Master Copy Sept 09

E2.12

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2009

ISE PART II: MEng, BEng and ACGI

SOFTWARE ENGINEERING 2

Friday, 12 June 2:00 pm

Time allowed: 2:00 hours

There are FOUR questions on this paper.

Q1 is compulsory.
Answer Q1 and any two of questions 2-4.
Q1 carries 40% of the marks. Questions 2 to 4 carry equal marks (30% each).

Any special instructions for invigilators and information for candidates are on page 1.

Examiners responsible First Marker(s) : L.G. Madden, L.G. Madden
Second Marker(s) : J.V. Pitt, J.V. Pitt

Special instructions for invigilators

Students should be allowed an extra 15 minutes at the start of the exam to read the exam paper. After the 15 minutes have elapsed the invigilator should announce to the students that they may start writing.

Each student should be given a separate sheet of paper which contains an enlarged version of:

- *Figure 1.1 that is needed for Question 1.*
- *Figure 2.2 that is needed for Question 2.*

Special instructions for students

You should use the extra 15 minutes at the start of the exam to decide which of the optional questions you will attempt and also to examine Figure 1.1 in preparation for answering the compulsory question 1. You may not start writing until the invigilator makes the announcement to do so.

Please note that there is no difference in content between the figures included in the exam paper with that provided on the separate sheet, other than the later is enlarged and may be easier to use.

[This page is blank by intent]

The Questions

1. **[Compulsory]**

   All the parts of this question refer to Figure 1.1 and each part is worth 1 mark. An enlarged version of the figure is available on a separate sheet of paper. The figure shows an architecture that is similar, but is not identical to the case study developed in the E2.12 Computing laboratories. Your answers should be based only on what is shown in the architecture in Figure 1.1

   Most parts can be answered with one or two line answers. Use fully qualified names for your answers where appropriate. When you are asked to list or identify something that you believe is *not* present in the architecture then you should state "*None*" for your answer. If you do not answer a question part, for example part (z) then please state "*Not attempted Q1(z)*".

   a) Identify *two different* kinds of relationships between classes that can be found in Figure 1.1. For each example name the classes involved in the relationship.

   b) Identify *any* OCCF members that are missing in `TFilter`.

   c) Identify an example of an *instance* data member and a *class* data member. State which is the example of the *instance* or *class* member.

   d) Identify a *relationship* with `TFilter` that could have been *drawn* (i.e. not text) and therefore could be added to Figure 1.1.

   e) Identify a generic UML stereotype that *should* be added to the class `TFilter`.

   f) Identify *two* UML stereotypes that are C++ specific, that is, they are *not* generic UML stereotypes.

   g) Identify *two* other details (i.e. not UML stereotypes) that are C++ specific, that is, they are *not* generic UML.

   h) Without loss of information content, use an alternative and more *concise* way to *redraw* the relationships between `TCircuit` and `std::complex<T>`. You do not need to include the class members or classes *not* involved in the aforementioned relationships in your diagram.

   i) Identify an example of *two* different *types* of string that are *not* a C++ string, that are used in the architecture. For each string type state what type it is and where it appears in the architecture.

   j) Identify *all* the classes involved in an example of a *multiple* inheritance relationship. For each class state whether it is a *derived* or *base* class.

Continued on the following page.

k) Identify *two* different types of polymorphism implicit in the architecture. For each type describe where it appears in the architecture.

l) Identify an example in the architecture where encapsulation *is* broken *knowingly*. Briefly explain your answer.

m) Identify an example in the architecture where encapsulation *might* be broken *accidentally*. Briefly explain your answer.

n) Is there evidence of Test Driven Development (TDD) in the architecture? Briefly explain your answer.

o) A STL class in the architecture may be swapped with an alternative STL class. Identify the *original* class and the name of *any* class that it may be swapped with.

p) Which coding problem would have to be addressed, if a user-written complex number class was added to the class hierarchy?

q) Briefly describe, in words or code, how a polymorphic *variable* involving *application* classes from Figure 1.1 might be used.

r) Briefly describe, in words or code, how a polymorphic *variable* involving *utility* classes from Figure 1.1 might be used.

s) What form of handling errors in *service* code is implicit in the architecture?

t) Identify any *two* design principles that are implicit in the architecture. Briefly explain your answer.

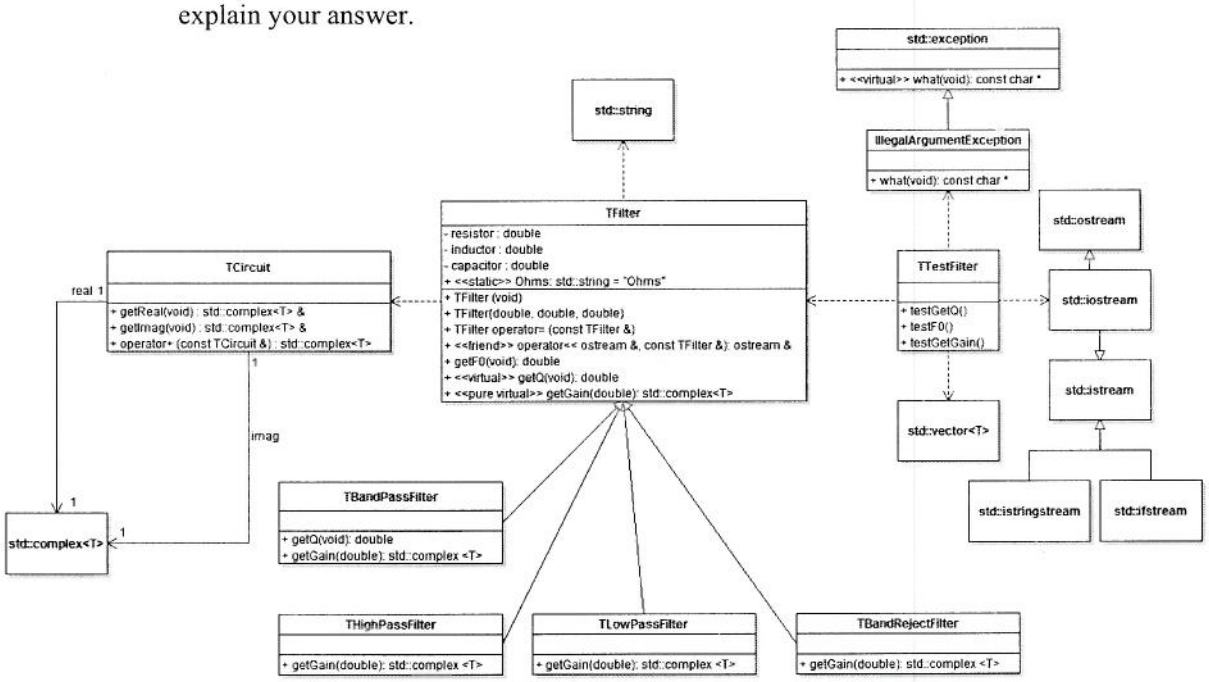


Figure 1.1 Architecture based upon the E2.12 Computing laboratories

2. The aim of this question is to explore C^{++} code reuse in the context of the Analogue filter case study developed during the E2.12 Computing laboratories. Figure 2.2 is one possible class diagram for the analogue filters based on the Engineering description in Figure 2.1

1. Every filter contains one each of the following primitive components: a resistor, a capacitor and an inductor.

2. For all the filters i.e. low pass, high pass, band pass and band reject, $f_0 = 1/(2\pi(LC)^{1/2})$

3. For the band pass filter $Q = (L/C)^{1/2}/R$ and for all the other filters $Q = R/(L/C)^{1/2}$

4. The frequency dependent complex transfer function (i.e. gain) is the complex impedance for the equivalent potential divider.

Figure 2.1 The Engineering description of the Analogue Filters

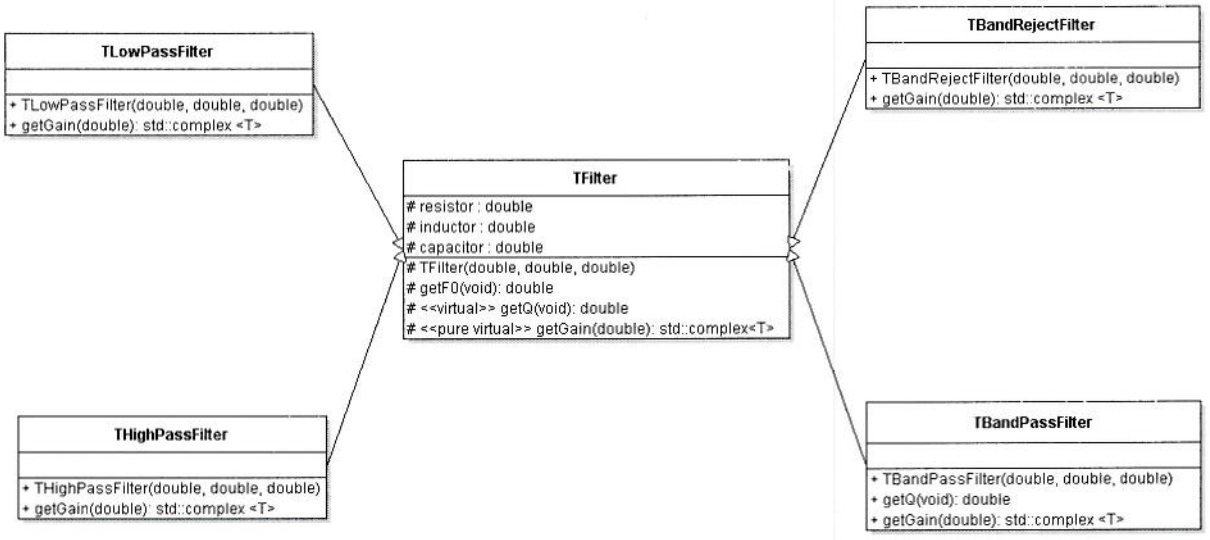


Figure 2.2 The filter class hierarchy based on the Engineering description (in Figure 2.1)

Continued on the following page.

a) In order to create instances of the derived filter classes in Figure 2.2 we need to provide implementation code in the five classes. Based upon Figures 2.1 and 2.2 make a list of fully qualified names for the *application* member functions where we *must* write *implementation* code in order to *completely* code the derived filter classes. Contrast the total *number* of entries in your list against the total *number* of code implementations there would be, if an inheritance hierarchy was *not* used (i.e. there is no base class).

[4]

b) Briefly, explain why it is tempting to think that `TFilter` *should not* include the OCCF members. Then briefly justify why we *should* include the OCCF members in `TFilter`.

[2]

c) Using only the information in Figures 2.1 and 2.2 briefly justify the inclusion or omission of the keyword `virtual` for each of the application member functions in `TFilter`. Distinguish between the *two* ways that the keyword `virtual` has been used.

[3]

d) Using only what is shown in Figure 2.2 write C++ *specification* code for the classes `TFilter` and `TBandPassFilter` (as it would appear in the *specification* file).

[4]

e) Using only what is shown in Figure 2.2 and *reusing* inherited code where possible, write C++ *implementation* code for the two class members `TBandPassFilter::TBandPassFilter(double,double,double)` and `TBandPassFilter::getQ(void)` (as it would appear in the *implementation* file).

[3]

f) List four distinct types of reuse of actual code implicit in Figure 2.2.

[2]

g) Drawing upon your answers above, explain why Figure 2.2 represents a useful framework for analogue filter construction. Use an example of another kind of analogue filter.

[2]

3.

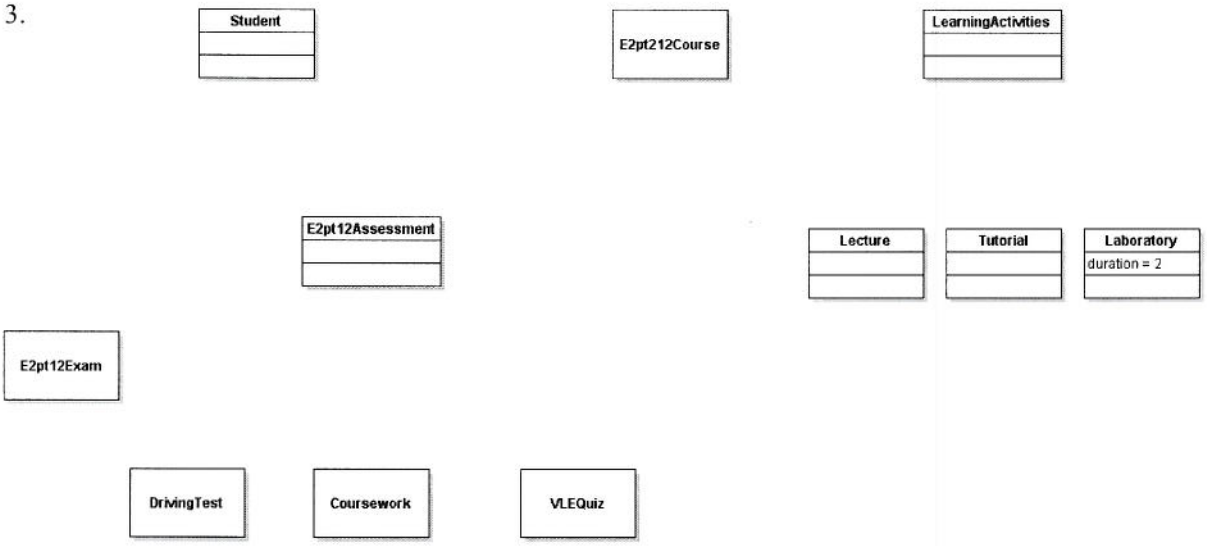


Figure 3.1 Partial class diagram.

a) Starting from Figure 3.1, draw the completed UML class diagram in your script, based upon a textual analysis of the textual description below. Include as much detail and precision as can be derived *only* from the text. Choose the most appropriate association type suggested by the text and meaningful identifiers for the association names, attribute names etc. You can assume a multiplicity of 1 where the text is not explicit. Only include the attributes and initial values that may be deduced from the text. The textual description follows:

> The E2.12 course consists of 16 learning activities. A learning activity records a fixed duration and subject content. There are various types of learning activity including: lecture, tutorial and laboratory which have duration of 2, 1 and 2 hours respectively.
>
> There are 20 students enrolled on the E2.12 course. The course has 2 assessments. There are 2 distinct types of assessment including: an exam and a coursework. Coursework consists of a driving test and a VLE based quiz.
>
> The exam maintains details for the 20 students that took the exam and also the single student with the top mark (there can only be one). Each student keeps a record of their E2.12 exam mark and their *unique* college identifier (CID).

[9]

Continued on the following page.

b)

i) One specific association in your diagram might be changed into a *derived* association, which one?

ii) Briefly describe the motivation for the change in (b i).

iii) Describe in words what the change in (b i) means in terms of coding.

iv) Illustrate the change in (b i) with a UML interaction diagram. You may use a sequence of two message sends to simulate iteration.

[4]

c)

i) One specific association in your diagram might be changed into a *qualified* association, which one?

ii) Briefly describe the motivation for the change in (c i).

iii) Illustrate the change in (c i) with a short code extract from a *generic* program showing the code *before* the change. You may assume that any additional getter that you need already exists.

iv) Illustrate the change in (c i) with a short code extract from a *generic* program showing the code *after* the change.

[4]

d) Redraw *only* that part of your class diagram that you produced for (a) that has *changed* as a result of the changes resulting from (b) and (c) above.

[2]

e) Using your completed class diagram suggest the need for another class that is *not* obvious in the textual description but follows from adopting an approach that aims to deliver reusable software components.

[1]

E2.12
Solutions 2009

[bookwork] [1]

1a) List *two* different kinds of relationships between classes. For each relationship identify an example in the diagram by naming the classes involved in that kind of relationship.

Uses e.g. `TFilter` uses `TCircuit`

Association e.g. `TCircuit` and `std::complex<T>`

Inheritance e.g. `TFilter` and `TBandPassFilter`

[bookwork] [1]

1b) List *any* OCCF members that are missing in `TFilter`?

```
~TFilter(),
```

```
TFilter(const TFilter &)
```

[bookwork] [1]

1c) List an example of an *instance* data member and a *class* data member. State which is the example of the *instance*/or *class* member.

`TFilter:resistor` is an instance data member

`TFilter::Ohms` is a class data member

[bookwork] [1]

1d) Identify a *relationship* with `TFilter` that could have been *drawn* and therefore could be added to Figure 1.1

`TFilter` *uses* `std::complex`

`TFilter` *uses* `std::string`

[bookwork] [1]

1e) Identify a generic UML *stereotype* that should be added to the class `TFilter`

Strong candidate is `<<abstract>>` because it is a pure virtual class.

Less strong candidates include `<<constructor>>`

[bookwork] [1]

[bookwork] [1]

1f) List *two* UML stereotypes that are C++ specific, that is, they are *not* generic UML stereotypes.

`<<static>>, <<virtual>>, <<pure virtual>>, <<friend>>`

[bookwork] [1]

1g) List *two* other details (i.e. not UML stereotypes) that are C++ specific, that is, they are *not* generic UML.

Reference variables, const, operator functions, templates type/functions/classes, friend function, scope resolution operator etc...

[bookwork] [1]

1h) Without loss of information, use an alternative and more *concise* way to redraw the relationship between `TCircuit` and `std::complex<T>`. You do not need to include the class members or classes *not* involved in the aforementioned relationship in your diagram.

one-to-many (i.e. 2) association replaces 2 one-to-one associations.

[bookwork] [1]

1i) Identify an example of *two* different *types* of string, that are *not* a C++ string, that are used in the architecture. For each string type state what type it is and where it appears in the architecture.

The literal for `TFilter::Ohms` is a C string. The value returned from the function `IllegalArgumentException::what()` is a C String.

A `istringstream` class is used in the architecture involved in the stream classes.

[bookwork] [1]

1j) Identify *all* the classes involved in an example of a *multiple* inheritance relationship. For each class state whether it is a derived or base class.

`std::iostream` is a derived from the base classes `std::istream` and `std::ostream`

[bookwork] [1]

1k) Identify *two* different types of polymorphism implicit in the architecture. For each string type state what type it is and where it appears in the architecture.

Template types in the template classes e.g. `complex<T>, vector<T>`

Overriden member functions e.g. `getGain(), getQ()`.

Overloaded operators e.g. `operator=(), operator<<()`

Overloaded constructors e.g. `TFilter(void)`, `TFilter(double,double,double)`

Overloaded member functions e.g. `string::get(void)`, `string::get(char)`, `ifstream::get(void)`, `ifstream::get(char)`

[bookwork] [1]

1l) Identify an example in the architecture where encapsulation is broken *knowingly*. Briefly explain your answer.

`TFilter` implements `operator<<()` as a friend function to provider the operator with direct access to the underlying object state.

[bookwork] [1]

1m) Identify an example in the architecture where encapsulation might be broken *accidentally*. Briefly explain your answer.

`TCircuit::getReal()` and `TCircuit::getReal()` both return references to objects.

[bookwork] [1]

1n) Is there evidence of Test Driven Development (TDD) in the architecture? Briefly explain your answer.

Yes, the `TTestTFilter` has member functions that appear to unit test each of the `TFilter` member functions.

[bookwork] [1]

1o) A STL class in the architecture may be swapped with an alternative STL class. Identify the *original* class and the name of *any* class that it may be swapped with.

`std::vector` could be swapped with `std::queue` and `std::deque`

[bookwork] [1]

1p) Which coding problem would have to be addressed, if a user written complex number class co-existed with `std::complex` in the class hierarchy?

Name collisions

[new computed example] [1]

1q) Briefly describe, in words or code, how a polymorphic *variable* involving *application* classes might be used.

```
TFilter *filter;
filter = new TBandPassFilter(rVal, lVal, cVal);
cout << filter->getGain(freq);
filter = new TBandRejectFilter(rVal, lVal, cVal);
cout << filter->getGain(freq); etc...
```

[new computed example] [1]

1r) Briefly describe, in words or code, how a polymorphic *variable* involving *utility* classes might be used.

```
ifstream fin("data.txt");
istringstream bin("c 1 1.0 one 1.1 2.2");
cin >> c >> i >> d >> s >> z;
fin >> c >> i >> d >> s >> z;
bin >> c >> i >> d >> s >> z;
```

Similar examples include: `vector/list/deque` etc...

[bookwork] [1]

1s) What form of handling errors in *service* code is implicit in the architecture?

Exception handling.

[bookwork] [1]

1t) List any *two* design principles that are implicit in the architecture, briefly explain your answer.

Information Hiding e.g. use of visibility modifier + and –

Separation of concerns e.g. getters, constructors, "do one thing and do it well" member functions.

Least privilege/const correctness – passing data into and out of operator/member functions as const.

Could argue there is evidence of low coupling/high cohesion. It would be harder to argue the case for Defensive Programming (including Design by Contract) with supporting source code, although the evidence of exception classes is a hint in that direction.

2a) Based only on what is shown in Figure 2.2, make a list of fully qualified names for where we *must* write *implementation* code for the application member functions in order to *completely* code the derived filter classes. Contrast the total *number* of entries in your list against the total *number* of code implementations there would be, if an inheritance hierarchy was *not* used (i.e. there is no base class).

[bookwork] [4]

```
//Note: TFilter::getGain() has no implementation code
TFilter::getF0(), TFilter::getQ(),

TBandPassFilter::getQ(),TBandPassFilter::getGain(),
THighPassFilter::getGain(),
TLowPassFilter::getGain(),
TBandRejectFilter::getGain().
```

Thus, we need to provide implementation code for 7 member functions when using inheritance as opposed to 12 (4*3) without inheritance.

2b) Briefly, explain why it is tempting to think that `TFilter` *should not* include the OCCF members. Then briefly justify why we *should* include the OCCF members in `TFilter`.

[bookwork] [2]

Since the class is abstract we can not create an instance of the class which would suggest we do not need the OCCF members e.g. assign. However, the OCCF exist for the benefit of the derived classes which may call base class constructors and member functions.

2c) Using only the information in Figure 2.2 briefly justify the inclusion or omission of the keyword `virtual` for each of the three application member functions in `TFilter`. Distinguish between the *two* ways that the keyword `virtual` has been used.

[bookwork] [3]

- The calculation of F_0 is the same for all the filters so the code is provided in the base class and is available to all the derived classes through drop-through inheritance.
- The calculation of Q is the same for all the filters except the band pass filter, so the code is provided in the base class and is available to all the derived classes through drop-through inheritance. However, the band pass filter needs to override the inherited code hence the member function is declared as virtual which means a derived class *may* override if needed.
- The calculation of gain is the different for all the filters, so no implementation code is provided in the base class. All the derived classes *must* override the inherited specification hence the member function is declared as virtual but is assigned equal to zero to indicate that there is no actual code.

2d) Using only what is shown in Figure 2.2 write C++ *specification* code for the classes `TFilter` and `TBandPassFilter` (as it would appear in the *specification* file).

[new computed example] [4]

```
class TFilter{
  public:
    TFilter(double rVal, double lVal, double cVal);
    double getF0(void);
    virtual double getQ(void);
    virtual complex <double> getGain(double)= 0;
  protected:
    double rVal, lVal, cVal;
};

class TBandPassFilter : public TFilter{
  public:
    TBandPassFilter(double, double, double);
    complex <double> getGain(double);
    double getQ(void);
};
```

2e) Using only what is shown in Figure 2.2 and *reusing* inherited code where possible, write C++ *implementation* code for the class members `TBandPassFilter::TBandPassFilter(double,double,double)` and `TBandPassFilter::getQ(void)` (as it would appear in the *implementation* file).

[new computed example] [3]

```
TBandPassFilter::TBandPassFilter(double rVal,
                                 double lVal,
                                 double cVal)
                                 :TFilter(rVal,lVal,cVal){
}


double TBandPassFilter::getQ(void) {
    return 1.0/TFilter::getQ();
}
```

2f) List four distinct types of reuse of actual code implicit in Figure 2.2

[bookwork] [2]

1. Reuse of utility classes e.g. `complex<T>`
2. Reuse of drop-through inherited code e.g. `getQ()`
3. Reuse of base class code by invocation from the derived class code e.g. `TFilter::TFilter(double,double,double)`, `TFilter::getQ(void)`
4. Reuse of base class e.g. `TFilter` to construct new derived classes e.g. `class THighPassFilter : public TFilter{ etc...`

2g) Drawing upon your answers above, explain why Figure 2.2 represents a helpful framework for analogue filter construction. Use an example of yet another kind of analogue filter.

[bookwork] [2]

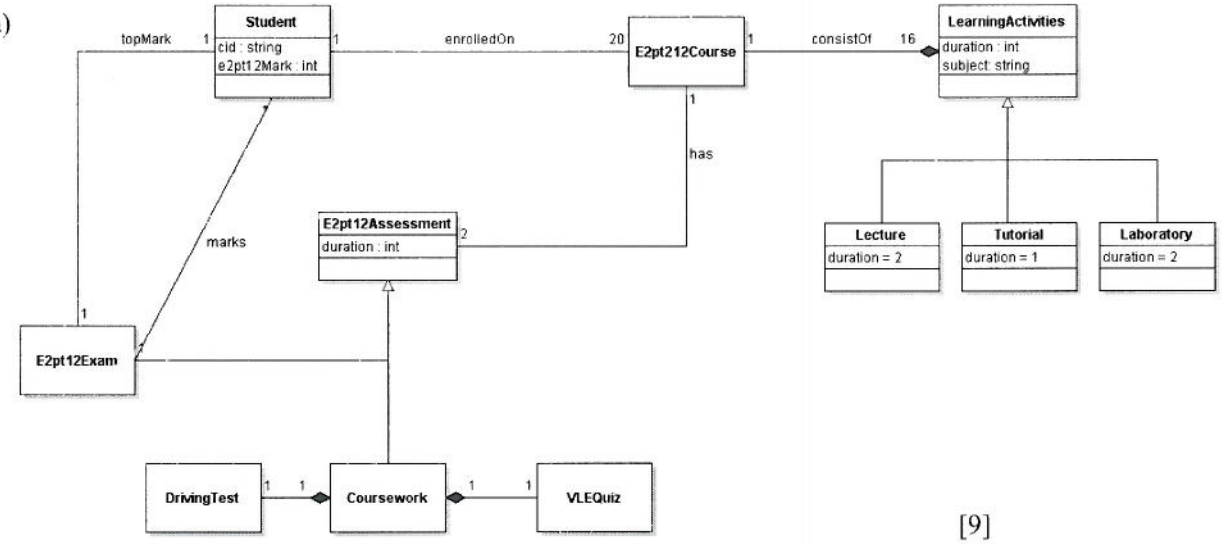
The answer to (f) reveals that we benefit greatly from reuse of existing code which reduces programming effort and saves time. We would also hope that this code is more reliable.

Taken as a whole, even the specification of member function (which does not provide actual code) but focuses our attention on what we need to provide.

The combination of implementation and specification code defines the essence of the concept we aim to build, acting as a blueprint for construction.

[20]

3a)



[9]

3b)

i) One specific association in your diagram might be changed into a derived association, which one?

`topMark`.

ii) Briefly describe the motivation for the change.

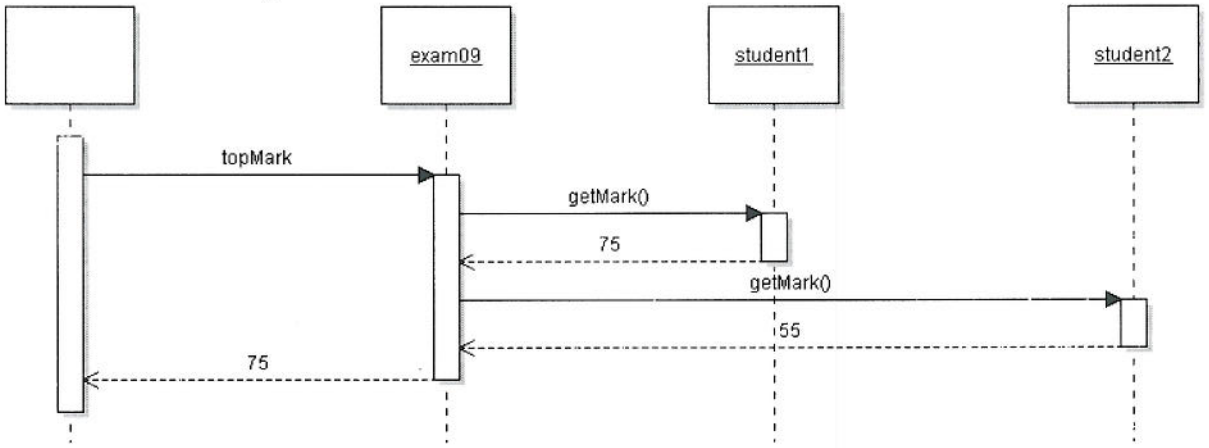
In code, an association becomes a data member that references an object. For a bi-directional association there is an instance variable in each of the two classes, referencing an object of the other class. The initialisation, updating and deletion of an object data member is potentially a high maintenance activity since it is easy to forget to create/modify/ delete links between pairs of objects involved in an association relationship. For example, in a bi-directional association we might delete one end of the link but not the other.

The association is shown with a forward slash in front of its name in the UML so that the programmer will be prompted to write the member function.

iii) Describe in words what this change means in terms of coding.

An object data member is replaced with a member function where the later can compute the same object reference as the link. Typically, an iteration through a set of objects (using another association e.g. `marks`) is made, checking a attribute(s) for some condition (e.g. `e2pt12Mark` student with the highest mark) is all that is required.

iv) Illustrate the change with a UML interaction diagram. Note: a couple of message sends can be used to simulate iteration.



[bookwork] [4]

3c)

i) One specific association in your diagram might be changed into a *qualified* association, which one?

`enrolledOn`.

ii) Briefly describe the motivation for the change.

If there are a large number of student instances enrolled on the E2.12 then a single student instance may be accessed more rapidly using a map/hashtable (as opposed to iterating through all the student instances until the required student is found). Effectively, a one-to-many association is converted into a one-to-one association.

iii) Illustrate the change with a short code extract showing the code *before* the change. Use C++ STL classes.

iv) Illustrate the change with a short code extract showing the code *after* the change. Use C++ STL classes.

Before	After

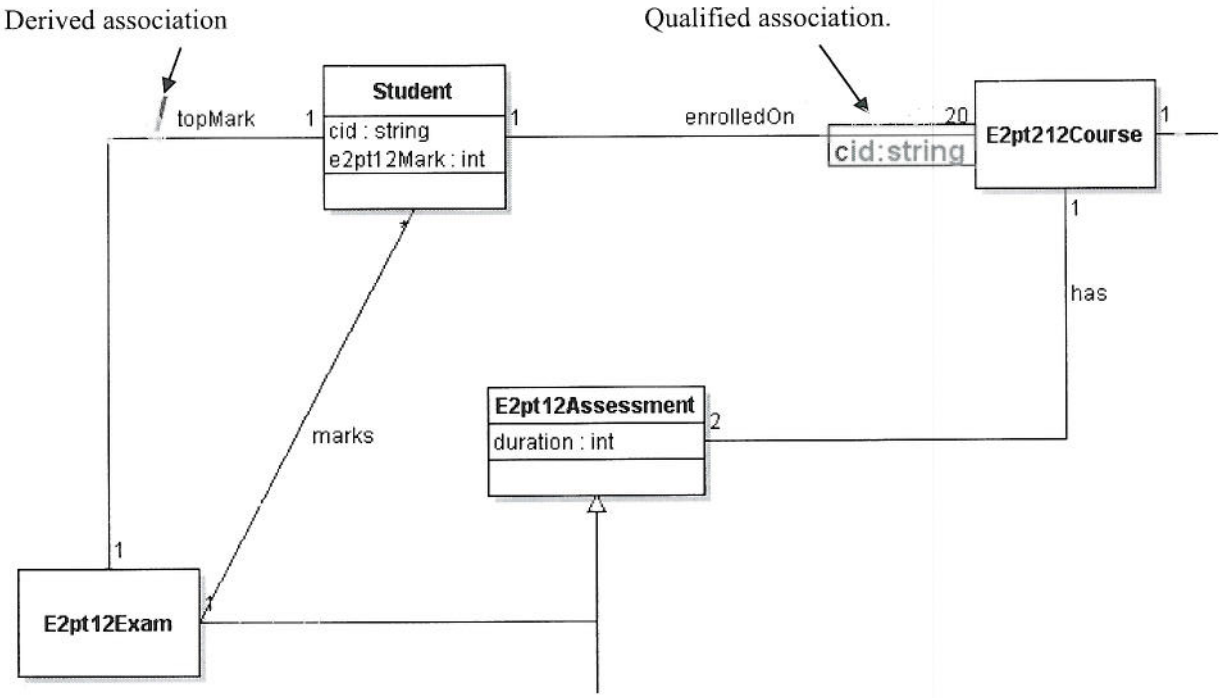
`typedef vector <Student> E2pt12Course;`
`E2pt12Course enrolledOn;`
`//enrol students etc…`
`enrolledOn.push_back(Member("123"));`
`// … then look for student with CID 007`
`E2pt12Course::iterator index;`
`for (index = enrolledOn.begin();`
`index!= enrolledOn.end(); index++)`
`if ((*index).getCID() == "007")`
`cout << (*index); //student found`
	`typedef map <string, Student> E2pt12Course;`
	`E2pt12Course enrolledOn;`
	`//enrol students etc…`
	`members.insert(make_pair("123",Student("123")));`
	`// … then look for student with CID 007`
	
	`cout << enrolledOn["007"]; //student found`

[new computed example] Code in italics not needed, incl. for completeness.

[4]

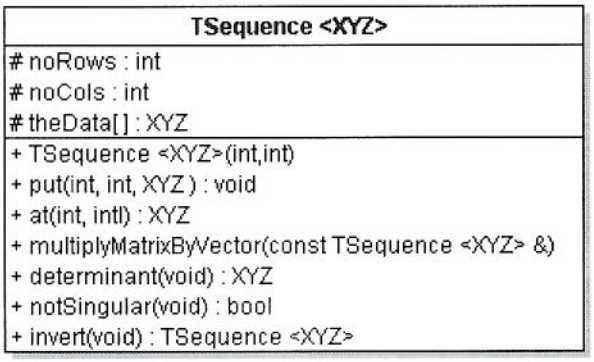
3d) Redraw *only* that part of your completed class diagram that has changed as a result of the changes resulting from (b) and (c) above.

[bookwork] [2]



4a) Translate the partial class specification shown in Figure 4.2 into a UML class diagram.

[new computed example] [3]



4b) Based upon the protocol shown in Figure 4.2, write a main program that implements the solution demonstrated by the Matlab™ code shown in Figure 4.1. The main program should display the calculated unknown currents.

[new computed example] [3]

```
TSequence <double> resistors(2, 2);
resistors.put(0, 0, 16); resistors.put(0, 1, -12);
resistors.put(1, 0, 4); resistors.put(1, 1, 6);

TSequence <double> voltages(2, 1);
voltages.put(0, 0, 24); voltages.put(1, 0, 24);

TSequence <double> resistorsInv(2,2);
resistorsInv = resistors.invert();

TSequence <double> unknownCurrents(2,1);
unknownCurrents = voltages.multiplyMatrixByVector(resistorsInv);
cout << unknownCurrents.at(0, 0) << unknownCurrents.at(1, 0);
```

4c) The main program could include a C++ assertion that is relevant to the matrix algebra. What members should have to be added to the current protocol in order that the assertion can work? Write the C++ assertion statement assuming that the member functions have been added.

[new computed example] [1]

Need to add the accessors: `getNoCols()` and `getNoRows()`. Assert checks dimensions of vector and matrix and that the matrix is invertible.

```
assert((resistors.getNoCols() == 2) &&
       (resistors.getNoRows() == 2) &&
       (voltages.getNoRows() == 2) &&
       (voltages.getNoCols() == 1) &&
       (resistors.notSingular() == true));
```

4d) Briefly describe what is a helper member function? Where might a helper function be applied in this application?

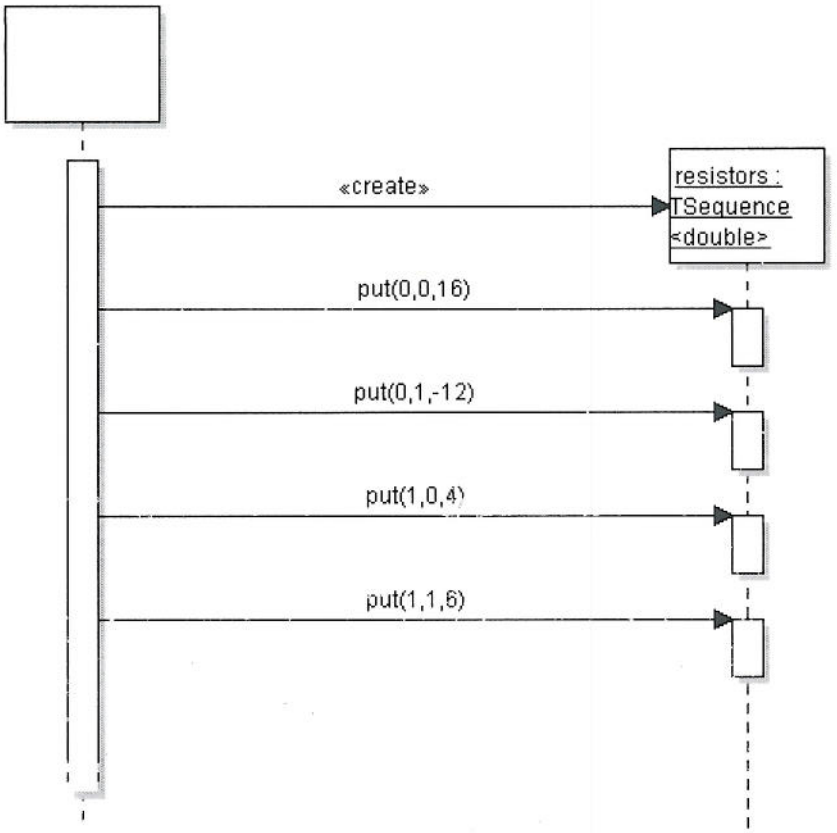
[new computed example] [2]

The mapping of the row and column indices into a linear array is could lead to out-of-bounds errors. A helper member function with private visibility called `isValid(int,int)` validates two integer values corresponding to a row and a column value for use with `theData` which is indexed from zero. Uses the member functions `getNoRows()` and `getNoCols()` to test the upper values. Code not required but for completeness, code:

```
private:
 bool isValid(int r, int c) {
    return ((r>=0) && (r<getNoRows()) &&
            (c>=0) && (c<getNoCols()));
 }
```

4e) Draw an interaction diagram corresponding to an extract of the main program showing the *instantiation* of any one of the objects in your program.

[new computed example] [2]



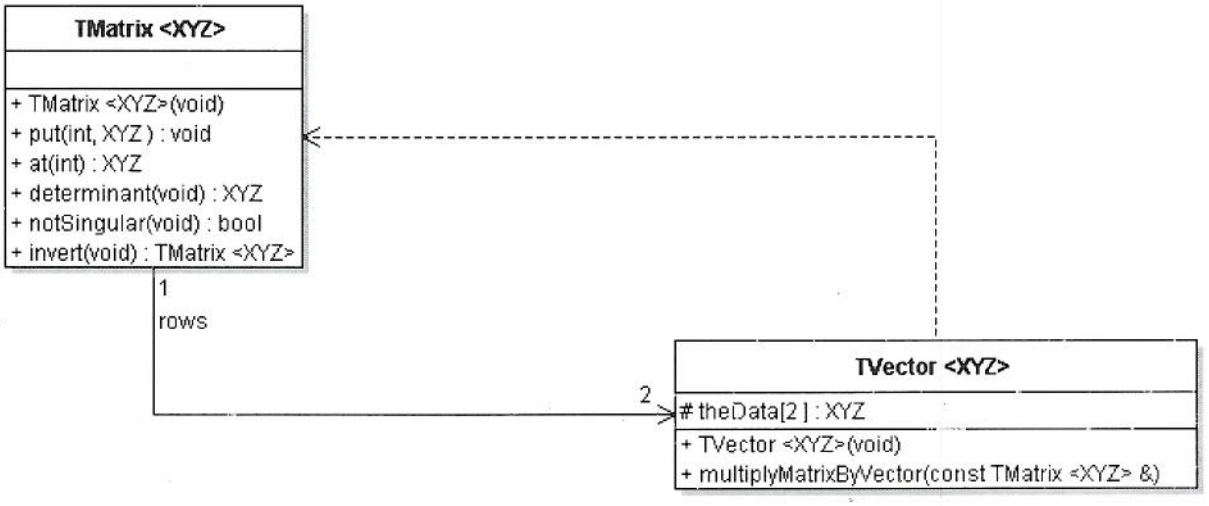
4f) Write C^{++} code for the *body* of the constructor as it would appear in an implementation file and based on the following software contract:
Precondition: None. Postcondition: Formal arguments copied to number of rows/cols data members, memory allocated to array and array elements initialized to zero.

[new computed example] [2]

```
noRows = nr;noCols = nc; theData = new XYZ [noRows*noCols];
for (int i = 0; i < noRows*noCols; i++)
  theData[i] = 0;
```

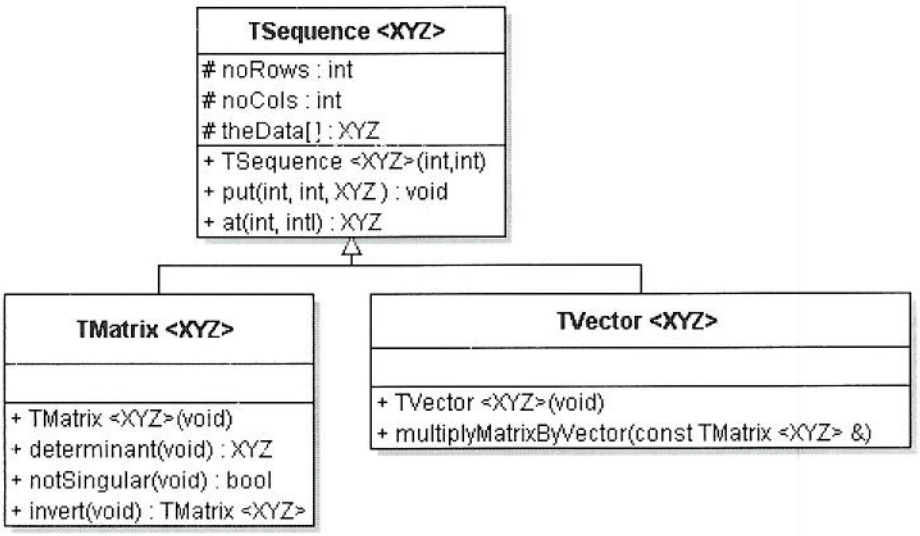
4g) Draw a UML class diagram for each of the two new architectures described below.

i) *Replace* the original class `TSequence` with its members distributed between two classes based on matrix algebra.



ii) *Refactor* the original class `TSequence` so that it is now a *base* class to the two matrix algebra classes created for (i) i.e. they are *derived* class

[new computed example] [6]



h) Name the *category* (i.e. group) of STL classes, any of one of which could be used to replace the dynamic array data member in the `TSequence` class.

[bookwork] [1]

First-class sequence