Marcia Gallant

# Demo Package

Demo.java is the class with the main method. For testing, you can uncomment the different tests for the amounts of products and for the different shipping strategies.

You can add lollipops or musical slippers to your shopping cart. I factored out the code which was similar to the addLollipops and addMusicalSlippers methods in the Demo class into the addProduct method. This is so when there’s more different products to sell, there’s less code to write because they can also use the addProduct method.

# Product Package

There’s an abstract class ShoppingPenguinBrand which all of shopping penguins products extend.

There’s also a product interface which ShoppingPenguinBrandimplements in case shopping penguin wants to create new brands or allow his friends to sell their items on his website in the future. This is useful because then I can keep all of the customers items in an array list and makes keeping track of all the items easier. It has an abstract method calculateCost since I need to be able to get the cost of all the items. If they are stored in a ArrayList<Product> I need this to be able to quickly get the price of the product. Otherwise, I would have to determine its type before being able to determine its price. This would make it hard to extend.

There’s also an Enum class BrandType which he can add all the brands he wants to sell on his site to keep track of them all.

# Lollipops Package

There’s an abstract class Lollipop which all of the different types of lollipop classes extend. This is useful because I only need one method to add Lollipops in the demo class and I can create a LollipopFactory object to create the correct lollipop.

There’s a class for each type of lollipop he sells, so, I can easily create what object the customer is buying. There’s also an Enum class LollipopTypes to keep track of all the types.

# Musical Slippers Package

There’s an abstract class Musical Slippers which all of the different types of lollipop classes extend. This is useful because I only need one method to add Musical Slippers in the demo class and I can create a MusicalSlippersFactory object to create the correct Musical Slippers.

There’s a class for each type of Musical Slippers he sells, so, I can easily create what object the customer is buying. There’s also an Enum class MusicalSlippersType to keep track of all the types.

# Product Factory Package

This is a design pattern: factory.

LollipopFactory and MusicalSlippersFactory have the responsibility of creating the right type of the product. This is useful because when I have a lot of products on the site, I would need to add lots of if statements and keep checking what product it is, so, I would know which product I need to create. Now, if I ever need to add more product types, I only need to create their classes and add them to the correct factory. This makes the program much easier to extend.

# Shipping Strategies

This uses a design pattern: strategy.

The strategy is how to ship the products. So, there’s an interface called shipping with the method ship(). Each country has a different class for its shipping method. All the countries ship differently, so, the ship() method has different behaviour in all classes. This is useful because I can just set the correct shipping strategy and it will use the method ship() in the right class and ship to the correct location. I do not need to create multiple shipToSpecificLocation() methods and have to check to see where they need their products to be shipped to.