Question 1

1a) i. No, it cannot.

1a) ii. Due to BodyGaurd requiring President and President requiring BodyGaurd, there is circular referencing. As there is no forward declaration it will not work.

1a) iii. Due to BodyGaurd requiring President, President requiring BodyGaurd, and Sniper requiring President there is circular referencing. As there is no forward declaration it will not work.

1b) i. The output will be: -
Employee salary :
Employee salary :

1b) ii. Due to the parent class (Employee) salary function not being declared as a virtual function, it is therefore not overridden in the children classes (Permanent and Temporary), and therefore the salary function call will use the Employee classes version.

1b) iii. I would declare the salary function as pure virtual as there is no need to have a salary of 0 in the employee class, as there will be no base employees, then override it in the children classes with the correct salary amount.

1b) iv.

Question 2

2a) It is not desirable since we try to prioritise encapsulation when dealing with OOP. A direct interface would violate encapsulation thus not being desirable.

2b) From the code snippet we can determine that Class A is the Memento and Class B is the originator.

Question 3

This is a close attempt at a Template Method. The reason I think this is since the Call class does not actually define any abstract operations and the VoiceCall and DataCall functions also don't seem setup to implement any of the non-existent primitive operations that Call would have defined. Instead they just redefine the would be primitive operation in each separate class.

Question 4

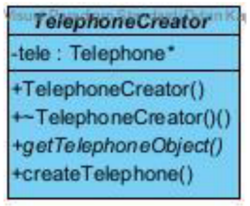
4a) i. Template Method function.

4a) ii. They will be implemented in the ConcreteClasses (the child classes), and they are private such that they can not be called from outside of their defined use cases.

4a) iii. This is a Template Method

4b) i. *-getTelephoneObject()* is the factory method and +createTelephone is the operation.

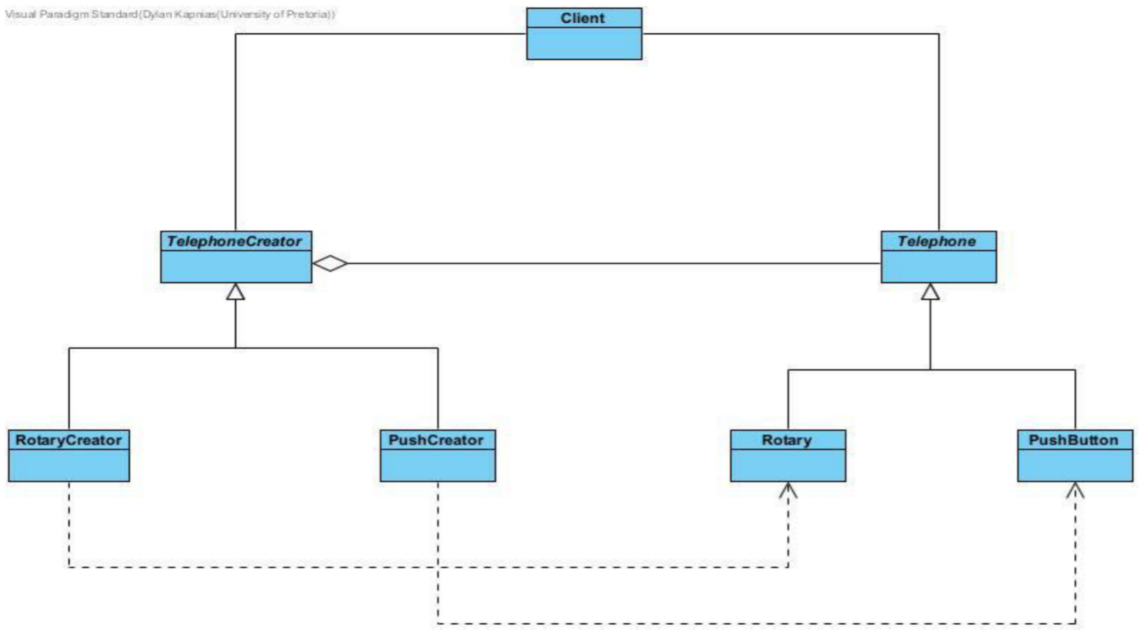
4b) ii.



4b) iii. Adding an input parameter into the createTelephone() function to determine the product.

4b) iv.
```
virtual Telephone* createTelephone(int input) {
        if (input == 0) {
                return new Rotary();
        } else {
                return new PushButton();
        }
}
```

4b) v.



4c) You would implement an Abstract Factory Method. Both the Creator and the Product hierarchies will need to be extended. The Creator will need extra methods added and they will be a factory method for each Concrete product type i.e. pushCreator::createOriginal(), rotaryCreator::createOriginal(), pushCreator::createNineteenTens(), rotaryCreator::createNineteenTens()

<u>Question 5</u>

5a) i. It would first need to be defined in the Telephone class.

5a) ii. public virtual Telephone* clone() = 0;

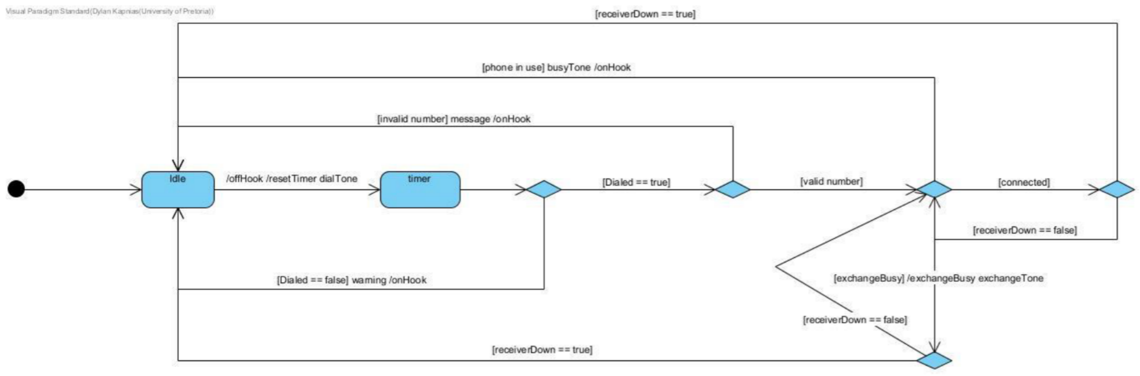
5a) iii.
```
Telephone* Rotary::clone(){
        return new Telephone(*this);
}
```
Assuming that the Telephone class has a copy constructor.

5b) i. Memento

5b) ii. Originator : Telephone

<u>Question 6</u>

6a)


Visual Paradigm Standard(Dylan Kapnias(University of Pretoria))
[receiverDown == true]
[phone in use] busyTone /onHook
[invalid number] message /onHook
Idle
/offHook /resetTimer dialTone
timer
[Dialed == true]
[valid number]
[connected]
[receiverDown == false]
[Dialed == false] warning /onHook
[exchangeBusy] /exchangeBusy exchangeTone
[receiverDown == false]
[receiverDown == true]

6b) i. A State Method to handle the changing of the tones as we have a class that has different behaviours that change based on a set of conditions. A Strategy to display the different tones as they are all tones and will just sound different based on the state.

6b) ii.