SOFTWARE ENGINEERING 2: OBJECT ORIENTED SOFTWARE ENGINEERING

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

   a) Compare and contrast how software complexity is handled in Structured Programming and Object Oriented Programming.

[9]

[bookwork]

In Structured Programming the focus is on the flow of program execution. Instructions are grouped in blocks with single entry and exit points. These blocks can be controlled by conditions (if...else) or repeated while a condition is true (e.g. while loops). Instructions are also enclosed in functions which can call each other and communicate (only) through the parameter list and the return value.

In Object Oriented Programming the focus is on the concept of state (of objects), which is represented in class member data and usually manipulated and accessed from outside the class using member functions (in turn usually implemented according to Structured Programming). Objects are instances of classes, classes can be organized following a conceptualization of the problem domain expressing different roles and responsibilities, relationships, hierarchies etc.

[Most students outlined with sufficient detail how complexity is handled in Object Oriented Programming (although few mentioned important concepts such as state). A few students provided a rather vague description as far as structured programming is concerned.]

   b) Explain the OOP concept of *encapsulation*. Illustrate your answer with an example in C++ code.

[8]

[bookwork, programming]

In Object Oriented Programming the state of objects is kept encapsulated in order to prevent direct access to member data from outside the class and enforce that state manipulation and access happen only through member functions. In C++ this can be achieved by declaring member data private, for instance:

```
class point{
    private:
        double x;
        double y;

    // etc...
```

```
};

int main(){
    point p;
    p.x = 10;
    // compiler error, x is private
    return 0;
}
```

[Most students answered correctly.]

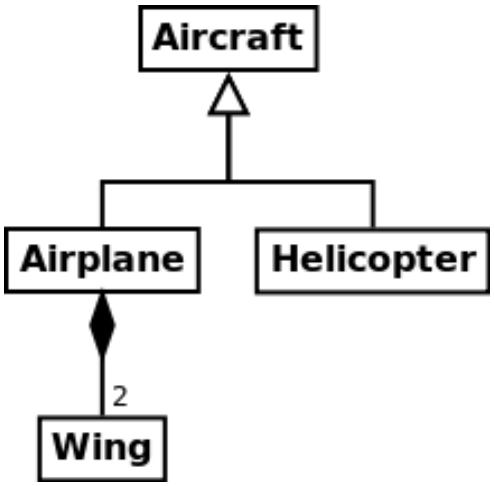

Figure 1.1 An UML diagram

c) Describe in words the software architecture represented in the UML class diagram in Fig. 1.1. What kind of problem domain could it model and what kind of software application?

[7]

[bookwork, example analysis]

The diagram represents four classes: Aircraft, Airplane, Helicopter and Wing. Airplane and Helicopter inherit from Aircraft. There is a relationship of composition between Airplane and Wing (an Airplane "has" two Wings).

[Most students answered correctly. Some did not mention the composition relationship.]

The domain appears to be aeronautics and the software application could be for instance an aerospace design software or a flight simulator.

[Some students answered vaguely.]

d) Compare and contrast the OOP concepts of *inheritance* and *composition* also expanding on the examples and the context of Fig. 1.1.

[9]

[bookwork, example analysis]

Inheritance is conceptually an "is-a" or, better, "extends" relationship between a (more general) "base class" and a (more specific) class inheriting from it. For instance both Airplanes and Helicopters are flying machines (Aircrafts) but they fly in different specific ways. Operations defined for general Aircrafts (e.g. "take off") will be also defined for more specific classes, however the implementation might differ reflecting the specific behaviour or characteristics.

Composition is a "has-a" relationship between classes: one class can have member data which are instances of several other classes and are "component pieces" sharing its life cycle. Several operations defined on the composed class are likely to be delegated or performed through the component classes, for instance operation "take off" on Airplane is likely to involve a change of the state (e.g. "flaps position") of the Airplane Wings.

[Most students outlined a correct answer ("is-a", "extends" vs. "has-a"), however a few did not expand on the context with a meaningful example.]

e) Write C++ declarations for all the classes in the UML diagram in Fig. 1.1. The declarations can be kept to the essential skeleton (e.g. constructors can be omitted) but all the elements related to available information (including relationships) should be included.

[7]

[bookwork, example analysis, programming]

```
class Aircraft {
    // etc...
};

class Wing {
    // etc...
};

class Airplane : public Aircraft {
    private:
        Wing lw;
        Wing rw;
    // etc...
};

class Helicopter : public Aircraft {
    // etc...
};
```

[Most students answered correctly. Some did not represent the composition relationship.]

2. This question deals with C++ references and features enabling or preventing value updates.

a) Consider the following code snippet. Would it compile (why or why not)? Would it (be likely to) cause runtime errors (why or why not)? Trace (if possible and meaningful) how the value of the variables would change.

```
int n = 10;
int& nr;
nr = n;
```

[6]

[bookwork, example analysis] The code would not compile because instruction `int& nr;` tries to declare a reference without initialization (the variable being referenced is not specified).

[Most students answered correctly, some did not remember that references cannot be uninitialized.]

b) Consider the following code snippet. Would it compile (why or why not)? Would it be likely to cause runtime errors (why or why not)? Trace (if possible and meaningful) how the value of the variables would change.

```
int n = 10;
int& nr = n;
int n2 = 3;
nr = n2;
n2 = 2;
```

[6]

[bookwork, example analysis]

The code would compile and run correctly.

Line 1: `n is 10`
Line 2: `n is 10, nr is 10`
Line 3: `n is 10, nr is 10, n2 is 3`
Line 4: `n is 3, nr is 3, n2 is 3`
Line 5: `n is 3, nr is 3, n2 is 2`

[Most students answered correctly, some made mistakes in the tracing of the values.]

c) When and why are function parameters in C++ declared `const`, usually? Illustrate your answer with an example in code including the implementation of such a function.

[8]

[bookwork, programming] In C++, objects are usually passed by `const` reference to functions which need read-only access to their state or content but do not need to change it. For instance:

```
void print_int_vector(const vector<int>& v){
    for(int i = 0; i<v.size(); i++){
```

```
            cout << v[i] << endl;
        }
    }
```

Passing by reference is often more efficient than doing a copy (in particular for large objects), however the advantage of having a copy is that the original object cannot be (accidentally) altered. Passing by const reference keeps both advantages.

[Most students outlined a correct answer, although some misunderstood the question, did not provide a meaningful example or omitted part of the reasons.]

d) When and why are member functions declared `const` in C++ (by placing the keyword after the parentheses enclosing the parameters)? Illustrate your answer with an example in C++ code including such a declaration (in its appropriate context) and a call of the function in relevant circumstances.

[10]

[bookwork, programming]

Member functions are declared `const` when they do not alter the object state. Consider for instance:

```
class point{
    private:
        double x;
        // etc...
    public:
        double get_x() const {
            return x;
        }
        // etc...
};

void print_point(const point& p){
    cout << p.get_x() << endl;
    // etc...
}
```

Member function `get_x` can be declared const because it doesn't change the state of the object. The const declaration allows function `print_point`, to which point `p` is passed by const reference, to call member function `get_x`.

[A few students did not remember this aspect. Some did not provide a meaningful example.]

3. Consider memory management in C++ and Java.

a) Consider the following function. Would it compile (why or why not)? Would it be likely to cause runtime errors (why or why not)? If the function is not correct,

propose and comment a correct alternative (keeping the function declaration unaltered).

```
int* make_it_double(int n){
    int m = 2 * n;
    return &m;
}
```

[10]

[bookwork, example analysis, programming]

The function would compile, however the compiler might notice (and thus issue a related warning) that the function is returning the memory address of a local variable (m). This variable goes out of scope when the function terminates, thus its memory is deallocated and its address cannot be used reliably to access the value, and such attempts might result in a runtime error.

The function could be written correctly (keeping the same declaration) making use of dynamic memory allocation:

```
int* make_it_double(int n){
    int* m = new int;
    *m = 2 * n;
    return m;
}
```

However in this case the caller has the responsibility to deallocate the memory.

[Most students identified the memory leak, however some erroneously identified wrong syntax or proposed unsuitable alternatives.]

b) What are memory leaks? How do they occur? What needs to be done in order to avoid them? Illustrate your answer with an example in C++ code of a situation with a memory leak and of how it can be fixed.

[12]

[bookwork, programming]

Dynamic memory allocation requires the program to allocate memory when it is needed and deallocate it when it is not needed anymore. Memory leaks occur when dynamically allocated memory areas become unreachable before being deallocated and thus are wasted; the repeated occurrence of this may deplete the available memory.

For instance considering the following function:

```
int* make_it_double(int n){
    int* m = new int;
    *m = 2 * n;
    return m;
}
```

And the following code snippet:

```
while(cin >> n){
    int* n2 = make_it_double(n);
    cout << *n2 << endl;
}
```

At each iteration the pointer containing the memory address returned from the function is overwritten without deallocating the pointed memory area. In the following version there are no memory leaks:

```
while(cin >> n){
    int* n2 = make_it_double(n);
    cout << *n2 << endl;
    delete n2;
}
```

[Most students had a general idea of memory leaks, however some did not provide a meaningful example or provided a vague explanation.]

c) Explain how Java differs from (standard) C++ in terms of memory management also mentioning related advantages and disadvantages.

[8]

[bookwork]

In Java there is no "delete" instruction deallocating dynamic memory for objects, instead a reference count is kept and the "garbage collector" periodically deallocates the memory for objects which are not referenced anymore. One advantage of this is that the programmer doesn't need to be concerned with memory leaks. However this system introduces overhead which can penalize performance.

[Most students correctly mentioned the Java garbage collector, however some provided only vague descriptions of the related advantages and disadvantages.]