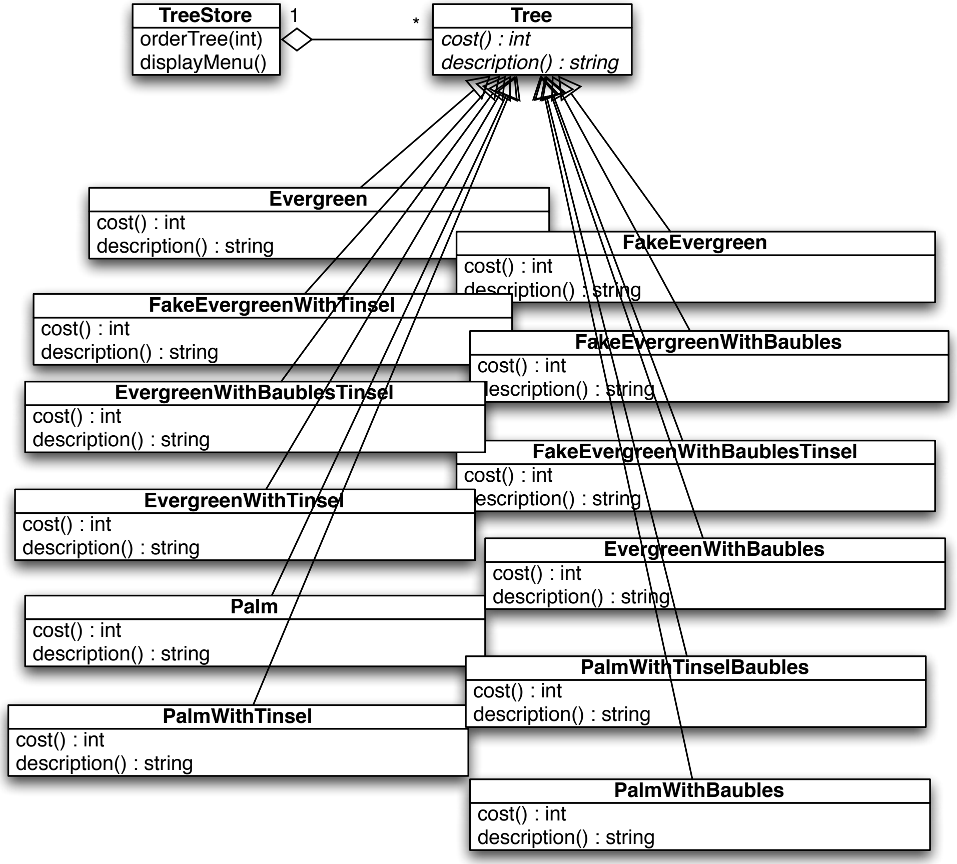
Question 3:

Part A:

A tree store offers a variety of trees for XMas holidays. Customers can inspect the tree store's 'menu' to see the combinations of trees and decorations that are available. The customer's can buy any combination that is available on the menu. I intend to use the **Decorator** pattern for this problem. Without the Decorator pattern, each combination of tree and decorations that the store sells and stocks would be a different type. The details of my implementation using the Decorator pattern are shown in *Part B*.

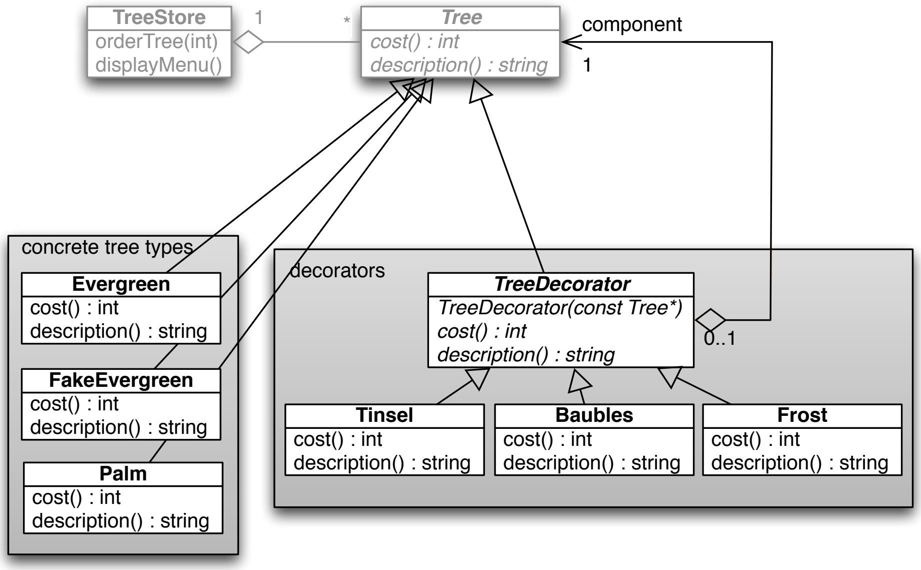
Problem Without Decorator Pattern:

Note: not all possible classes are shown in the figure below. For every new combination of trees and decorations the store wants to sell, a new class would be added to the figure below.



Part B:
You can see that all the original subclasses of Tree have been removed. In there place, there are now three concrete classes representing tree plant types and a second abstract class for decorations that can be added to the plants. There are three concrete types of decorations inheriting from this abstract decoration class.

Problem With Decorator Pattern:


TreeStore
orderTree(int)
displayMenu()
1
*
Tree
cost() : int
description() : string
component
1
concrete tree types
Evergreen
cost() : int
description() : string
FakeEvergreen
cost() : int
description() : string
Palm
cost() : int
description() : string
decorators
TreeDecorator
TreeDecorator(const Tree*)
cost() : int
description() : string
0..1
Tinsel
cost() : int
description() : string
Baubles
cost() : int
description() : string
Frost
cost() : int
description() : string