Homework Manual

by

Hazem Abo-Donia

Stevens.edu

September 20, 2024

© Hazem Abo-Donia
Stevens.edu
ALL RIGHTS RESERVED

Homework Manual

Hazem Abo-Donia
Stevens.edu

Table 1: Document Update History

Date	**Updates**
09/12/2024	Initial: • Created inital Homework Manual version and incorporated sswTeam.tex and sswGitHomework.tex files.
09/19/2024	Homework: • Completed Homework Two and incorporated sswHomeworkTwo.tex.

Table of Contents

Chapter 1

Team

Hazem Abo-Donia

1.1 A Little About Me

I'm Hazem Abo-Donia, a software engineering student at Stevens Institute of Technology. I grew up in Brooklyn, and my family background is Egpytian. I've always been fascinated by computers, which led me to pursue projects ranging from developing sensory toys to improving data transfer systems. I've also gained experience as an intern at Regeneron and serve as Vice President of the INCOSE chapter at Stevens. When I'm not working on tech, I enjoy spending time with my cat and dog and exploring ways to solve real-world problems through engineering.

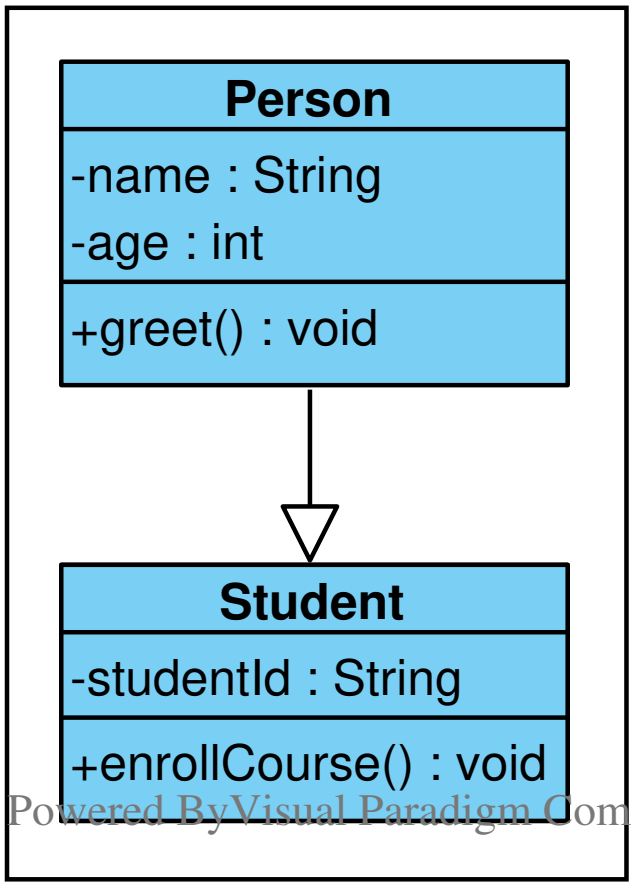

Figure 1.1: Sample Diagram 1 in Visual Paradigm

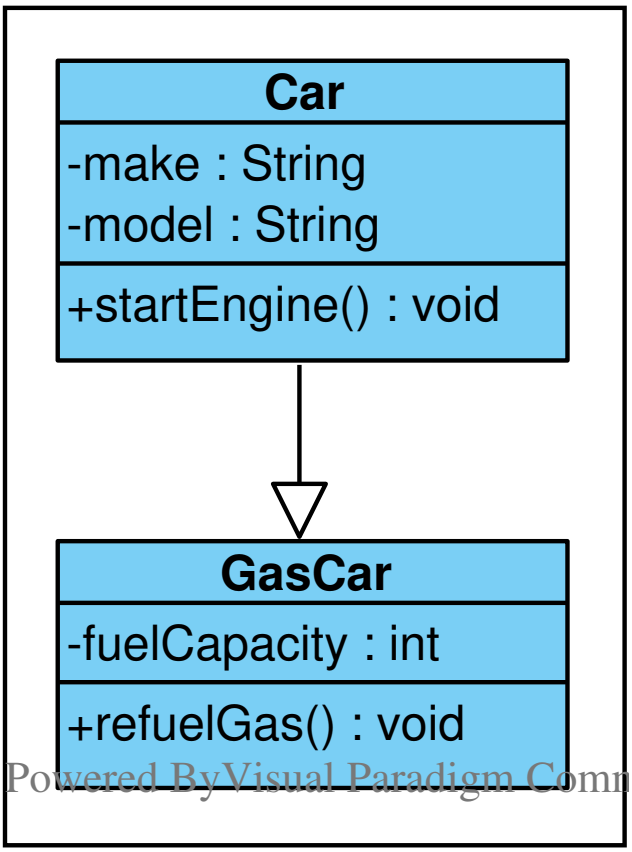

Figure 1.2: Sample Diagram 2 in Visual Paradigm

Chapter 2

Homework One

Hazem Abo-Donia

2.1 Learning Git

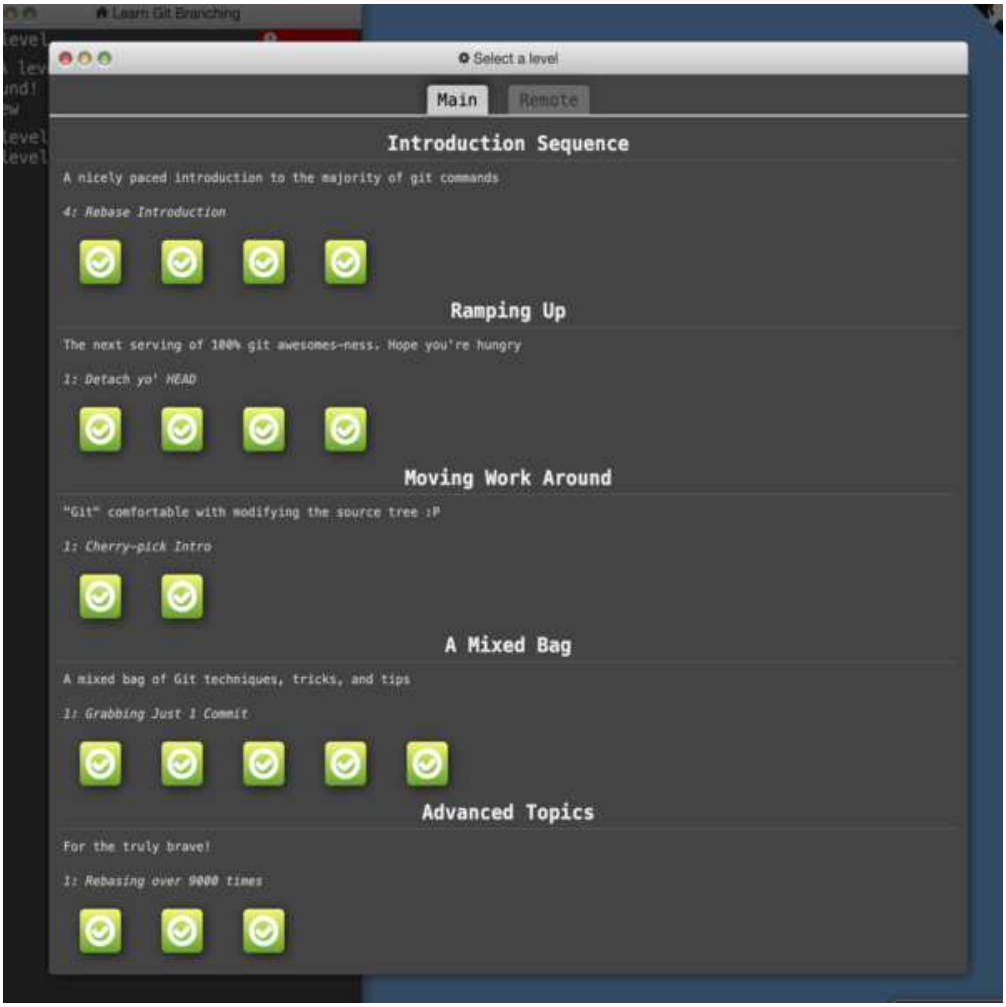

Figure 2.1: Learning Git Task Completion Proof

Chapter 3

UML Class Modeling

Hazem Abo-Donia

3.1 Class Modeling Exercise 2.1

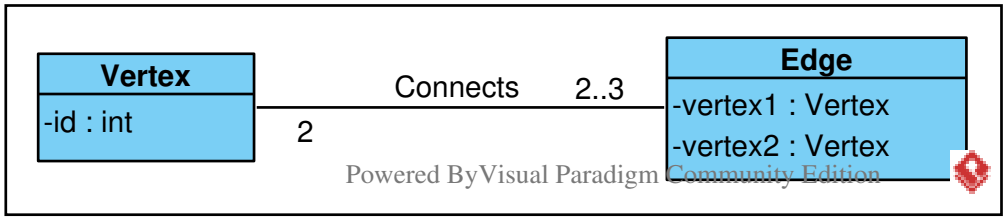

Figure 3.1: Undirected Graph UML

3.2 Class Modeling Exercise 2.2

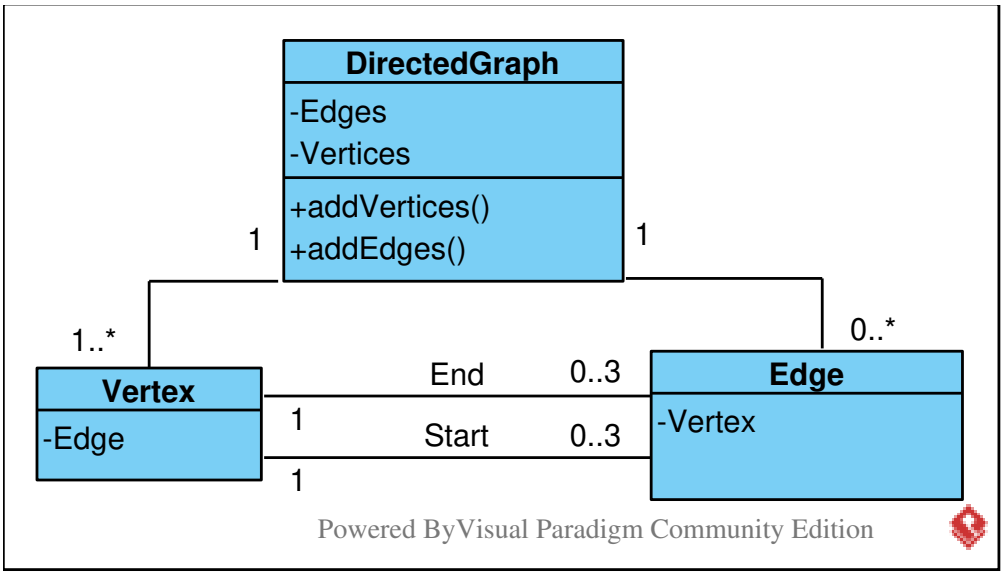

Figure 3.2: Directed Graph UML

3.3 Class Modeling Exercise 2.3

1. **Window:**
   - *Inheritance:* Window is the parent class of ScrollingWindow, Canvas, and Panel. Each of these subclasses inherits attributes (x1, y1, x2, y2) and methods (display, undisplay, raise, lower).
   - *Association:* No explicit associations or multiplicity are defined here.
2. **ScrollingWindow:**
   - *Inheritance:* Inherits from Window, gaining the properties and methods of Window.
   - *Association:* No direct associations are shown in the diagram for ScrollingWindow aside from the inheritance from Window.

3. **Canvas:**

   - *Inheritance:* Inherits from Window.
   - *Association with Shape:* There is an association between Canvas and Shape, with a multiplicity of *, indicating that one Canvas can contain multiple Shape objects.
   - *Multiplicity:* The relationship is one-to-many (1 Canvas to * Shapes).

4. **Shape:**

   - *Inheritance:* Serves as a parent class for Line and ClosedShape. These subclasses inherit the color and lineWidth attributes from Shape.
   - *Association:* Shape is associated with Canvas (as described above).

5. **Line:**

   - *Inheritance:* Inherits from Shape, gaining attributes like color and lineWidth.
   - *No Additional Associations:* No further relationships are explicitly shown for Line.

6. **ClosedShape:**

   - *Inheritance:* Inherits from Shape.
   - *Association with Polygon and Ellipse:* ClosedShape is the parent of Polygon and Ellipse, which inherit properties such as fillColor and fillPattern.

7. **Polygon:**

   - *Inheritance:* Inherits from ClosedShape.
   - *Association with Point:* Polygon has an association with Point, with the multiplicity * indicating that a Polygon can have multiple Point objects, which likely define its vertices.

- *Multiplicity:* One Polygon is associated with multiple Point objects.

8. **Ellipse:**

   - *Inheritance:* Inherits from ClosedShape.
   - *No Additional Associations:* No further relationships are explicitly shown for Ellipse.

9. **Panel:**

   - *Inheritance:* Inherits from Window.
   - *Association with PanelItem:* There is an association between Panel and PanelItem, with a multiplicity of 0..1. This means that a Panel can contain zero or one PanelItem.
   - *Multiplicity:* One-to-zero or one (Panel to PanelItem).

10. **PanelItem:**

    - *Association with Event:* There is an association between PanelItem and Event, indicating that each PanelItem can be associated with exactly one Event, and each Event can have multiple PanelItems.

11. **Event:**

    - *Association with PanelItem:* As described above, an Event is associated with multiple PanelItem instances.
    - *Association with TextItem:* Event can be associated with multiple TextItems, and TextItem can be associated with 1 event.

12. **Button:**

    - *Inheritance:* Inherits from PanelItem, gaining attributes such as x, y, and label.
    - *No Additional Associations:* No further relationships are explicitly shown for Button.

13. **ChoiceItem:**

    - *Inheritance:* Inherits from PanelItem.
    - *Association with ChoiceEntry:* There is an association between ChoiceItem and ChoiceEntry, with the multiplicity *, indicating that one ChoiceItem can be associated with multiple ChoiceEntry objects (the possible choices within the item).
    - *Multiplicity:* One-to-many relationship between ChoiceItem and ChoiceEntry.

14. **ChoiceEntry:**

    - *Association with ChoiceItem:* As described above, a ChoiceEntry is part of a ChoiceItem and represents an individual choice.
    - *No Additional Associations:* No other relationships are explicitly defined for ChoiceEntry.

15. **TextItem:**

    - *Inheritance:* Inherits from PanelItem, gaining attributes such as x, y, and label.
    - *No Additional Associations:* No other relationships are explicitly defined for TextItem.

16. **Point:**

    - *Association with Polygon:* Point is associated with Polygon as described earlier. A Polygon is composed of multiple Point objects, representing the vertices of the polygon.
    - *Multiplicity:* One-to-many relationship between Polygon and Point.

3.4 Class Modeling Exercise 2.4

1. **MailingAddress:**

- *Association with CreditCardAccount:* One MailingAddress can be linked to multiple (*) CreditCardAccount objects, representing the address used for various accounts.

2. **CreditCardAccount:**

   - *Association with MailingAddress:* A CreditCardAccount is connected to exactly one MailingAddress.
   - *Association with Statement:* A CreditCardAccount can have zero or one (0..1) Statement generated.
   - *Association with Institution:* A CreditCardAccount is associated with exactly one Institution.

3. **Institution:**

   - *Association with CreditCardAccount:* Each Institution can have (0..1) CreditCardAccount objects under its management.

4. **Customer:**

   - *Association with MailingAddress:* A Customer can have multiple (*) MailingAddress objects, representing different addresses associated with the customer.

5. **Statement:**

   - *Association with CreditCardAccount:* Each Statement belongs to exactly one CreditCardAccount.
   - *Association with Transaction:* A Statement can contain (0..1) Transaction entries.

6. **Transaction:**

   - *Inheritance:* Transaction is a parent class for CashAdvance, Interest, Purchase, Fee, and Adjustment.

- *Association with Statement:* Each Transaction belongs to exactly one Statement.

7. **CashAdvance:**

   - *Inheritance:* Inherits from Transaction, representing a cash advance transaction.

8. **Interest:**

   - *Inheritance:* Inherits from Transaction, representing an interest-related transaction.

9. **Purchase:**

   - *Inheritance:* Inherits from Transaction.
   - *Association with Merchant:* A Purchase is associated with one Merchant.

10. **Fee:**

    - *Inheritance:* Inherits from Transaction, representing a fee-related transaction.

11. **Adjustment:**

    - *Inheritance:* Inherits from Transaction, representing an adjustment made to the account.

12. **Merchant:**

    - *Association with Purchase:* A Merchant can have multiple (*) Purchase transactions associated with it.

3.5 Code to Modeling Exercise

```python
# 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

    def __init__(self, sale: Sale, inventory: int):
        self.__lastSale = sale
        self.__inventory = inventory

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

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

    def updateInventory(self, quantitySold: int):
        if self.__inventory >= quantitySold:
            self.__inventory -= quantitySold
            print(f"{quantitySold} units sold, {self.__inventory} remaining.")
        else:
            print("Not enough inventory.")

    @property
    def getInventory(self) -> int:
```

```
31          return self.__inventory
32
33      def __getitem__(self, item):
34          return self
35
36
37  # no forward reference needed since Product is defined
38  class Sale:
39      __saleTimes = 0
40      __productSold: List[Product] = None
41      __saleNumber: int = 0
42
43      def __init__(self, products: List[Product]):
44          Sale.__saleTimes += 1
45          self.__product = products
46          self.__saleNumber = Sale.__saleTimes
47          for product in products:
48              product.setLastSale(self)
49              product.updateInventory(1)
50
51      @property
52      def getSaleNumber(self) -> int:
53          return self.__saleNumber
54
55
56  # Creating products with initial inventory
57  productOne = Product(sale=None, inventory=10)
58  productTwo = Product(sale=None, inventory=5)
59
60  # Making sales
```

```
saleOne = Sale([productOne, productTwo])  # Sale involves productOne and
    productTwo
saleTwo = Sale([productOne])  # Another sale involving productOne
saleThree = Sale([productTwo])  # Sale involving productTwo

print(f"Last sale number for Product 1: {productOne.getLastSale.getSaleNumber}
    ")
print(f"Last sale number for Product 2: {productTwo.getLastSale.getSaleNumber}
    ")
```

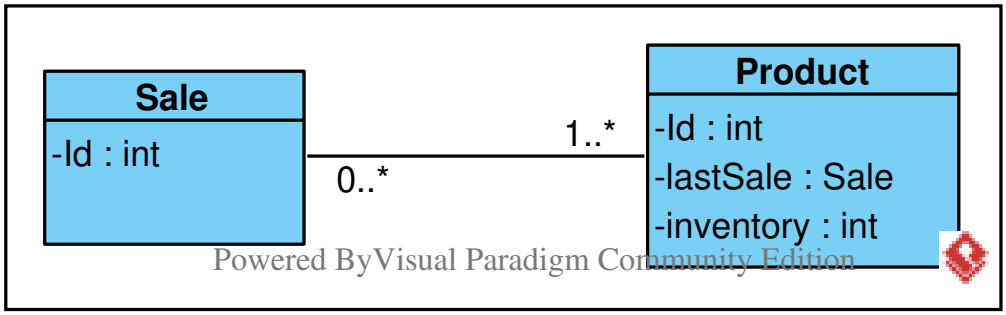

Figure 3.3: Updated Product Sale UML

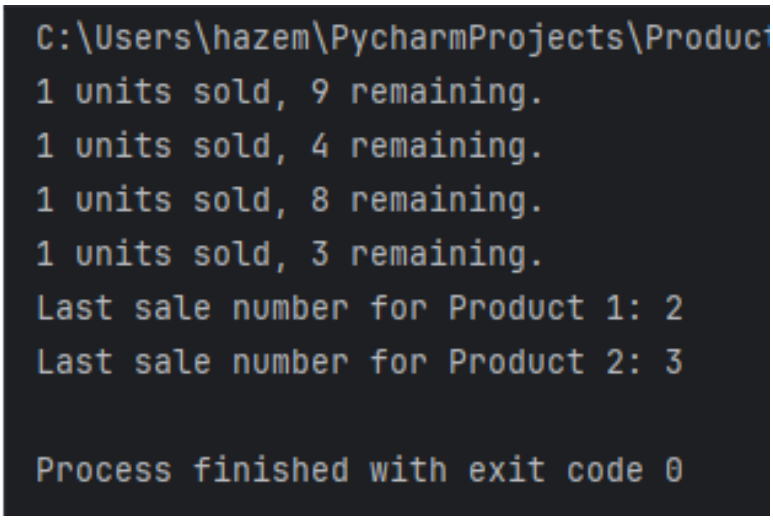

Figure 3.4: Code Output