THE UNIVERSITY OF WESTERN ONTARIO
LONDON CANADA

COMPUTER SCIENCE 3307A
FINAL EXAMINATION
DECEMBER 10, 2019
3 HOURS

NAME: Ali Al-Musawi

STUDENT NUMBER: _____

Question	
1-20.	_____
21-22.	_____
23.	_____
24.	_____
25.	_____
26.	_____
27.	_____
28.	_____
29.	_____
30.	_____
31.	_____
32.	_____
33.	_____
34.	_____
35.	_____
36.	_____
TOTAL	_____

(Out of 180 marks)

There are no cheat sheets, books, or other reference materials allowed for this exam. No calculators or other electronic devices are permitted either.

Part I -- Multiple Choice, True/False -- Choose the best answer from the choices given. Circle your answer on the paper. [40 marks total, 2 marks each]

1. In C++, a non-null pointer always "points to" data allocated on the heap.
   a. True.
   b. False.

2. If a variable `i` is declared as:

```
const int i = 0;
```

   then the value of `i` cannot be changed in subsequent code.
   a. True.
   b. False.

3. The syntax `const int *bob` indicates that `bob` is a:
   a. Regular pointer to regular int type.
   b. Constant pointer to regular int type.
   c. Regular pointer to constant int type.
   d. Constant pointer to constant int type.

4. Under what circumstances should you declare a function as `const`?
   a. When the function modifies only the object's member data.
   b. When you want to call the function on references to objects of that class.
   c. When the function does not modify `*this`.
   d. When the function creates no local variables.
   e. None of the above.

5. Space is always allocated on the stack when:
   a. `new` is invoked.
   b. An assignment statement is executed.
   c. A function is called.
   d. A local object variable is modified.
   e. None of the above.

6. Space is always allocated on the heap when:
   a. `new` is invoked.
   b. A local object variable is modified.
   c. An object parameter is passed by reference.
   d. A constructor is called.
   e. None of the above.

7. Consider the following C++ code snippet:

```
1   int i = 4;
2   int j = 5;
3
4   int* k = &i;
5   int& l = j;
6
7   *k = 9;
8   l = 44;
9
10  cout << i << " ";
11  cout << j << " ";
12  cout << *k << " ";
13  cout << l << endl;
```

What will be the output of the snippet? Assume `cout` and `endl` have been imported.

a. `4 5 9 44`
b. `4 5 4 5`
c. `9 5 9 44`
d. `9 44 9 44`
e. The snippet will not compile.

8. Consider the following function:

```
1 template <class T>
2 T average(T *atArray, int nNumValues) {
3          T tSum = 0;
4          for (int nCount=0; nCount < nNumValues; nCount++)
5             { tSum += atArray[nCount]; }
6          tSum = tSum / nNumValues;
7          return tSum;
8 }
```

Which of the following statements about the class/type `T` must be true in order for the code to compile and run without crashing?

a. It must be some kind of numeric type.
b. It must have the < operator defined.
c. It must have the `[]` access operator defined.
d. All of the above.
e. None of the above.

9. In C++, the destructor for an object is always called when the object is destroyed.
a. True.
b. False.

10. In C++, only operations that are declared to be virtual can be overridden in derived classes.
    a. True.
    b. False.

11. All member functions in a C++ class have a `this` pointer.
    a. True.
    b. False.

12. The story points of a user story:
    a. Have a work-hours equivalent defined specifically in the user story specifications.
    b. Cannot change once set.
    c. Are used to indicate the approximate size and complexity of a story.
    d. Are estimated by the customer.
    e. None of the above.

13. In a UML class diagram, there can never be more than one association or relationship between any two classes in the diagram.
    a. True.
    b. False.

(a - d)

14. Which of the following statements is false?
    a. Design principles define guidelines for "good" code.
    b. Extensibility is a trait of well-designed software.
    c. In specific coding situations, design patterns should be applied as they follow good design and make code more readable to other developers.
    d. Low coupling and high cohesion can go together.
    e. All of the above statements are true, not false.

15. Why does the Interface Segregation Principle aim to avoid fat interfaces?
    a. High level modules should not depend on lower level modules, because higher level interfaces are less likely to change.
    b. Because a fat interface prevents derived classes from being used as if they were the interface class.
    c. While fat interfaces reduce coupling between the subclasses and the interface, the large interface becomes difficult to use.
    d. Any class inheriting the interface must provide an implementation for all interface functions, which may be empty or meaningless.
    e. None of the above.

16. Which of the following are true about a successful implementation of the Singleton design pattern?
    a. It relies on public constructors.
    b. It must use the keyword `static`.
    c. It ensures that exactly one copy of the object always exists during execution.
    d. It prohibits refinement through subclassing.
    e. None of the above.

17. Which of the following statements about the Decorator pattern are true?
    a. It can be used to add and remove behaviour dynamically at run time.
    b. It is still viewed logically as a single unit, regardless of the number of decorator layers applied.
    c. It works by wrapping one object inside another with the same interface.
    d. All of the above.
    e. None of the above.

18. Creational design patterns:
    a. Are all based on some sort of factory pattern.
    b. Are used to encapsulate the creation of objects.
    c. Are only useful when constructors are statically dispatched.
    d. All of the above.
    e. None of the above.

19. The Adapter pattern:
    a. Can be implemented in C++ using multiple inheritance.
    b. Can be implemented in C++ without using multiple inheritance.
    c. Uses an adapter object to translate requests to an existing interfaces.
    d. Both a) and c) are true.
    e. All of the above are true.

20. The Builder pattern is used to:
    a. Create objects by merging classes with each other.
    b. Control the structure of complex, hierarchical class relationships, by building up from the bottom.
    c. Control the creation of objects where there are many constructors with long and complex parameter lists.
    d. Create objects through a similar process of steps, but allow the actual representation to vary.
    e. None of the above.

Part II -- Fill In The Blank -- Fill in the blanks as instructed. [34 marks total]

21. Consider the following class declarations [8 marks total, 1 mark per line]:

```
1  class Person
2  {
3    void sleep();         // 1
4    virtual void walk(); // 2
5    virtual void eat();  // 3
6  };
7
8  class Student : public Person
9  {
10   void sleep();          // 4
11   void walk();           // 5
12   virtual void study(); // 6
13 };
14
15 int main()
16 {
17   Person* p1 = new Student();
18   Student s1;
19   Person p2 = s1;
20
21   // insert line here
22 }
```

For each of the following, indicate the method called (by its number in the comments above) if we were to replace line 21 with the line given. If the line would not compile, indicate WNC.

`p1->sleep();` WNC

`p1->walk();` WNC

`p1->eat();` WNC

`p1->study();` WNC

`s1.sleep();` WNC

`s1.eat();` WNC

`p2.sleep();` WNC

`p2.walk();` WNC

« Mike, you psycho  »

22. Given the following class definitions [26 marks total, 1 mark per line]:

```
#include<iostream>
using namespace std;

class X {
 public:
  virtual void a() { cout << "X::a"; }
  void b() { cout << "X::b"; }
  virtual void c() { cout << "X::c"; }
  virtual void c(int r) { cout<<"X::c(r)"; }
  void d() { cout << "X::d"; }
};
```

```
class Y : public X {
 public:
  void a() { cout << "Y::a"; }
  void b() { cout << "Y::b"; }
  virtual void d() {cout<<"Y::d";}
  void e() { cout << "Y::e"; }
};

class Z : public Y {
 public:
  void c() { cout << "Z::c"; }
};
```

Comment on any errors in these statements (and show output where requested).

```
X * xx = new X;     no error
X * xy = new Y;     no error
X * xz = new Z;     no error
Y * yx = new X;     compilation error
Y * yy = new Y;     no error
Y * yz = new Z;     no error
Z * zx = new X;     compilation error
Z * zz = new Z;     no error
```

```
xx->a();      Output: X::a
xx->b();      Output: X::b
xx->c();      Output: X::c

xy->a();      Output: Y::a
xy->b();      Output: X::b
xy->c();      Output: X::c
xy->c(10);    Output: X::c(10)
xy->d();      Output: X::d

xz->a();      Output: Y::a
xz->b();      Output: X::b
xz->c();      Output: Z::c

yy->a();      Output: Y::a
yy->c();      Output: X::c

yz->a();      Output: Y::a
yz->c();      Output: Z::c
yz->c(20);    Output: X::c(20)

zz->a();      Output: Y::a
zz->c();      Output: Z::c
```

Part III -- Short Answer -- For the following questions, write your answer in the space provided, using diagrams as appropriate. You do not need to write full sentences, and can use point form if that is your preference. [48 marks total]

23. There is a considerable memory leak in your software system. You suspect after hours of debugging that something is wrong with the following object usage. Here, `TimedSession` inherits from `Session` (i.e. your code contains the declaration: `class TimedSession: public Session`).

```
1 Session * currentSession = new TimedSession(2000);
2 ...
3 delete currentSession;
```

Is it possible, when using the object in this way, for it to cause a memory leak? If so, what is the likely cause and why would that cause the leak? If not, why is it not possible? [6 marks]

Yes. If TimedSession opens a resource file (or a socket, etc), closes this resource in the destructor, but Session destructor is not declared virtual, then line 3 causes the Session destructor only to be called, which means the destructor of TimedSession is not called due to static dispatch. As a result, the resource is never closed properly.

24. Default parameters in C++ allow functions to be called without providing one or more trailing parameters, which can be convenient to software developers. Default parameters, however, are not without issues. Provide a scenario in which the use of default parameters leads to ambiguity when overloading a function in C++. [6 marks]

Consider the following class:

```
class A {
 public:
    int doX(int x = 0){ return x;}   // method 1
    int doX() {return -1;}           // method 2
};
```

notice when calling doX() on an object of type A, the compiler cannot resolve which method the user wants since not providing an argument to method 1 implies x=0, but the user could have also simply wanted to use method 2.

25. A regular polygon is a shape with any number of sides, where all sides are the same length and all angles are equal in degrees. Consider this class declaration:

```
1  class RegularPolygon {
2      public:
3          RegularPolygon();
4          RegularPolygon(int numSides, int sideLength);
5          virtual ~RegularPolygon();
6          virtual int numberOfSides();
7          virtual int area();
8      private:
9          double interiorAngle;
10         int sideLength;
11 }
```

Suppose you created a subclass of RegularPolygon called Square, overriding the above methods as necessary and providing a new constructor that takes a single parameter, the side length for the square in question. Would this scenario violate the Liskov Substitution Principle as we did in class when we derived a Square class from a Rectangle class? Give a basic description of the Liskov Substitution Principle and explain why there is or is not a violation in this case. [6 marks]

No, it does not violate LSP. LSP states that if X extends Y, then we can always use X whenever Y is expected without affecting the correctness of the program. The reason it does not violate the LSP is because a Regular Polygon with 4 sides behaves exactly as a Square. They have the same area and perimeter as well as interior angle.

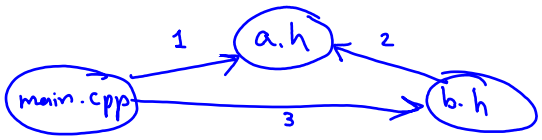
26. For memory management, does C++ have built-in garbage collection? Provide one benefit and one drawback of C++'s approach to memory management. [6 marks]

C++ does not have a built-in garbage collector as opposed to Java. One of the greatest benefits of this is performance. Garbage collection has a run time overhead. In C++, this overhead is not incurred. Another benefit is flexibility. In Java, the garbage collector works automatically when it wants, so there are times in which the stack and heap pointers become too close (i.e. we are running out of memory) but Java does not let you reclaim memory from the heap, so a JVM Out of Memory exception could be thrown or the program might hang until memory is reclaimed. In C++, the programmer manually reclaims heap memory as soon as a resource/object becomes unneeded so memory is immediately available for use. A drawback is that if a programmer is not careful, allocates memory and forgets to deallocate, then a memory leak occurs, which can affect the run time of a program and other processes on the same OS.

27. What type of error might you get if you omitted the include guard (example below) in your C++ header files? Why might such an error occur? [4 marks]

```
1 #ifndef SOMESOURCEFILE_H
2 #define SOMESOURCEFILE_H
3 ...//header content
4 #endif
```

We get a compilation error such that a class/function/variable is already declared (multiple declaration). This occurs because the preprocessor simply copies the contents of the header file when it is included in another file. So, assume we have the following dependency graph where each node represents a file and each edge represents an "includes" relationship:



notice that the contents of a.h are copied into b.h, which are then copied into main.cpp by edge 3. Moreover, by edge 1, the contents of a.h are copied again into main.cpp.

28. Why should you never place a `using namespace` directive inside of a header file? Why do some C++ programmers argue that you should avoid making use of `using namespace` directives altogether, even in code files? [4 marks]

This pollutes the code by causing unintended imports of functions, classes, types, and variables into all files that include that header file and transitively all files that include files that include this header. This also causes naming collisions as a local variable whose name is the same as one used in the namespace will cause compilation error. Purists argue it is bad practice because when multiple namespaces are used, it is not possible to determine where some of the variables come from, which makes the code unreadable. Moreover, the same variable can be defined in multiple namespaces, which can cause an ambiguous reference compilation error when referred to without qualifying with the namespace

29. For each of the following user stories, indicate whether or not the story is acceptable according to the INVEST criteria we studied in class. If not, indicate which criteria the story violates. Briefly justify your answer. [6 marks total]
    a. *An agent should be able to manage tickets.* [3 marks]

Not acceptable, violates the "Small" criterion. Managing is a macro task that can be decomposed into smaller tasks, each of which is a story of its own. This can be used as an epic.

    b. *The GUI must be aesthetically pleasing to the user.* [3 marks]

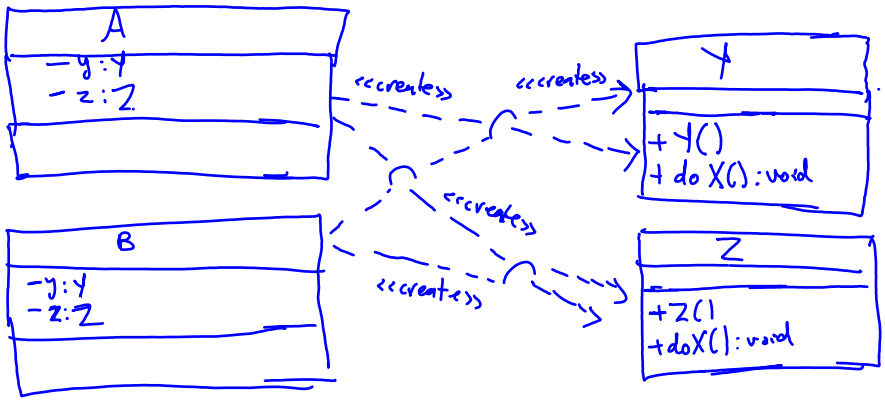
Not acceptable, violates the "Testable" criterion. The specification is non-functional and uses subjective language. Ideally, tests should be automated, but it is not possible to automate a test for "aesthatically pleasing". Different people have different notions of what passes as "aesthetically pleasing".

30. Why might we want to pass a parameter as a constant reference? For example: `void myfunct(const MyClass &c)` [4 marks]

The default is passing by value in C++, which implies the copy constructor is invoked. However, if an object has lots of data, copying it entirely to the stack is computationally expensive. Passing by reference uses the same object and gives it the alias c, so the cost is negligable. Moreover, if we do not intend to let myfunc change the passed object, we can get the compiler to enforce this intention by adding "const". In this case, the function can only use accessors and methods labelled const.

31. Describe high coupling, using at least one example or symptom of high coupling. [3 marks]

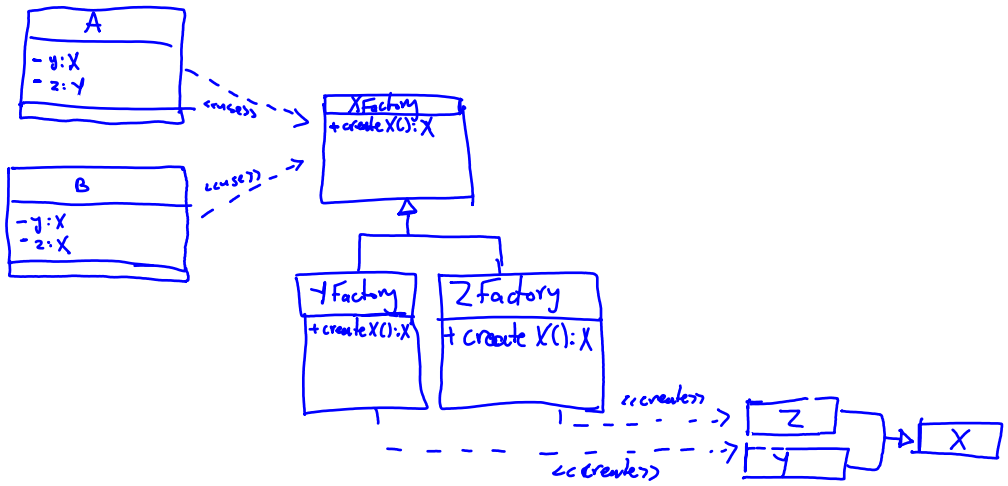
High coupling is when different classes/components are heavily intertwined and depend on each other such that changing one class/component breaks the code. An example is:



If we decide to add a parameter to the constructor of Y or Z, then both A and B will break.

32. Describe low coupling, using at least one example or symptom of low coupling. [3 marks]

Low coupling is when different components/classes do not heavily depend on each other. Changing the implementation in some classes does not break the code. To lower the coupling in the above example, we can use a factory:



changing the constructor of Y for example now means we only change YFactory rather than changing A, B, and any other class that instantiates Y.

Part IV -- Long Answer -- For the following questions, write your answer in the space provided, using diagrams as appropriate. Again, you do not need to write full sentences, and can use point form if that is your preference. [58 marks total]

33. Recall the file utilities created in the individual assignment earlier in this course. This assignment entailed the creation of a class to manipulate files and directories in the file system, as well as a set of utilities using this class that could replace a number of standard command line file utilities. [20 marks total]

a. Craft a user story capturing one of the requirements for this assignment. Use the front-of-card and back-of-card areas below to capture and record all of the requisite information for this user story. [8 marks]

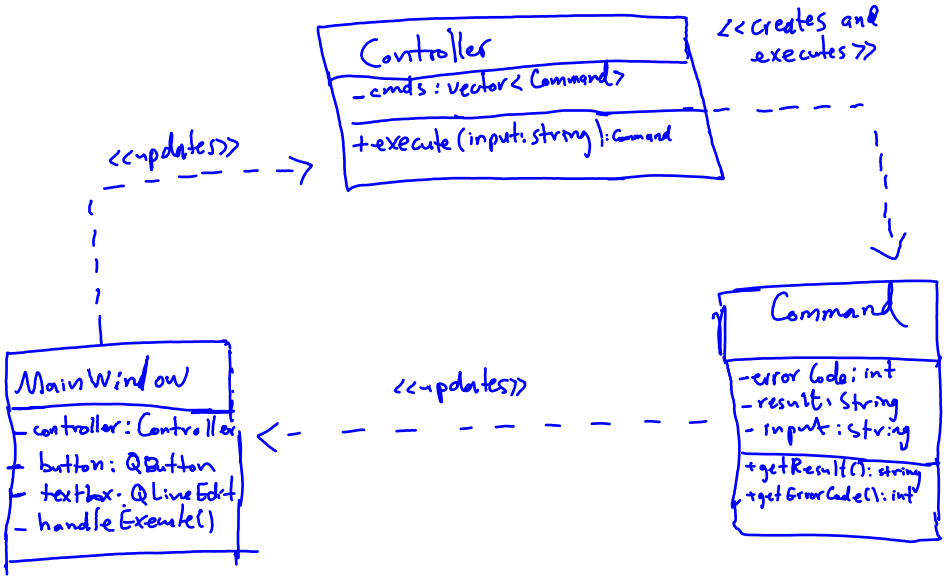
Front of Card
• id : 0 • points: 5 • priority: high • description: as a user, I would like to input a command, click a button, and it gets executed as though I entered it into the command line.

Back of Card
• Acceptance Tests: 0. User can write any number of characters in a textbox (pass). 1. The user enters an invalid command, gets an error code with an error message as output (pass). 2. The user enters a valid command, gets the output of the command (pass).

b. In class, we identified one software pattern that would have been of particular use to this assignment. Which pattern did we identify? How would the use of this pattern improve the design and quality of this software? [8 marks]

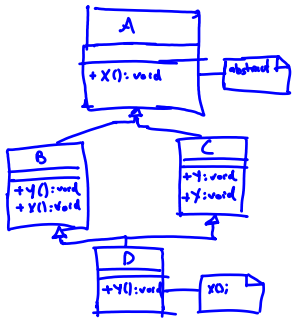
Model-View-Controller. This pattern improves the design and quality of the software by ensuring the GUI is decoupled from changes to the implementation and methods of the Command class, and therefore only the Controller has to be changed. Moreover, it encourages separation of concerns, so the GUI is only responsible for performing changes to the GUI, the controller is responsible for handling user commands by updating the state of the Command class, and the Command class is responsible for working without any knowledge of the Controller or the GUI.

c. Provide a UML class diagram, like those given in class accompanying our design pattern examples, to illustrate the use of and integration of the design pattern from part b) above into the assignment. [4 marks]



34. Consider multiple inheritance in C++. [12 marks total]

a. Not every object-oriented language supports multiple inheritance, but C++ does. What are the advantages of supporting multiple inheritance? What are the drawbacks or potential problems with multiple inheritance? [6 marks]



Multiple inheritance allows for flexibility and code reuse in that a class can extend two or more unrelated classes and inheriting all of their functionalities. This is particularly useful when the base classes deliver atomic functionality and the derived class groups the desired functionalities into a single class. This has its drawbacks. For instance, a class can inherit from 2 classes whose behavior is complementary (for example, shape that inherits from both Circle and Square). In addition one can run into the diamond problem: which implementation of X does Y call in D?

b. In class, we used multiple inheritance in a number of our Structural patterns, with a mixture of public and private inheritance. For example, in our discussion of the Bridge pattern, we created a set list implementation that was declared as follows:

```
template <class T>
class SetListImpl : public ListImpl<T>, private std::set<T> {
   …
};
```

In this case, `ListImpl` was an implementor in the pattern, `SetListImpl` was a concrete implementor, and `set` was a standard container class used internally within `SetListImpl` as its underlying data structure. Why did we use a mixture of public and private inheritance? What purpose can such a mixture serve in general in the creation of C++ classes with multiple inheritance? [6 marks]

This is important so that the derived class SetListImpl does not expose the implementation using std::set, i.e. all methods inherited from std::set are inaccessible to clients using SetListImpl. In general, this helps us re-use code implemented in some classes and use it to implement methods in other classes. We use public inheritance to implement or extend classes, and private inheritance to reuse code and not duplicate without exposing this detail.

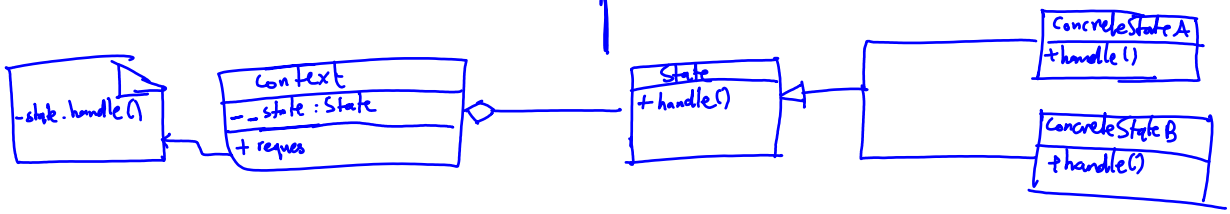
35. Compare and contrast the State and Strategy design patterns. Draw a UML diagram outlining each pattern, and briefly describe the context in which each are used. Include some additional points comparing the two patterns, and describe in what situation you would use each over the other. [14 marks]

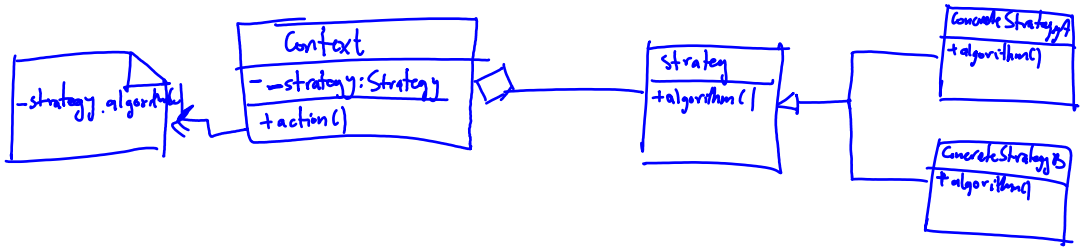
State

- Behavioural DP.
- Behavior of the class owning a state changes dynamically depending on the state.
- State changes occur according to a finite state machine (i.e. not explicitly changed by the client).
- Used when the behaviour of the context class (owning the state) changes depending on its state even if the same method is called.
- Encapsulates a state.

Strategy

- Behavioral DP.
- Behavior of the class owning a strategy changes dynamically depending on the strategy/algorithm.
- The client selects the strategy explicitly.
- Used when there are different ways to achieve the same objective. The client can decide which algorithm to use depending on needs.
- Encapsulates an algorithm.





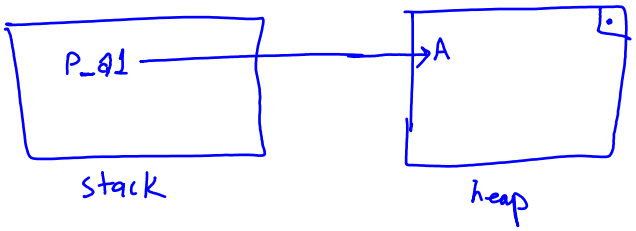
36. Assume that you are using some class A, declared in a file called A.h. Consider the following code that makes use of this class.

```
#include "A.h"

int main (int argc, char*argv[]){
     A * p_a1 = new A();
     A * p_a2 = p_a1;
     delete p_a1;
     p_a1 = 0;
     delete p_a2;
}
```

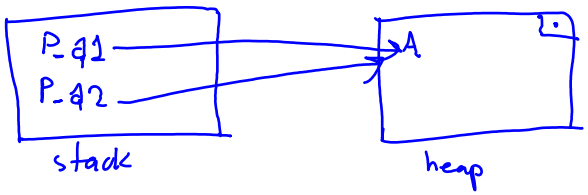
Create a memory diagram depicting the execution of the above code step-by-step. Using your diagram, identify and explain in detail the memory management issue(s) that can be found in this code, if any. If there are no such issues present, explain why this is the case in detail. [12 marks]

1.



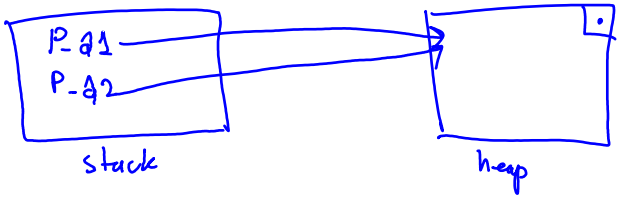
no issues, memory is allocated on the heap for a new object of type A.

2.



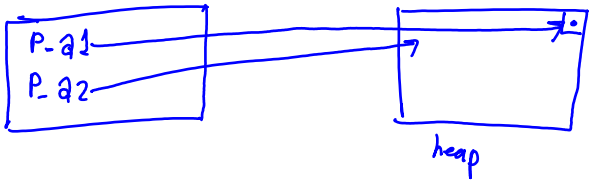
no issues, a pointer is pointing to an existing object on the heap.

3.



no issues, an object on the heap got destroyed.

4.



no issues, a pointer got changed so that it points to 0. (Nullptr).

5. This causes a problem (segmentation fault). P_a2 does not point to an object of type A, but it is attempting to destroy such an object at the memory location it is pointing to.

This page has been left intentionally blank. Use it as additional workspace or extra space for answers if necessary.