Assignment 4: Garment Factory

COMP 3522 Assignment #4

Garment Factory (aka I’ll take one of everything!)

Chris Thompson (some modifications by Rahul Kukreja)

cthompson98@bcit.ca

BCIT CST — November 2019

Introduction

Welcome to your fourth COMP 3522 assignment.

For assignment four, you will:

1. Use the pandas Python Data Analysis Library to open and parse information provided in an MS Excel .xlsx file
2. Implement the abstract factory pattern
3. Flex your programming muscles and impress me with your ongoing mastery of Object Oriented Programming.

Submission Requirements

1. This assignment is due before **11:59 PM on Friday 22nd November 2019. I will not check out any commits past this date and time.**
2. Submit your .py files and UML diagrams in the Assignment 4 folder of your GitHub repository.
3. This is an individual assignment. I strongly encourage you to share ideas and concepts, but sharing code or submitting someone else's work is not allowed.
4. Include a Readme file that describes how your application works. Tell me (very briefly) what each class does. Please also tell me about any bugs, errors, or omissions. List any errors or edge cases that are not

handled/accounted for. Additionally list any behavior or requirement that is not implemented (if any are left out) as well as any peculiar behaviors or extra functionality that I should keep an eye out for.

Grading

This assignment will be marked out of 15. For full marks, you must:

1. **(5 marks)** Correctly implement the requirements described in this document
2. **(2 marks)** Correctly format and comment your code. Eliminate all warnings offered by PyCharm, follow PEP 8 guidelines, choose excellent function and variable identifiers, craft code that is easy to understand and lends itself to future maintenance, use whitespace wisely, write good and appropriate docstrings, etc.
3. **(2 marks)** Write unit tests which verify the correctness of your program
4. **(2 marks)** Include a UML Class Diagram depicting appropriate use of OOP principles, inheritance and design in your code
5. **(1 mark)** Manage errors and unexpected input gracefully
6. **(1 mark)** Format your output. Make sure your messages display the correct information in a pleasant, highly readable manner. You know I *am* a fan of ASCII art…
7. **(2 marks)** Your answer to the final question at the end of the list of requirements. Include this in your mandatory README file.

Implementation Requirements

In programming, we often say that **interfaces are less volatile than implementations**. That is, we tend to avoid changing them. This makes good sense when you consider how we program. Changing a concrete implementation doesn't usually mean we have to change the interface(s) it implements. Changing an interface, however, invariably means we need to change the classes that implement it (You could say, the implementation is coupled to the interface). So we avoid modifying interfaces. They become less volatile. Most developers respect this, and those who don't quickly learn to pretend.

Suppose you own a garment factory. Through hard work, sound business decisions, and a bit of luck, you've built a mini-empire of your own, complete with employees, machinery, a dependable supply chain for textiles and other materials, a reliable logistics operator for moving your product to your customers, and (of course!) more than enough customers to keep the lights on.

I don't know the ins and outs of the textile industry, but I do know that a single garment factory may produce garments for any number of different brands. How might this work? Well, you probably have employees who create the garments, and there's probably machinery for them to use. Or maybe you've upgraded and automated your production line. Maybe you have a robotic assembly line that you configure for each run of each brand or each kind of clothing. When your salesforce brings in a new order, you identify what's required, reconfigure the machinery, load up the textiles, and press Go. (Or something like that, anyway!)

Let's model this. Let's pretend your factory produces garments for some off-brand clothing lines called **Lululime, Pineapple Republic, and Nik**a. For each brand, your factory produces **men's shirts**, **women's shirts**, and **unisex socks**. Your salesforce collects orders and creates an entry (row) for each order on a master Excel spreadsheet. You review the Excel spreadsheet each morning and ensure the correct types of clothing are made according to the correct Brand specifications.

In order to capitalize on your BCIT education, you've decided to write software to automate the process. Yes, it may have been wiser to hire an expensive consulting company to do this, but that would cost tens of thousands of dollars and take a long time. You have until next Friday.

Your implementation must include the following:

1. An abstract factory class called **BrandFactory**. This abstract class must have the following methods:
    1. createShirtMen which returns a ShirtMen (see below)
    2. createShirtWomen which returns a ShirtWomen
    3. createSocksUnisex which returns a SockPairUnisex
2. The **concrete factory classes** that extend the abstract BrandFactory class:
    1. LululimeFactory

    2. PineappleRepublicFactory
    3. NikaFactory
3. An **abstract class for ShirtMen**. A men’s shirt has the following attributes:
    1. style (a string) for the trendy name of the shirt style
    2. size (S, M, L, XL, XXL)
    3. colour
    4. textile, i.e., cotton, polyester, cotton-poly blend, etc.
4. An **abstract class for ShirtWomen**. A women’s shirt has the following attributes:
    1. style (a string) for the trendy name of the shirt type
    2. size (XS, S, M, L, XL, XXL)
    3. colour
    4. textile, i.e., cotton, polyester, cotton-poly blend, etc.
5. An **abstract class for SockPairUnisex**. A men’s shirt has the following attributes:
    1. style (a string) for the trendy name of the shirt type
    2. size (S, M, L)
    3. colour
    4. textile, i.e., cotton, polyester, cotton-poly blend, etc.
6. The ShirtMen class must be extended by **ShirtMenLuluLime**, **ShirtMenPineappleRepublic**, and **ShirtMenNika** classes:
    1. ShirtMenLululime has additional attributes for:
        1. Whether the garment is designed for yoga or for running
        2. The number of hidden zippered pockets in the shirt
    2. ShirtMenPineappleRepublic has additional attributes for:
        1. Whether the shirt requires ironing or not (no-iron)

        2. The number of buttons on the shirt
    3. ShirtMenNika has an additional attribute for whether the garment is for indoor or outdoor sports
7. The ShirtWomen class must be extended by **ShirtWomenLuluLime**, **ShirtWomenPineappleRepublic**, and **ShirtWomenNika** classes:
    1. ShirtWomenLululime has additional attributes for:
        1. Whether the garment is designed for yoga or for running
        2. The number of hidden zippered pockets in the shirt
    2. ShirtWomenPineappleRepublic has additional attributes for:
        1. Whether the shirt requires ironing or not (no-iron)
        2. The number of buttons on the shirt
    3. ShirtWomenNika has an additional attribute for whether the garment is for indoor or outdoor sports
8. The SockPairUnisex class must be extended by **SockPairUnisexLuluLime**, **SockPairUnisexPineappleRepublic**, and **SockPairUnisexNika** classes:
    1. SockPairUnisexLuluLime has additional attributes for:
        1. Whether the garment contains silver to combat athletic odour
        2. The colour of the contrasting stripe (all Lululime socks have a contrasting stripe)
    2. SockPairUnisexPineappleRepublic has an additional attribute for whether the socks require dry-cleaning or not
    3. SockPairUnisexNika has additional attributes for:
        1. Whether the sock is articulated or not.
        2. Sock length (ankle, calf, or knee)
    4. **NOTE:** the concrete factory class methods must return brand-specific apparel. For example, the LululimeFactory class' createShirtMen method must return a ShirtMenLululime, the createShirtWomen method must return a ShirtWomenLululime, and the createSocksUnisex method must

return a SockPairUnisexLuluLime. Ensure all three factory classes follow this pattern.

5. Included with this assignment is an MS Excel spreadsheet called COMP_3522_A4_orders.xlsx. This MS Excel spreadsheet contains your orders for a day. Your program must open this Excel spreadsheet and process it one row at a time. This is not difficult. You may remember we began to explore spreadsheets and CSV files in COMP 1510. There is a very popular, very helpful Python data analysis library called **Pandas** (we pronounce it Panda-Ess, not pandas). Check it out. You will find it helpful.
6. Your program requires a driver class called **GarmentMaker**. GarmentMaker must contain the main method that drives your program. GarmentMaker must also contain a constructor which initializes three ArrayList instance variables called shirtsMen, shirtsWomen, and socksUnisex. We will place the day's orders into each ArrayList and then "send" everything at the end of each working day.
7. Your program requires a class that is responsible for processing the Excel spreadsheet. Please call this class **OrderProcessor**. Ensure its design is object oriented. This object must be instantiated by your GarmentMaker class, and you must add an instance variable for your OrderProcessor object to your GarmentMaker. Some useful methods may include:
    1. openOrderSheet
    2. processNextOrder
8. Your program must ask the user for the name of the Excel spreadsheet that contains the day's orders. **OrderProcessor must use the pandas library** to extract the orders from the spreadsheet, one row at a time.
9. The Order processor is also responsible for identifying the correct factory to use for each order. **It stores the order details and the associated factory in an object of type Order and returns it.** (You will need to make a separate order class for this)
    - HINT: you may want to save the order details in a dictionary.
10. Your **GarmentMaker driver class should receive this order object** and based on the type of product being ordered (remember the

GarmentMaker should not be dependent on any concrete product) invoke one of its three methods:

- shirtMenMaker(Order shirt_men_order)
- shirtWomenMaker(Order shirt_women_order)
- socksUnisexMaker(Order socks_unisex_order).
- Each of these methods will invoke the correct method on the BrandFactory passed to it in order to create the correct number of the correct type of clothing. Ensure the constructed garments are added to the correct ArrayList in the GarmentMaker.

11. Once the day's orders have been processed, **GarmentMaker must produce a report summarizing the day's work**. The report must contain a single line per order. The line must include the following data:
    1. Brand
    2. Garment produced
    3.
12. Answer this two-part question in your **README file**:
    1. Suppose you just won a contract to produce shirts and socks for Goosie, another off-brand clothing maker. Exactly what changes do you need to make to your code to make this
    2. Suppose you won a contract to start producing women's activewear pants for each of these three brands. Exactly what changes do you have to make to your code base in order to make this possible?
13. **Remember to program to an interface**. We use abstract data types or interface data types as much as possible for parameter and return value data types. This makes it easier to:
    1. Isolate concrete classes
    2. Separate business logic
    3. Exchange modules or product families easily
    4. Promote consistency in our code.

That’s it! Good luck, and have fun.