IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2015

EIE PART II: MEng, BEng and ACGI

SOFTWARE ENGINEERING 2: OBJECT-ORIENTED SOFTWARE ENGINEERING

Wednesday, 10 June 2:00 pm

Time allowed: 2:00 hours

Corrected Copy

There are THREE questions on this paper.

Answer ALL questions.
Q1 carries 40% of the marks. Questions 2 and 3 carry equal marks (30% each).

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

Examiners responsible	First Marker(s) :	M. Cattafi
	Second Marker(s) :	J.V. Pitt

SOFTWARE ENGINEERING 2: OBJECT ORIENTED SOFTWARE ENGINEERING

1. This is a general question about Object Oriented Software Engineering.

   a) Consider the concept of *state* in Object Oriented Software Engineering.

      i) Explain what is meant by "state of an object". Illustrate your answer using an example in C++ code dealing with points on the Cartesian plane.

      [6]

      ii) Explain what "inconsistent state" means in this context. Illustrate your answer using an example in C++ code expanding on the previous one.

      [6]

      iii) Explain how encapsulation and abstraction can avoid inconsistent states while keeping the object mutable. Illustrate your answer using an example in C++ code expanding on the previous one.

      [6]

   b) Consider an application domain related to geometric entities, in particular dealing with triangles, circles and points. Shapes are represented in terms of points (and other attributes when needed). A translation operation should be available for any entity (effect of applying a geometric vector expressed by a point). For instance if a point p1 is at coordinates (1, 2) and a point p2 is at coordinates (3, 4), after translating p1 by p2, p1 should be at coordinates (4, 6).

      i) Describe in words how you would model this domain in an object oriented architecture.

      [6]

      ii) Draw a UML class diagram of the architecture.

      [6]

      iii) Write a declaration for all the classes. The declarations can be kept to the essential skeleton (e.g. constructors can be omitted) but all the relevant elements needed in order to express the architecture should be included. Moreover, include the definition (where needed) for the member function which implements the translation.

      [10]

2. This question deals with container classes and memory management in C++. Write the implementation (you can keep the definition and the declaration together) of a container class representing a *stack* of integers, i.e. a data structure on which elements (in this case integer numbers) can be pushed (making the stack grow) and from which elements

can be popped (making the stack shrink). Stacks are "last in, first out" data structures.

a) Declare suitable member data for the class. The implementation should be based on a dynamically allocated array.

[3]

b) Define a constructor which takes as argument an integer representing the initial physical size of the stack.

[4]

c) Define the copy constructor.

[4]

d) Define the assignment operator. FOR ASSIGNMENT FROM OTHER STACKS.

[6]

e) Define the destructor.

[3]

f) Define a `push` member function which models the operation of pushing an element (passed as argument) on the stack.

[6]

g) Define a pop member function which models the operation of popping an element (which is also returned) off the stack. The attempt to pop an element off an empty stack does not need to be handled, it is assumed that it may result in undefined behaviour.

[3]

h) Define an `empty` member function which returns true if the stack is empty and false otherwise.

[1]

3. This question deals with C++ templates and related concepts.

a) Explain why the headers where template classes and functions are declared usually also contain their definitions.

[8]

b) The Standard Template Library includes a container template class `pair` with two (public) member data fields, `first` and `second` (not necessarily both of the same type). Write a skeleton of this class including the member data and a constructor initializing them. Show with a code snippet how the class can be instantiated and used (e.g. in the main).

[8]

c) Using the following code as a starting point, explain why iterators are useful illustrating your answer with an example in C++ code.

Let `items` be a `vector` (e.g. containing some integers).

```
for(int i = 0; i < items.size(); i++){
    cout << items[i] << endl;
}
```

[8]

d) Consider the (global) function `sort` included in the header `<algorithm>` which takes as arguments the initial and final iterators of a sequence (contained in a container class) in order to sort its elements. Describe the conditions that the container and its elements need to respect in order to be used with this function.

[6]