1 (a) DNS translates **domain name given by the user to the IP address** that the computer can read and process. DNS server first checks if the requested domain name is available in its **cache**. If not found, it sends request to **the local DNS** by the Internet Service Provider. If still not found, DNS searches from a **hierarchy of distributed database** to locate the domain name. Once found, DNS server sends the IP address back to the user.

**Marker's Comment: many students did not mention the check of cache/local DNS before searching the hierarchy of distributed database. Some students gave answers for DHCP instead.*

(b) Advantage: Server can control the access rights of resources; If one client is down, the server and other clients are not affected; Resources can be updated faster, and it is easier to perform backup. Disadvantage: If server is down, the whole network is down; Centralized server is more expensive to build up, and requires professional to maintain.

2

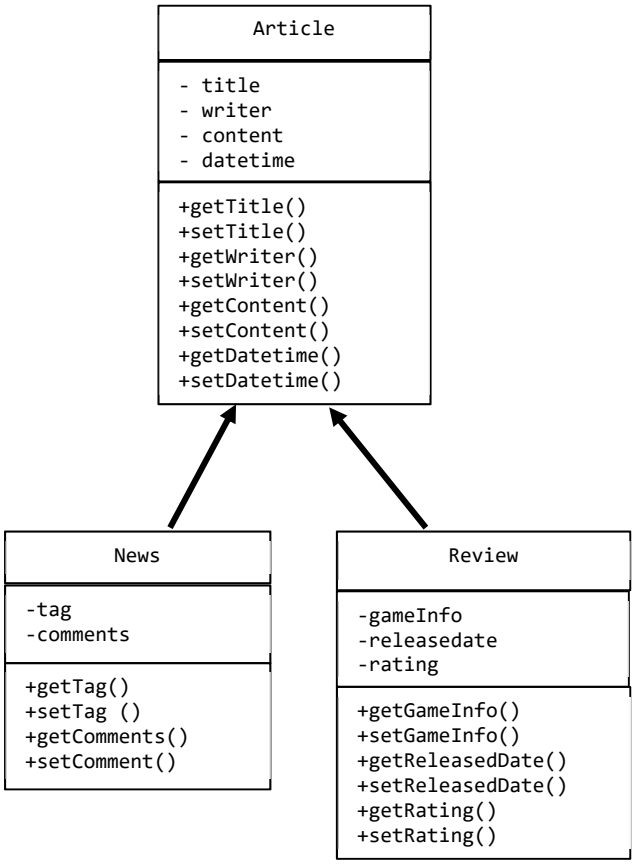
(a) The table has a repeated group of attributes
-There are several reviews for the same Reviewer/ReviewerID
(b) Data redundancy leads to data anomalies and data corruption.
Same Reviewer and Platform are being stored more than once, which will cause data anomalies during inserting, updating and deleting of data from the database.
(c) There are transitive dependencies in `REVIEW` table.
`Platform` is transitive dependent on `ReviewID`.
(d) Table descriptions:

`REVIEWER(ReviewerID, Reviewer)`

`PLATFORM(PlatformID, Platform)`

`REVIEW(ReviewID, Title, ReviewDate, ReviewerID, PlatformID)`

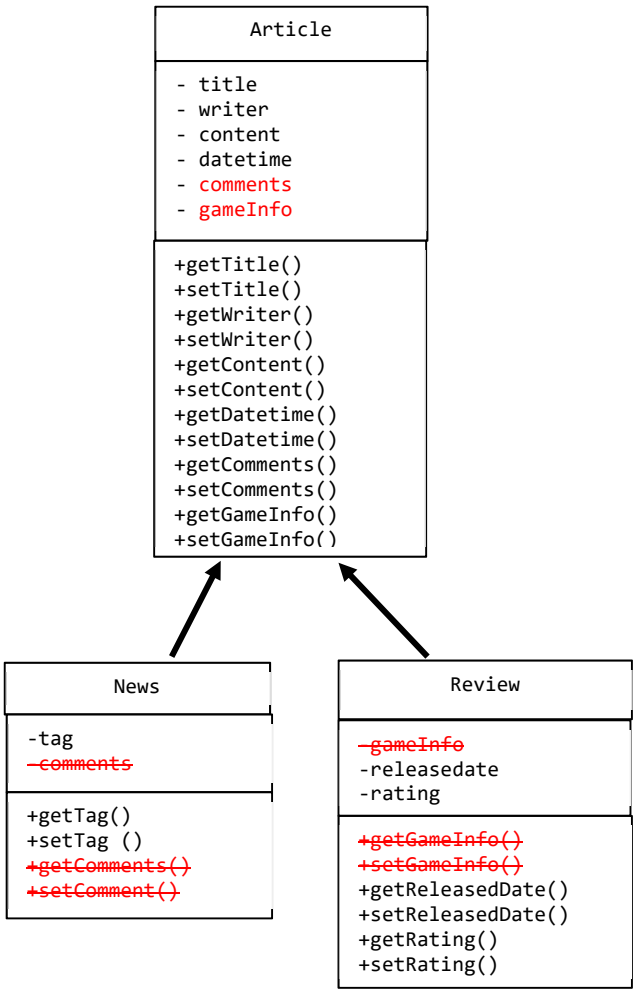
(e)



(f) Inheritance allows real world objects to be easily modeled, where some objects are specialize d versions of more general objects. Inheritance promotes code reusability as subclasses inherit all data attributes and methods of their superclasses and do not need to declare the attributes and recode the methods. For News class, the title, writer, content and datetime attributes are inherited and need not be declared in the subclass.

(g) Encapsulation is the **bundling** of attributes and methods that operates on the class. It allows **data hiding** so that internal attributes will not be corrupted by external operations. The attributes title, writer, content and datetime in Article are private and is not accessible by code outside the class. In order to access or change the value of the attributes, the code outside will need to make use of the public methods provided by the class. Eg. To change the value of title attribute, the code outside will use setTitle().

(h)



**Marker's Comment:*

(a,b,c) - most students provide an explanation but did not provide an example from the question

(h) many students mentioned the adding of attributes and methods to the superclass but did not mentioned the removing of the attributes and methods from the subclass. The answer needs to mentioned clearly what is/are added to the superclass and what is/are removed to the subclasses.

3 (a) Asymmetric encryption uses a pair of keys to encrypt a message so that **only authorized personnel can access the message**. The sender uses the **receiver's public key** and an **encryption algorithm** to encrypt the message to a cipher text. The receiver uses **its own private key** and a **decryption algorithm** to decrypt the cipher text to the original message.

**Marker's Comment: many students did not mention the use of encryption/decryption algorithms. The use of public or private keys must be clearly stated. Some students gave answers for digital signature instead.*

(b) DOS attack **exhaust the resource and bandwidth** in the network by sending huge amounts of requests, so that the system cannot fulfil legitimate requests. We may use **firewall or Intrusion Protection System** to protect from DOS attack.

4

(i)

```
class Node: # Node class not needed in qn
    def __init__(self, data):
        self.data = data
        self.next = None

class Customer:
    def create(self):
        self.head = None
        self.tail = None
    def display(self):
        if self.head == None:
            print("No customer.")
        else:
            print("Customers: ", end="")
            ptr = self.head
            while ptr != None:
                print(ptr.data, end = ', ')
                ptr = ptr.next
```

(ii)

```
class Queue(Customer):
    def enqueue(self, person):
        if self.tail == None:
            self.head = Node(person)
            self.tail = Node(person)
        else:
            self.tail.next = new_node
            self.tail = new_node

    def dequeue(self):
        if self.head == None:
            print("Queue is empty")
        else:
            data = self.head.data
            self.head = self.head.next
            if self.head == None:
                self.tail = None
            return data
```

Some students forgot that when an empty queue is enqueued, both the head & tail must be updated.
Many students forget that when the last item is dequeued, the head & tail must be updated.

(iii)	As the footpath is narrow, a **'first in, first out'** approach using the queue structure is inefficient and **'first in, last out'** approach using **stack** would be better. While there are other possible structures, dequeuing everybody together is not accepted as data still has to be accessed one at a time, and having everyone move at the same time with no system is messy and can cause a stampede Imagine dequeuing from the middle of a row of seats… A popular alternative accepted is alternating dequeue from head and tail if customers can exit from both ends of their seats (ie. dequeue 2 people max) Using array or circular does not improve the situation as it is still messy.

	`class BetterQueue(Queue):`
	
	`    def enqueue(self, person):`
	`        if self.head == None:`
	`            self.head = Node(person)`
	`            self.tail = Node(person)`
	`        else:`
	`            temp = Node(person)`
	`            temp.next = self.head`
	`            self.head = temp`
	
	Since BetterQueue is a standard Stack structure, it can be inherited from Queue, where only enqueue needs to be changed as dequeue can be fully inherited from Queue (dequeue and enqueue both at head)
	Since BetterQueue is inherited from Queue or Customer, it has the tail-attribute. This attribute must also be updated.
	
	Alternatively, a completely new class (without using inheritance) can be created.

Note: Written exam should NOT have Python coding question. Questions should be answered in pseudocode or text. (with the exception of SQL query statements) However, as required in this year's paper, the code is done in Python.

(creating class in pseudocode requires CLASS & ENDCLASS which are not in A-level syllabus)