Parts 1 & 2 , Version 1 CST8288 - Quiz 1 W13

1 Assume that the following code snippet exists in a java program that is otherwise valid . (2 points)

```
int thisArray[ ] [ ] = {    {1,  2,  3,  4,  5},
                            {2,  4,  6,  8, 10},
                            {3,  6,  9, 12, 15},
                            {4,  8, 12, 16, 20},
                            {5, 10, 15, 20, 25}};
nextRow:
    for (int row = 1; row <= 5; row ++){
        for ( int column = 1; column <= 5; column++ ) {
            if (column > row)
                continue nextRow ;
            System.out.print(thisArray[row-1][column-1] + "\n" );
        }
        System.out.println( " " );
    }
    System.out.println( "\n*** " );
```

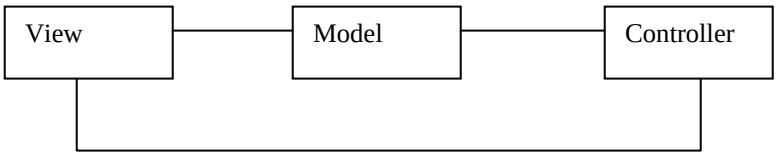
Neatly, Hand print the output from this snippet.

1
2
4
3
6
9
4
8
12
16
5
10
15
20
25

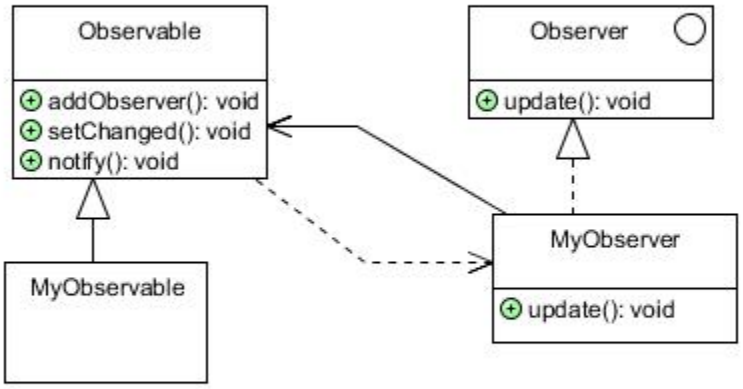
***

2 Provide a full description of the get/set pattern. (4 marks)
The get/set pattern provides the functionality for retrieving and updating attribute values. Get returns an instance of the value object, or a collection of values, set changes the value or a collection of values. No other functionality should be included unless required by a specific design pattern implementation.

3 Draw and label the UML diagram for model-view-controller architecture. (4 marks) (arrows should be included in this diagram)



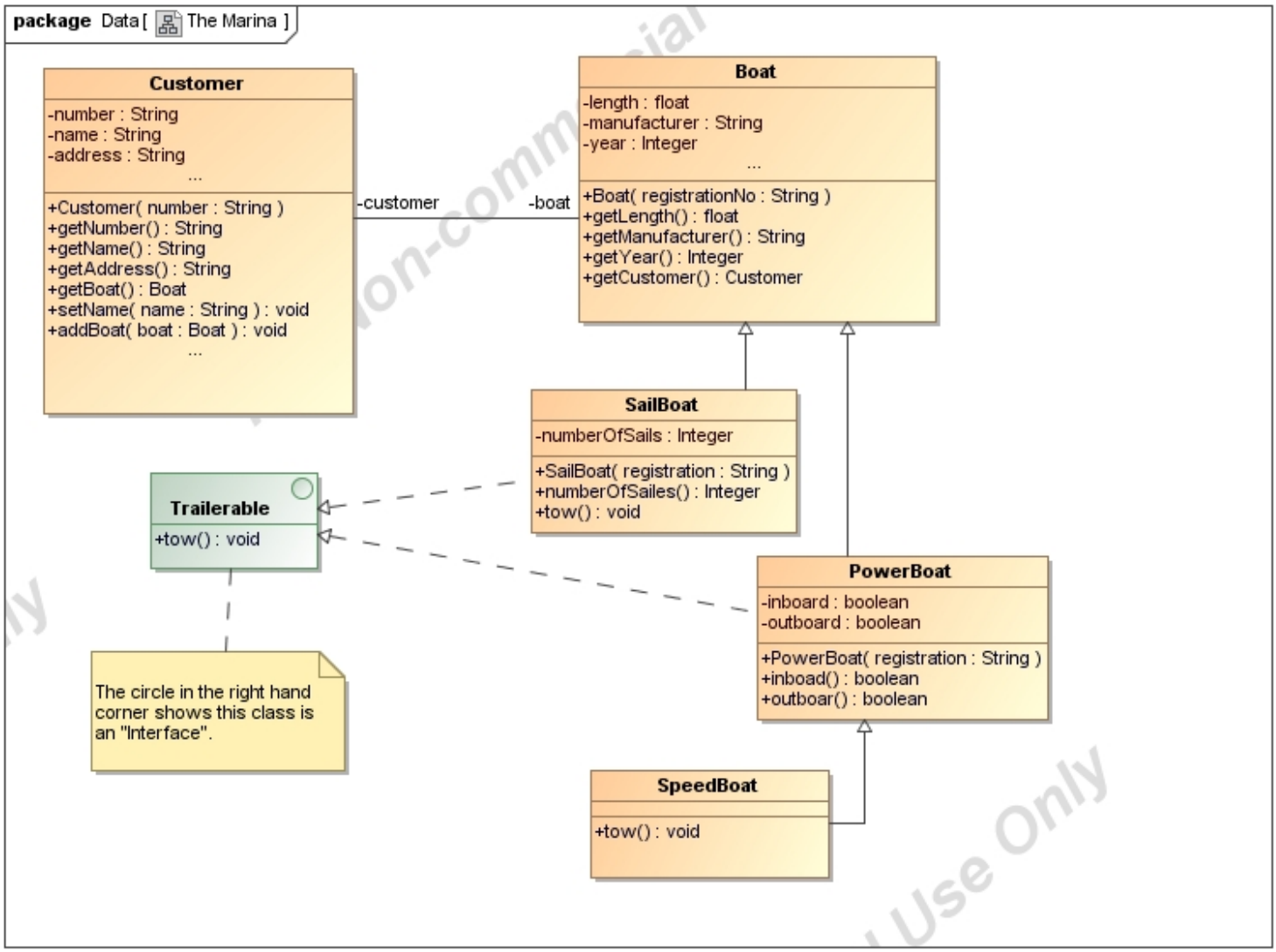
4 Explain the Observer pattern (including it's purpose , include a UML diagram) (4 marks)
The observer pattern provides a way for implementing an interrupt driven architecture between classes, or quite simply it allows one object to "observe" another object. The pattern implements the model-view paradigm. The Observer pattern allows the subject (observable object) to send events to registered observer objects normally indicating that a change has occurred in the subject object. This pattern decouples the observer from the observable object (subject). The observable object doesn't need to know anything about its observers other than their ability to respond to an update() message.



5 The Java API contains a number of implementations of the "Adapter" design pattern. Explain one of the implementations and where/why you might use it in your code. (6 marks)

Any …Adapter class from the API would do. Include not only the adaption aspect but the ability to extend the class and override method implementation (saves writing code)

6 Write the code that implements the UML Class Diagram below (comments may be omitted) (35 marks)




```
public class Customer     {
        public Customer(String number) { this.number = number; }
        public String getNumber()                 { return number;          }
        public String getName()           { return name;    }
        public String getAddress()                { return address;         }
        public Boat getBoat()             { return boat;    }
        public void setName(String name)          { this.name = name;       }
        public void addBoat(Boat boat)    { this.boat = boat;         }
        private String number;
        private String name;
        private String address;
        private Boat boat;
}

public class Boat{
        public Boat()     {}
        public Boat(String registrationNo)        {}
        public float getLength()          { return length;  }
        public String getManufacturer()  { return manufacturer;   }
        public Integer getYear()          { return year;    }
```

```
        public Customer getCustomer()   { return customer;}
        private float length;
        private String manufacturer;
        private Integer year;
        private Customer customer;
}

public interface Trailerable        {
        public void tow();
}

public abstract class PowerBoat extends Boat implements Trailerable {
        public PowerBoat(String registrationNo)  { super(registrationNo);  }
        public boolean inBoard()                 { return inboard;         }
        public boolean outBoar()                 { return outboard;        }
        private boolean inboard;
        private boolean outboard;
}

public class SailBoat extends Boat implements Trailerable {
        public SailBoat(String registrationNo)     { super(registrationNo);  }
        public Integer numberOfSailes()            { return numberOfSails; }
        public void tow()           {}
        private Integer numberOfSails;
}

public class SpeedBoat extends PowerBoat          {
    public void tow()      {}
}
```

BONUS QUESTION: Earn up to 4 bonus marks for a COMPLETE answer (test score cannot exceed 100%).

From the following code, describe where the memory leak(s) occur(s) and explain why.

```
/**
 *    Stack implementation using an array.
 */
public class Stack<ELE> {

   public Stack( ) {     this( 10 );     }

   public Stack(int initialCapacity ) {
      this.elements = new ELE[ initialCapacity ];
            }

            public void push(ELE e) {
               ensureCapacity( );
               elements[ size++ ] = e;
            }

            public ELE pop( ) throws Exception {
               if (size == 0) throw new Exception("Empty queue");
               return elements[--size];       /* THIS IS THE LEAK, OBJECT REFERENCES MUST BE SET
                                                 *    TO A NULL FOR THE GARBAGE COLLECTOR TO RELEASE
                                                 *    THE MEMORY.  Here we decrement the index into the array,
                                                 *    however, we leave the object reference in the array.  The
                                                 *    reference would not be set to null until we quit the program or
                                                 *    over-write the reference at this element position.
                                                */
            }
           }

            public void ensureCapacity( ) {
               if (elements.length == size) {
                  ELE[ ] oldElements = elements;
                  elements = new ELE[2 * elements.length+1];
                  System.arraycopy( oldElements, 0, elements, 0, size );
               }
            }


            private ELE[ ] elements;
            private int size = 0;

         }       /* End of Class: stack.java */
```