a.

i Outline **one** advantage of polymorphism. [2]

ii Outline **one** advantage of encapsulation. [2]

iii Outline **one** disadvantage of inheritance. [2]

An invoice is created every time a customer purchases one or more products.

The `Invoice` class keeps details of each invoice. The following shows part of the code for this class:

```
public class Invoice {

   private String invoiceID; // identifies a unique invoice


   // list of products purchased
   private static Product[] products = new Product[20];


   // number of items of a particular product purchased
   private static int[] prodQuantity = new int[20];


   private boolean qualifiesForDiscount; // default value is false
   private int numOfProducts; // how many products in this invoice

   // constructor is defined, code not shown



   public String getInvoiceID(){
       return invoiceID;
   }


   // all accessor and mutator methods are present but not shown

   public void addProduct(Product product, int quantity) {
       // code missing
   }
```

```
    public void setQualifiesForDiscount(){
        // if total value of purchases is more than 3000,

        // qualifiesForDiscount value is set to true

        // code missing

    }
} // end of Invoice class
```

(b) Describe how encapsulation has been used in this code. [2]

(c) Construct the method `addProduct(Product product, int quantity)` that will update an invoice.
The method should:

- Update the `products` array
- Update the `prodQuantity` array
- Increment the `numOfProducts`.

[3]

If the total value of the purchases is greater than 3000, the invoice qualifies for discount.

(d) Construct code for the method `setQualifiesForDiscount()` to change the status of an invoice.
The method should:

- calculate the total value of an invoice
- change the value of `qualifiesForDiscount` if needed.

[6]

(e) Outline **one** advantage of using modularity in program development.

[2]

f) HL ONLY
A company stores its Products using Java's built-in `LinkedList<>` class.

Each `Product` has a unique product code.

```
public class Product {
    private String code;
    private String name;
    private double price;

    public Product(String code, String name, double price) {
        this.code = code;
        this.name = name;
        this.price = price;
    }

    public String getCode() { return code; }
}
```

The company stores all products in:

```
LinkedList<Product> products = new LinkedList<>();
```

You may use the following linked list methods.

Method	Description
`addFirst(E e)`	adds an element to the front of the list
`add(E e)`	adds element to the end
`get(int index)`	returns the element at index

`size()`	returns the number of elements
`isEmpty()`	checks if the list is empty
`iterator()`	returns an iterator
`listIterator()`	returns a bidirectional iterator

i) Explain why a LinkedList may be more efficient than an ArrayList when frequently adding items to the front of a list.

[4]

ii) Write a method containsCode(String code) that returns true if any product in the list has the given code. [4]

iii)Write a method addIfNew(Product p) that:

- checks whether the product already exists (same code),
- if it exists: returns false and does not add the product,
- if it does not exist: adds the product to the front of the list using addFirst, and returns true.

You must use at least one of the LinkedList API methods listed in the table above.

Method signature:

```
public boolean addIfNew(LinkedList<Product> products, Product p)
```

[4]

.