Claude Cockfield.
Spring 2024.
CSCI-540 Object Oriented Development.
Exercise #3 – Object Oriented Principles.

1a. **Define Class**

A class is a descriptive template of a conceptually bound collection of data (attributes) and behaviors (methods).

Such a definition is an abstraction that may be used to represent a physical object, a process, a common software pattern or any other conceptual grouping that the designer correctly determines shares a common identity or common overarching purpose.

Class design typically identifies nouns in the problem domain as primary initial candidates for class representation.

A class definition can also be called a ***type***.

1b. **Define Object**

An object is an instance of a class. An object is typically created by using the corresponding class definition as a template to create an instance of that class in program memory.

1c. **Define Encapsulation**

Encapsulation is the hiding of the details of a class' internal implementation away from the users of that class.

This follows the maxim of maintaining a separation of concerns.

The user of a class should neither be exposed to nor have to concern themselves with the internal implementation of a class.

A class is an abstraction and the user of the class should generally be able to use it at the level of the abstraction as if it were a primary indivisible thing.

Computing as a discipline thrives due to multiple levels of abstraction and if we were not able to operate at higher levels in the hardware-software systems abstraction hierarchy, many tasks would become unfeasibly complex and/or intractably convoluted.

1d. **Define Inheritance**

Inheritance is a mechanism where a class definition can be automatically based on the definition of another class. That is, if class B inherits publicly from class A, class B will automatically possess the same data and attributes as class A (although some may not be visible).

Any subsequent changes to the definition of class B will be layered on top of the base inherited definition that it gets from class A.

If class B inherits from class A, class B is the ***subclass*** and class A is the ***superclass***. Class B is also said to ***extend*** class A.

There are language specific visibility modifiers of class inheritance, for example private, public, protected.

Additionally there are language specific limitations on what can be changed, overridden or overloaded within a subclass.

In the general public inheritance sense, when a class inherits from another class it is said to have an "***is-a***" relationship with it. That is every instance of the subclass is also an instance of the superclass.

This is important for polymorphism.

1e. **Define Polymorphism**

Polymorphism allows us to programatically treat objects of different types as if they were the same *type T*, as long as they all share an “***is-a***“ relationship with that type T.
It allows us to treat an object of any type, as if it were any of the “***is-a***” relationship classes in its inheritance chain.
In Java where all classes inherit from the class Object, we are able to treat all objects of any class like an instance of Object.

2.

class: Order

Order represents a single customer’s single order and contains enough information, captured at the point of order inception, for the Order to be managed through its complete life cycle.

class: OrderItem

OrderItem represents a product sub-order within an owning Order. OrderItem objects cannot exist independently of Orders. Orders can contain a maximum of 10 instances of OrderItem.

class: OrderManager

OrderManager contains the current session Order and read-only-access classes for the ancillary static data required to create and manage Orders.

3.

class: Order

order-ID	ID field containing automatically generated order ID.
customer-ID	Customer ID field.
order-status-ID	Order Status ID field.
order-date	Order date-time.
shipping-mode	Character indicating shipping mode – with ‘A‘ or ‘G‘
order-item-list	Array/List containing max 10 instances of OrderItem objects.

class: OrderItem

order-ID	Here for completeness. Not needed in code as OrderItem class can only exist within an order, so order-ID is implicit in that relationship.
order-item-ID	The order item number within the Order. This should range from 1 – 10.
product-ID	Product-ID of product in this OrderItem.
order-quantity	Quantity of Product-ID in this OrderItem.
order-price	Total price of total quantity of Product-ID in this OrderItem.
product-category-ID	Product category ID field – saved here as categories can change over Order life-cycle.

warranty-period-ID	warranty period ID field – saved here as warranty periods can change over Order life-cycle.

class: OrderManager

current-order	Order currently being worked on.
products	ProductQuery – allows read only access to product static data.
customers	CustomerQuery – allows read only access to customer static data.

NOTES:

The above **OrderManager** would be contained within a Session Object of some sort.
Product and Customer read-only static data access is required to manage Orders.
Read-only access is facilitated through ProductQuery and CustomerQuery read-only access objects.
General static data management classes exist for ProductCategory(ProductCategoryUnary, ProductCategoryCompound), WarrantyPeriod, WeightClassification, Supplier, Warehouse, Stock, OrderStatus, Customer(CustomerPerson, CustomerCompany) and Product.
Static data managers inherit and extend corresponding read-only Query classes.

4.

class: Order

addOrderItem	Add a new OrderItem to Order
calculateTotalOrderPrice	Calculate and set the total price for all OrderItems and the individual prices for each OrderItem using the supplied Customer discount.
init	Constructor initializing most **Order** attributes.
removeOrderItem	Remove an OrderItem from Order
setOrderStatus	Set OrderStatus. Used to move Order through processing stages through the Order life-cycle.

class: OrderItem

init	Constructor initializing most **OrderItem** attributes.

class: OrderManager

commitOrder	Save current-order to database and set status to live.
createOrder	Create a new order.
findCustomerOrders	Find orders for a specific customer. Can be filtered to live orders or orders over a period.
init	Constructor initializing most **OrderManager** attributes.
resetOrder	Resets current-order back to its freshly created new state. This can only be done on uncommitted Orders.
retrieveLiveOrders	Find and return summary details for all live Orders.
retrieveOrder	Find and load a specific order.

SaveOrder	Save current state of current-order to database.
setOrderDispatched	Set order to dispatched status.
setOrderVoid	Set current-order to status void. Orders are never deleted, only voided. This is a terminating final life-cycle event.

5.

The most natural inheritance hierarchy class amenable classes, given the problem description, are the Customer and the ProductCategory classes. Order, OrderItem and OrderManager (from 2,3,4 above) are less natural fits.

class: ProductCategory – abstract class

product-category-ID	Auto-generated product category ID

ProductCategoryUnary inherits ProductCategory

product-category-name	Category name
product-category-desc	Category description

ProductCategoryCompound inherits ProductCategory

product-categories	Array of **ProductCategory** objects references to **ProductCategoryUnary** and **ProductCategoryCompound** objects – thereby creating a tree.

class: Customer – abstract class

customer-ID	Auto-generated customer ID
name	Customer name – multiple string fields
address	Customer address – multiple string fields
telephone-number	Telephone number format string field

CustomerPerson inherits Customer

license-number	License number of customer

CustomerCompany inherits Customer

contact-person	Customer name – multiple string fields
discount	Company discount

-

class: ProductCategory

isCategory	Checks whether Category is a given Category. No definition if

	language allows it (just declare). Else returns false.

ProductCategoryUnary

isCategory	Checks whether Category is a given Category by direct checking of **ProductCategory** parameter with this object.

ProductCategoryCompound

isCategory	Checks whether Category is a given Category by checking of **ProductCategory** parameter with all ProductCategory objects in hierarchy.

-

class: Customer

getDiscount	Returns customer discount. Not defined here if language allows it (just declare). Else returns zero.
getContact	Not defined here if language allows it (just declared). Else returns empty.

CustomerPerson

getDiscount	Returns zero.
getContact	Returns **CustomerPerson** name and telephone number.

CustomerCompany

getDiscount	Returns CustomerCompany discount.
getContact	Returns **CustomerCompany** contact-name and telephone number.