Lab Manual

by

Kyle Ponte, Enoch Chan

Stevens.edu

October 3, 2024

Lab Manual

Kyle Ponte, Enoch Chan
Stevens.edu

Table 1: Document Update History

Date	**Updates**
10/2/2024	DDM: • Created lab manual and created a new chapter called Advanced Class Modeling Lab (Chapter 1).

Table of Contents

List of Tables

List of Figures

Chapter 1

Advanced Class Modeling Lab

– Kyle Ponte, Enoch Chan

1.1 Section 3.1 Exercise

First, a buffer represents an area where lines and boxes are temporarily stored when cut or copied. Next, a selection represents a section of lines and boxes that have been highlighted by the user. Moreover, a sheet contains lines and boxes. Additionally, a line is symbolic of a graphical line that is a part of the sheet and also connects two boxes, while a box is a square graphical element on the sheet. A line segment is a part of a line, and is defined by two points. Lastly, a point shows the endpoints and other important locations in the diagram where lines or line segments meet.

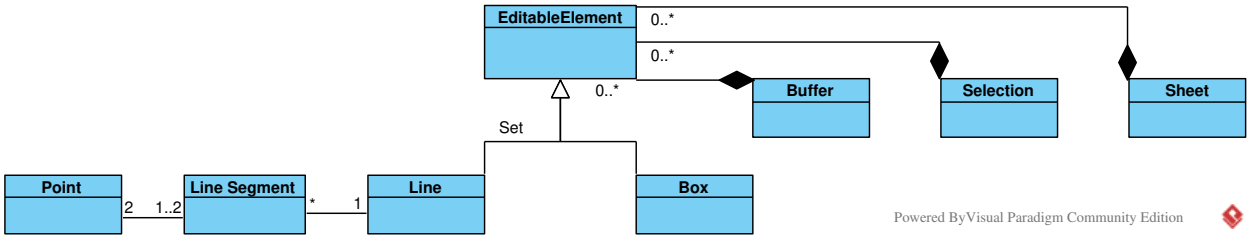


Figure 1.1: Revised Diagram of Interactive Editor

The revised design uses composition to establish clear ownership of EditableElement objects (which include Line and Box). The use of composition ensures that a Line or Box can only belong to one Buffer, Selection, or Sheet at a time, which prevents issues like shared or conflicting ownership of elements. Introducing the EditableElement superclass provides a flexible and scalable structure by applying the generalization concept. This allows common attributes and methods for Line and Box to be defined once in EditableElement, reducing redundancy and improving code reusability. Additionally, managing the association between EditableElements and their respective containers (Buffer, Selection, Sheet) becomes easier due to the unified handling of these objects through a single superclass. The revision enforces the constraint that a Line or Box can belong to exactly one Buffer, Selection, or Sheet through the diagram's structure. This eliminates the possibility of a Line or Box being in more than one place simultaneously, reducing the risk of data integrity issues and simplifying validation.

1.2 Section 3.2 Exercise

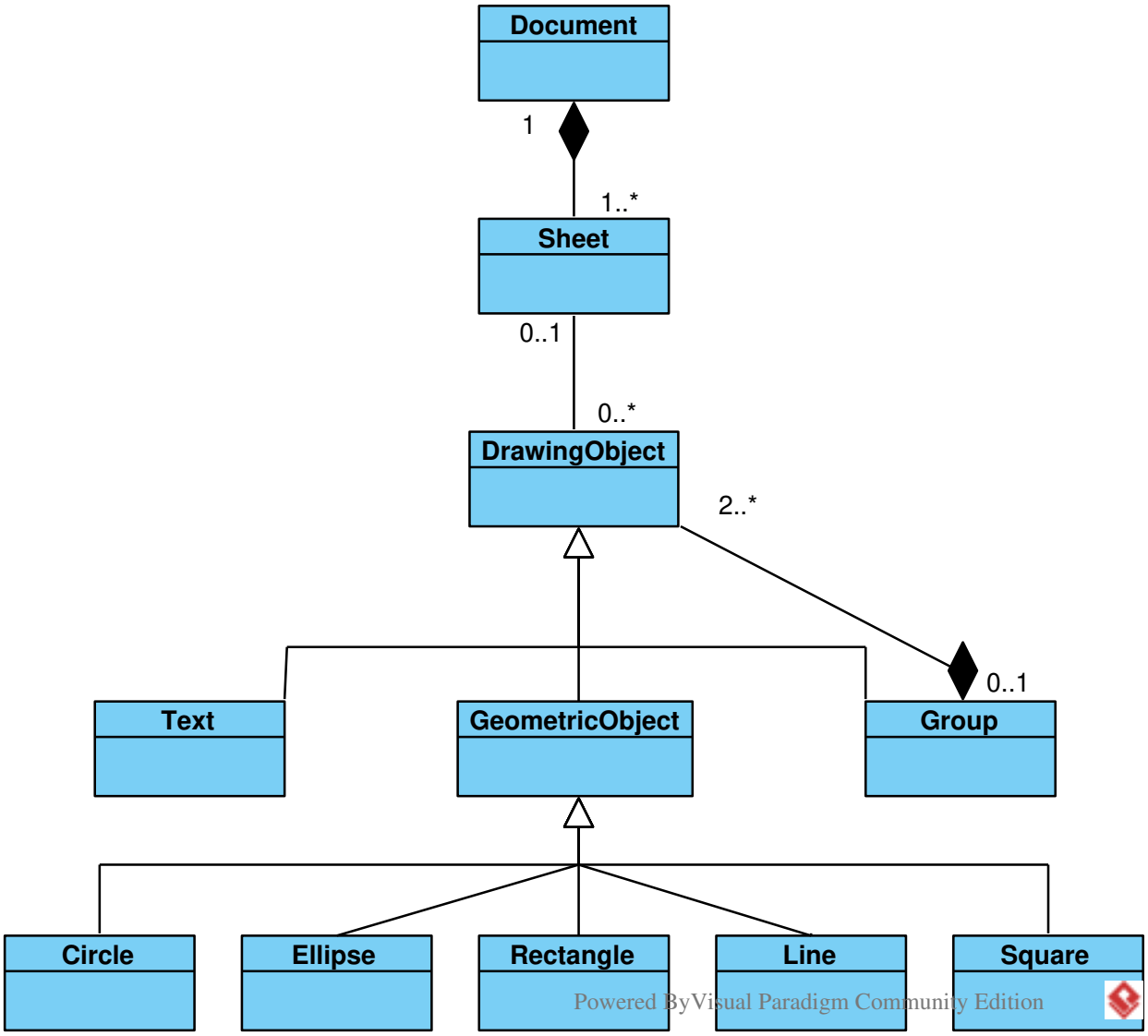


Figure 1.2: Diagram for Graphical Document Editor

1.3 Section 3.3 Exercise

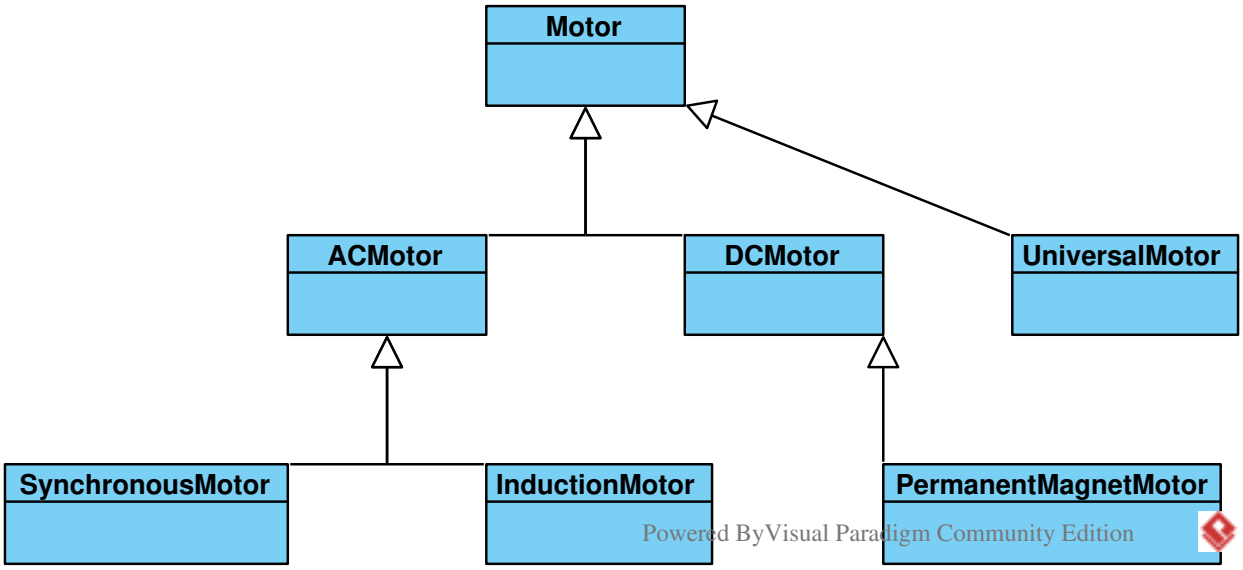


Figure 1.3: Motor Diagram

I did not use multiple inheritance in the initial diagram. This is because ACMotor, DCMotor, and UniversalMotor all inherit from Motor independently. Furthermore, UniversalMotor inherits directly from Motor, not from both ACMotor and DCMotor. This avoids multiple inheritance because UniversalMotor is a standalone subclass of Motor, capable of running on both AC and DC, without needing to inherit from both ACMotor and DCMotor.

1.4 Section 3.4 Exercise

```
1  from abc import ABC
2
3  class SM(ABC):
4      state = 0
5      startState = 0
6
7      def start(self):
8          self.state = self.startState
9
10     # step returns the next output.
11     def step(self, inp):
12         (s, o) = self.getNextValues(self.state, inp)
13         self.state = s
14         return o
15
16     def transduce(self, inputs):
17         self.start()
```

```
        return [self.step(inp) for inp in inputs]

    def run(self, n=10):
        return self.transduce([None] * n)

    def getNextValues(self, state, inp):
        nextState = self.getNextState(state, inp)
        return (nextState, nextState)

    def getNextState(self, state, inp):
        pass


class VendingMachine(SM):
    startState = ('waiting', 0)  # The machine starts waiting with 0 money inserted
    drink_price = 75  # Price of the drink in cents

    def getNextValues(self, state, inp):
        current_state, total_money = state
        if inp == 'cancel':
            return (('canceled', 0), f"Transaction canceled. Returning ${total_money / 100:.2f}.")
        elif inp in [5, 10, 25, 100]:  # Valid inputs: nickel, dime, quarter, dollar
            total_money += inp
            if total_money >= self.drink_price:
                change = total_money - self.drink_price
                return (('dispensing', 0), f"Drink dispensed. Returning ${change / 100:.2f} in change.")
            else:
                return (('waiting', total_money), f"Amount entered: ${total_money / 100:.2f}. Insert more money.")
        else:
            return (state, "Invalid input.")

# --- Test Scenarios ---

# Scenario 1: User inputs three quarters
vending_machine = VendingMachine()
scenario1 = [25, 25, 25]  # Three quarters
print(vending_machine.transduce(scenario1))

# Scenario 2: User inputs one quarter and cancels the transaction
vending_machine = VendingMachine()  # Reset the machine
scenario2 = [25, 'cancel']  # One quarter, then cancel
print(vending_machine.transduce(scenario2))

# Scenario 3: User inputs a dime and a dollar bill
vending_machine = VendingMachine()  # Reset the machine
scenario3 = [10, 100]  # A dime, then a dollar bill
print(vending_machine.transduce(scenario3))
```

```
['Amount entered: $0.25. Insert more money.', 'Amount entered: $0.50. Insert more money.', 'Drink dispensed. Returning
$0.00 in change.']
['Amount entered: $0.25. Insert more money.', 'Transaction canceled. Returning $0.25.']
['Amount entered: $0.10. Insert more money.', 'Drink dispensed. Returning $0.35 in change.']


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Figure 1.4: Updated Code Output for Exercise 3.4

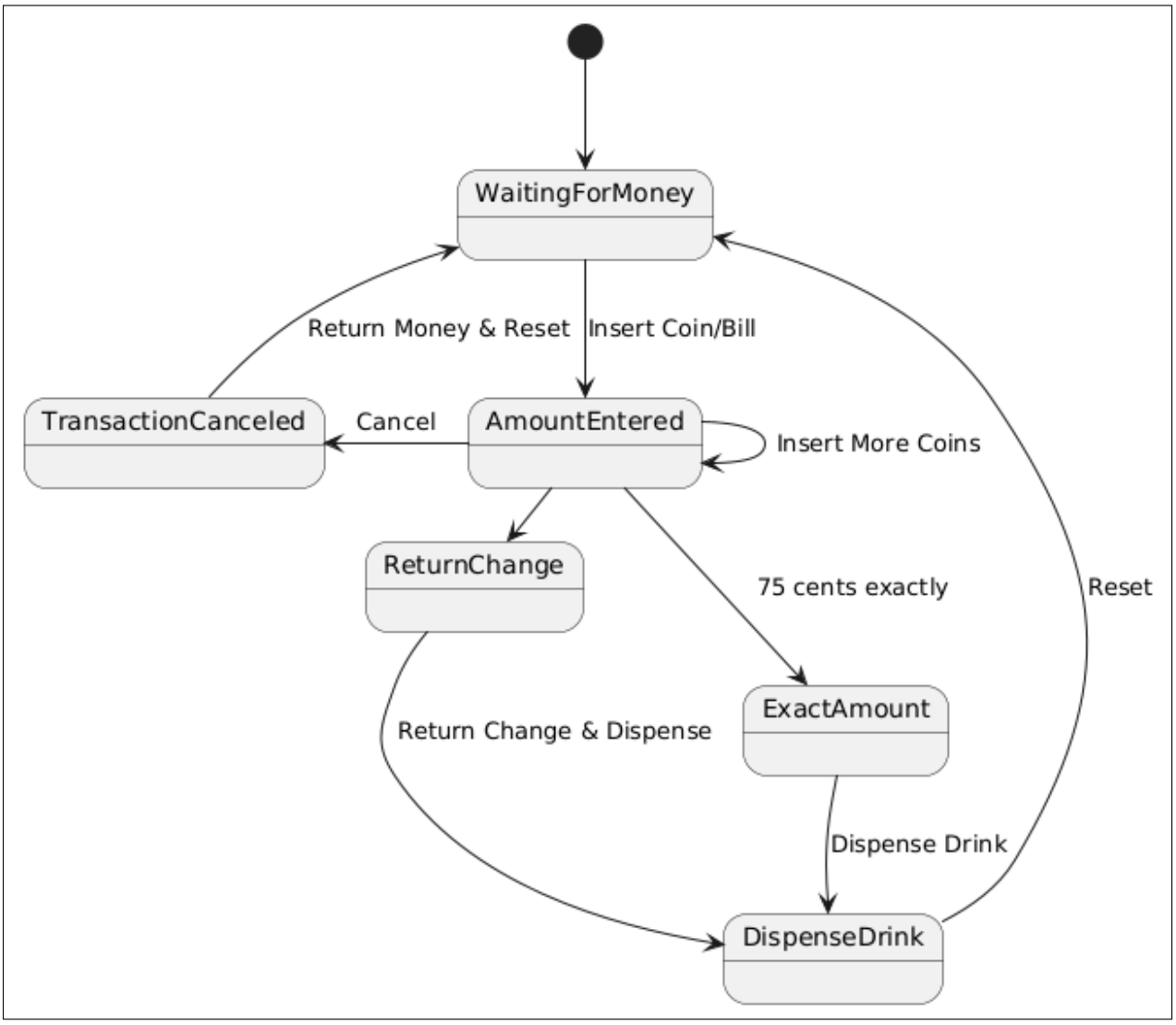
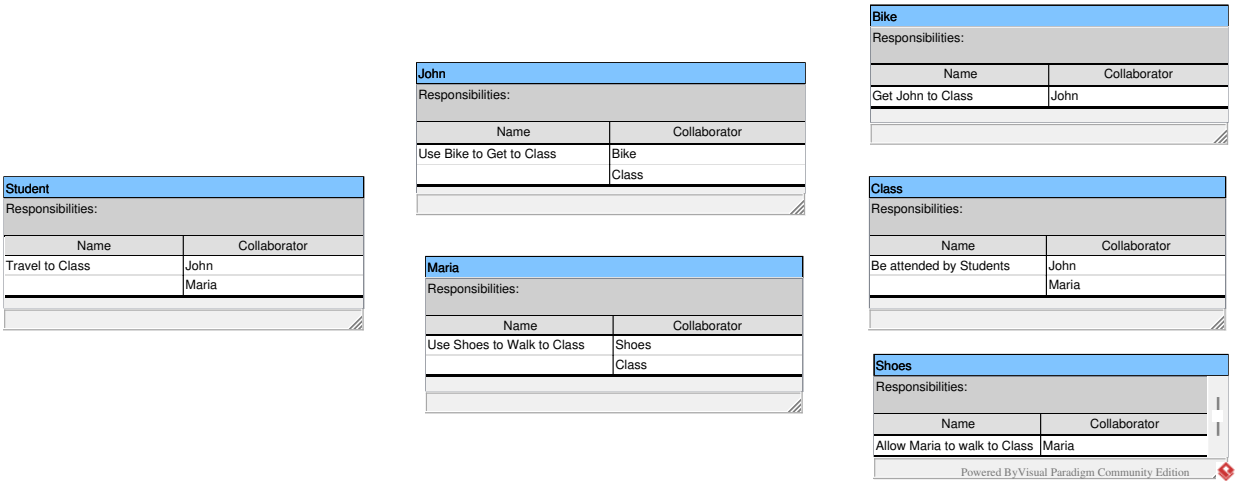


Figure 1.5: State Diagram for Vending Machine

1.5 Section 3.5 Exercise



Student

Responsibilities:

Name	Collaborator
Travel to Class	John
	Maria

John

Responsibilities:

Name	Collaborator
Use Bike to Get to Class	Bike
	Class

Maria

Responsibilities:

Name	Collaborator
Use Shoes to Walk to Class	Shoes
	Class

Bike

Responsibilities:

Name	Collaborator
Get John to Class	John

Class

Responsibilities:

Name	Collaborator
Be attended by Students	John
	Maria

Shoes

Responsibilities:

Name	Collaborator
Allow Maria to walk to Class	Maria

Figure 1.6: CRC Cards

The CRC cards identify the primary classes (Student, Bike, and Shoes) and their responsibilities in the context of students commuting to school. The Student class coordinates interactions with Bike or Shoes, with each class's card specifying its role in these interactions. This breakdown helps outline responsibilities and collaborations effectively.

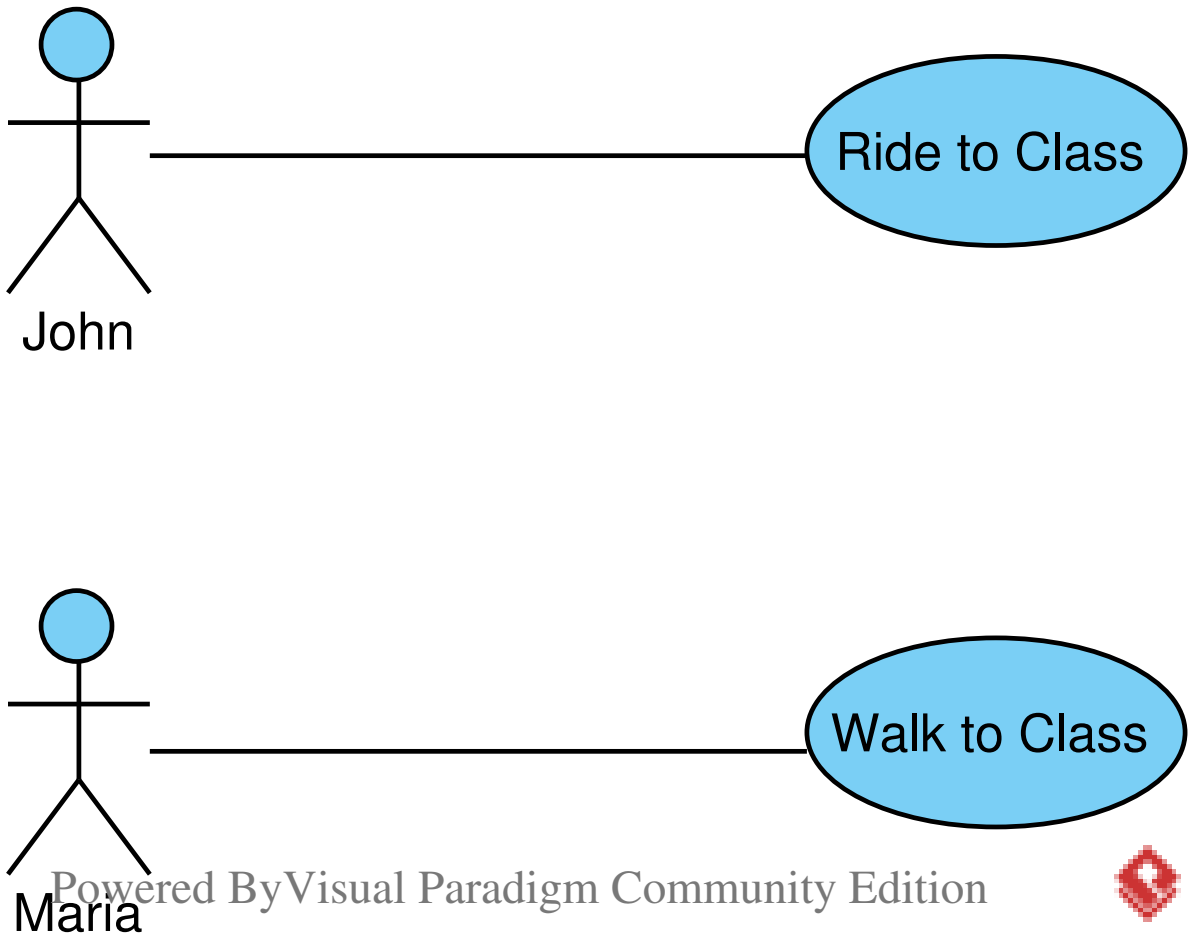


Figure 1.7: Use Case Diagram

The use case diagram depicts the different paths John and Maria take to get to class: John rides a bike, while Maria uses shoes to walk. This diagram visually separates each student's use case, showing how different resources can achieve the same goal of commuting to school.

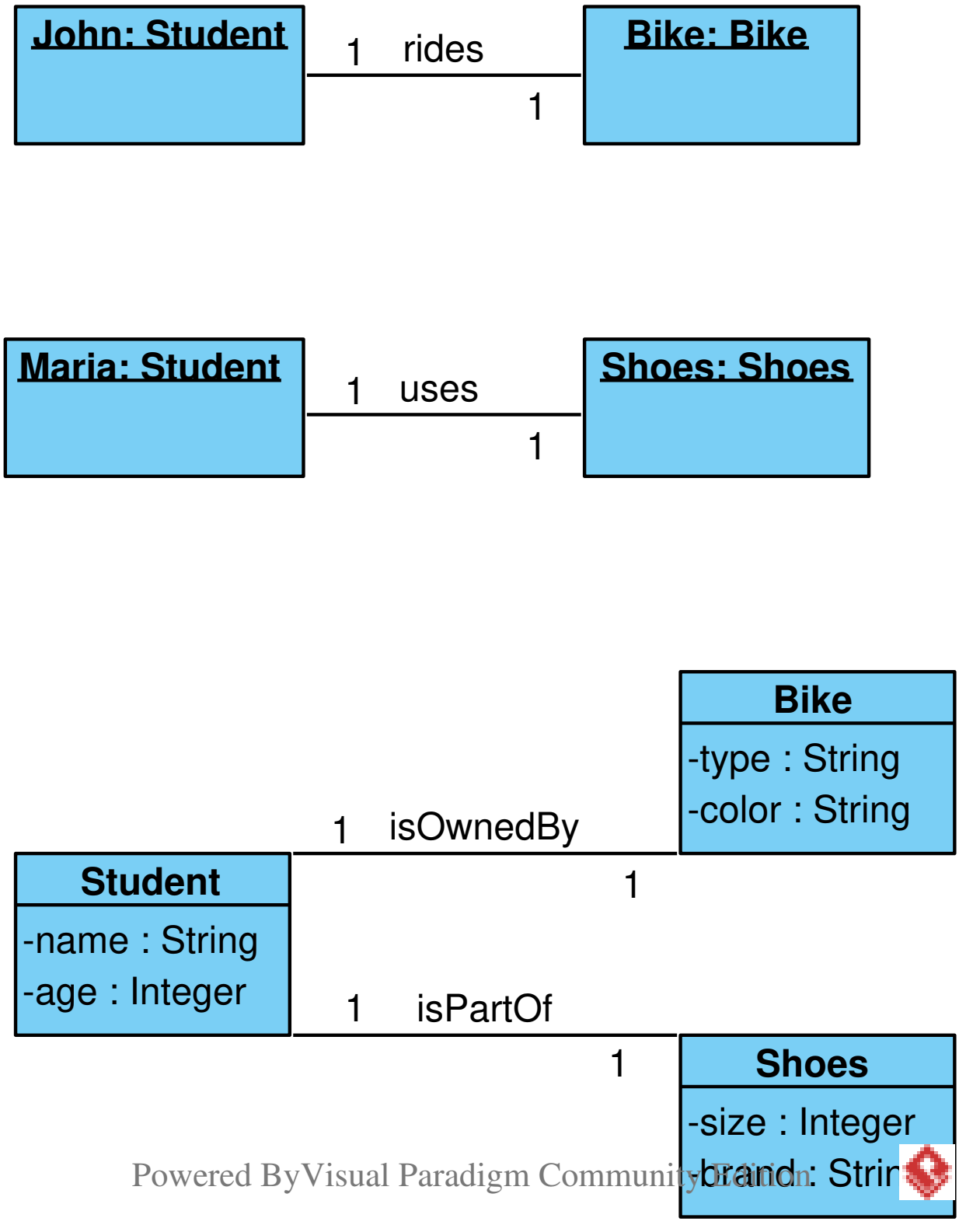


Figure 1.8: Object Diagram and Class Diagram

The class diagram shows the structural relationships between Student, Bike, and Shoes. The association between Student and Bike is modeled as aggregation, since a bike can exist independently but is linked to a student for commuting. The relationship between Student and Shoes is a composition, as shoes are integral to the student's equipment.

1.6 Section 3.6 Exercise

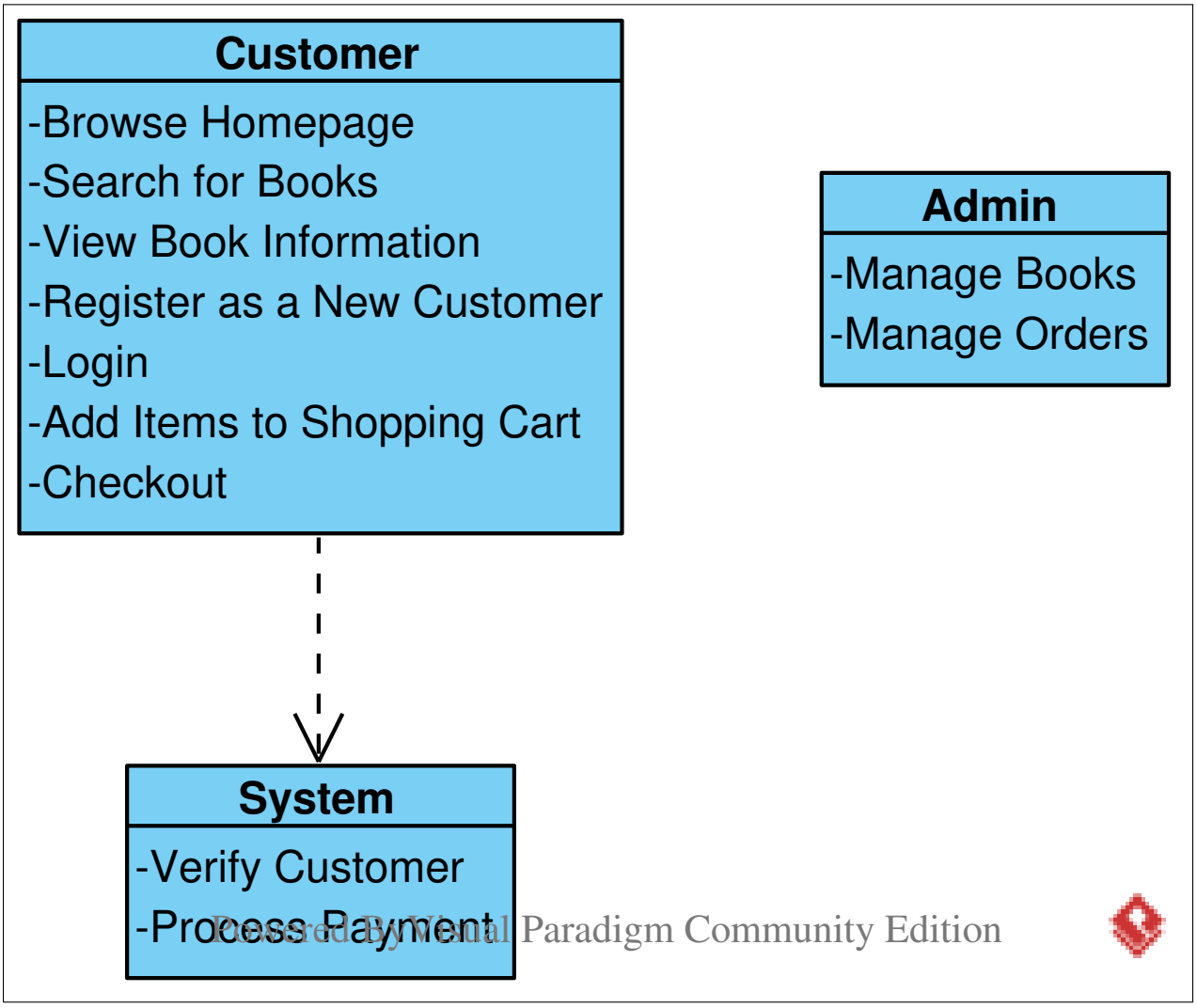


Figure 1.9: Use-case Diagram for E-Book Website

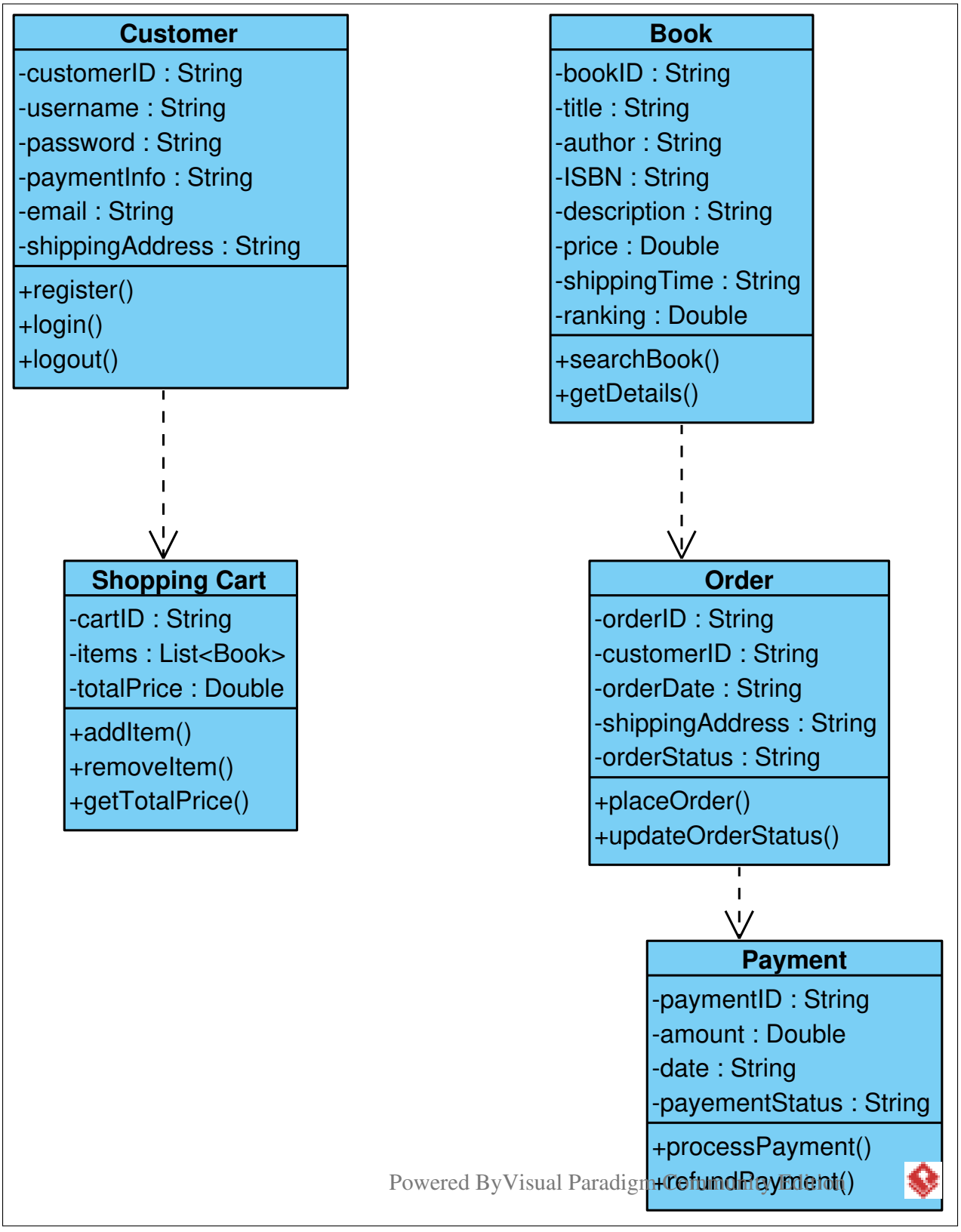


Figure 1.10: Class Diagram for E-Book Website

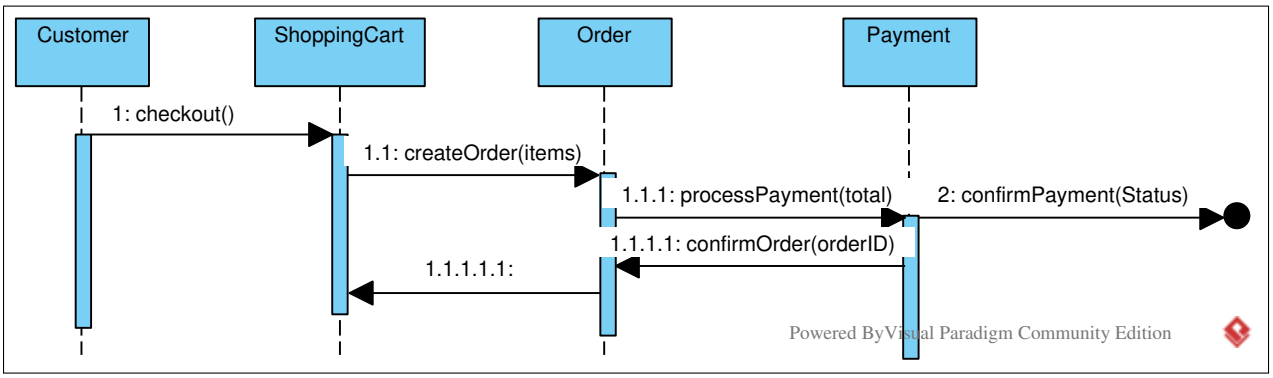


Figure 1.11: Sequence Diagram for E-Book Website