SWEN30006 Software Modelling and Design
Workshop 5: Fundamental Design Patterns (2) – Partial Solution

School of Computing and Information Systems
University of Melbourne
Semester 2, 2019

Exercise 1

By this stage you should be getting your UML diagram notation correct. Some relevant sections of the textbook: Chapter 9 Domain Models – especially 9.5 Guideline: How to find Conceptual Classes; and 9.14 Associations

Exercise 2

This is like a follow on from Workshop 4. Some relevant sections of the textbook:

- Chapter 17 GRASP: Designing Objects with Responsibilities (esp.)

- Chapter 19 Designing for Visibility

- Chapter 25 GRASP: More Objects with Responsibilities

Your design model will vary according to your interpretation of the requirements and the domain model you produce, and you should refine your design model after dynamic modelling. The diagram that follows is an indicative result after that refinement.

Example sequence diagram: note that your sequence diagram will likely look different given your domain models and static design model.

UML Class Diagram

Customer
- name :string
- dob :date
- phoneNo :int
- vip :boolean

«abstract» Card
- cardNo :int
- expirationDate :date
- cvv :int
- name :string

Store Card
- balance :float

Credit Card
- limit :float

Staff
- name :string
- dob :date
- staffId :int

Supervisor
- department :string

MST
- user :Staff
- currentSales :ArrayList<Sale>
- + scan(barcode :Barcode) :Item
- + login(user :string, pwd :string) :boolean
- + checkStock(i :Item) :int
- + checkSales(u :User) :ArrayList<Sale>
- + makeSale() :Sale
- + finaliseSale(s :Sale) :boolean
- + addItem() :float
- + processPayment(cardData :CardData) :boolean

«abstract» Card Agency
- + validate(c :Card) :boolean
- + charge(c :Card, price :float) :boolean

Visa Agency

Store Card Agency
- + checkBalance(c :Card) :float

Policy

Sale
- items :arrayList<Item>
- totalPrice :float
- discountPolicy :Policy
- + addItem(i :Item)
- + calculatePrice() :float
- + checkOut() :boolean
- + addDiscountPolicy(p :Policy) :boolean

Item
- barcode :int
- stockNum :int
- price :float

Sales Server
- sales :hashMap<Sale>
- policies :hashMap<Policies>
- + submitSale(s :Sale) :void
- + retrieveSales(user :string) :arrayList<Sale>
- + retrieveDiscountPolicy(s :Sale) :Policy

Security Server
- + releaseSale(s :Sale) :void

Stock Inventory Server
- stock :hashMap<Item>
- + queryStock(i :Item) :int
- + removeStock(i :Item, num :int) :boolean
- + increaseStock(i :Item, num :int) :boolean
- + addItem(i :Item, num :int) :boolean
- + removeItem(i :Item) :boolean
- + loadItem(barcode :int) :Item

Relationships / associations:
- Customer — owns — Card (1 to 1..*)
- Staff — serves — Customer
- Staff — uses — MST (+user 1 to 0..*)
- MST — read — Card
- MST — creates — Sale (1 to 0..*, +currentSales)
- MST — queries — Card Agency (0..* to 0..1)
- MST — queries — Stock Inventory Server
- Sale — includes — Item (0..* to 0..1)
- Sale — has — Policy (1 to 1)
- Policy — +discountPolicy / manages — Policy (0..*)
- Sales Server — manages — Sale (-sales 0..*)
- Sales Server — manages — Policy (-policies 0..*)
- Sales Server — updates — Security Server
- Security Server — releases — Sale
- Stock Inventory Server — recorded by — Item (1 to +stock 0..*)
- Security Server — updates — Stock Inventory Server

2

sd scan item

scan(Barcode) → :MST

:MST → :Stock Inventory Server : loadItem(barcode)

:Stock Inventory Server : lookup(barcode) :item

:Stock Inventory Server ⇢ :MST : :item

:MST ⇢ :item

---

sd read card

processPayment(cardData) : boolean → :MST

:MST → :Card Factory : buildCard(cardData)

:Card Factory : type= determineType(cardData)

Note: Card is really just a representation of the information needed by the relevant credit agency

alt

[type = credit]

:Card Factory → :Credit Card : build(data) :Card

[type = store]

:Card Factory → :Store Card : build(data) :Card

:Card Factory ⇢ :MST : :Card

:MST ⇢ :Card