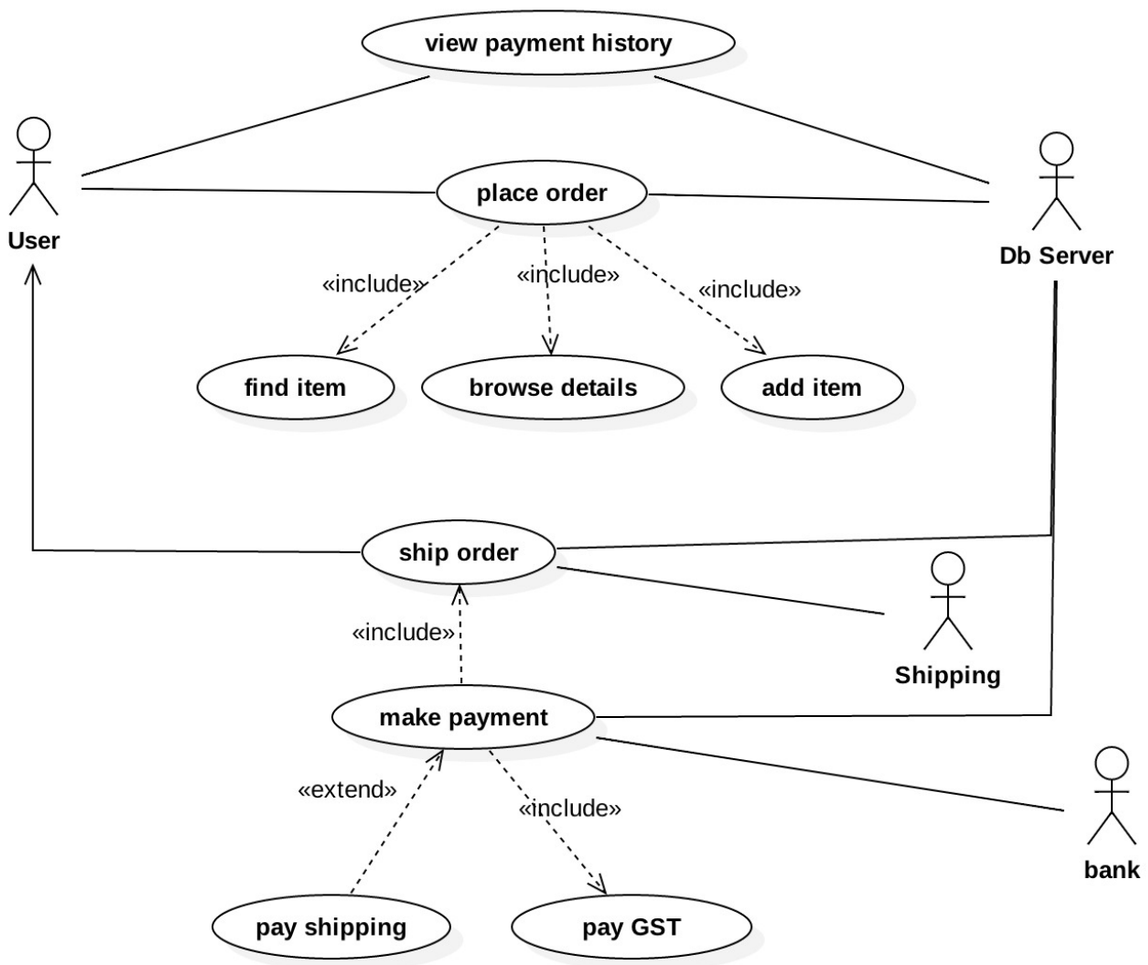
ENSF 480 Quiz II Solutions
Fall 2019

Question I (8 marks)

In the following space draw a very simple use case diagram that ONLY illustrates the following requirements for a Online Shopping System:

- Users should be able to place orders. Orders will be shipped to the user, once relevant payments are made by the user and confirmed by the bank or financial institute.
- Users should be able to make a payment, where payment always include GST, but payment for shipping is optional depending on factors such as distance form point of delivery (in other words means not every purchase may have a shipping cost).
- User should be able a view his payments history



Question II (6 marks):

We would like to design a class called Vector with the following memory management policy:

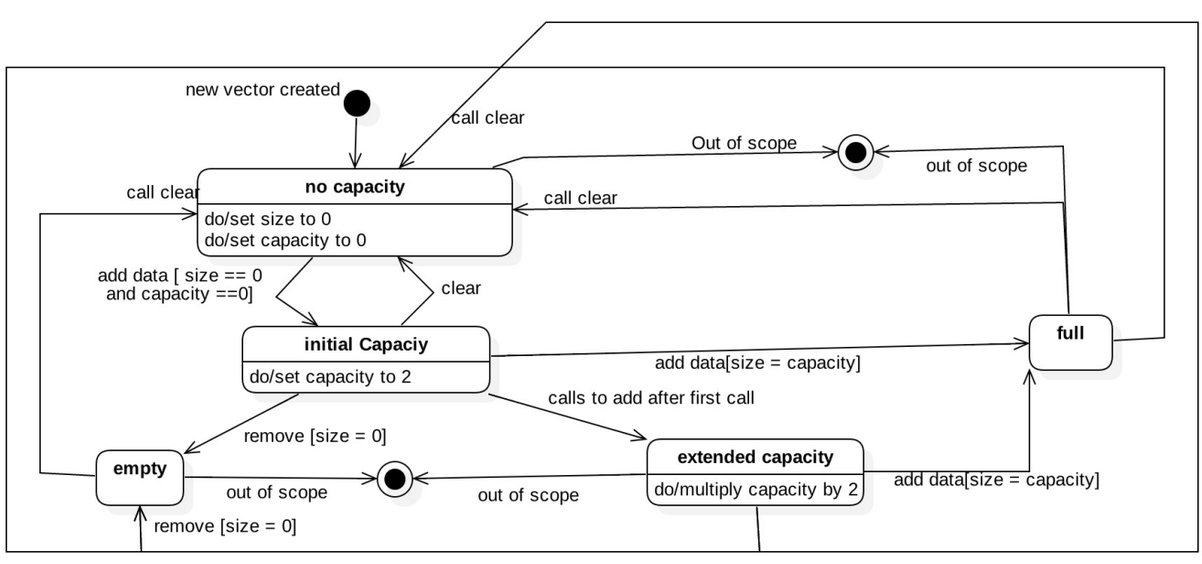
- When constructor is called, it will create the object with no data and no capacity. Means, its size and capacity are both zero. but it will be available to the user to add data to it.
- When adding new data for the first time, capacity will be set to 2, and then the data will be added.
- After the first data is added, when user adds new data, if list is full its capacity will be doubled (will be multiplied by 2), and then the new data will be added.
- If the data is removed from list, the data in the elements after the removed element will be shifted to the left and size will be reduced by one. However the capacity remains unchanged.
- Users may also use other functions such as clear that means removes all the elements of the list, and reduces the values of size and capacity to zero.
- When object goes out of the scope it will be fully destroyed

To give you a better idea about the lifecycle of a Vector object the following code is given.

```
class Vector {
public:
    // creates an that is ready to be used
    Vector() : storeM(0), sizeM(0), capacityM(0){ }
    int size() const { return sizeM; }
    int capacity() {return capacityM;}
    bool isFull() {if (sizeM == capacity) return true; return false; }
    bool isEmpty() {if (sizeM == 0) return true; return false; }
    bool hasCapacity (){if (sizeM < capacityM) return true; return false; }
    void add(int el_val);
    // PROMISES:
    //              - if sizeM == 0 and capacityM == 0 sets capacity to 2. Then
    //                allocates memory.
    //              - else if sizeM == capacityM multiplies capacity by 2. Then
    //                allocates memory
    //              - else if sizeM < capacityM does notthing
    //              - Then, adds el_val into the allocated space for storeM

    void remove(int val);
    // PROMISES: If one or more elements match val all of those elements are
    // removed, remaining data will be shifted to the left, and the value of
    // sizeM will be reduced. But the value of capacityM remains unchanged.

    void clear();
    // PROMISES: sizeM and capacity will be set to zero, and memory will be
    // deallocated

    ~Vector();
    // PROMISES: to dellocate memory space on the heap that is used by storeM
private:
    int *storeM;
    int sizeM, capacityM;
};
```

Based on above description and particularly the given code, draw a State Transition Diagram that shows the possible **states** of a Vector object. PLEASE do not attempt to draw a flowchart that shows the flow of logic.

Solution:



Question III (6 marks):

The following Java program shows partial implementation of Observer Pattern to notify objects of class **Renter** to receive the list of ALL available properties when a NEW property is added to the list of properties. You may assume that `java.util.ArrayList` is imported and you can use its methods such as: `add`, `remove`, `get`, etc.

```
interface Subject {
  public void register(Observer o);
  public void notifyObservers();
}
interface Observer {
  public void update();
}
```

```
class Property{
  String title;
  public Property(String title)
  {
      his.title = title;
  }
}
```

```
class Renter implements Observer {
  PropertyManager propertyManager;
  String name;
  public Renter(String name, PropertyManager pm) {
      this.name = name;
      propertyManager = pm;
  }
  // displays all available propertys
  public void update(){
      System.out.println("List of Available Properties: ");
      for(int i = 0; i < propertyManager.properties.size(); i++){
            System.out.println(propertyManager.properties.get(i).title);
      }
  }
}
```

In the following space partial implementation of class PropertyManger is given. You should complete the BLANK lines and incomplete definition of member functions of this class :

```
class PropertyManager implements Subject {
      ArrayList<Property> properties;
      ArrayList<Observer> renters;

     public PropertyManager() {
        properties = new ArrayList<Property>();
        renters = new ArrayList<Observer>();

      }


      // STUDENTS MUST COMPLETE THE FOLLOWING BLANK LINES
      public void register(Observer o) {
            renters.add(o);
            o.update();
      }


      public void addProperty(Property p){
            properties.add(p);
            notifyObservers();
      }
     public void removeProperty(Property p){
            for(int i = 0; i < properties.size(); i++)
                  if(properties.get(i).equals(p))
                        properties.remove(i);
            notifyObservers();
      }


      // STUDENTS MUST COMPLETE THE FOLLOWING FUCNTION
      public void notifyObservers() {
            for(int i = 0; i<renters.size(); i++){
                  Observer o = renters.get(i);
                  o.update();
            }
      }


     // STUDENTS MUST COMPLETE THE FOLLOWING BLANK LINE
      public void unregister(Observer o) {
            renters.remove(o);
      }
}
```

Question IV (8 marks):

The following code shows the implementation of a poorly designed class diagram.

```
class Cat{
private:
 int weight;
 // animal's location of birth:
 string country;
 string province;
 string continent;
public:
 Cat(int w, string c, string p, string n);
 void move(){cout << "Cat moves";}
};
```

```
class Dog{
private:
 int weight;
 // animal's location of birth
 string country;
 string province;
 string continent;
public:
 Dog(int w, string c, string p, string n);
 void move(){cout << "Dog moves";}
};
```

```
class Bear{
private:
 int weight;
 // animal's location of birth
 string country;
 string province;
 string continent;
public:
 Bear(int w, string c, string p,
                          string con);
 void move(){cout << "Bear moves";}
};
```

```
void move_animal(Cat* x, Dog* y, Bear* z){
 if(x != nullptr) x -> move();
 else if( y!= nullptr) y -> move();
 else if (z != nullptr) z -> move();
 else cout << "ERROR";
}

int main() {
  Cat* cat = new Cat(...);
  Dog* dog = new Dog(...);
  Bear* bear = new Bear(...);
  Move_animal(cat, dog, bear);
  return 0;
}
```

You should suggest a better design by drawing a **class diagram**.
NOTE: We are not concern about using any design pattern this question. The only concern is the basic elements of good object oriented design (**Abstraction, Encapsulation, Modularity and Hierarchy).** Your suggested design should make the implementation of function **move_animal** much simpler by considering concept of **polymorphism**..
DRAW YOUR CLASS DIAGRAM IN THE FOLLOWING SAPCE