Question 1

1 a.

- Encapsulation is the process of encapsulating a pieces of application code with shared behaviours and attributes as self-contained and separate units.
- One way encapsulation can be achieved is using classes to encapsulate attributes and methods (which operate on the attributes) into a single unit.
- Encapsulation allows information hiding as it hides some attributes and methods from a user through visibility modifiers. For example, some details of a class (such as private methods and private attributes) can be hidden from other classes 'outside' the given class.
- This in turn allows privacy by ensuring a user of a program is unable to access the private attributes and data of any class directly.

1 b.

- UML is useful as it enables software developers to identify and model classes and class relationships without the added complexity of having to code the classes and debug errors. This thus allows developers to gain a 'big picture' view of a software system, allowing them to make better design decisions.
- UML is also useful as a communication tool between software developers. For example, several software developers may work on designing a software system, and thus require a conventional and widely known 'language' to express their ideas. This also allows for the delegation of tasks, as one software developer can design the system with the UML, while another will translate it into code.

1 c.

- Design patterns are descriptions of solutions to recurring problems in software design.
- Design patterns should be used when designing software as they allows us to save a lot of time by reusing solutions to difficult problems. We can also extend upon the existing solutions to make it more specific to our problem.
- Secondly, we use design patterns when writing software as it enables easier communication between software developers. This is because many design patterns are commonly used in software development, so during the development of a piece of software, using design patterns reduces the need for complicated explanations about software solutions.

1 d.

- Cohesion is the measure of how the specificity of modules in solving clear, focused problems. For example, classes with high cohesion have methods and attributes which are highly related to each other and work towards a common objective.
- Coupling is the measure of the degree of interaction between modules. Coupling should be reduced as much as possible, with a software system with low coupling having minimal dependency between classes as possible.

1 e.

- Automated software testing is the testing of code with automated, purpose built software.
- We prefer automated testing as it is easily scalable and can be used with large software systems. This is because automated testing is not reliant on human resources.
- Automated testing is also preferable because it is much faster and more reliable than manual testing. This is because building software to test for errors is less time intensive and less prone to human error than a human debugger testing code line by line.

1 f.

- This stream pipeline first selects elements from the list 'members' which isActive and has a Follower count of over 100000.
- Then, the result list is sorted in descending order.
- Finally, a new list is created out of the sorted result list, which is reassigned to the 'result' variable.
- We might use this code with a social media platform for finding the most famous active member (with the most followers). This could be useful for finding famous influencers to promote brands.

Question 2

2 a.

- The Strategy Pattern solves the problem of separating a generic algorithm from a detailed design. It solves this problem by using a common Strategy interface (a generic component) which can be extended and adapted by to a specific Application by classes which implement it.

2 b.

- One benefit of using the Strategy Pattern is that in uses minimal coupling. Thus, an entirely different Application which builds upon the same algorithm can be used. For example, with the sorting example, the Application can be BubbleSort or QuickSort, and both utilizes the same Strategy interface.
- In contrast, the template method solves a similar problem as the Strategy Pattern with inheritance. Using the template method, there is a strong dependency to the parent class which contains the generic algorithm, so for example, only one sorting algorithm can be used.

2 c.

- The Strategy interface contains generic methods for an algorithm.
- Strategy 1 and Strategy 2 are classes which implement the specific behaviour of the methods in the Strategy interface.

- Application contains code for a generic operation (such as sorting) which uses the generic methods from the Strategy interface. A class which implements the Strategy interface will be passed into the operation method as a parameter, thus giving the operation its specific implementation (for example, sorting integers).

2 d.

- A real world application of the Strategy pattern is printing. It is highly likely that a user would want to print a large variety of data types (such as images, text, etc). At the same time, the application using the printing method might be different (such as a different printer software).
- Thus, the Strategy pattern is used to enable all data types to be printed by any printer. For example, the Strategy interface can hold a 'print' operation, which is implemented in classes such as 'ImagePrint'. Then, a printer application will use the print operation implemented in the concrete ImagePrint class.

Question 3

3 b.

- Behaviour: Ensure that each DefenceItem must have at least the minimum number of soldiers.
  Input: Create a DefenceItem object with a list of soldiers of size less than the item's minimum number of soldiers.
  Output: An exception is thrown and the object cannot be created.
- Behaviour: Ensure that a Location object cannot be created without all three attributes (x,y,z) initialized.
  Input: Give a location object only x and y coordinates.
  Output: An exception is thrown and the object is not created.

Question 4

```
public class Card implements Comparable<Card>{
    private final Rank rank;
    private final Suit suit;

    public Card(Rank rank, Suit suit) {
        this.rank = rank;
        this.suit = suit;
    }

    public Rank getRank() {
        return rank;
    }

    public Suit getSuit() {
        return suit;
    }

    @Override
    public int compareTo(Card c2) {
```

```
        Card c1 = this;

        return (c1.rank.getPoints()-c2.getRank().getPoints());

    }


    @Override
    public String toString() {
        return "["+rank +
                ", " + suit +
                ']';
    }
}
```

```
import java.util.ArrayList;
import java.util.Collections;

public class Player {
    private int playerID;
    private ArrayList<Card> cards;


    public Player(int playerID, ArrayList<Card> cards) {
        this.playerID = playerID;
        this.cards = cards;
    }

    public void addCard(Card card){
        this.cards.add(card);
    }

    public int totalScore(){
        int total = 0;

        for (Card card : this.cards){
            total += card.getRank().getPoints();

        }

        return total;
    }

    public void sortHand(){
        Collections.sort(this.cards);
    }

    public Card playBestHand(){
        this.sortHand();

        Card bestCard = this.cards.get(0);

        this.cards.remove(0);

        this.cards.trimToSize();

        return(bestCard);
    }
```

```
    public int getPlayerID() {
        return playerID;
    }

    public void setPlayerID(int playerID) {
        this.playerID = playerID;
    }

    public ArrayList<Card> getCards() {
        return cards;
    }
}
```

Question 5

```
public String rectEncryption(String text, int n){

    // Convert all characters to upper case
    text = text.toUpperCase();
    text = text.replace(' ','#'); // replace spaces with #

    // Make rectangle
    ArrayList<String> rectangle = new ArrayList<String>();

    int j = 0;

    while (j<text.length()){
        String row = "";

        for (int i=0; i<n;i++){
            if (j>= text.length()){
                row += '*';
            }else{
                row += text.charAt(j);
            }

            j++;
        }

        rectangle.add(row);
    }

    // Take columns of rectangle
    String encryption = "";

    for (int i=0; i<n; i++){
        for (String row : rectangle){
            encryption += row.charAt(i);
        }
    }


    return encryption;
}
```

Question 6

```
public interface Categoriser<In, Out> {

    public Out categorise (In input);
}
```

```
public class StringCategoriser implements Categoriser<String, String>{

    @Override
    public String categorise(String input) {

        if (!input.equals("")){
            return (Character.toString(input.charAt(0)));
        }

        return null;
    }
}
```

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

public class CategorisedMap<K, V, C extends Categoriser<V,K>> {
    HashMap<K, ArrayList<V>> hashMap;
    C categoriser;


    public CategorisedMap(C categoriser) {
        this.hashMap = new HashMap<K, ArrayList<V>>();
        this.categoriser = categoriser;
    }

    public void put (V value){
        K key = categoriser.categorise(value);

        if (hashMap.containsKey(key)){
            ArrayList<V> list = hashMap.get(key);
            list.add(value);
            hashMap.replace(key, list);
        }else{

            ArrayList<V> newList = new ArrayList<V>();
            newList.add(value);
            hashMap.put(key, newList);
        }


    }

    public int getCategoryCount(V value){
        K key = categoriser.categorise(value);

        if (hashMap.containsKey(key)){
            return (hashMap.get(key).size());
```

```
        }else{

            return 0;
        }
    }


    public String toString(){

        List<K> keyList = new ArrayList<K>();
        keyList.addAll(hashMap.keySet());

        String string = "";
        for (K key : keyList){
            string += key.toString() + ": ";
            ArrayList<V> valueList = hashMap.get(key);

            for (V value : valueList){
                string += value.toString();
                string += ", ";
            }

            string += "\n";
        }

        return(string);

    }
}
```