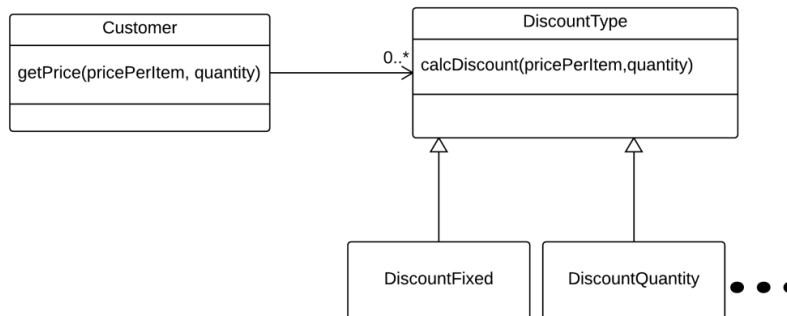


# Exercise - JPA Entity Mappings -3

## Mapping Inheritance

Imagine that we had different kinds of customers which would get a discount according to their type. The simple and often stupid solution would be to derive specialized Customers from the Customer class. Stupid because it won't allow a design where some customers actually could hold more than one type (get a better discount). This can be implemented using the design below.



Before you start this exercise you must know the basics of JPA inheritance mapping using the default SINGLE\_TABLE strategy (see the slides/presentation.md)

- 1) Implement the classes from the diagram above (all as Entity classes).
- 2) In the class DiscountFixed add the following code:

```
double discount = 0.1; //10% on all
@Override
public double calcDiscount(double priceItem, int quantity) {
    return priceItem * discount * quantity;
}
```

- 3) In the class DiscountQuantity add the following code:

```
int quantityForDiscount = 3;
double discount = 0.2; //20% on all if quantity is 3 or more
@Override
public double calcDiscount(double priceItem, int quantity) {
    return quantity >= quantityForDiscount ? priceItem * quantity * discount : 0;
}
```

- 4) Run the project and investigate the generated table. Explain the content of each column (especially the DTYPE column) and how it relates to the object model.
- 5) What we have done so far is using the default Inheritance Strategy which is SINGLE\_TABLE.

Add a `@Inheritance` annotation on top of the DiscountType class and select `InheritanceType.JOINED` for the strategy.

Regenerate the tables and explain the purpose/content of each table and each column (especially the DTYPE column) and how they relate to the object model.