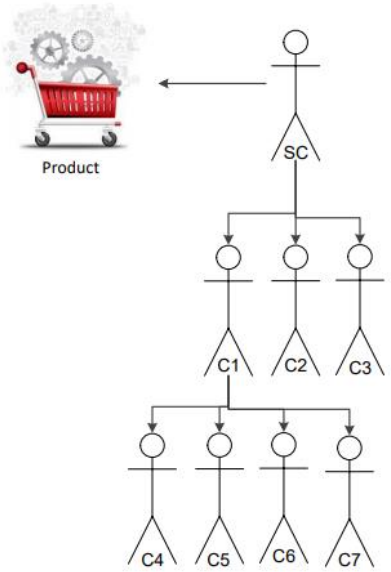
Problem Scenario:



Suppose, a small business company hired you to develop a medium-sized software. Basically, the company wants to develop a “marketing net” type software. The brief specification of the targeted software is as follows (modelled in Figure):

(i) It supports the online selling of the products as manufactured by the company.
(ii) A single customer (SC) is required to purchase at least one product.
(iii) Each customer also needs to bring/introduce three more customers (i.e. C1, C2, C3) in order to avail the special concession (package) as offered by the company.

Moreover, each customer is assigned to packages based on their number of invitations for other customers i.e Gold, Silver, Bronze. A customer who introduces three customers is counted as normal customer.

Identify the suitable structural and creational design patterns (as discussed in the class) to deal with the different situations of the given problem scenario of marketing net software. Note that more than one design pattern can be required to completely solve the given problem situations. It means that a single design pattern should be consider/use to handle the single situation of the problem scenario.

The problem is based on working both on structural and creational design patterns. It creates a product and assign it to the deserved customer. We need a combination of patterns that ease the design by identifying a simple way to realize relationships between entities and also, we need control on each object creation in a manner suitable to the situation. According to the Rule of Thumb, *Builder* pattern often builds a *Composite* but the problem is, *Builder* focuses on constructing a complex object step by step. *Abstract Factory* emphasizes a family of product objects (either simple or complex). *Builder* returns the product as a final step, but as far as the *Abstract Factory* is concerned, the product gets returned immediately.

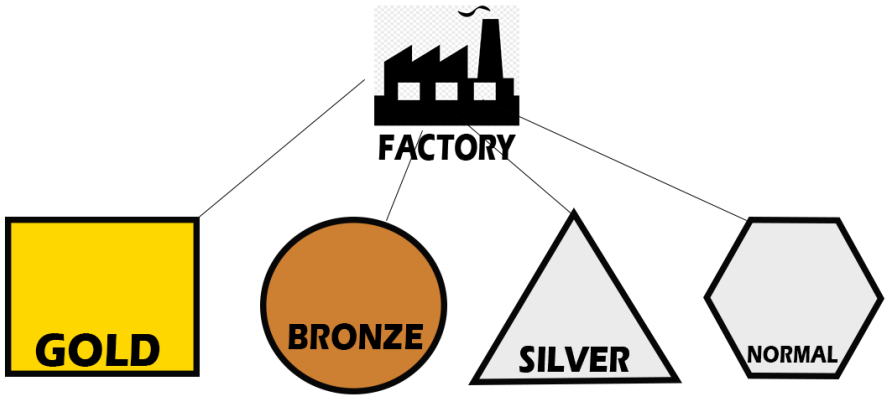
Structural Pattern: *Abstract Factory Pattern*

Creational Pattern: *Composite Pattern*

Formally, we need to build a hybrid pattern called **Composite Factory.**

Pattern: **COMPOSITE FACTORY**

Rationale: Imagine that you have two types of objects: Customers and Packages. A Customer can contain several Packages as well as a number of other Customers. These other Customers can also hold some Packages or even other Customers, and so on. In other words, we create a upside down tree.

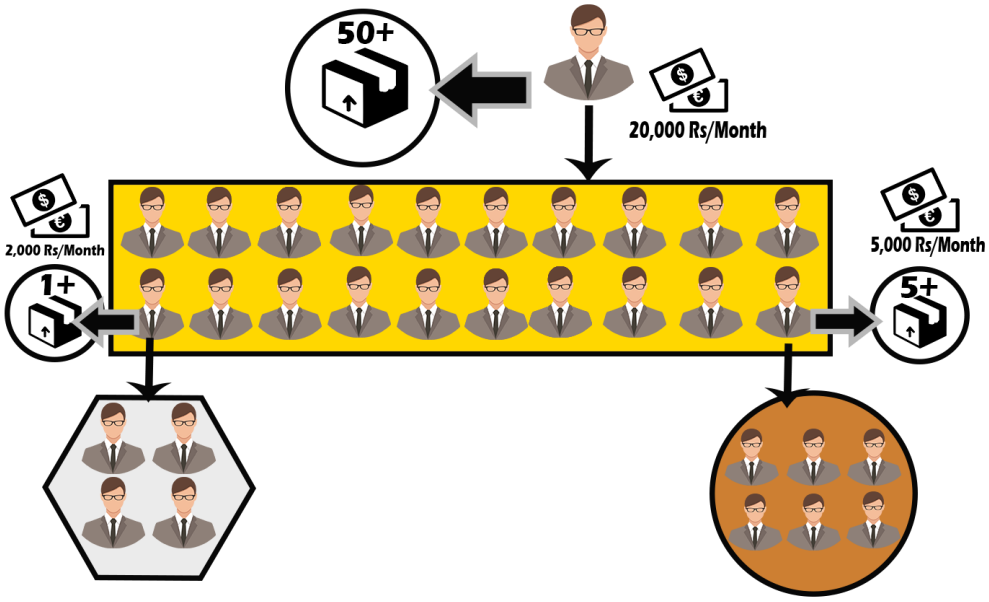


The below figure shows a scenario:

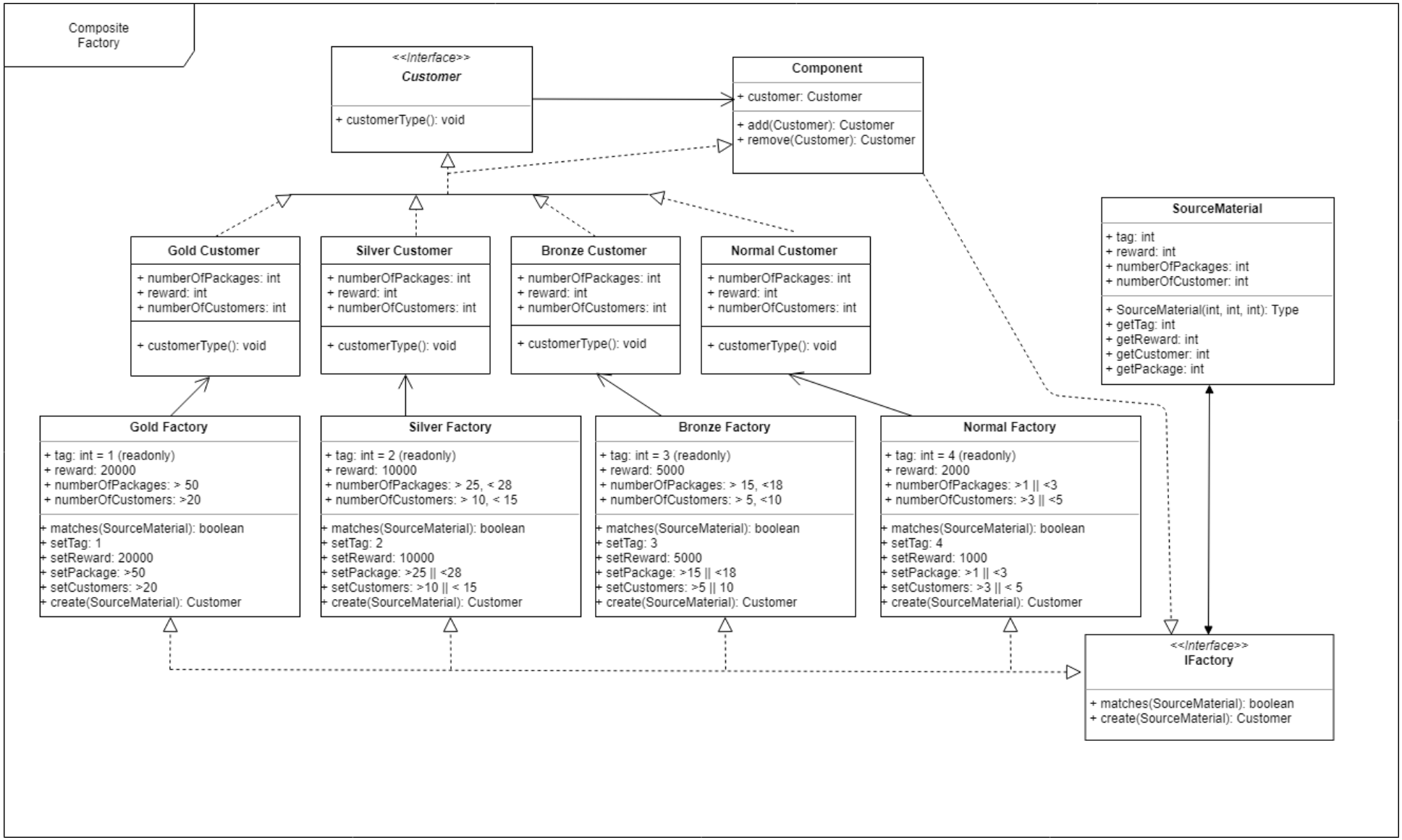
Gold customer sells 50+ packages and earns 20,000 Rs/Month reward. The sub-customers form a tree to further Normal Customer and Bronze Customer. A hierarchal representation to sub-objects and for each specific type of customer and package is made. With Abstract Factory, we will form a customer with its specific package, number of customers and reward. With Composite Pattern, we form a tree to each specific type of customer. To create a hybrid of Abstract Factory and Composite Pattern, we made concrete factories for each category of customer to generate the type of customer according to each requirement.

Interfaces: *IFactory and Customer*

Classes: *Source Material, Component, Gold Customer, Silver Customer, Bronze Customer, Normal Customer, Gold Factory, Silver Factory, Bronze Factory, Normal Factory,*



Class Diagram:



Implementation: