Project Title

by

Author(s) Name

Stevens.edu

September 20, 2024

Project Title

Author(s) Name
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 **??**).

Table of Contents

List of Tables

List of Figures

Chapter 1

Team

– Enoch Chan

My name is Enoch. I am a software engineer major at Stevens Institute of Technology. I was born and raised in Staten Island, New York. I like to play volleyball and go to the gym. I am currently working at the Front Desk of Fitness Factory on 2nd Street. Below are my .EPS figures:

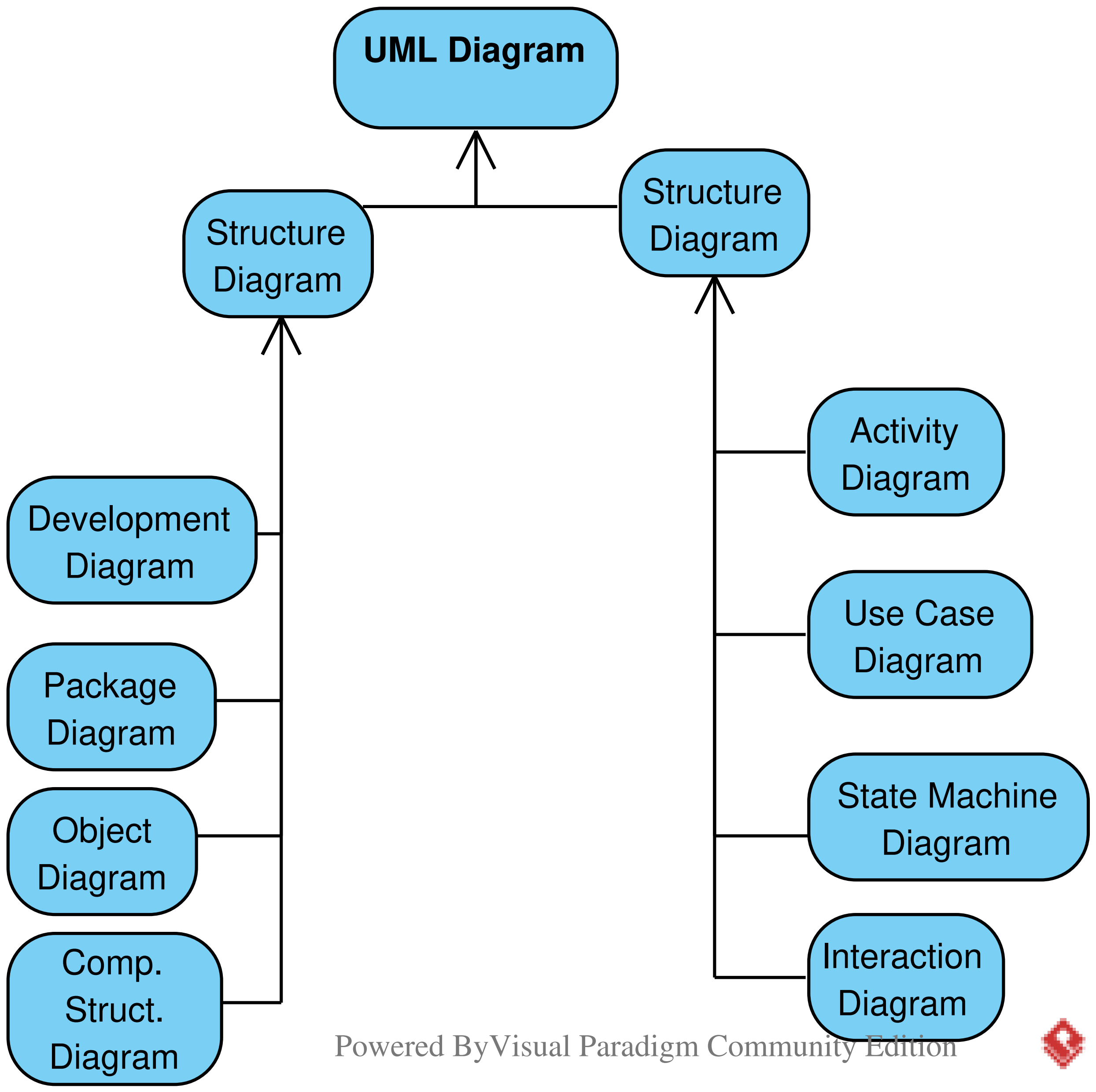


Figure 1.1: Diagram 1

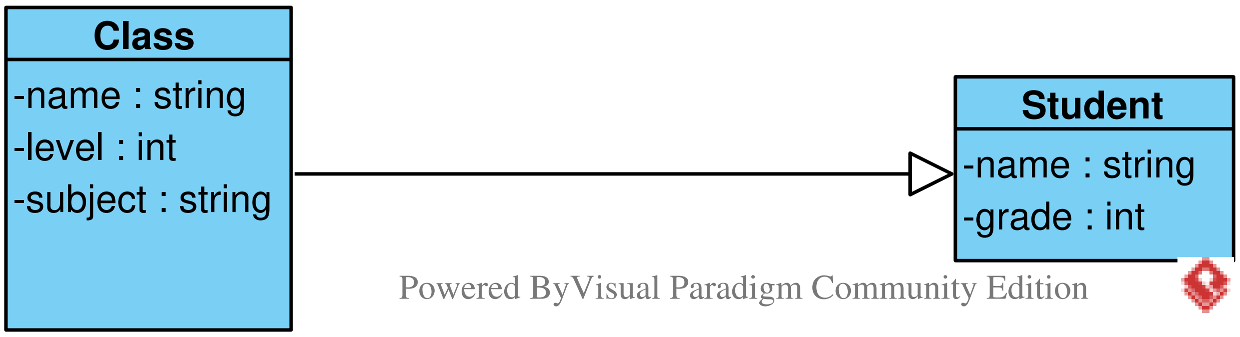
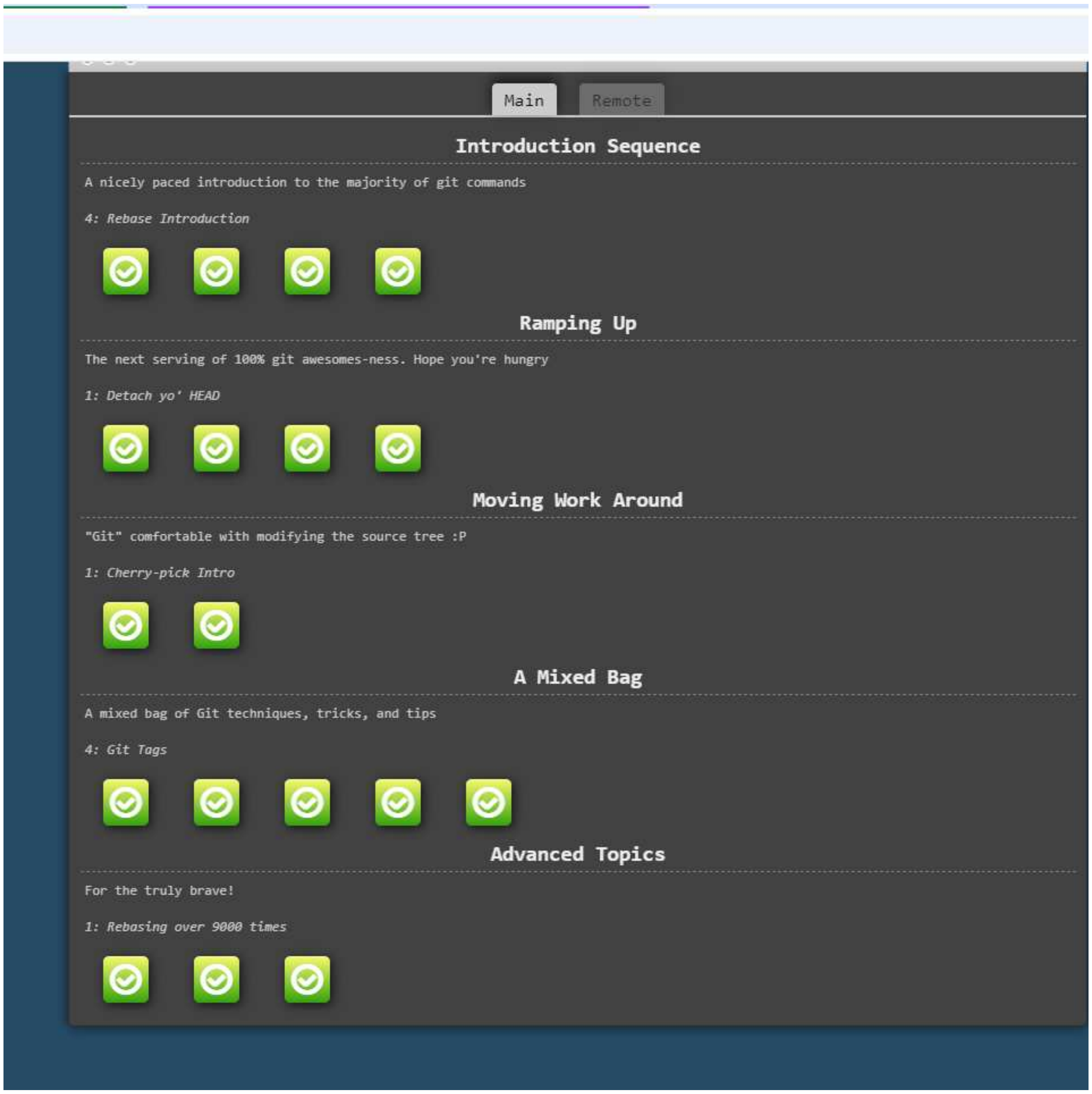


Figure 1.2: Diagram 2

Chapter 2

gitHomework

– Enoch



Introduction Sequence

A nicely paced introduction to the majority of git commands

4: Rebase Introduction

Ramping Up

The next serving of 100% git awesomes-ness. Hope you're hungry

1: Detach yo' HEAD

Moving Work Around

"Git" comfortable with modifying the source tree :P

1: Cherry-pick Intro

A Mixed Bag

A mixed bag of Git techniques, tricks, and tips

4: Git Tags

Advanced Topics

For the truly brave!

1: Rebasing over 9000 times

Figure 2.1: git Homework Screenshot

Chapter 3

UML Class Modeling

– Enoch

3.1 2.1 Exercise

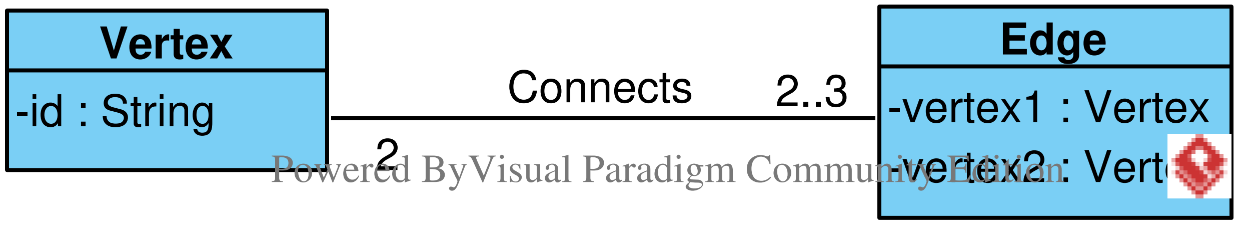


Figure 3.1: Diagram 1

3.2 2.2 Exercise

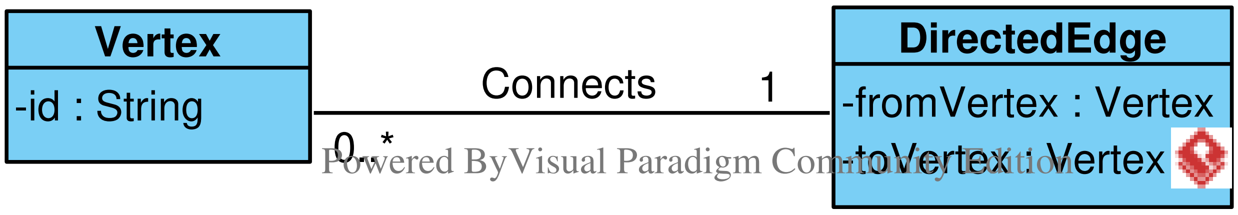


Figure 3.2: Diagram 1

3.3 2.3 Exercise

Window → ScrollingWindow
Association: A ScrollingWindow is a special kind of Window.

Description: The ScrollingWindow allows for scrolling within the Window area.

Canvas → Shape
Association: A Canvas can have many shapes.
Description: The Canvas holds different shapes that can be drawn, like lines or closed shapes.

Canvas → ScrollingCanvas
Association: A ScrollingCanvas is a type of Canvas.
Description: The ScrollingCanvas can scroll to show more of the drawing area.

Canvas → Line
Association: A Canvas can have many lines.
Description: A Line is drawn on the Canvas and is defined by its start and end points.

Shape → ClosedShape
Association: A ClosedShape is a type of Shape.
Description: A ClosedShape is a shape that forms a complete boundary, like a circle or polygon.

Shape → Line
Association: A Line is a type of Shape.
Description: A Line is a straight shape with a start and end point.

ClosedShape → Polygon
Association: A ClosedShape can be a Polygon.
Description: A Polygon is a shape with multiple sides.

ClosedShape → Ellipse
Association: A ClosedShape can be an Ellipse.
Description: An Ellipse is an oval shape.

Canvas → Panel
Association: A Canvas can be part of a Panel.
Description: The Panel can hold a Canvas and other interface elements.

Panel → PanelItem
Association: A Panel has many PanelItems.
Description: PanelItems are objects within the Panel, such as buttons or text fields.

Panel → Button
Association: A Panel can have many Buttons.
Description: A Button is a clickable item in the Panel.

Panel → TextItem
Association: A Panel can have TextItems.
Description: A TextItem displays text or allows the user to input text.

Panel → ChoiceItem
Association: A Panel can have ChoiceItems.
Description: A ChoiceItem allows the user to choose from a list of options.

ChoiceItem → ChoiceEntry
Association: A ChoiceItem has multiple ChoiceEntries.
Description: A ChoiceEntry is one of the options in a ChoiceItem.

PanelItem → Event
Association: A PanelItem can trigger an Event.
Description: When a PanelItem (like a button) is clicked, it triggers an Event.

ScrollingWindow → Scroll
Association: A ScrollingWindow allows scrolling.
Description: The scroll function moves the visible area of the ScrollingWindow.

TextWindow → Text
Association: A TextWindow manages text.
Description: The TextWindow allows text to be inserted or deleted.

3.4 2.4 Exercise

Customer → MailingAddress
Association: A Customer has one or more MailingAddresses.
Description: The MailingAddress is associated with the Customer and holds information like address and phone number for communication purposes.

MailingAddress → CreditCardAccount
Association: A CreditCardAccount can have one or more MailingAddresses.
Description: The MailingAddress is linked to the CreditCardAccount for billing and correspondence.

CreditCardAccount → Statement
Association: A CreditCardAccount can generate multiple Statements.
Description: The Statement shows important details like the payment due date, finance charges, and minimum payment, summarizing the account's activity.

CreditCardAccount → Institution
Association: A CreditCardAccount is associated with one Institution.
Description: The Institution is the financial entity responsible for the CreditCardAccount, holding details like name, address, and phone number.

CreditCardAccount → Transaction
Association: A CreditCardAccount can have multiple Transactions.
Description: Each Transaction represents an action on the CreditCardAccount, such as purchases, cash advances, fees, or adjustments.

Statement → Transaction
Association: A Statement includes multiple Transactions.
Description: Each Transaction is part of a Statement, showing detailed records of charges, payments, or fees on the account.

Transaction → Purchase
Association: A Transaction can be a Purchase.
Description: A Purchase is a type of Transaction where goods or services are bought from a Merchant.

Transaction → Fee
Association: A Transaction can be a Fee.
Description: A Fee is a type of Transaction that includes extra charges like late fees or service fees.

Transaction → Interest
Association: A Transaction can include Interest.
Description: Interest is a finance charge added as a result of carrying a balance on the CreditCardAccount.

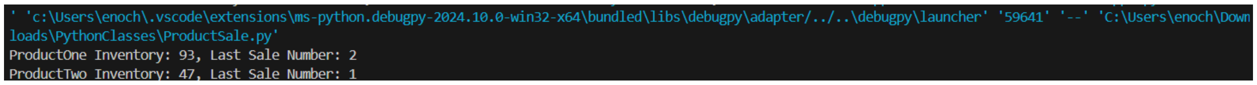
Transaction → CashAdvance
Association: A Transaction can be a CashAdvance.
Description: A CashAdvance is a type of Transaction where cash is withdrawn from the credit card account.

Transaction → Adjustment
Association: A Transaction can be an Adjustment.
Description: An Adjustment modifies or corrects a previous transaction, either as a credit or debit to the account.

Purchase → Merchant
Association: A Purchase is linked to a Merchant.
Description: The Merchant is the business or entity from which a Purchase is made, identified by name.

3.5 ProductSale

```python
from __future__ import annotations
from typing import List

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

    def __init__(self, inventory: int = 0):
        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:
            raise ValueError(f"Not enough inventory to sell {amount} units. Current inventory: {self.__inventory}")

# No forward reference needed since Product is already defined
class Sale:
    __saleCounter = 0  # Static variable to track number of sales
    __productSold: List[Product] = None

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

        for index, product in enumerate(products):
            product.decreaseInventory(quantities[index])
            product.setLastSale(self)

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

# Example usage

productOne = Product(inventory=100)
productTwo = Product(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.getLastSale.getSaleNumber}")
print(f"ProductTwo Inventory: {productTwo.getInventory}, Last Sale Number: {productTwo.getLastSale.getSaleNumber}")
```



```
' 'c:\Users\enoch\.vscode\extensions\ms-python.debugpy-2024.10.0-win32-x64\bundled\libs\debugpy\adapter/../..\debugpy\launcher' '59641' '--' 'C:\Users\enoch\Downloads\PythonClasses\ProductSale.py'
ProductOne Inventory: 93, Last Sale Number: 2
ProductTwo Inventory: 47, Last Sale Number: 1
```

Figure 3.3: Results Screenshot

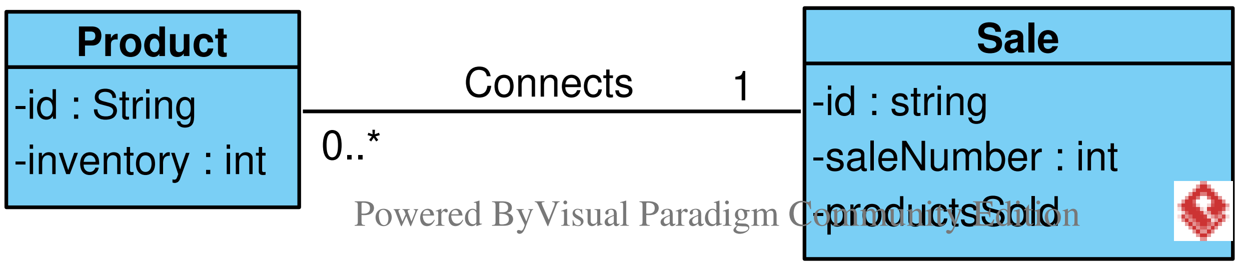


Figure 3.4: UML

Bibliography

Index