Midterm Examination

9.30am-10.20am, Friday, October 18, 2013 Instructor: Jacek Wolkowicz

Name: .....

Student Number: .....

Student Signature: .....

Duration: 45 minutes
Aids allowed: None

Fine print:

1. Place your student card on the table beside you and fill out the cover page with your name, student number and signature. An invigilator will check your ID during the exam.
2. This examination has **8** pages. Ensure that you have all pages
3. The use of any electronic or analog devices during the test, apart from a pen, a pencil and your brain is strictly prohibited. Turn off all your electronic devices and leave them in your bag. Place all your bags, coats, and books in the centre lane.
4. You must hand in the exam. You may not remove the exam from the room.
5. You may not ask questions of invigilators, except in cases of supposed errors or ambiguities in examination questions.
6. Write your answers in the space provided. If you need more space use the reverse side of the page and indicate this in the provided space.
7. **Your answers should be at most 3 sentences long!**
8. Write legibly and neatly.
9. Complete as much of the exam as you can.
10. Good Luck!

Question	Value
1	/5
2	/9
3	/6
4	/10
5	/10
6	/10
Total	/50

1. [5 points] Answer True or False for the following questions and **provide a brief (one or two sentence) justification**:

(a) [1] Most complex systems are organized into a few objects.

False. Most complex systems comprise many objects, which are organized into hierarchies.

(b) [2] Encapsulation is useful for increased coupling.

False. Encapsulation decreases coupling. Encapsulation hides implementation, preventing other modules from making assumptions about the implementation.

(c) [2] Composition is similar to aggregation relationship.

True. Composition is aggregation where the containment is strict. In composition the aggregate is responsible for the collection, but in aggregation this is optional.

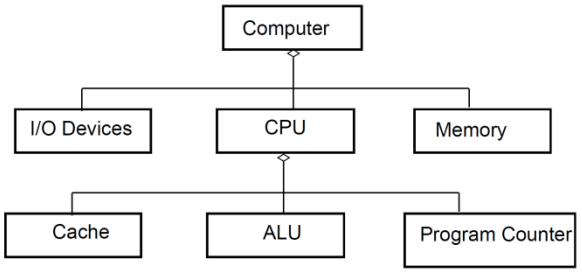
2. [9 points] These questions deal with the foundations of the object oriented approach.

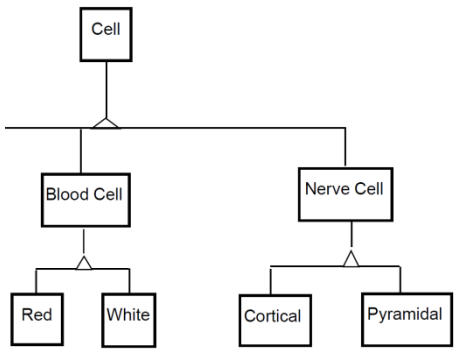
(a) [4] List two (2) common kinds of hierarchies and briefly describe their parent-child relationships. Draw an example of each of the two hierarchies. Each example should comprise at least three objects.

"Is A" : The child is a specialization of the parent.

"Part Of" : The child is a component of the parent.

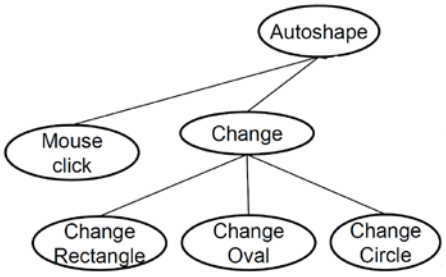
"Uses" : The child is used by the parent.





(b) [3] What is the difference between algorithmic and object-oriented decomposition? Give an example of each.

Algorithmic decomposition identifies what is to be done and divides the task into smaller subtasks until the subtasks can easily be solved. Objectoriented decomposition, identifies the entities within the problem statement and their interactions to divide the problem into manageable portions.



Any class diagram or object diagram would do

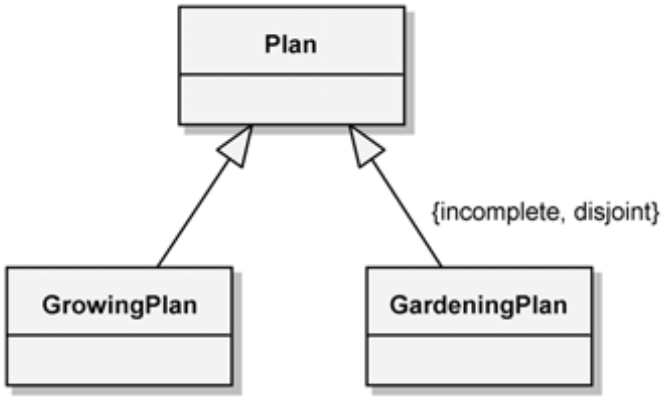
(c) [2] How does encapsulation facilitate abstraction?

Encapsulation hides the implementation of an abstraction and decouples the behaviour of the abstraction from the implementation.

3. [6 points] For each of the following relationships identify the conditions under which it should be used and give an example of the relationship. Use UML class diagrams to depict the examples.

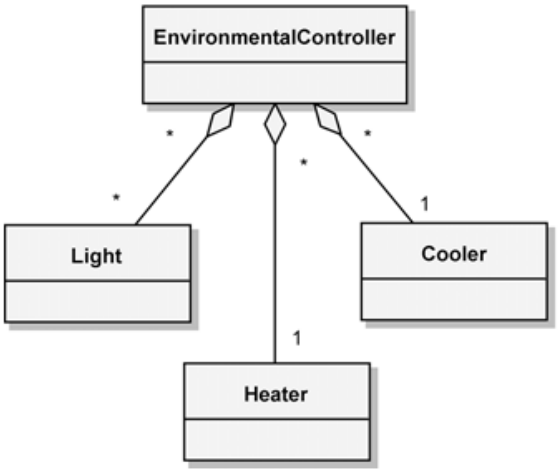
(a) [2] inheritance

Should only be used when the class is specialization of the superclass, i.e., satisfies the "is a" property. E.g. Circle inherits from Shape.



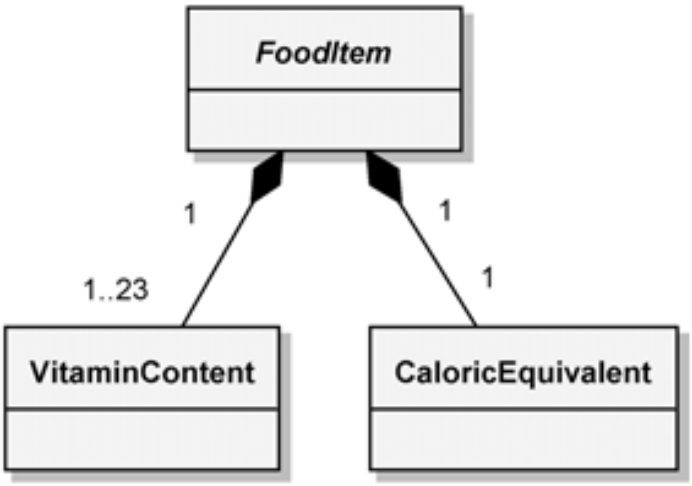
(b) [2] aggregation

Should be used when the aggregate class/object comprises a collection of other classes/objects that are not strictly contained in the class/object and/or whose lifetimes are not subsumed by the lifetime of the class/object. E.g., Library is a collection of books.



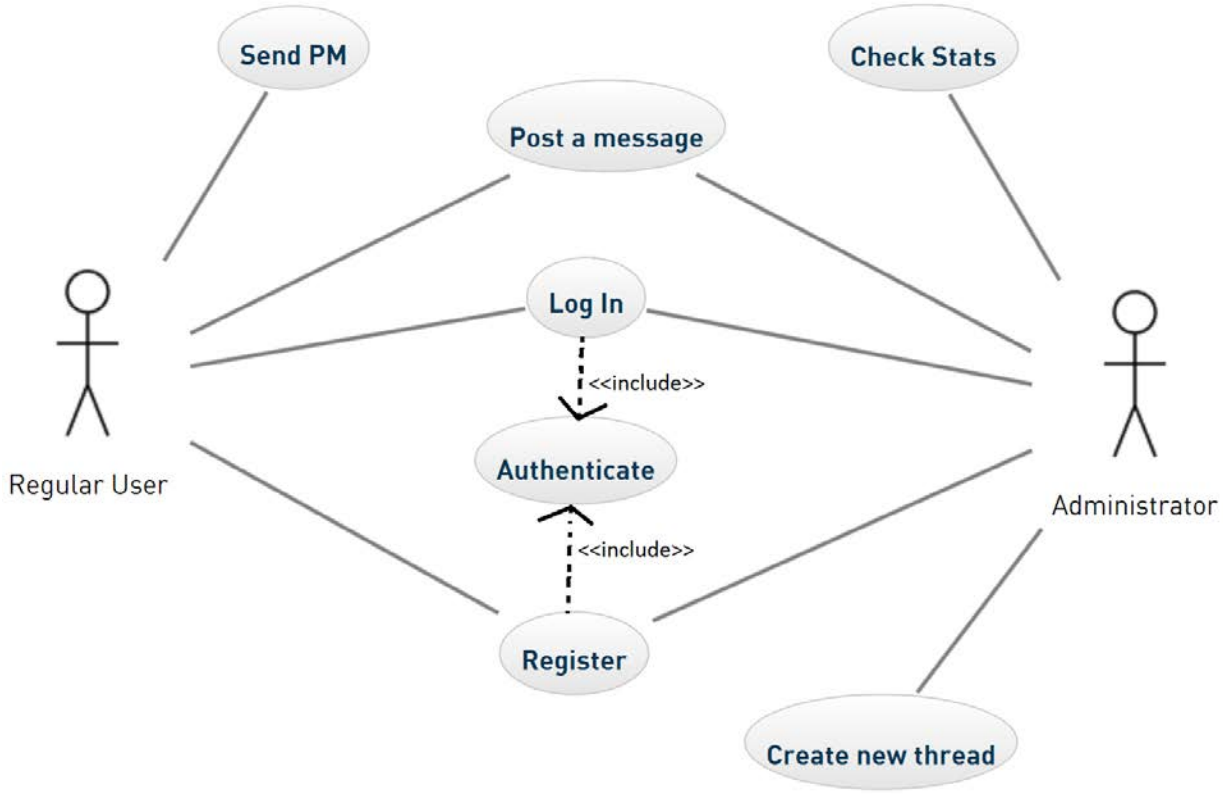
(c) [2] composition

Should be used when the aggregate class/object comprises a collection of other classes/objects that are strictly contained in the class/object and whose lifetimes are subsumed by the lifetime of the class/object. E.g., Car is a collection of parts.



4. [10 points] Imagine that you are analyzing requirements for an online forum system. Forums can get very complex, but imagine that we have only two kinds of users that interact with our system with different responsibilities: Regular Users and Administrators. Both can log in to the system, and part of logging in is an internal authentication process. Both can also register with the system, which also uses internal authentication. After logging in, everybody can post new messages to the board, however only Administrators can check statistics and create new threads. Regular users on the other hand can send private messages to other users, while administrators do not have this ability.

Draw a Use Case diagram that contains Actors, Use Cases and their relationship from the scenario described above.

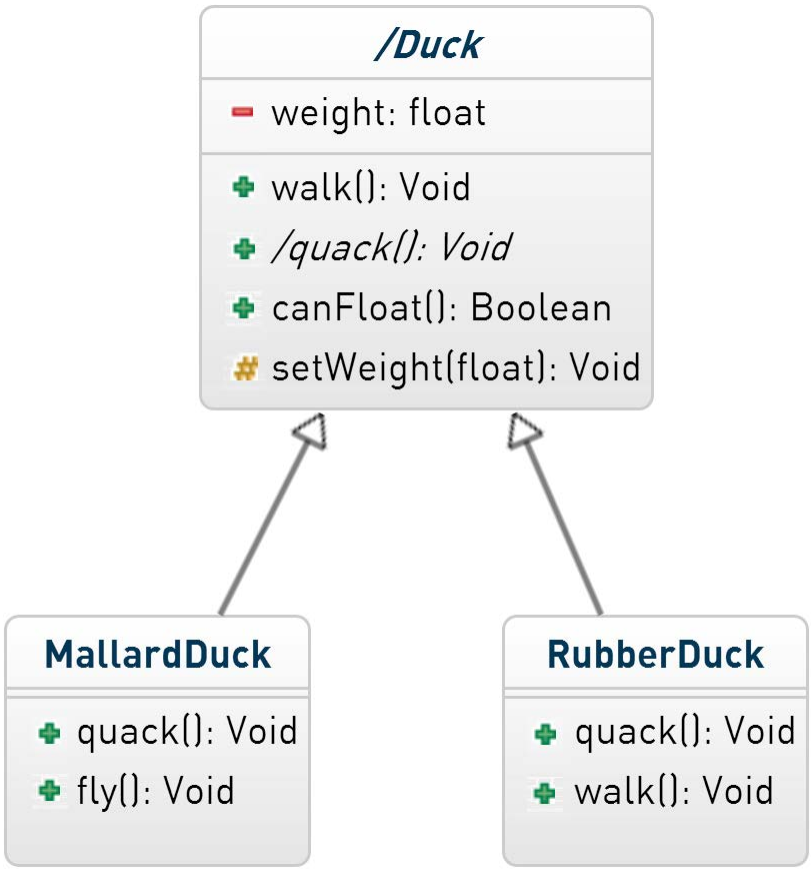


5. [10 points] Consider an abstract class Duck, representing all available ducks. Every duck can quack and walk, which are publically accessible functionalities of every duck. While every duck walks the same way, they quack differently which makes quacking an abstract feature of a duck. Every duck has a weight (float) which determines their ability to float (no pun intended), which ability can be checked by everybody through canFloat method. The weight of a duck can be set through a method available only for all concrete ducks implementations. A rubber duck, a kind of a duck, has all the features of a duck (yes, it quacks), but when asked to walk – it acts differently, because a rubber duck can't walk, so it overrides a duck walk feature and does nothing. An example of a duck is also a mallard duck that apart of quacking and walking, can also fly.

Draw a class diagram depicting classes, with names, attributes and methods along with proper relationship between classes. Skip constructors and destructors, but don't forget to annotate elements visibility using UML notation. For the actual attributes types or method signatures you can use Java or C++ notation, if you wish. Preceed a name with a '/' symbol to indicate slanted (italicized) text.

Use identifiers (class, attributes and method names) from the following list (you can repeat them if you need it.) (Hint: you should include all of them in your solution):

- Duck, MallardDuck, RubberDuck, walk, quack, fly, weight, canFloat, setWeight



6. [10 points]

[8 points] Write the body of a function template (myfilter) which copies elements satisfying a predicate from one container to another. The first two parameters are InputIterators marking respectively the beginning (included) and end (excluded) of the input range. The third parameter is an OutputIterator marking the beginning of the range where the result is to be put. The fourth parameter is the Predicate to check, which can be used as a boolean function on the stored elements. The function must return with the OutputIterator to the element right after the last copied one.

If the function template works correctly, the following code should yield the result given in the comment:

```
class EvenIntegers
{
public:
    bool operator()(int x) { return x%2==0; }
};
int main()
{
    int v[8] = { 5, 6, 7, 8, 9, 11, 2, 4 };
    int w[8];
    int *p = myfilter(v, v+8, w, EvenIntegers()); //w contains {6,8,2,4,...},
}                                                 //p equals w+4
```

Here is the myfilter definition (place your solution between braces):

```
template<class InputIterator, class OutputIterator, class Predicate>
OutputIterator myfilter(InputIterator start, InputIterator end,
                        OutputIterator result, Predicate pred)
{
   while (start != end)
   {
      if (pred(*start))
         *(result++) = *start;
      start++;
   }
   return result;

}
```

[2 points] What are the remaining values of elements of the w[] array, after the fourth element and why is that?

They have undefined garbage values, but are still accessible