Homework PDF

by

Matthew Smith

msmith20@stevens.edu

September 19, 2024

msmith20@stevens.edu

Homework PDF

Matthew Smith
msmith20@stevens.edu

This document provides the requirements and design details of the PROJECT. The following table (Table 1) should be updated by authors whenever major changes are made to the architecture design or new components are added. Add updates to the top of the table. Most recent changes to the document should be seen first and the oldest last.

Table 1: Document Update History

Date	**Updates**
08/22/2023	DDM: • Updated dsnManual.tex with *newcommand(s)*{} for easier references of requirements, figures, and other labels.
10/25/2023	DDM: • Added chapters on use cases (Chapter **??**) and user stories (Chapter **??**).
10/11/2023	DDM: • Added chapters on requirements (Chapter **??**) and glossary.
09/18/2023	DDM: • Added chapter on development plan (Chapter **??**).
09/12/2024	DDM: • Added chapter 1.
09/18/2024	DDM: • Added chapter 2.

Table of Contents

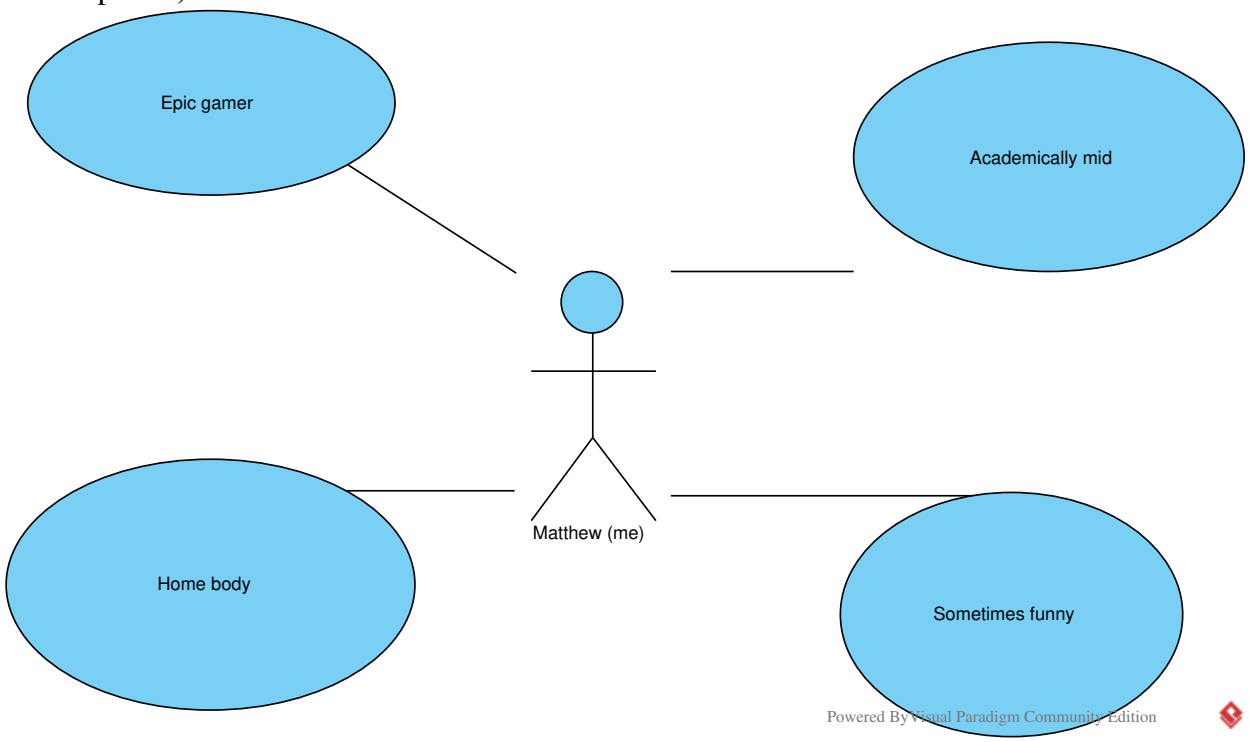
List of Tables

List of Figures

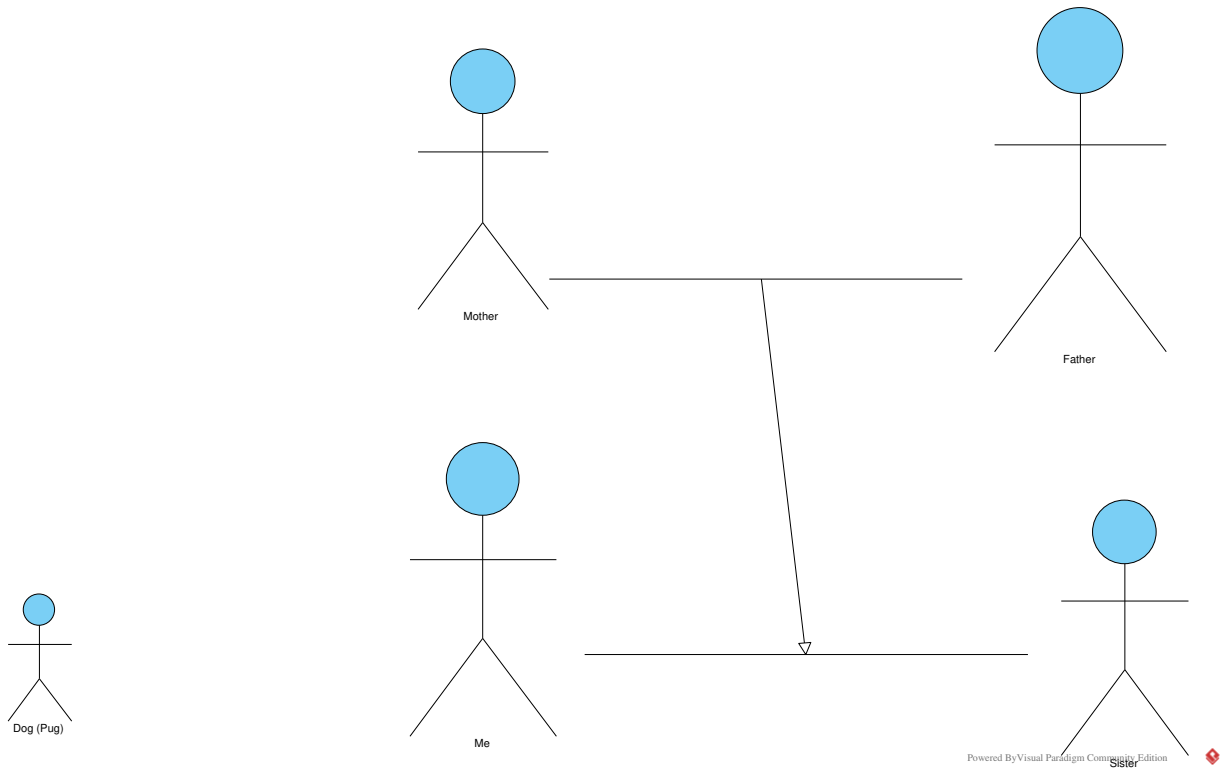
Chapter 1

Team

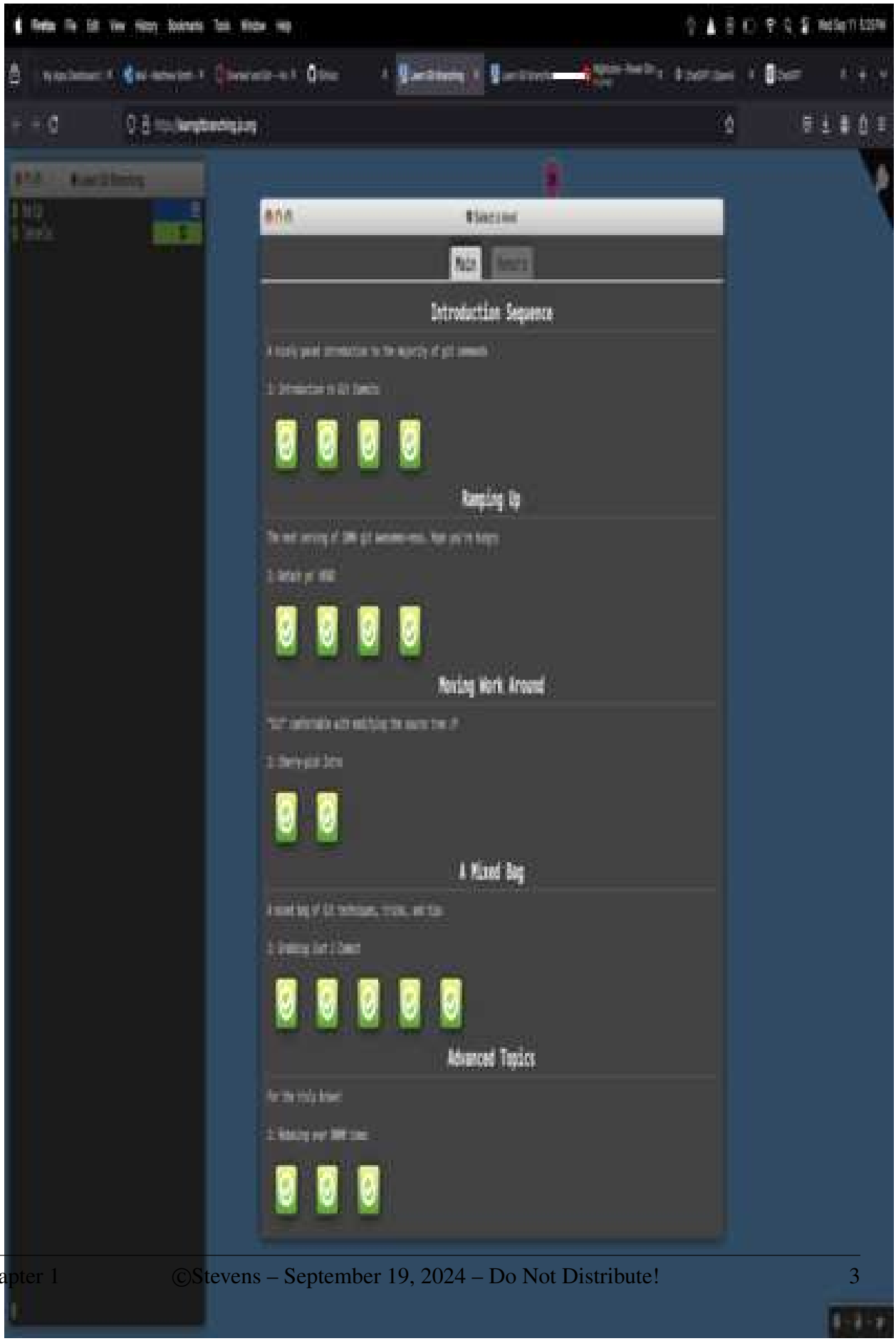
Hello, my name is Matthew Smith. I am a software engineer in the 3rd year, trying to become one in the future. I mostly am a homebody, but currently trying to become more outgoing this year. I have a pretty good background in programming as well as academics, and hope to increase my knowledge on this in the future. For task 3: https://github.com/JKDX4567/SSW-345-HW-MS (should be public)



Visual paradigm 1



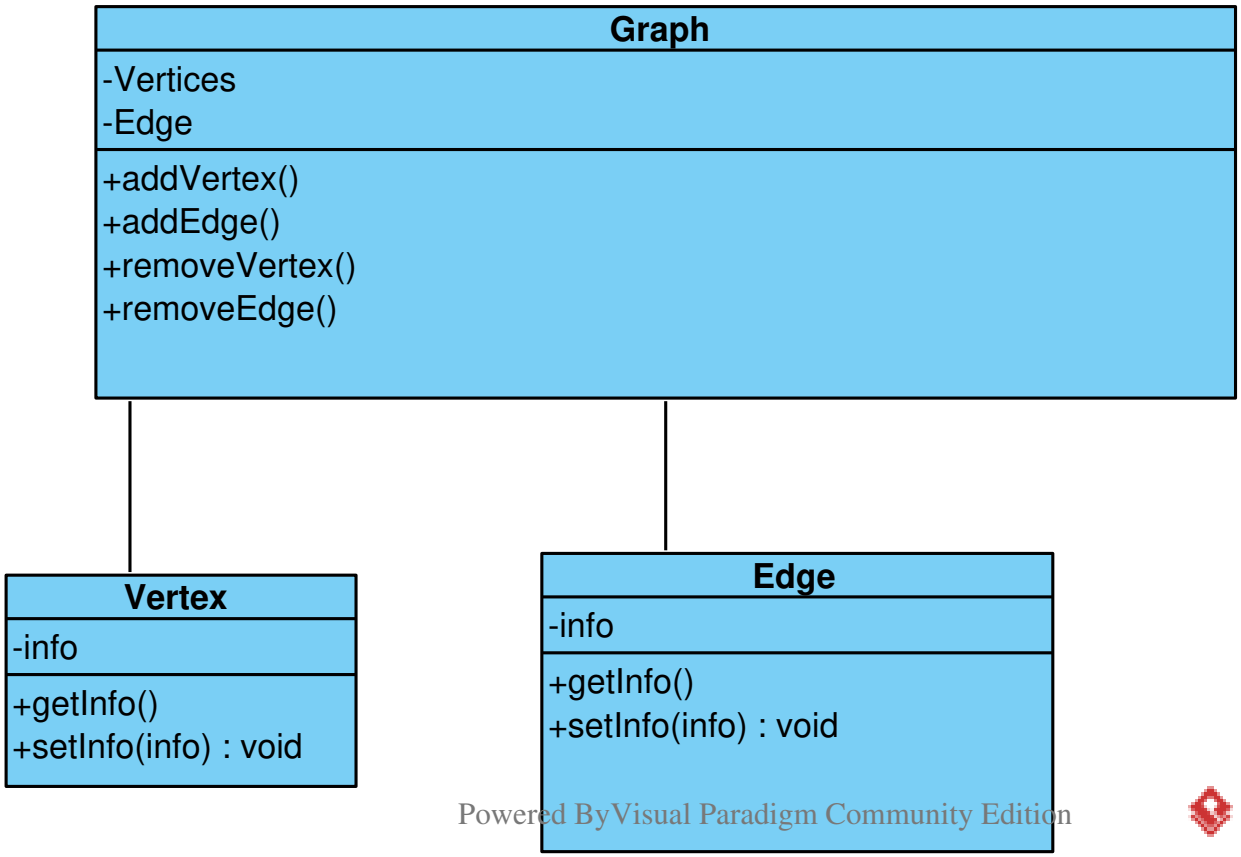
Visual paradigm 2


Introduction Sequence
Ramping Up
Moving Work Around
A Mixed Bag
Advanced Topics

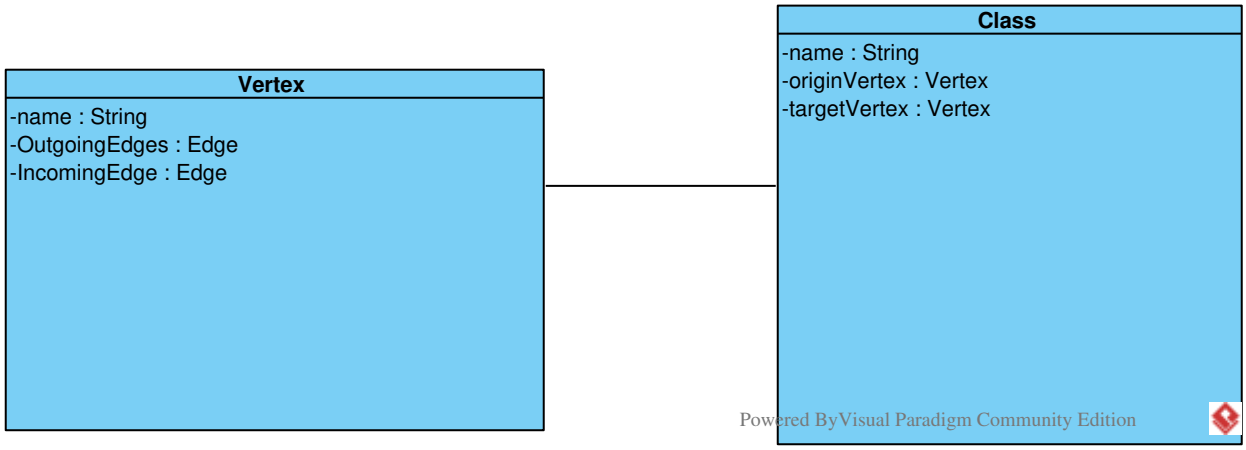
Github comments proof

Chapter 2

UML Class Modeling



2.1
Undirected UML



2.2 Directed UML

2.3 The following associations exist within the model: 1. Windows is a composition of everything, being directly above scrolling window, canvas, and panel 2. Scrolling Window has aggregation with Canvas, Text Window, and Scrolling canvas 3. Canvas has aggregation with Scrolling-Window and Scrolling Canvas, and has a direct relationship with shape, a canvas can have infinitly many shapes, and goes through it via elements, while a shape can have only 1 canvas, through window, and is an example of composition 4. Panel is tied to PanelItem, a panel can have 0-1 PanelItem's, while a PanelItem has 1 Panel 5. Line is the parent of Line and Closed Shape, which also has the subclasses Polygon and Elipse. 6. PanelItem has 3 sub classes being ChoiceItem, Butoon, and TextItem 7. Event can have infinitly many PanelItems, while a PanelItem can have only 1 event, and gets it via notifyEvent. Event also can have infinite TextItem, while they can only have one Event and acess it via Event. 8. ChoiceItem is directly related to ChoiceEntry, with two accesses, currentChoice, with 0-1 ChoiceItems for 1 ChoiceEntry, which is a subset of the choices acess, which a choice Item can have infinitly many of, while ChoiceEntry can have only 1 ChoiceItem. 9. Finally, Points are directly related to polygons, with a point being only on one shape, but a shape can have infinite points, called verticies.

2.4 1.MailingAddress is related to customer, with both having an infinite number of potential relations beetween them, and a MailingAddress sees the customer via accountHolder. MailingAddress can also have an infinite number of CreditCardAccounts, but they can only have 1 MailingAddress. 2. CreditCardAccounts can have 0-1. Statements with statementDate, with the reverse being only 1. It works the same with Insitutions, with accountNumber accessing CreditCardAccounts 3. Statements, via transactionNumber, can have 0-1 Transactions, and each transaction can only have 1 statement 4. Transaction has composition relationships with CashAdvance, Intrest, Purchases, Fees, and Adjustments. A Purchase is related to Merchant, with a Merchant having as many Purchases as necessary, but a purchase can only have 1 Merchant.

[CODE FOR PART TWO OF HW] needed for forward reference of Sale in Product, since Sale is not yet defined. from $_{future_{importannotationsfromtypingimportList}}$

forward reference used for class Sale class Product: $_{lastSale:Sale=Noneinventory=0}$

def $_{init_{(self,sale:Sale,inven):self._{lastSale=saleself.inventory=inven}}}$

def setLastSale(self, lastSale: Sale): self.$_{lastSale=lastSale}$

@property def getLastSale(self) -¿ Sale: return self.$_{lastSale}$

def $_{getitem_{(self,item):returnself}}$

def getInventory(self) -¿ int: return self.inventory
def gotInventory(self, count) -¿ int: self.inventory = self.inventory + count
no forward reference needed since Product is defined class Sale: $_{saleTimes=0}{}_{productSold:List[Product]=None}{}_{saleNumber:int=0}$
def $_{init}{}_{(self,product:List[Product]):,saleNumber:int=1):Sale.}{}_{saleTimes+=1self.}{}_{product=productself.}{}_{saleNumber=Sale.}{}_{saleTimesforindex,productinenumerate(product):product[index].setLastSale(self)p}$
def setProductsSold(self, productSold: List[Product]): self.$_{productSold=productSold}$
@property def getSaleNumber(self) -¿ int: return self.$_{saleNumber}$
productOne = Product(sale=None, inven=1) productTwo = Product(sale=None, inven=25) print (”Product one has: ” + str(productOne.getInventory()) + ” units”) print (”Product two has: ” + str(productTwo.getInventory()) + ” units”)
saleOne = Sale([productOne, productTwo]) saleTwo = Sale([productOne]) saleThree = Sale([productTwo]) print (”Product one has 2 sales”) print (”Product two has 2 sales”)
print (”Product one has: ” + str(productOne.getInventory()) + ” units”) print (”Product two has: ” + str(productTwo.getInventory()) + ” units”) productOne.gotInventory(10) print (”Product One has added 10 units”) print (”Product one has: ” + str(productOne.getInventory()) + ” units”) print(f”productOne.getLastSale.getSaleNumber, productTwo.getLastSale.getSaleNumber”)

Updated UML

Bibliography