Homework

by

Benjamin Moks

Stevens.edu

September 20, 2024

Homework

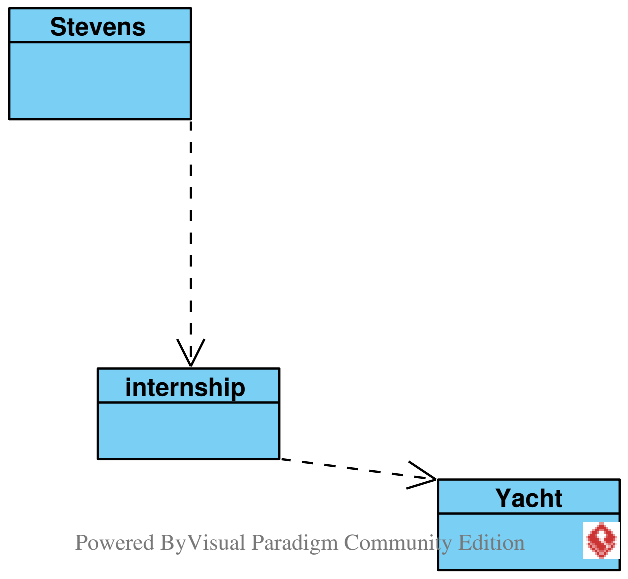
Benjamin Moks
Stevens.edu

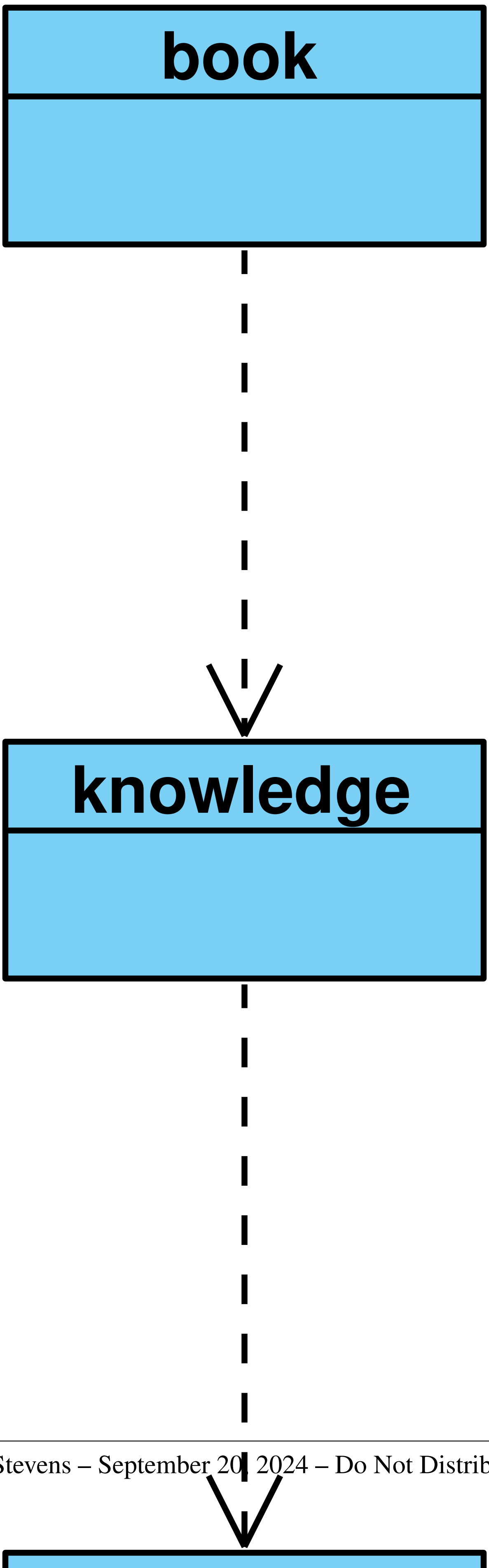
Table of Contents

Chapter 1

– Benjamin Moks

Hello My name is Benjamin but most people call me Ben. I'm from Long Island New York and have a pet dog. My favorite place to eat at is Shahs halal because it is open late and is cheap. I also enjoy playing volleyball.

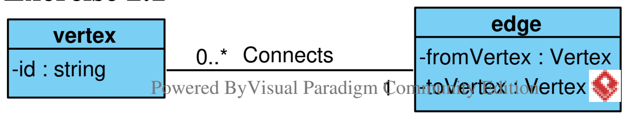

Stevens
internship
Yacht
Powered By Visual Paradigm Community Edition


book
knowledge

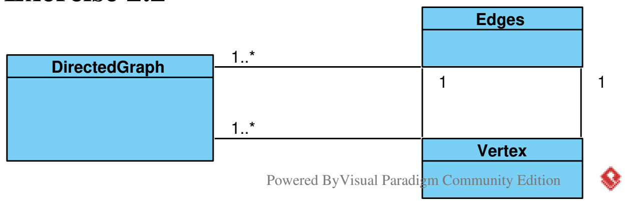
Chapter 2

UML Class Modeling

Exercise 2.1



Exercise 2.2



Exercise 2.3

Window - ScrollingWindow: ScrollingWindow inherits from Window, adding scrolling features with xOffset and yOffset.

Window - Canvas:
Each Window contains one Canvas, used for adding and removing graphical elements.

Canvas - Shape:
A Canvas holds multiple Shapes (e.g., lines, polygons), each with attributes like color and lineWidth.

Shape - Line/ClosedShape:
Line and ClosedShape inherit from Shape. Line has coordinates, while ClosedShape includes filled shapes.

Shape - Polygon/Ellipse:
Polygon and Ellipse are types of ClosedShape, with Polygon defined by points and Ellipse by its dimensions.

Window - Panel:
A Window contains multiple Panels, which organize PanelItems like buttons or text fields.

Panel - PanelItem:
A Panel holds multiple PanelItems, which have attributes like x, y, and a label.

PanelItem - Button/ChoiceItem/TextItem:
PanelItem has specific subclasses, including Button, ChoiceItem, and TextItem, each with unique attributes.

ChoiceItem - ChoiceEntry:
ChoiceItem contains multiple ChoiceEntry objects, each representing a selectable option.

PanelItem - Event:
PanelItem triggers Event objects (e.g., button clicks or keyboard events).

Canvas - ScrollingCanvas/TextWindow:
ScrollingCanvas and TextWindow inherit from Canvas, adding scrolling and text features.

Point - Polygon:
Polygon consists of ordered Points, defining its shape.

Exercise 2.4
Customer - MailingAddress:
A Customer can have one MailingAddress, representing their address and phone number.

Customer - CreditCardAccount:
A Customer is associated with multiple CreditCardAccounts (*), each representing their credit limit, current balance, and account details.

CreditCardAccount - Statement:
Each CreditCardAccount can have one Statement (0..1), representing the payment details, finance charge, and minimum payment.

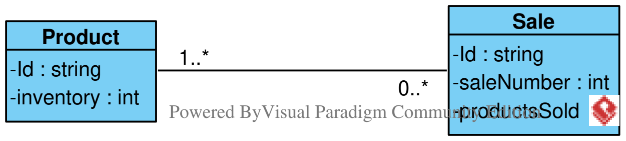
Statement - Transaction:
A Statement can have multiple Transactions (*), with each Transaction representing details like the transaction date, explanation, and amount.

Transaction - CashAdvance/Interest/Purchase/Fee/Adjustment:
A Transaction can be a CashAdvance, Interest, Purchase, Fee, or Adjustment, representing different types of transactions.

Transaction - Merchant:

A Transaction can involve a Merchant, which includes the merchant's name.

CreditCardAccount - Institution:
Each CreditCardAccount is associated with one Institution, representing the bank or financial institution that issues the card.



```
# Needed for forward reference of Sale in Product,
# since Sale is not yet defined.
from __future__ import annotations
from typing import List

# Forward reference used for class Sale
class Product:
    __lastSale: Sale = None
    __inventory: int = 0  # New attribute to track inventory

    def __init__(self, sale: Sale = None, inventory: int = 0):
        self.__lastSale = sale
        self.__inventory = inventory

    def setLastSale(self, lastSale: Sale):
        self.__lastSale = lastSale

    @property
    def getLastSale(self) -> Sale:
        return self.__lastSale

    @property
    def getInventory(self) -> int:
        return self.__inventory

    def decreaseInventory(self, amount: int):
        if self.__inventory >= amount:
            self.__inventory -= amount
        else:
            print("Not enough inventory")

# No forward reference needed since Product is defined
```

```
class Sale:
    __saleTimes = 0
    __productSold: List[Product] = None
    __saleNumber: int = 0

    def __init__(self, products: List[Product], quantities: List[int]):
        Sale.__saleTimes += 1
        self.__productSold = products
        self.__saleNumber = Sale.__saleTimes

        for index, product in enumerate(products):
            product.setLastSale(self)  # Set the sale for each product
            product.decreaseInventory(quantities[index])  # Decrease inventory for eac

    @property
    def getSaleNumber(self) -> int:
        return self.__saleNumber


# Example usage

productOne = Product(sale=None, inventory=100)
productTwo = Product(sale=None, inventory=50)

# Sale 1: Selling 5 units of productOne and 3 units of productTwo
saleOne = Sale([productOne, productTwo], [5, 3])

# Sale 2: Selling 2 units of productOne
saleTwo = Sale([productOne], [2])

print(f"ProductOne Inventory: {productOne.getInventory}, Last Sale Number: {productOne
print(f"ProductTwo Inventory: {productTwo.getInventory}, Last Sale Number: {productTwo
```

Figure 2.1: UML Class Diagram

Index