COMP0004 Object-Oriented Programming
Example Answer for Programming Exercises 3
Question 1

Q1.

a) Write a `class AddressBookEntry` to represent an entry in an address book. An object of the class should store the name, phone number and email address for one person.

The class should have a constructor to initialise a new object to represent a person, and getter methods to access each piece of information:

```
public AddressBookEntry(String name, String phone, String email);
public String getName();
public String getPhone();
public String getEmail();
```

The class must not do any input or output. The information needed to represent a person entry is passed via the constructor. The instance variables storing the data must be private.

Example Answer:

This is a straightforward data class, with no setter methods, hence instance objects will be immutable.

```
public class AddressBookEntry
{
  private String name;
  private String phone;
  private String email;

  public AddressBookEntry(String name, String phone, String email)
  {
    this.name = name;
    this.phone = phone;
    this.email = email;
  }

  public String getName()
  {
    return name;
  }

  public String getPhone()
  {
    return phone;
  }

  public String getEmail()
  {
    return email;
```

```
  }
}
```

b) Next write a `class AddressBook`, that uses a
`private ArrayList<AddressBookEntry>`
instance variable to store entries in an address book. The class should include methods to add, remove and search for entries. Note again, this class should not do any input or output.

Example Answer:

This is a basic version of the class.

```
import java.util.ArrayList;

public class AddressBook
{
  private ArrayList<AddressBookEntry> entries;

  public AddressBook()
  {
    entries = new ArrayList<>();
  }

  public ArrayList<AddressBookEntry> getEntries()
  {
    return new ArrayList<>(entries);
  }

  public void addEntry(String name, String phone, String email)
  {
    entries.add(new AddressBookEntry(name, phone, email));
  }

  public ArrayList<AddressBookEntry> searchFor(String name)
  {
    ArrayList<AddressBookEntry> matches = new ArrayList<>();
    for (AddressBookEntry entry : entries)
    {
      if (entry.getName().equals(name))
      {
        matches.add(entry);
      }
    }
    return matches;
  }

  public void removeEntry(String name)
  {
    for (AddressBookEntry entry : entries)
    {
      if (entry.getName().equals(name))
      {
        entries.remove(entry);
        return;
      }
```

```
    }
  }
}
```

Note that in order to list all the entries in the address book, a getter method called `getEntries` has been added. This returns a shallow copy of the `entries` ArrayList, without copying the `AddressBookEntry` objects, but is not ideal as it is exposing too much of the internal structure of the class. Fortunately in this basic version of the program `AddressBookEntry` objects are immutable, so the entries cannot be modified by other code. This part of the design would need re-working if the program were developed into a more robust version.

The `searchFor` method returns a list of entries that match the given name, but uses `equals` to compare names. This could be enhanced to do at least some basic pattern matching. The `removeEntry` method also uses `equals` to find a match, but just deletes the first match and then returns. This could also be improved on!

c) Write a class with a main method that creates an address book and allows it to be used by adding, removing and searching for entries. This class does the input and output, as well as the main method.

Think carefully about the public methods and their parameters needed by all three classes. What are the correct methods for the abstractions that the classes represent? All instance variables should be private, so that data is accessed only by the object it belongs to.

Example Answer:

This class provides a simple command line interface and the main method.

```
import java.util.ArrayList;

public class SimpleAddressBook
{
  private static final int LIST_ENTRIES = 1;
  private static final int ADD_ENTRY = 2;
  private static final int REMOVE_ENTRY = 3;
  private static final int SEARCH_FOR_ENTRY = 4;
  private static final int QUIT = 10;

  private Input in = new Input();

  private AddressBook addressBook;

  private SimpleAddressBook()
  {
    addressBook = new AddressBook();
  }

  private void run()
  {
    while(true)
    {
      displayMenu();
      int option = getMenuInput();
      if (option == QUIT)
      {
```

```
      break;
    }
    doOption(option);
  }
}

private void displayMenu()
{
  System.out.println("\nSimple Address Book Menu");
  System.out.println(LIST_ENTRIES + ". List all entries");
  System.out.println(ADD_ENTRY + ". Add entry");
  System.out.println(REMOVE_ENTRY + ". Remove entry");
  System.out.println(SEARCH_FOR_ENTRY + ". Search for entry");
  System.out.println();
  System.out.println(QUIT + ". Quit");
}

private int getMenuInput()
{
  System.out.print("Enter menu selection: ");
  int option = in.nextInt();
  in.nextLine();
  return option;
}

private void doOption(int option)
{
  switch (option)
  {
    case LIST_ENTRIES -> listEntries();
    case ADD_ENTRY -> addNewEntry();
    case REMOVE_ENTRY -> removeEntry();
    case SEARCH_FOR_ENTRY -> searchForEntry();
    default -> System.out.println("Invalid option - try again");
  }
}

private void listEntries()
{
  System.out.println("\nAddress Book Entries:");
  ArrayList<AddressBookEntry> entries = addressBook.getEntries();
  for (int n = 0 ; n < entries.size(); n++)
  {
    AddressBookEntry entry = entries.get(n);
    System.out.printf("%d. Name: %s, Phone: %s, Email: %s\n",
                      n + 1, entry.getName(),
                      entry.getPhone(), entry.getEmail());
  }
}

private void addNewEntry()
{
  System.out.println("\nAdd new address book entry");
  System.out.print("Enter name: ");
  String name = in.nextLine();
  System.out.print("Enter phone number: ");
```

```
    String phone = in.nextLine();
    System.out.print("Enter email address: ");
    String email = in.nextLine();
    addPerson(name, phone, email);
  }

  private void addPerson(String name, String phone, String email)
  {
    addressBook.addEntry(name, phone, email);
  }

  private void removeEntry()
  {
    listEntries();
    System.out.print("\nEnter name of entry to remove: ");
    String name = in.nextLine();
    addressBook.removeEntry(name);
  }

  private void searchForEntry()
  {
    System.out.print("Enter name to search for: ");
    String name = in.nextLine();
    ArrayList<AddressBookEntry> matches = addressBook.searchFor(name);
    int count = 0;
    for (int n = 0 ; n < matches.size() ; n++)
    {
      AddressBookEntry entry = matches.get(n);
      if (entry.getName().equals(name))
      {
        System.out.printf("Name: %s, Phone: %s, Email: %s\n”,
          entry.getName(), entry.getPhone(), entry.getEmail());
        count++;
      }
    }
    if (count == 0)
    {
      System.out.println("No names found");
    }
  }

  private static void addExampleData(SimpleAddressBook addressBook)
  {
    addressBook.addPerson("Person 1", "12345245", "p1@test.com");
    addressBook.addPerson("Person 2", "24553463", "p2@test.com");
    addressBook.addPerson("Person 3", "78678678", "p3@test.com");
    addressBook.addPerson("Person 4", "45656433", "p4@test.com");
  }

  public static void main(String[] args)
  {
    SimpleAddressBook simpleAddressBook = new SimpleAddressBook();
    addExampleData(simpleAddressBook);
    simpleAddressBook.run();
  }
}
```

For convenience while writing the code a method `addExampleData` has been added that adds some test data to the address book, avoiding having to type it in manually all the time. Note that this method is static (i.e., a class method), as it is a utility method that does not need to be called on an object.

Notice that this class does all the input and output for the program and holds the address book object. If the program were developed further then this code would undergo considerable change, but the principle of the `AddressBook` and `AddressBookEntry` classes not doing input or output would be maintained, aiming to keep them as cohesive as possible.