NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

CS3005 – Software Design & Architecture Lab

LAB Instructors: Sobia Iftikhar “Sobia.iftikhar@nu.edu.pk”

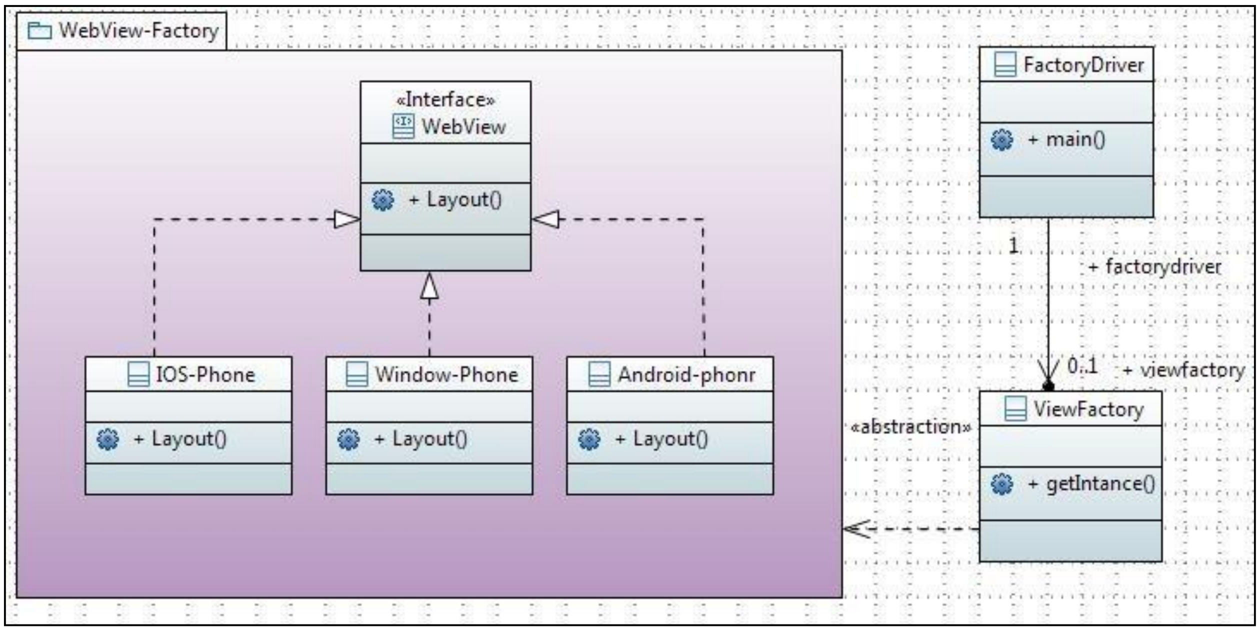
Lab 13-Task

Task 01

User experience (UX) encompasses a wide variety of aspects. iOS and Android both provide desirable UX features, but they have different priorities, which may influence which is the best fit for you. IPhones tend to be easier to use than Android phones. Their interface is more intuitive, and there have been few changes in how the phones work from release to release. Once you pick up how to use an iPhone, you shouldn’t have to relearn much after an upgrade. Even so, Androids have a much more customizable interface. You can set default apps, add widgets, choose from various lock screens, change your keyboard, and more. Unlike an iPhone, you can really make an Android your own and modify it to better suit your needs.

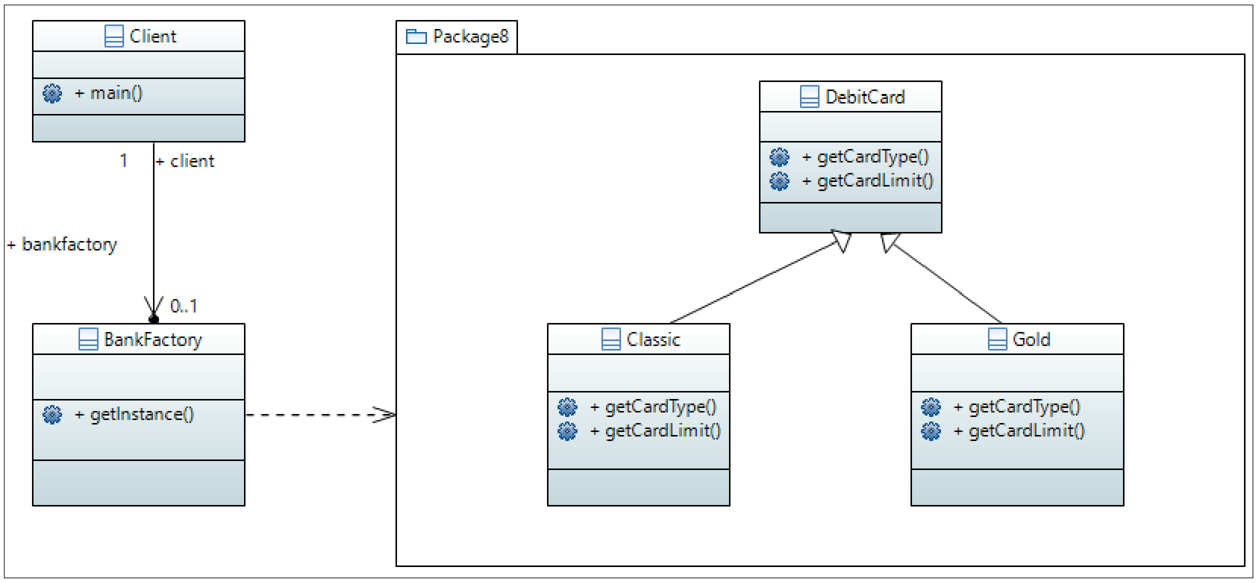
Let suppose user have multiple and different implementation of same module (Web-View). User wants to invoke **Layout ()** function for all kind of phone, but make sure we need to hide the logic of object creation for every particular class from user.
Implement **Layout ()** for all the concrete class. Make sure don’t create direct instance of concrete classes in factory classes.
Layout () print three different message for respective class.

1. ***IOS provide light color web view***
2. ***Window provide Multicolor color web view***
3. ***Android provide dark base color web view use as a***



Task 02

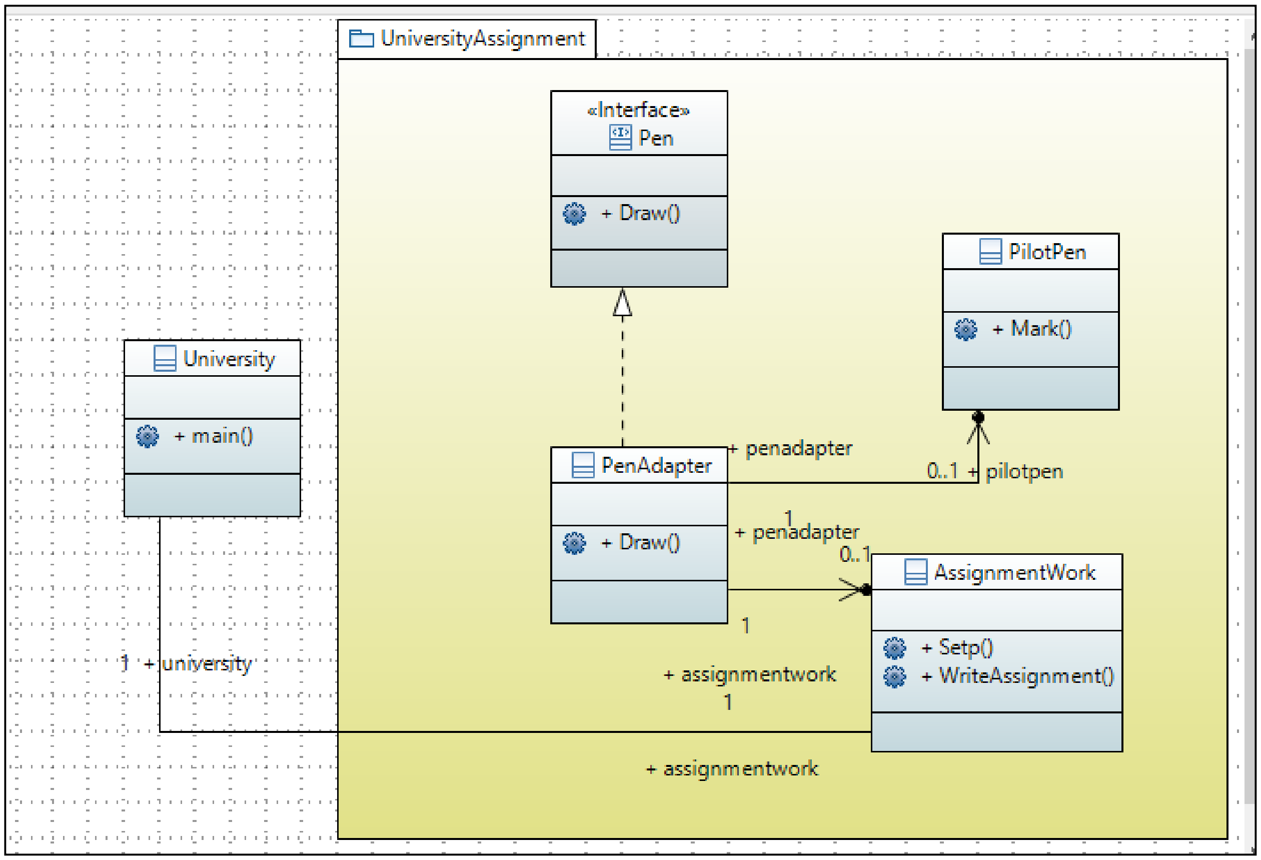
Every bank offers their customers different types of cards for their convenience in shopping, withdraw, ordering etc. Bank has some different types of cards like DebitCard can be classic orgold based on the income of the customer. Gold is offered to the customers with income more than 70K per month and classic is for the customers with more than 20K per month. In this example DebitCard is the interface having two methods getCardType and getCardLimit to get the type of card and the withdraw limit of card. Classic and Gold are the concrete classes implementing DebitCard interface. BankFactory is the factory class having getInstance method. Client is interacting with the BankFactory as card logical details are hidden from him.



Task 03

Every device—smartphone, tablet, or laptop—seems to come with its own charger.But do you really need all these different cables and charging blocks? Can you re- use the same charger for multiple devices? While this used to be a far more complextopic, standards have (finally) started to come into play that make it much easier to manage. Every Company made cellphone version with particular chargerADAPTER. Let’s suppose we are recently using iPhone 6 cellphone, and charger isnot working, but we have another iPhone 7’s charger. Is it possible to charge our recent phone with other kind of charger? If yes, then what kind of techniques we need to implement to get from it.

How we utilize the pattern techniques, and which pattern need to implement.



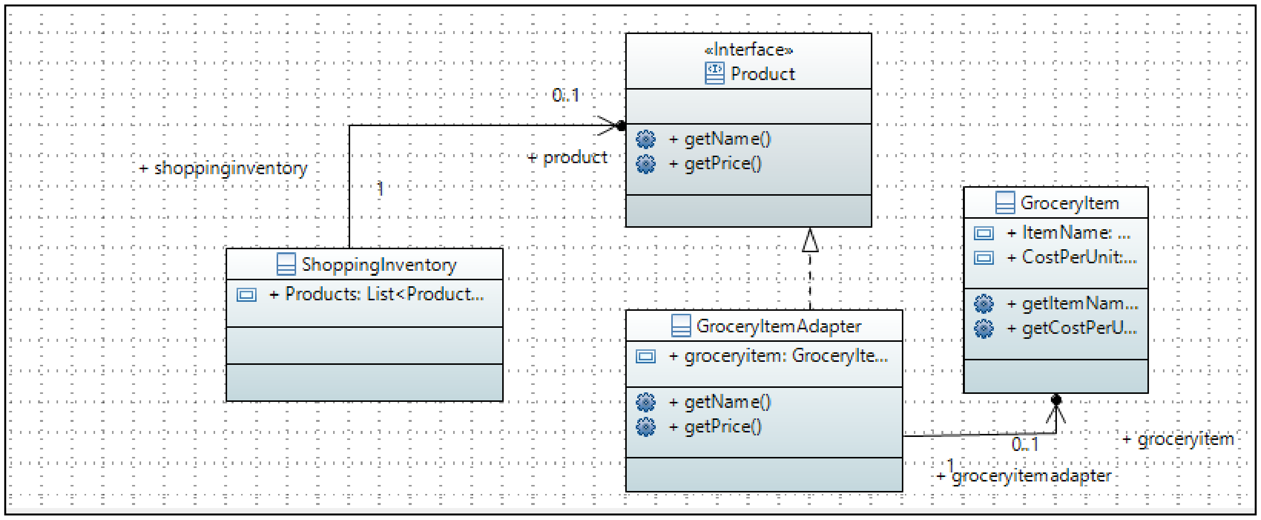
Task 04

Let’s say we have a *ShopInventory*-Class which maintains a list of products. Later on, we took over another store inventory which sells groceries. We now want to addthose items to our *ShopInventory-class*. The problem we have here is that although the *GroceryItem-class* is just a type of product but is unrelated to the *Produc- classt* interface.

To solve this problem, we’ll use the adapter pattern. We’ll create a *GroceryItemAdapter-class* which will implement the *Product* interface:With thehelp of an adapter, we’ll now be able to treat the *GroceryItem-class* as a *Product* without changing anything in the third-party code(*GroceryItem-class*).

The adapter pattern helps us to connect two incompatible interfaces exposing thesame business functionality.

With an adapter pattern, we convert an existing interface to another interface thatthe client code expects.


«Interface»
Product
+ getName()
+ getPrice()
0..1
+ product
+ shoppinginventory
1
ShoppingInventory
+ Products: List<Product...
GroceryItem
+ ItemName: ...
+ CostPerUnit:...
+ getItemNam...
+ getCostPerU...
GroceryItemAdapter
+ groceryitem: GroceryIte...
+ getName()
+ getPrice()
0..1
+ groceryitem
1
+ groceryitemadapter