Extreme	5 and/or 50 (the boundary values)

(Fuel used today for a van is valid between 5 and 50 litres inclusive, so 25 is a typical normal value, 51/–5/"abc" are all invalid abnormal examples, and 5 and 50 are the two extreme boundary values.)

(e) Benefits of superclass/inheritance [2m]

Any two of the following (1 mark each):

- Reduces code duplication — shared attributes and methods (registration number, status, distance travelled, efficiency score, IsActive(), etc.) are written once in Vehicle instead of three times.
- Easier to maintain — if a shared attribute or method needs to change, it only needs to be updated once in the superclass rather than in every subclass.
- Easier to extend — a new vehicle type (e.g. Truck) could be added simply by creating a new subclass of Vehicle, automatically gaining all shared attributes and methods.
- Better models the real-world relationship between the vehicle types, making the code clearer and better organised.

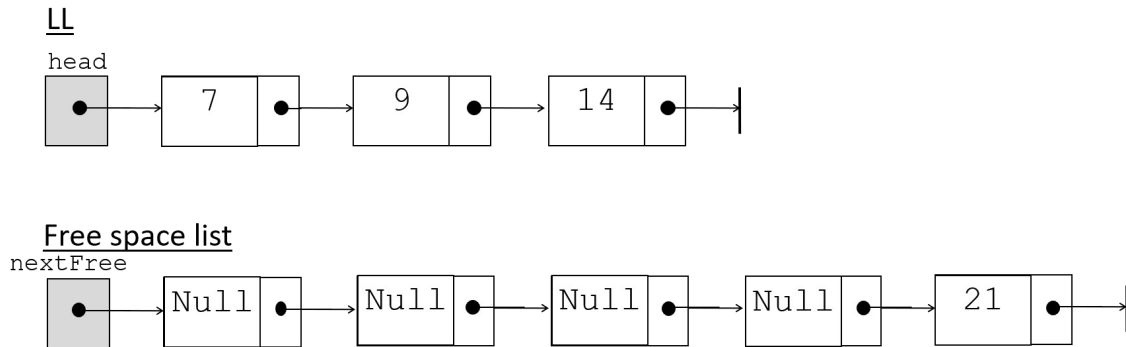
3	(a) Total number of years = $2090 - 1990 + 1 = 101$ – 1m Let n be the min. number of bits $2^n \geq 101 \quad \text{– 1m}$ $n \ln 2 \geq \ln 101$ $n \geq 6.658$ Hence the minimum number of bits is 7 – 1m Also accept: $2^6 = 64 < 101 \leq 128 = 2^7$, hence 7 bits. A candidate who uses 100 years instead of 101 loses the first mark only.
	(bi) 156 divided by 2 = 78 r 0 78 divided by 2 = 39 r 0 39 divided by 2 = 19 r 1 19 divided by 2 = 9 r 1 9 divided 2 = 4 r 1 4 divided 2 = 2 r 0 2 divided 2 = 1 r 0 1 divided 2 = 0 r 1 Read remainder from bottom to top: 10011100_2 1m – attempts to use repeated division or any other methods 1m – final answer correct SR: 0m if no workings shown but with correct answer
	(bii) $10010110_2 = (1 \times 2^7) + (0 \times 2^6) + (0 \times 2^5) + (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0)$ $= 128 + 16 + 4 + 2$ $= 150_{10}$ 1m – any correct method used 1m – correct answer SR: 0m if no workings shown but with correct answer
	(biii) $110111001_2 = 0001_2 \ 1011_2 \ 1001_2$ Given that: $0001_2 = 1_{16}$ $1011_2 = 11_{16} = B_{16}$ $1001_2 = 9_{16}$

Hence $110111001_2 = 1B9_{16}$

1m – final correct answer

Note for Q3(b): no deduction if the base is not stated; a correct value earns the answer mark.

4 (a)



Mark Scheme

1m – for LL with head pointer point to first node, all nodes in correct order and linked with arrows and last node denoted by an end pointer.

1m – for free space list with nextFree pointer referencing to the first node of the freespace list with the correct data and order, linked with arrows and last node denoted by an end pointer.

(bi)

As the linked list is not full, the first available node is at index 0, as indicated by:

$\text{nextFree} = 0$

1. Checks if there is free space to insert 12
2. Since there is an available node space at index 0.
3. Store 12 in the data field of node 0.
4. Update nextFree to the value of next pointer of node to be inserted.

i.e. $\text{nextFree} = 6$:

5. Traverse the linked list from $\text{Head} = 1$ until the correct insertion position to insert new node is found. Since:

$9 < 12 < 14$

New node must be inserted between the nodes with index 5 and 3. i.e. node at index 5 will have $\text{next} = 0$. New node's $\text{next} = 3$

Mark Scheme

1m – for points (1) and (2) (Check if linked list is full)

	1m – for points (3) and (4) (Store new data into free node and update nextFree) 1m – for point (5) (Traverse LL to find the insertion position and relink: node at index 5 has next = 0, new node at index 0 has next = 3)
	(bii) • Unlike array that requires all stored items to be contiguously arranged and packed close to one another. The items stored in a list are ordered by pointers. Adding or removing any items from a list only requires the value of the pointers to change to reflect the new order, whilst adding or removing items of an array will require existing items to move from one memory location to another, which can be very resource costly. Mark Scheme 1m for the point above with some explanation of why it is an advantage. Do not accept "a list is dynamic in size / memory is allocated at run time": in this context LL is implemented inside a fixed-size array of 8 nodes.

5 (a)

```

PROCEDURE enqueue(msg : STRING)
  IF (rear + 1) MOD 4 = front
    THEN
      OUTPUT "Queue is full"
    ELSE
      IF front = -1
        THEN
          front ← 0
          rear ← 0
        ELSE
          rear ← (rear + 1) MOD 4
        ENDIF
      CQ[rear] ← msg
    ENDIF
  ENDPROCEDURE

```

Mark Scheme**1m – check full using condition $(rear + 1) \text{ MOD } 4 = front$** **1m – check empty using condition $front = -1$** **1m – assign $front = rear = 0$ if empty, else $rear = (rear + 1) \text{ MOD } 4$** **1m – store msg in $CQ[rear]$** **(b)**

index	0	1	2	3
	Gaia	Annie	Carrie	Hollie

front = 2**rear = 3****Mark Scheme****1m – front = 2 and rear = 3****1m – Names correctly shown in each element of CQ, including Gaia and Annie still in index 0 and 1 (dequeued data is not deleted).****(c)**Any **one** below:

- Printer spooling**, where print jobs are processed in the order they are received while freed buffer positions are reused.
- Network data buffering**, where incoming or outgoing data packets are temporarily stored in a fixed-size buffer before being transmitted or processed.
- CPU round-robin scheduling**, where each process receives a fixed time slice

	before the scheduler moves to the next process. 4. Keyboard input buffering, where keystrokes are temporarily stored before being processed. 5. Audio or video streaming buffers, where data is continuously written and read in sequence. 6. Traffic-light control systems, where signal phases repeat in a fixed cycle. 7. Customer service call systems, where callers are handled in arrival order within a bounded queue. 8. Sensor data logging, where the newest readings overwrite the oldest readings when the buffer becomes full. 9. Network router buffers, where packets wait temporarily before transmission. 10. Multiplayer game turn management, where players take turns repeatedly in a fixed order. Mark Scheme 1m – award 1m for stating either one of the points above.
6	(a) • An array is suitable because it allows fast direct access to an index location by specifying an integer within its size. Therefore, the hash value can be used directly as the array index to insert or retrieve an item efficiently. • Time taken for insertion, lookup, and removal of an item in array is $O(1)$ • Since the hash table is fixed in size, an array also provides a fixed number of storage locations with predictable memory usage. Mark Scheme 1m – array allows fast direct access of stored element or location 1m – both hash table and arrays are fixed in size.
	(b) The hash function calculates an array index directly from the product code, allowing access to the array's index location directly for the key-value pair to be stored without searching through any other locations. Marks Scheme

	1m – allocates an array index to store an item based on its product code. 1m – with an array index, access to the item can be performed directly without searching through any other locations.				
	c(i) A collision occurs when two or more different product codes are processed by the same hash function and produce the same hash value, causing them to be assigned to the same index in the hash table. Mark scheme 1m – majority of bolded phrases included				
	c(ii) Whenever a collision is detected, linear probing will handle it by: 1. Checking if the next index location is empty. 2. If it is empty, the data will be stored in that location. 3. If it is not empty, it will continue checking consecutive indices until an empty location is found or when all locations of the hash table have been inspected. Marks Scheme 1m – Inspect the index location that succeeds the current location and check if it is empty 1m – if empty, the data will be stored in that location. Else linear probing will repeat this process until an empty location is found or when all locations of the hash table have been inspected.				
7	a) Insertion sort works as follows: 1. The first element is treated as an already-sorted sublist of size 1. 2. The next unsorted element (the "key") is taken and compared against elements in the sorted sublist, moving from right to left. 3. Each element in the sorted sublist that is greater than the key is shifted one position to the right. 4. Once an element smaller than (or equal to) the key is found (or the start of the list is reached), the key is inserted into that position. 5. This is repeated for each remaining element until the whole list is sorted. Mark scheme (4 marks): <table border="1"> <thead> <tr> <th>Point</th><th>Mark</th></tr> </thead> <tbody> <tr> <td> </td><td> </td></tr> </tbody> </table>	Point	Mark		
Point	Mark				

	Treats the first element as a sorted sublist	1			
	Takes the next element as the "key" and compares it against the sorted sublist	1			
	Shifts elements greater than the key one position to the right	1			
	Inserts the key into its correct position once found; repeats until all elements are placed	1			
(b) [3m]					
	list				
Pass	2200	1600	3000	1800	2500
1	1600	2200	3000	1800	2500
2	1600	2200	3000	1800	2500
3	1600	1800	2200	3000	2500
4	1600	1800	2200	2500	3000
Pass 1 and 2 – 1m Pass 3 – 1m Pass 4 – 1m					
(c)					
Worst-case time complexity: $O(n^2)$ -- 1m					
(d)					
Insertion sort is preferred over quick sort when the items are already sorted or nearly sorted according to the sort criteria. Most items are then already close to their correct positions, so only a small number of comparisons and shifts are needed; in the best case (already sorted) insertion sort runs in $O(n)$, while quick sort still takes $O(n \log n)$ and can degrade to $O(n^2)$ on sorted input with a first or last element pivot. Marks Scheme 1m – condition stated: items already sorted or nearly sorted (also accept: a small number of items, where quick sort's recursion overhead outweighs its advantage) 1m – description: few comparisons and shifts needed, best case $O(n)$ (or: quick sort degrades to $O(n^2)$ on sorted input with a first/last element pivot; or for small n: less overhead than quick sort's recursion)					

8	(a) Any three functions of router: - Connects different networks together (e.g. office LAN to the Internet). - Forwards data packets to the correct destination network. - Determines the best path for packets using routing tables. - Directs traffic based on destination IP addresses. - Separates broadcast domains to reduce unnecessary traffic. - Also accept: assigning IP addresses to devices (DHCP), translating private addresses to the public address (NAT), filtering traffic as a firewall. Mark Scheme Award 1 mark for each correctly described router function.	[3]
	(b) Attack: - Phishing. (1 mark) State and describe any one preventive measure: (1m for stating the measure, 1m for describing how it reduces the risk of the attack succeeding) - Verify sender: Check that the email address belongs to the legitimate organisation before trusting the email. - Avoid clicking links: Suspicious links may lead to fake websites designed to steal login credentials. - Multi Factor Authentication (MFA): Even if a password is stolen, an attacker cannot log in without the second authentication factor. - Training: Employees learn to recognise warning signs such as urgent requests, spelling mistakes and suspicious URLs.	[1] [2]
	c)i) Max two from: - It is the unpublished/secret key that is never transmitted - It has a matching public key - It is used to decrypt data that was encrypted with its matching public key - It is used to encrypt data that can only be decrypted by its matching public key c)ii) One mark for each:	[2] [2]

	1. The message to be sent is encrypted with the recipient's public key 2. Eavesdropping is prevented as the message can only be decrypted with the recipient's private key	
	c)iii) Max four from: The sender hashes the message to produce a message digest, then encrypts the message digest with the sender's private key to produce the digital signature The digital signature received is decrypted with the sender's public key to recover the hash digest The received message is hashed to reproduce the message digest from the message received The two digests are compared if both digests are the same the message has not been modified // if the digests are different the message has been modified	[4]
	(d) Any two valid benefits (for each: 1m identify, 1m describe): 1. Better performance because it is compiled for the specific operating system and hardware for the mobile phone. 2. Can access device features such as the camera, GPS, fingerprint scanner, microphone or push notifications on the mobile phone. 3. Provides a better user experience because it follows the operating system's interface guidelines and responds more smoothly. 4. Improved security, as it can make use of OS security features such as biometric authentication and secure storage. Sample answer: • Better performance and faster response because it is optimised for the mobile device. • Can access built-in hardware features such as the camera, GPS and fingerprint scanner.	[4]
	Teacher's Notes • In (a), candidates should describe router functions, not simply list terms like "routing" or "forwarding". • In (b), award the final explanation mark only if the candidate explains how a preventive measure reduces the risk, rather than merely naming it. For example, "Enable MFA" alone earns one mark, while "Enable MFA because a stolen password alone cannot be used to access the account" earns the additional explanation mark.	

8 (e) PDPA (4 marks)

Accept any two. For each: 1 mark for identifying the breach, 1 mark for an explanation in the context of the scenario. (Maximum 4 marks)

1. **Consent obligation: personal data was disclosed to a third party (the advertising company) without the users' consent.**
 - Consent was given only for event registration, so fresh consent was needed before the data was shared with anyone else. Also accept: users should have been given a way to opt in to or out of the sharing.
2. **Purpose limitation obligation: the data was used for a purpose (advertising products) different from the purpose for which it was collected (registration for a school event).**
 - Data may only be used for purposes that a person would consider reasonably appropriate and that were made known at collection; advertising by a third party is not connected to running the event.
3. **Notification obligation: users were not told, at or before collection, the purposes for which their data would be used or that it would be passed to a third party.**
 - The app should have stated these purposes (for example in a privacy notice) before the data was collected, so that users could decide whether to register.

Also accept, with a scenario-based explanation, any other PDPA obligation the candidate names or clearly describes, for example:

- transfer limitation (data passed to another party without ensuring comparable protection);
- retention limitation or protection (keeping or exposing photographs and contact details beyond what the event needed);
- or a reasonable-purpose argument that photographs and mobile numbers exceed what registration for a school event requires.

Award the identification mark only if the obligation is named or clearly described, and the explanation mark only if the explanation says why the student's action breaches it in this scenario.

Do not accept:

- "Collected the data without consent": users supplied it knowingly during registration.
- Two answers that describe the same act under different names. "Shared without consent" and "did not tell users it would be shared" count as separate breaches only if the candidate names or clearly describes both obligations (consent and notification); if the second answer merely repeats the sharing fact, award identify and explain once and look for a distinct second obligation.
- The explanation mark for restating the breach, for a general statement that it "violates the PDPA", or for describing what the PDPA is.

END OF PAPER