Francois Smith

u21649988

Task 1:

1.1 a) Stack – Because it an intrinsic type it will be created on the stack.
b) Heap – Because the word "new" is used this creates an object on the heap.
c) Stack – Because no references or new objects are created it is on the stack.
d) Stack – Because no references or new objects are created it is on the stack.
e) Stack - Because no new memory is allocated by creating an int.
f) Stack – No new memory is allocated, it merely a pointer to a memory location(anything can be stored in this location)
g) Stack - It is an intrinsic type, thus it's on the stack.
h) Stack – It is an intrinsic type, thus it's on the stack.
n) Stack – It is an intrinsic type with a constant value, thus it's on the stack.

1.2 Because h is not a pointer so it cannot be assigned a null pointer value, it needs to be assigned an integer value.

1.3 - **"const int h = NULL;"**: NULL cannot be assigned to the type int, it must be a pointer to be able to assign a value of NULL.
- **"c [10] = *&*e":** Will cause an out of bounds exception because the array created was a size of 10 but the indexes go up to 9, so accessing 10 will break the program.
- **"c [10] = *&*e":** *&*e is a pointer and c[10](if it existed) is not a pointer variable stored in the array it is a normal integer, can't assign a memory address to an integer.

Task 2:

2.1 Whenever the constructor of the derived class is called. Thus ClassC, ClassD and ClassD will call ClassA() because they inherit.

2.2 Whenever the destructor of the derived class is called. Thus ClassC, ClassD and ClassD will call it.

2.3 It is called after, because the class it inherits from is constructed first to ensure all base variable and functions are present, before the expanded items are added by the derived class's constructor.

2.4 Depends on what class it inherits first, if it is a "public ClassA, public ClassB", then it will call A first and vice versa.

2.5 The opposite from the constructors, so ClassD will be called then after the second parent class and finally the first. To go back to 2.4's example ClassB's destructor will be called first then ClassA's.

Task 3:

3.1 Yes it would be possible as integers support division. Because the class is generic, instantiating the variable to use int, all methods will work as intended.

3.2 Yes it would work because floats/doubles can be added together. Because the class is generic, instantiating the variable to use floats, all methods will work as intended.

3.3 Yes because strings support addition. It will result in a concatenation of the values.

3.4 No because strings do not support multiplication, thus it would result in an error.

Task 4:

4.1 1) Because both ptr_b and ptr_a are pointers, when setting the value b will point to the same memory allocated by a, by dereferencing the pointers "*" it allows us to output the 15 that was set earlier.

2) ptr_b geta appointed a new location in memory and it is filled with the value 4, thus when outputting ptr_b will output the value stored in its new memory location which is 4.

3) By altering the value stored in ptr_b's memory location to that of ptr_a they both have the same value but different locations in memory.

4) By deleting ptr_a's value we freed the memory it was pointing to, now ptr_a is pointing to the same value as ptr_b. Within the cout, ptr_b has a lot of trailing 'indirection' and 'address of ' symbols, this will output the value that ptr_b is pointing to.

5) ptr_c will output the memory address where ptr_a is pointing to whereas ptr_a will output the value saved in this address.

Task 6:

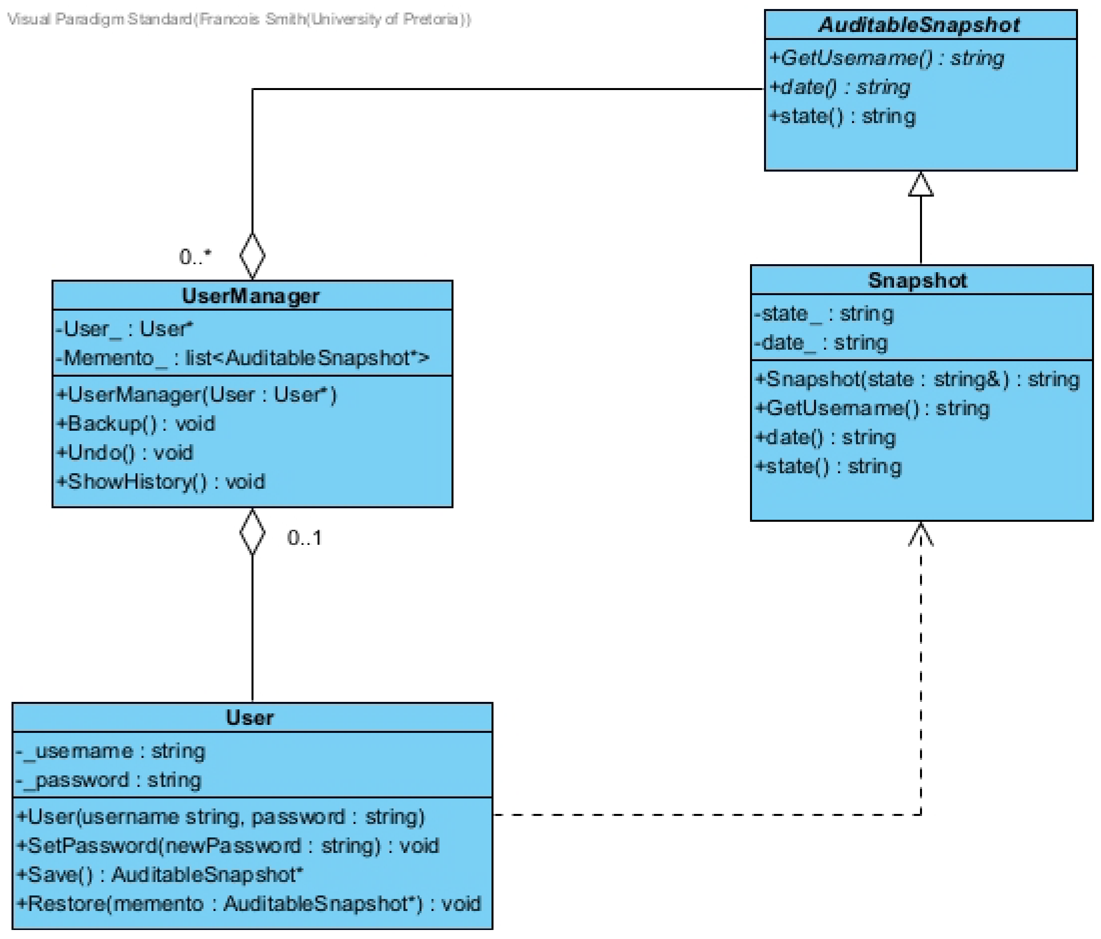
6.1 AudibleSnapshot abstract class

6.2 Snapshot class

6.3 User class

6.4 UserManager Class

6.7


Visual Paradigm Standard(Francois Smith(University of Pretoria))
AuditableSnapshot
+GetUsername() : string
+date() : string
+state() : string
0..*
UserManager
-User_ : User*
-Memento_ : list<AuditableSnapshot*>
+UserManager(User : User*)
+Backup() : void
+Undo() : void
+ShowHistory() : void
0..1
Snapshot
-state_ : string
-date_ : string
+Snapshot(state : string&) : string
+GetUsername() : string
+date() : string
+state() : string
User
-_username : string
-_password : string
+User(username string, password : string)
+SetPassword(newPassword : string) : void
+Save() : AuditableSnapshot*
+Restore(memento : AuditableSnapshot*) : void