Question 1

```
/**
 * Devante Wilson - 100554361
 * Software Design and Architecture - Assignment 2
 * December 4, 2016
 *
 * Interface defines the general functions needed
 * for a tree and its nodes.
 */

interface Component
{
    // add a component
    public abstract void add(Component c);
    // remove a component
    public abstract void remove(Component c);
    // display the depth of a component
    public abstract void stateName(int depth);
}
```

```
/**
 * Devante Wilson - 100554361
 * Software Design and Architecture - Assignment 2
 * December 4, 2016
 *
 * Class describes a leaf node of a tree structure.
 */

class Leaf implements Component
{
    // declare instance variable(s)
    private String name;

    // Constructor
    public Leaf(String name)
    {
        this.name = name;
    }
    public void add(Component c)
    {
        System.out.println("Cannot add to a leaf");
    }

    public void remove(Component c)
    {
         System.out.println("Cannot remove from a leaf");
    }

    public void stateName(int depth)
    {
         System.out.println(depth  + "  " + this.name);
    }
}
```

```java
/**
 * Devante Wilson - 100554361
 * Software Design and Architecture - Assignment 2
 * December 4, 2016
 *
 * Class impersonates a composite component node
 * of a tree structure.
 */

// import libraries
import java.util.ArrayList;
import java.util.List;

class Composite implements Component
{
    // declare instance variable(s)
    private List <Component> children = new ArrayList<Component>();
    private String name;

    // Constructor
    public Composite(String name)
    {
          this.name = name;
    }

    public void add(Component component)
    {
        children.add(component);
    }

    public void remove(Component component)
    {
        children.remove(component);
    }

    public void stateName(int depth)
    {
        System.out.println(depth + "    " + this.name);
        // Recursively display child nodes
        for (int i = 0 ; i < children.size(); i++)
        {
             children.get(i).stateName(depth + 1);
        }
    }

}
```

```
/**
 * Devante Wilson - 100554361
 * Software Design and Architecture - Assignment 2
 * December 4, 2016
 *
 * Class embodies client of a Bank.
 */

class Client
{
    public static void main(String[] args)
    {
        // create a tree structure
        Composite root = new Composite("President");

        Composite comp1 = new Composite("Manager a");
        root.add(comp1);
        comp1.add(new Leaf("Manager d"));
        comp1.add(new Leaf("Clerk c"));

        Composite comp2 = new Composite("Manager b");
        root.add(comp2);
        comp2.add(new Leaf("Clerk a"));
        comp2.add(new Leaf("Clerk b"));

        Composite comp3 = new Composite("Manager c");
        root.add(comp3);
        comp3.add(new Leaf("Teller"));

        root.add(new Leaf("Teller a"));

        // Recursively display tree
        System.out.println("Depth:  Object:");
        root.stateName(1);
    }
}
```

```
Depth:  Object:
1       President
2       Manager a
3       Manager d
3       Clerk c
2       Manager b
3       Clerk a
3       Clerk b
2       Manager c
3       Teller
2       Teller a
```

Devante Wilson
100554361

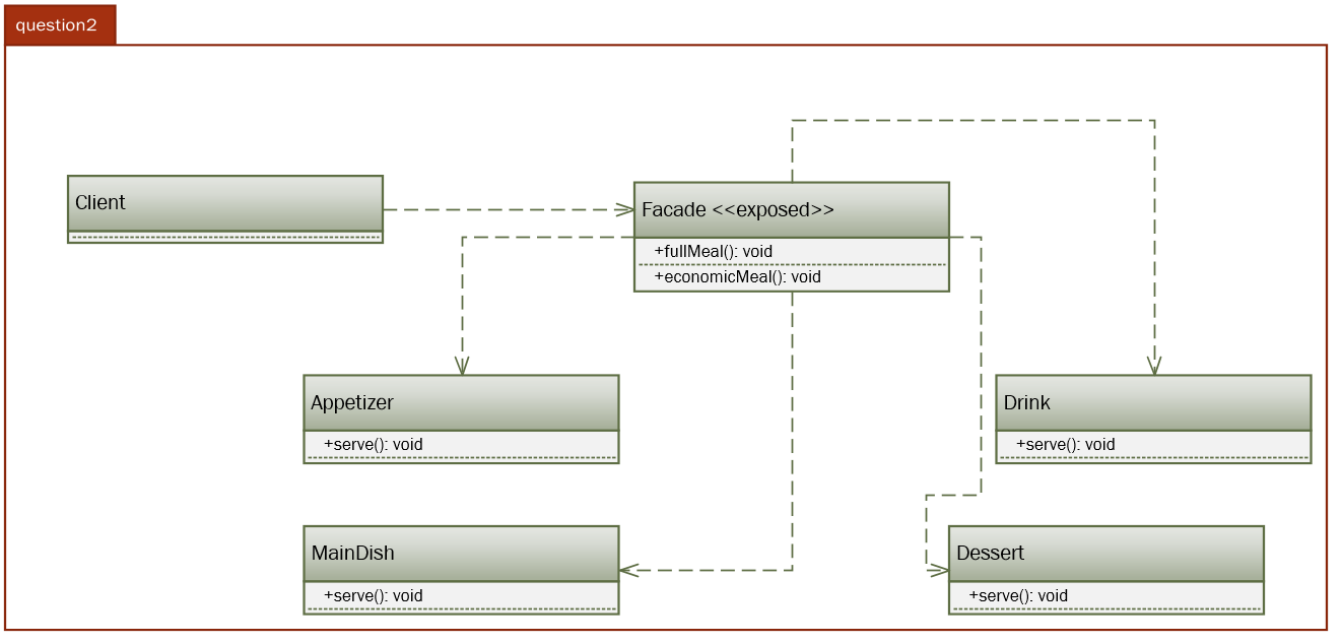
Assignment 2 – 3650U

Question 2

a) Identify the most appropriate design pattern that can be used to allow a customer to only order using one of the two types of meals provided and that the meal components must be served in the given order.

The ***Facade*** design pattern would be the most appropriate as it not only allows the meal components to be served in the given order, but it also allows for encapsulating the food items into the two types of meals (full meal and economic meal).

b) Draw the class diagram using the design pattern you chose.



c) Provide the Java code for the design pattern.

```java
/**
 * Devante Wilson - 100554361
 * Software Design and Architecture - Assignment 2
 * December 4, 2016
 *
 * Class mimics an appetizer to be served
 */

// import package
package question2;

public class Appetizer
{
    public void serve()
    {
        System.out.println("Appetizer");
    }
}
```

```java
/**
 * Devante Wilson - 100554361
 * Software Design and Architecture - Assignment 2
 * December 4, 2016
 *
 * Class mimics a drink to be served
 */

// import package
package question2;

public class Drink
{
    public void serve()
    {
        System.out.println("Drink");
    }
}
```

```java
/**
 * Devante Wilson - 100554361
 * Software Design and Architecture - Assignment 2
 * December 4, 2016
 *
 * Class mimics a main dish to be served
 */

// import package
package question2;

public class MainDish
{
    public void serve()
    {
        System.out.println("Main Dish");
    }
}
```

```java
/**
 * Devante Wilson - 100554361
 * Software Design and Architecture - Assignment 2
 * December 4, 2016
 *
 * Class mimics a dessert to be served
 */

// import package
package question2;

public class Dessert
{
    public void serve()
    {
        System.out.println("Dessert");
    }
}
```

```java
/**
 * Devante Wilson - 100554361
 * Software Design and Architecture - Assignment 2
 * December 4, 2016
 *
 * Class serves as a facade - with what the client interacts
 */

// import package
package question2;

public class Facade
{
    // declare instance variables
    private Appetizer appetizer;
    private Drink drink;
    private MainDish mainDish;
    private Dessert dessert;

    // constructor
    public Facade()
    {
        appetizer = new Appetizer();
        drink = new Drink();
        mainDish = new MainDish();
        dessert = new Dessert();
    }

    // serve a full meal
    public void fullMeal()
    {
        System.out.println("Full Meal:\n");
        appetizer.serve();
        drink.serve();
        mainDish.serve();
        dessert.serve();
    }

    // serve an economic meal
    public void economicMeal()
    {
        System.out.println("\nEconomic Meal:\n");
        drink.serve();
        mainDish.serve();
    }
}
```

```java
/**
 * Devante Wilson - 100554361
 * Software Design and Architecture - Assignment 2
 * December 4, 2016
 *
 * Class embodies a client ordering a type of meal
 */

// import package
package question2;

public class Client
{
    public static void main(String []args)
    {
        // create instance variable(s)
        Facade facade = new Facade();

        facade.fullMeal();
        facade.economicMeal();
    }
}
```

```
Full Meal:

Appetizer
Drink
Main Dish
Dessert

Economic Meal:

Drink
Main Dish
```

Devante Wilson
100554361

Assignment 2 – 3650U

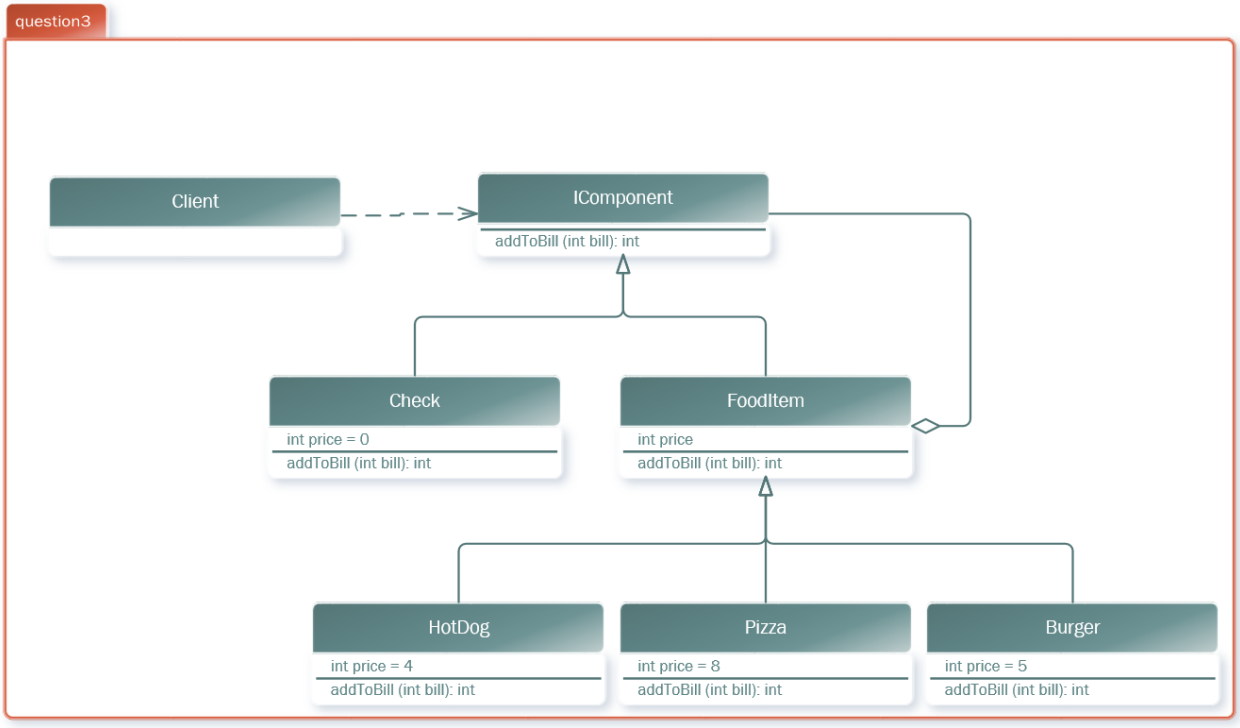
Question 3

a) Identify the most appropriate design pattern that can be used to allow a customer to only order using one of the two types of meals provided and that the meal components must be served in the given order.

In this case, the ***Decorator*** design pattern would seem to be the most appropriate.

b) Draw the class diagram using the design pattern you chose. Also explain in writing how the class structure you have will operate to satisfy the restaurant's conditions.

Since it is a linked list, the meal components can be served in the given order. Additionally, The addToBill() function can be easily added to upon ordering a new item.



c) Assuming the client ordered 2 hot dogs, 1 pizza and 2 burgers. Provide the Java code for the design pattern and that would correctly calculate the total price of the order. (*You can make the client create the list of food items including the tail Check object*).

```java
/**
 * Devante Wilson - 100554361
 * Software Design and Architecture - Assignment 2
 * December 4, 2016
 *
 * Class serves as a component node
 */

// import package
package question3;

interface IComponent
{
    int price = 0;
    // operation to be carried out by the components
    int addToBill(int bill);
}
/**
 * Devante Wilson - 100554361
 * Software Design and Architecture - Assignment 2
 * December 4, 2016
 *
 * Class mimics a hot dog food item
 */

// import package
package question3;

class HotDog extends FoodItem
{
    // declare instance variables
    IComponent component;
    static int price = 4;

    // constructor
    public HotDog(IComponent c)
    {
        super(c);
    }

    public int addToBill(int bill)
    {
        return super.addToBill(bill) + price;
    }
}
```

```java
/**
 * Devante Wilson - 100554361
 * Software Design and Architecture - Assignment 2
 * December 4, 2016
 *
 * Class mimics a pizza food item
 */

// import package
package question3;

public class Pizza extends FoodItem
{
    // declare instance variables
    IComponent component;
    static int price = 8;

    // constructor
    public Pizza(IComponent c)
    {
        super(c);
    }

    public int addToBill(int bill)
    {
        return super.addToBill(bill) + price;
    }
}
```

```java
/**
 * Devante Wilson - 100554361
 * Software Design and Architecture - Assignment 2
 * December 4, 2016
 *
 * Class mimics a burger food item
 */

// import package
package question3;

public class Burger extends FoodItem
{
    // declare instance variables
    IComponent component;
    static int price = 5;

    // constructor
    public Burger(IComponent c)
    {
        super(c);
    }

    public int addToBill(int bill)
    {
        return super.addToBill(bill) + price;
    }
}
```

```java
/**
 * Devante Wilson - 100554361
 * Software Design and Architecture - Assignment 2
 * December 4, 2016
 *
 * Class mimics a total order bill
 * (leaf node)
 */

// import package
package question3;

public class Check implements IComponent
{
    // declare instance variable
    static int price = 0;

    public int addToBill(int bill)
    {
        return price;
    }
}
```

```java
/**
 * Devante Wilson - 100554361
 * Software Design and Architecture - Assignment 2
 * December 4, 2016
 *
 * Class embodies a client ordering food items
 */

// import package
package question3;

public class Client
{
    static void display(IComponent c)
    {
        System.out.println("Price:" + c.price);
    }

    public static void main(String[] args)
    {
        // define new object(s)
        //IComponent component = new Check();
        IComponent component = new HotDog(new HotDog(new Pizza(new Burger(
                new Burger(new Check()))))));
        display(component);
    }
}
```

```java
package question3;

public abstract class FoodItem implements IComponent
{
    protected IComponent food;
    int price;

    public FoodItem(IComponent c)
    {
        this.food = c;
    }

    public int addToBill(int bill)
    {
        return food.addToBill(bill) + price;
    }
}
```