**Diagrams explanation**

***CLOTHINGSTORE INC.***

PIELEANU, A. ANDREI

# Domain classes

The domain classes follow quite a simple principle: KYSS. They only contain the variables, constructors, getters, setters, methods inherited from Object (toString, equals, hashCode). The reason behind this is that their corresponding classes in the Service layer (UserService for User) handles the logic behind it. This ensures that the Single-Responsibility principle is respected, each class having exactly one responsibility.

A Category object can have a parent above it of the same kind and so on. This enables expandability in case of a larger chain of categories and subcategories. A Clothing object corresponds a subcategory and a subcategory only has one parent category and so on.

Order class has access to a multitude of classes from where it can retrieve information, such as Address, User, Order items and OrderStatus.

I decided to separate the linkage of Clothing into 2 different classes, one for Shopping Cart, and the other one for Order. The shopping cart item is linked to the product’s price directly, so that if the product’s price changes, so it changes in the cart item too. The order item is more independent from Clothing item because it has his own price and amount that you have bought. This will ensure that when the price changes, the price in the order stays the same.

# 3-layer architecture

I decided to go with the 3-layer architecture, because the separation of concerns can be clearly seen, each layer having its own responsibility (Repository handling database operations, Service handling logic and Controller handling requests/responses). Interfaces are set in between each layer so that the Dependency Inversion principle is respected.

All the CRUD methods are retrieved from the database into the repository, but the Services will utilize some of them in some cases (for example, ClothesService will not use the delete method, as this might cause some anomalies in the database. Instead of this, enable/disable is used). This ensures that YAGNI principle is respected. Only the needed functionality is added, thus making the application lighter, less prone to errors and less time spending on testing.