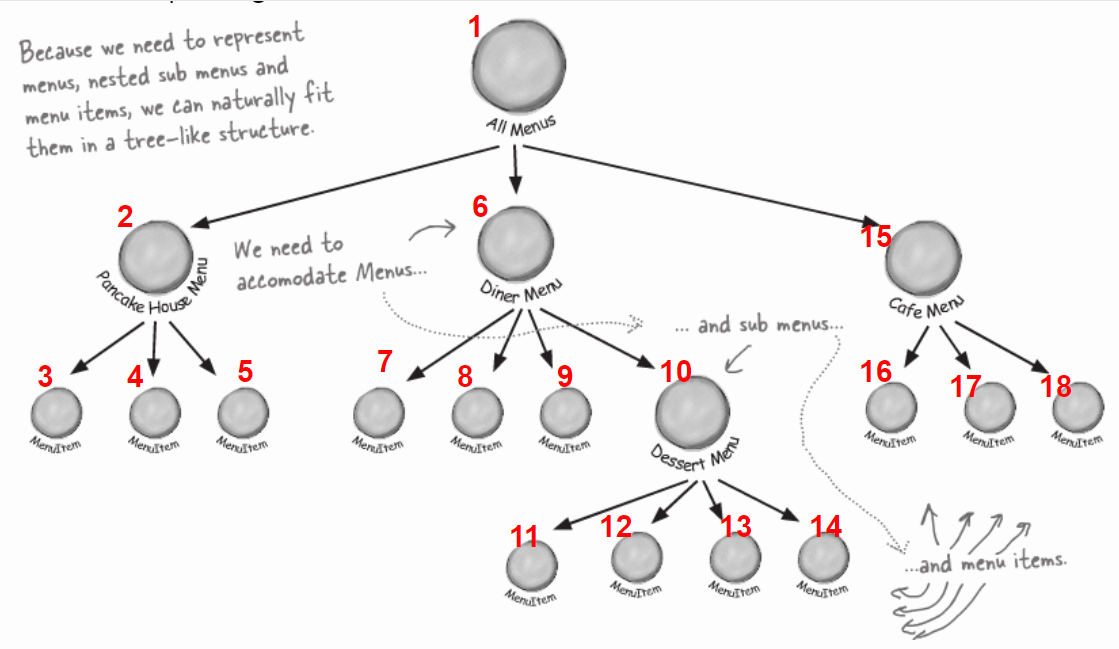
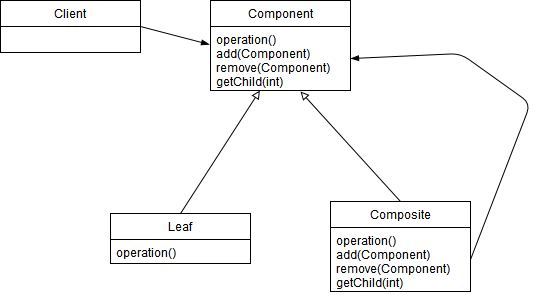
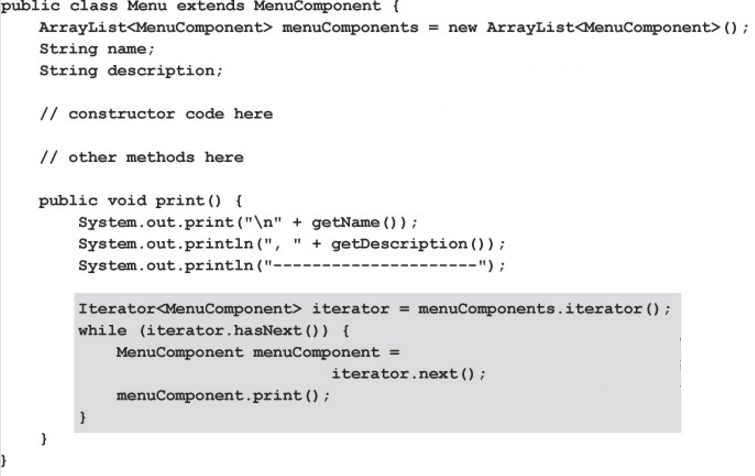
Workout 2018-03-22 – Iterator & Composite Patterns **Name: Hannah Laws**

Instructions: In this workout, create a workout directory. Place your answers for questions 1-17 in this word document. Put all the code for question 18 in the workout directory. Zip up the directory and submit it to ASULearn.

1. Last time, we had plenty of examples of sequential iterators (beginning to end and in reverse) that move from the beginning to the end of a list (one that rendered every other item instead of every item). What other types of Iterators can you imagine? Use your imagination and list three other orderings that might be useful and an example application where it might be used.
   1. iterator for grabbing items based on
   2. iterator for every other item,
   3. iterator for grabbing certain items depending on the day of the week, this could be for cafes that have weekend specials or something of the sort.
2. How does adding Dessert Menu to the Diner Menu in the book’s example fundamentally change our problem?
   1. We have to go through each diner menu item, and then create a separate label to identify the desserts.
3. Number the following tree in a pre-order ordering. (Number the first node visited as 1, the second as 2, etc.) You can create text boxes in Word, place numbers in those text boxes, and place them over the corresponding tree node.



1. State the Composite Pattern.
   1. The Composite Pattern allows you to compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.
2. Does the Composite Pattern also hide the underlying composite representation from the client (like the Iterator Pattern)? Explain.
   1. Yes, the Composite Pattern makes it so that the client can only see the Component class which controls the leaf and composite classes uniformly.
3. Draw the generic UML for the Composite Pattern.
   1. 
4. How is the Iterator Pattern “embedded” in the Composite Pattern?
   1. When going from branch to leaves, the iterator pattern would be used to go through every leaf.
5. State, as a general rule, what methods should go in the Component abstract class that are used in the Composite Pattern.
   1. Any methods in the Component abstract class should be used in the Composite Pattern.
6. What methods from the Component class are meaningless in the Leaf class? What should happen if the client calls one of these methods on a Leaf object?
   1. Methods that involve adding, removing, or retrieving other components.
   2. If the client calls one of these methods on a leaf object, an UnsupportedOperationException should be thrown.
7. Examine the Menu print method to the right. Explain in which statements recursion and “pseudo-recursion” occur.
   1. recursion happens in the while loop after declaring the Iterator Arraylist. Specifically when the while loop is checking to see if there is another component, and then pulls up that component if there is one.
8. Define transparency? How does it violate the Single Responsibility Principle?
   1. Allowing the Component interface to contain child management operations AND leaf operations.
   2. The Components are able to add and remove nodes while also performing operations not related to said nodes.
9. If instead of using a Null Iterator in implementing the Composite Pattern, we had the iterator for a Leaf object return null, what would the client have to do in order to work properly?
   1. The client would need to move onto the next leaf or component, essentially skip over it.
10. Study the textbook's code for the Composite Iterator. State what stack.peek() does in the context of the Composite Pattern. (We know that peek returns the top element on the stack without removing it. The question here are "What will be on the top of the stack in the Composite Pattern?" and "Why must we 'peek' it rather than 'pop' it?".)
    1. The peek method, in this case, is looking to see if there is another “branch” to start iterating through. If there is another branch, the next method is suppose to get the corresponding iterator.
11. What changes are needed in the MenuComponent class to create a low-calorie Menu? (You don't need specific calorie counts; simply mark certain menu items "low-calorie".)
    1. the class would need to add a boolean method called, say, isLowCal().
12. Rewrite the print method in MenuItem (page 369) to add a "lc" designation for low-calorie. If an item is both vegetarian and low-calorie, let the designator be "(v, lc)". If it is just low-calorie, the designator should be "(lc)". All other items will be unchanged from the result in the book's program. (Just copy the code for print here as the answer to this question.)

public void print(){

System.out.print(“ “ + getName());

if(isVegetarian()){

System.out.print(“(v)”);

}

if(isLowCal()){

System.out.print(“(lc)”);

}

System.out.println(“, ” + getPrice());

System.out.println(“ --” + getDescription());

}

1. Write a method for the Waitress class that prints out a low-calorie menu. (Just copy the method here as the answer to this question.)

public void printMenu(){

System.out.println(“\nLow Calorie”);

printMenu(lowCalIterator);

}

1. Does it seem odd using try…catch for essentially "normal operations"? How so? Do you think that's a good programming practice or not? Explain.
   1. using a try-catch is meant for error handling and not program logic. If there’s a possibility that a function would return an error, then we could change the code for said function to return something else instead of throwing an exception.
2. A file on ASULearn contains code that simulates a directory structure in linux along with a recursive "ls" operator. Copy the code into eclipse (or your favorite IDE). In main, write code that creates the following directory structure (directories below start with uppercase letters, individual files start with lowercase letters). Then run the ls operation on that structure. Copy your code for main and the output of your finished program below.

public static void main(String[] args)

{

Directory Root = new Directory("Root");

Directory Documents = new Directory("Documents");

Directory Pictures = new Directory(" Pictures");

Directory Music = new Directory("Music");

File a = new File("fileA.txt");

File b = new File("fileB.txt");

File c = new File("fileC.jpg");

File d = new File("fileD.png");

File da = new File("fileD.txt");

Root.add(Documents);

Root.add(Pictures);

Root.add(Music);

Root.add(da);

Documents.add(a);

Documents.add(b);

Pictures.add(c);

Pictures.add(d);

Root.ls();

}

Root

Documents

fileA.txt

fileB.txt

Pictures

fileC.jpg

fileD.png

Music <empty directory>

fileD.txt