

Abstract Factory Pattern

Scenario:

Abstract factory pattern is used to wrap the methods of the function and objects and other factory classes can use the methods without knowing the details. This factory is also called a factory of factories. It's a creational pattern that provides objects for other factories to use.

Implementation:

For this project, we have a products class that has product price, quantity, and other product details with exposing all methods of the main product class we made the product class as abstract and imported base classes as Electronics and furniture which are factory classes in our cases by creating this class, we can set prices and other details for each category of the products.

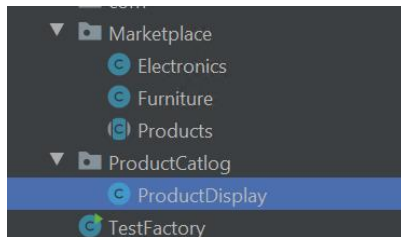
Product
productId: Integer quantity: Integer price: Float description: String
getQuantity() getPrice() getProductStatus()

All the objects are made private and methods to access them are made public so that other classes can access the data without knowing the details of the main product line

Code:

We have a marketplace package consisting of the following classes

Marketplace (Package)



Abstract class → Products

Factory class 1 → Electronics

Factory class 2 → Furniture

Product Display (Package)

Factory Driver Class → ProductDisplay

Test Factory class → Client access to the Program

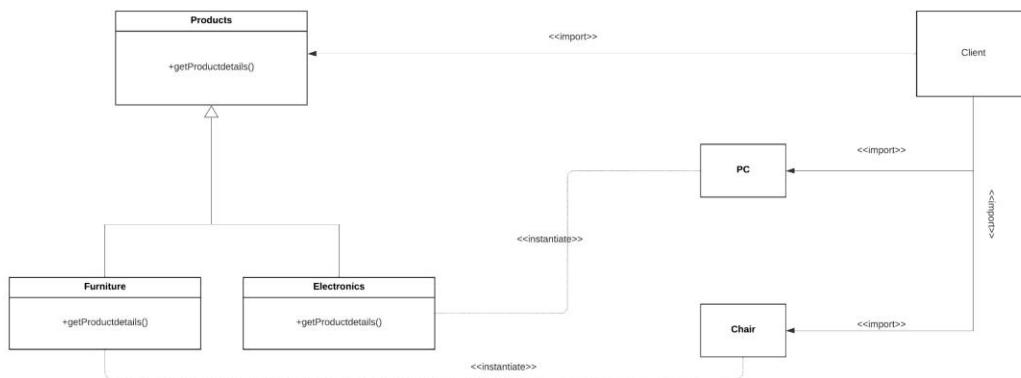
```
Products.java
1 // This is a abstract class
2 package Marketplace;
3
4 public abstract class Products {
5
6     public abstract int getQuantity();
7     public abstract float getPrice();
8     public abstract String getProductStatus();
9     public abstract String getDescription();
10
11     @Override
12     public String toString(){
13         return " Quantity = "+this.getQuantity()+ "\n Price per Item = "+ this.getPrice() / this.getQuantity()  +"\n Price";
14     }
15 }
16
```

Output:

```
Run: TestFactory x
"C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.3.1\jbr\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.3.1\jbr\bin\idea_rt.jar"
Final Price After Electronics Sales Tax::
Quantity = 2
Price per Item = 525.0
Price = 1050.0
Product Status = In stock
Product Description = This a 2gb computer
Final Price After Furniture Sales Tax::
Quantity = 16
Price per Item = 105.0
Price = 1680.0
Product Status = In stock
Product Description = This chair is comfortable for children

Process finished with exit code 0
```

Class Diagram:



Sequence State Diagram:

