UNISA university of south africa

COS3711

October/November 2016

ADVANCED PROGRAMMING

Duration : 2 Hours 80 Marks

EXAMINATION PANEL AS APPOINTED BY THE DEPARTMENT.

Closed book examination.

EXAMINATION PANEL:	
FIRST:	MR CL PILKINGTON
SECOND:	MR K HALLAND
EXTERNAL:	DR L MARSHALL (UNIVERSITY OF PRETORIA)

Instructions

1. Answer all questions.
2. All rough work must be done in your answer book.
3. The mark for each question is given in brackets next to each question.
4. Please answer questions in order of appearance.
5. Note that no pre-processor directives are required unless specifically asked for.

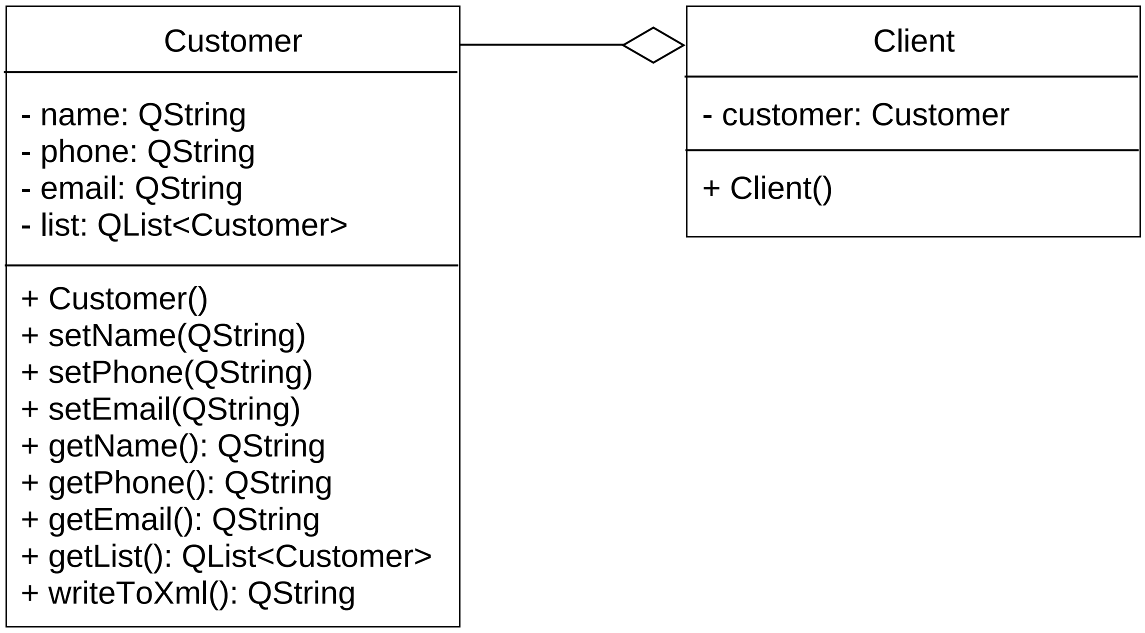
This examination paper consists of 6 pages

Good luck!

All the questions in this paper are based on the software requirements for a book store (yes, people do still read books). Classes will be required that manage an OOP implementation of books and customers, as well as lists of both customers and books.

Question 1 **[11 marks]**

The application needs to manage customer records. A second-year student provided the following UML class diagram as part of the solution.



1.1 Identify one design anti-pattern in the above design and explain why you think it is a problem. (2)

1.2 Using a UML class diagram, provide a better design for the solution. (9)

Question 2 **[26 marks]**

The application will need to manage a list of books. The following classes have already been defined.

```
class Book
{
  public:
    Book();
  private:
    QString title;
    QString getTitle() const;
    void setTitle(QString t);
};
```

```
class Fiction: public Book
{
  public:
    Fiction();
  private:
    QString genre;
    QString getGenre() const;
    void setGenre(QString g);
};
```

```
class NonFiction: public Book
{
  public:
    NonFiction();
  private:
    QString subject;
    QString getSubject() const;
    void setSubject(QString g);
};
```

```
class BookList
{
  public:
    BookList();
    void addBook(Book b);
    QList<Book> getList() const;
  private:
    QList<Book> booklist;
};
```

2.1 Rewrite the `Book`, `Fiction`, and `NonFiction` classes so that they can be used in a reflective programming approach. You do not need to rewrite the whole class – you only need to add the necessary code to the outline for each of the classes given below.

```
class Book
{
  public:
      // Existing attributes and functions
  private:
      // Existing attributes and functions

};

class Fiction: public Book
{
   public:
       // Existing attributes and functions
   private:
       // Existing attributes and functions
};

class NonFiction: public Book
{
   public:
       // Existing attributes and functions
   private:
       // Existing attributes and functions
};
```

(6)

2.2 Given the `BooksToXml` class definition and partial implementation of the `write()` function below, complete the implementation of the `write()` function so that it produces the XML given below. Use the reflective abilities of the classes that were implemented in 2.1. Comments have been included to guide you where the code has to be added.

```
class BooksToXml
{
  public:
    BooksToXml ();
    void write(BookList *list, QString filename);
  private:
    QXmlStreamWriter writer;
};

void BooksToXml::write(BookList *list, QString filename)
{
  QFile file(filename);
  file.open(QIODevice::WriteOnly);
  writer.setDevice(&file);
  writer.writeStartDocument();
  // add code to start the booklist element

  QList<Book> books = list->getList();

  foreach (Book book, books)
  {
    // for each book
    // use meta-objects to find what type of book it is
    // use meta-objects to find the book title
    // write code to generate the required XML
  }

  // add code to end the booklist element
  writer.writeEndDocument();
  file.close();
}
```

Required XML format (note that the `genre/subject` from the derived classes is not required):

```
<booklist>
  <book type="Fiction">
    <title>Book 1 Title</title>
  </book>
  <book type="NonFiction">
    <title>Book 2 Title</title>
  </book>
</booklist>
```

(13)

2.3 Which one other approach could have been used to write the `BookList` to XML apart from using `QXmlStreamWriter`? (1)

2.4 Write the class header and implementation code that will implement the Factory Method design pattern so that it could be used to create the 2 different types of `Books`. (6)

Question 3 **[20 marks]**

This question relates to the `BookList` class given in question 2.

3.1 Rewrite the `BookList` class and its implementation file so that
a) there can ever only be one instance of this class in the application, which is also properly deleted, and
b) the list implements a backup facility.
Use the appropriate design patterns to meet these requirements. Note that you are not required to write implementations for the `addBook()` and `getList()` functions.

You can assume that the following class has been defined and implemented to provide this backup facility.

```
class Backup
{
  private:
    friend class BookList;
    Backup();
    void setBackup(QList<Book>);
    QList<Book> getBackup() const;
    QList<Book> backup;
};
```
(15)

3.2 When, or why, would a user request that there be only one book list? (2)

3.3 Which design pattern was used to implement the backup facility in question 3.1? How would this design pattern differ from a Serialiser design pattern? (3)

Question 4 **[15 marks]**

Consider the following class used to search a `BookList`.

```
class Search
{
  public:
    Search(QList<Book> list, QString title);
    Book find();
  private:
    QList<Book> books;
};
```

4.1 Rewrite this class definition so that it can be run as a thread (using the recommended approach to threading), and that it will return a `found` signal with a `Book` instance. (5)

4.2 Complete the following partially written `Client` function that will pass a `QList<Book>` instance to a `Search` instance and run this instance as a thread (remembering that there is only one instance of the list). Use the comments in the code as a guide to where you should add code.

```
void Client::findBook(QString titleToFind)
{
  QList<Book> b = // access the book list
  QThread *thread = new QThread();
  Search *search = new Search(b);
  // write the code necessary to run the Search instance as a thread
  // ensuring that all signals/slots are handled
}
```

You can use the following slot in `Client` to handle found `Book`s:

```
private slot:
  void handleFound(Book);
```

(7)

4.3 What is a dynamic property, and how does it differ from an instance's other properties? (2)

4.4 Write the code necessary to add a dynamic property (named `staff`, with the value `supervisor`) to the `Search` instance in question 4.2. (1)

Question 5 **[8 marks]**

It has been decided to use a model-view approach to manage staff records at the bookshop, where the staff records would be displayed in rows with a column for each field in the record. Two different programmers attempted this, and there were two different proposed solutions.

a) `QAbstractTableModel *model = new QAbstractTableModel();`
b) `QStandardItemModel *model = new QStandardItemModel();`

5.1 Which of these two would be the most appropriate approach? Explain your answer fully. (2)

5.2 Write the code to attach this model to an appropriate view class. (2)

5.3 What is the purpose of a delegate in the Qt approach to model-view programming? (2)

5.4 Given a `QLineEdit` in a graphical user interface, explain 2 ways a regular expression could be used to validate text a user enters in the line edit box. (2)