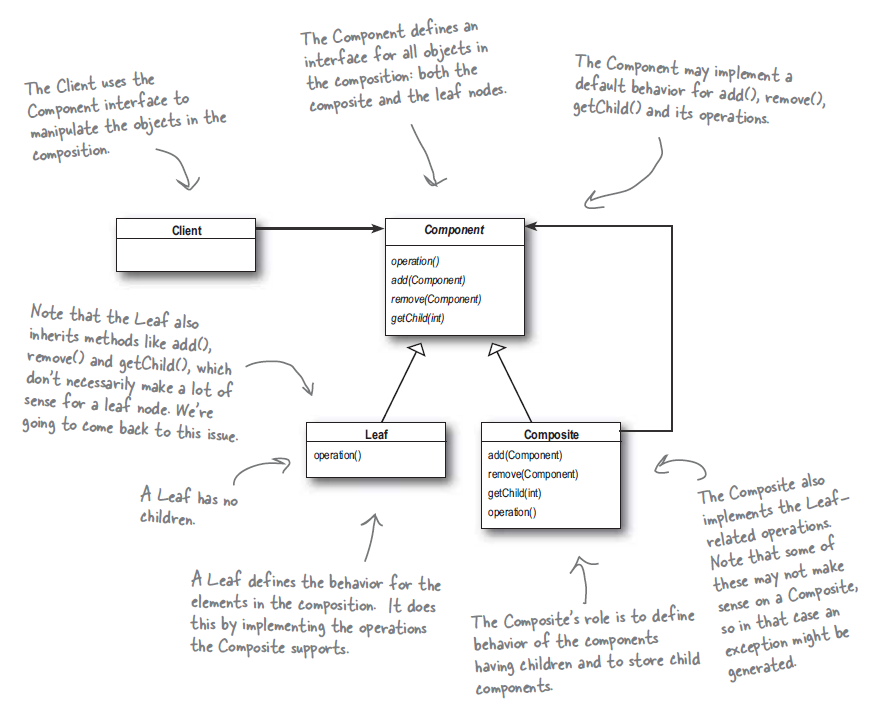
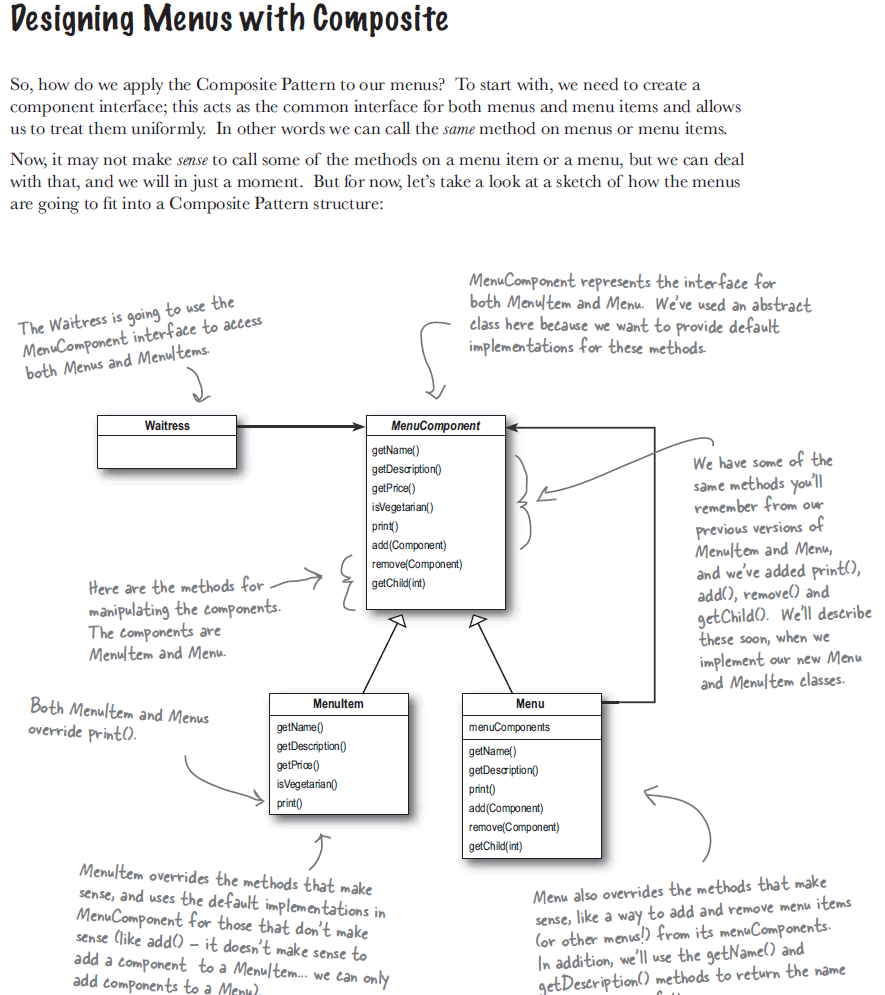
**Composite Pattern:**

1. **Definition**: The Composite Pattern allows you to compose objects into tree structures to represent part-whole hierarchy such objects and composition of objects could be treated uniformly.
   1. This pattern allows us to build structure of objects in form of trees that contains both composition of objects and individual objects as nodes.
   2. Using this pattern, we can apply same operation to both composition of object and individual object, thus ignore differences between composite and individual objects.
   3. When we structure data in this way we have composites at roots and branches of the composites growing up to leaf.
   4. **Class diagram**:



1. Both menu and menu component has same interface but they should only implement methods that make sense for them, giving default implementation for the rest or throw runtime exception.
2. A composite pattern manages hierarchy also operations on composite and leaf node. So composite patterns take single responsibility principle and trades it off for transparency. Here we have two different find of nodes but they implement same interface.
3. Keeping the order of the children is a challenge in this pattern. If traversing is expense we may have to implement caching of composite nodes.



1. **Java example**: To level component JFrame, JPanel contains other components like scrollbars, texts, buttons etc. So, GUI consists of several parts but when we display it we think it as whole. So, we just call display on top level component.