Lab

Ryan Davis
Cory Vitanza

October 2024

Contents

Chapter 1

Advanced Class Modeling

GitHub link to lab repository: `https://github.com/ryry91021/ssw345-labs/tree/main/AdvancedClassModelingLab`

1.1 Exercises

1.1.1 Exercise 3.1

Part 1

- The **Buffer** is associated with both the **Sheet** and **Selection** classes, with a multiplicity of 0..1, meaning it is optional.
- The **Selection** class is also optional, with the same multiplicity (0..1) and is linked to the **Sheet**.
- The **Sheet** class is the central element, connected to the **Buffer**, **Selection**, **Line**, and **Box** classes, each with a multiplicity of 0..1, meaning a **Sheet** can have at most one instance of each.
- The **Line** class is associated with the **Sheet** (multiplicity 0..1) and must have one **LineSegment** (multiplicity 1).
- The **LineSegment** class is linked to the **Point** class with a multiplicity of 1..2, meaning it must have one or two **Points**.
- The **Point** class represents the points that define the geometry of a **LineSegment**.

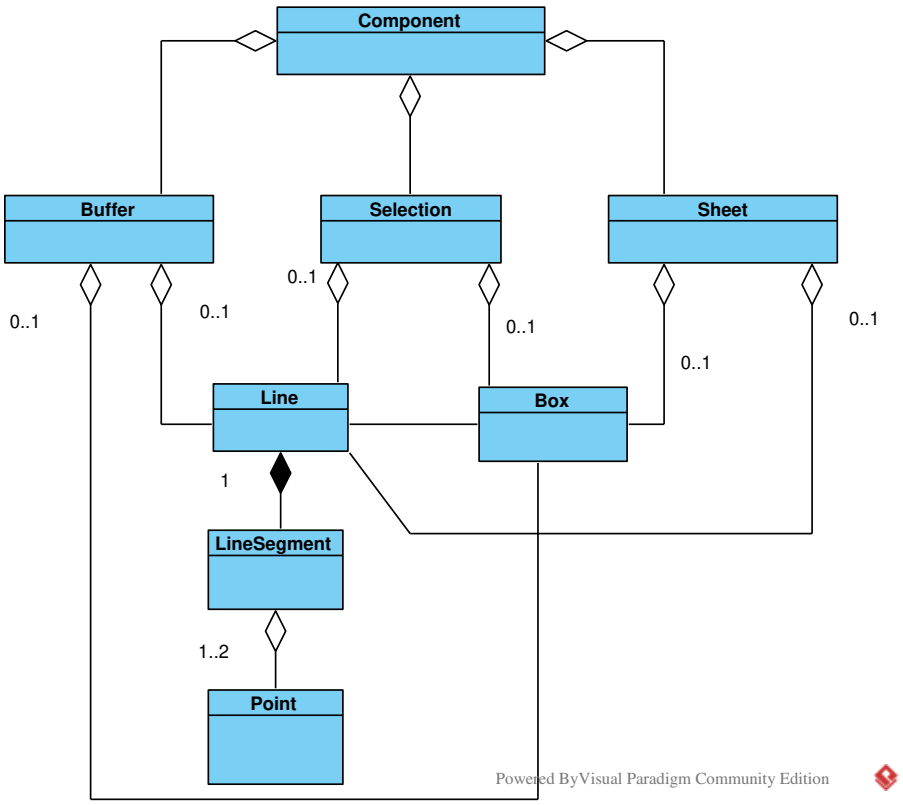


Figure 1.1: Revised simple editor class hierarchy

Part 2

Merits of the Revision

- **Simplified Maintenance:** Having a base class like Component reduces redundancy, as the common logic related to managing collections of lines and boxes is encapsulated in a single place.
- **Enforcement of Constraints:** This revision enforces the rule that a Line or Box can only belong to one Component (either Buffer, Selection, or Sheet), preventing any ambiguity in ownership.
- **Flexible Extension:** If, in the future, more types of collections (other than buffer, selection, or sheet) need to be added, they can simply inherit from Component.
- This revised diagram would address the problem stated in part 2 by clearly enforcing the "exactly one" rule, while also making the class structure more flexible and easier to extend.

1.1.2 Exercise 3.2

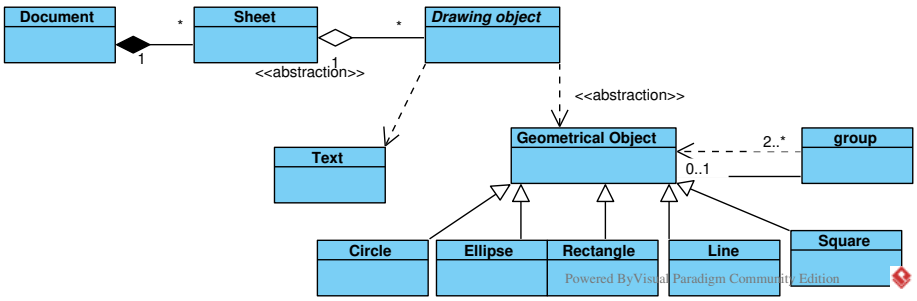


Figure 1.2: Class diagram for a graphical Document editor

1.1.3 Exercise 3.3

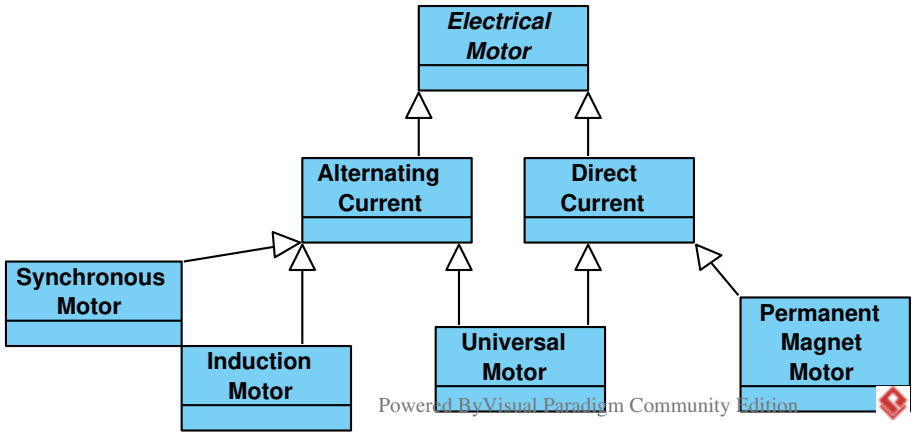


Figure 1.3: Electrical motors class diagram

Multiple inheritance was used for the electric motor, as it can be generalized into either a type of alternating current or direct current.

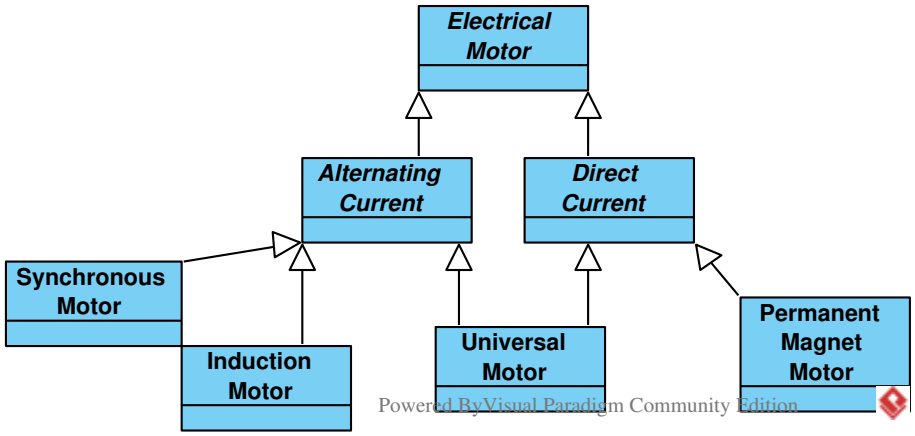


Figure 1.4: Revised Electrical motors class diagram

By abstracting the AC and DC current classes, we are able to allow each individual motor to be derived from a different type of current.

1.1.4 Exercise 3.4

`https://github.com/ryry91021/ssw345-labs/blob/main/AdvancedClassModelingLab/wk3StateMachines.py`

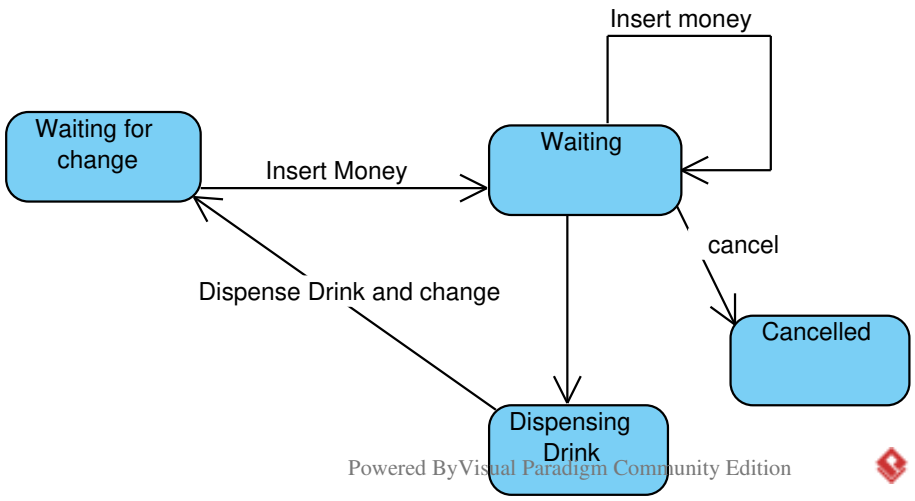


Figure 1.5: Vending machine class diagram

1.1.5 Exercise 3.5

Class	Responsibilities	Collaborations
Student	- Manage student info - Select transportation - Go to class	- Transportation - Class
Transportation	- Manage transport modes - Provide transport info	- Student - Bike - Shoes
Bike	- Represent bike attributes - Facilitate riding to class	- Transportation - Student
Shoes	- Represent shoe attributes - Facilitate walking to class	- Transportation - Student
Class	- Manage class info - Schedule class	- Student

Table 1.1: CRC Cards/Table for Student Transportation Simulation

Part 1: CRC Cards The CRC card creation process began by identifying the key classes present in the user story, specifically focusing on John and Maria's actions and interactions. The main classes came from the characters and their transportation methods: Student, Transportation, Bike, Shoes, and Class. Each class was analyzed to determine its responsibilities and correlation with other classes. For instance, the Student class was responsible for managing student information and selecting transportation, while the Transportation class served as a parent for specific transport methods. This initial analysis helped clarify the structure of the system and laid the groundwork for the subsequent diagrams.

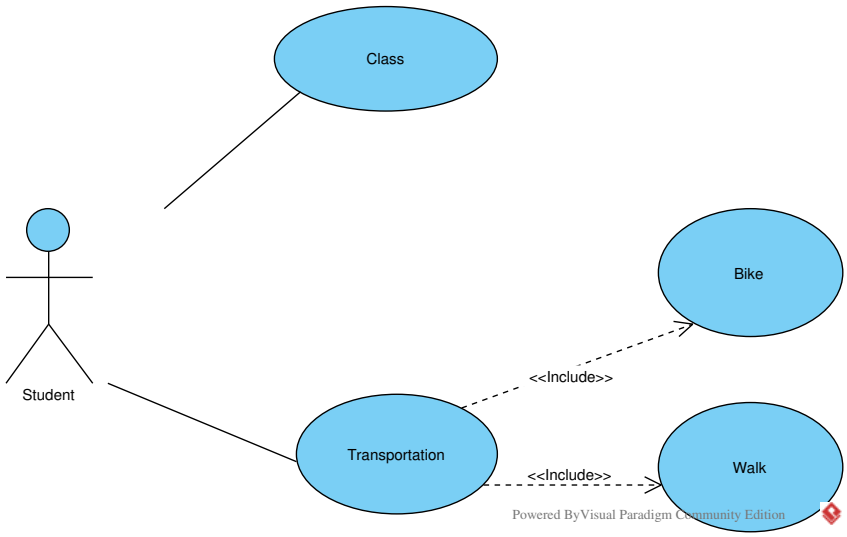


Figure 1.6: Use Case Diagram for Students

Part 2: Use Case Diagram When constructing the use case diagram, I aimed to capture the interactions between the Student and their actions in relation to transportation and class attendance. The main use cases were identified as selecting transportation, attending class, riding a bike, and walking to class. I chose to use inclusion relationships to show that selecting transportation leads to either riding a bike or walking to class. This diagram succinctly represents the students' journey from selecting how to get to class to actually attending the class. The use case diagram effectively visualizes the system's functionality from the user's perspective, providing a clear view of the processes involved.

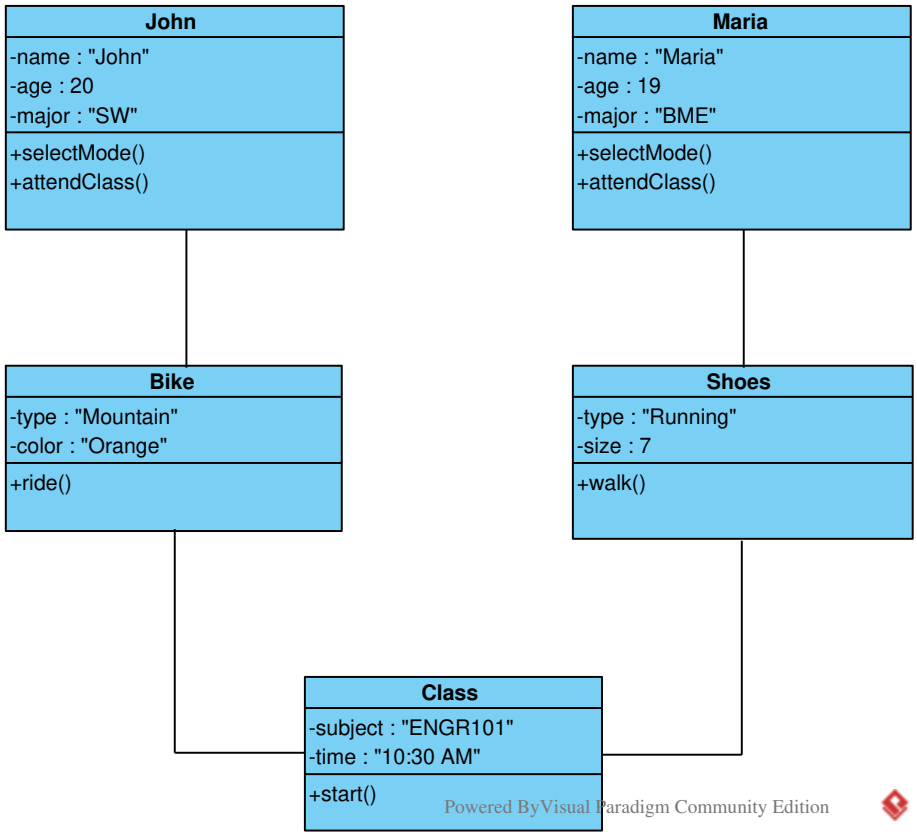


Figure 1.7: Object Diagram: Student Transportation Scenario

Part 3: Object Diagram The object diagram was created as a concrete snapshot of the system based on the previously defined classes and their relationships. I focused on instantiating the main objects, such as John, Maria, Bike, Shoes, and Class, to reflect the specific scenario outlined in the user story. The associations among these objects were carefully mapped to illustrate how John uses a bike and Maria uses shoes to reach the same class. This diagram highlights the real-time interaction between the objects and provides insight into how the system behaves during a specific instance. It serves as a bridge between the abstract class structures and the practical, real-world scenario presented in the user story.

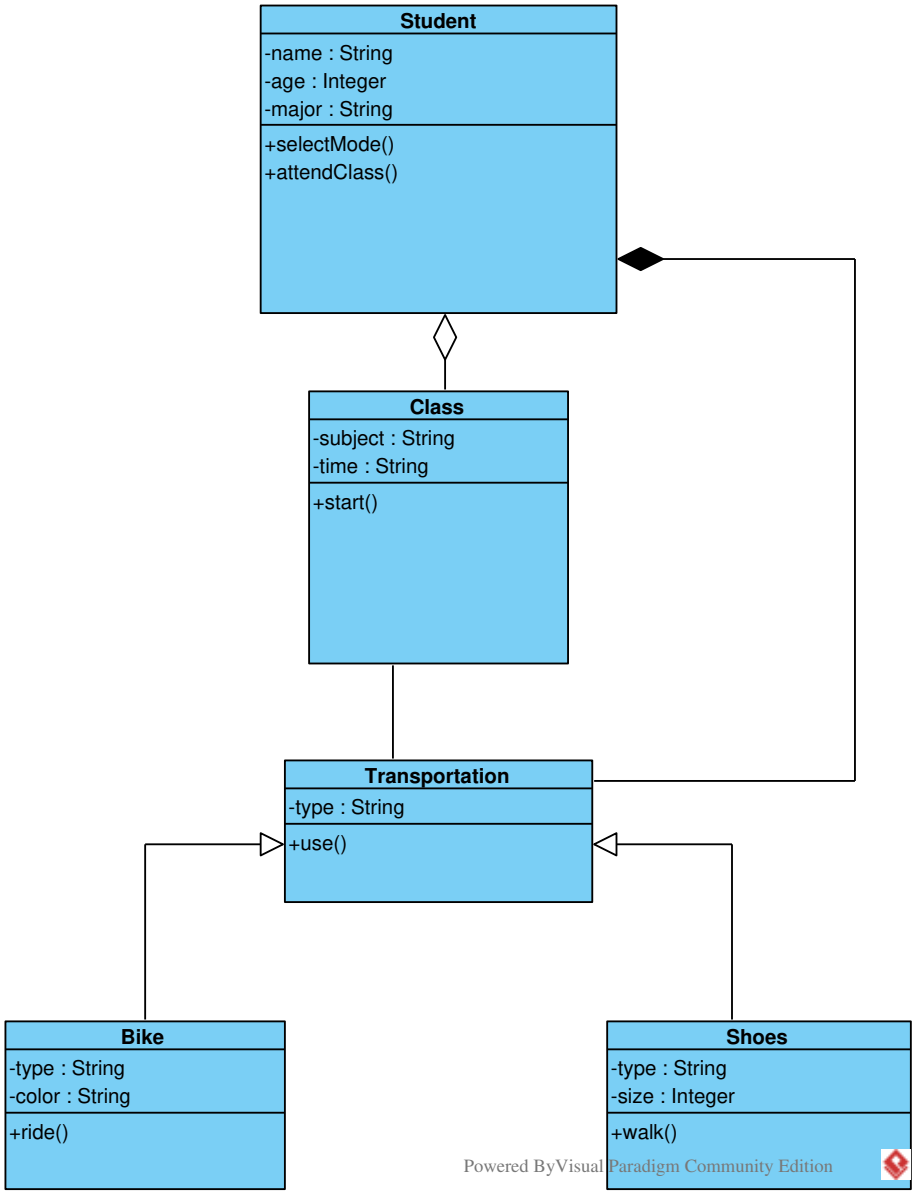


Figure 1.8: Class Diagram: Student Transportation System

Part 4: Class Diagram In developing the class diagram, I synthesized information from the previous diagrams and refined the relationships between the classes. I recognized that Transportation could serve as an abstract class with sub-classes for Bike and Shoes, reflecting the generalization of transportation modes. I included attributes and methods for each class to provide a compre-

hensive view of their functionality. The relationships were explicitly defined, using multiplicities to clarify the number of instances involved, such as a Student having exactly one Transportation method and a Class potentially having multiple Students. This diagram serves as a more structured representation of the system's architecture, emphasizing the organization and hierarchy of classes while highlighting their interactions.

1.1.6 Exercise 3.6

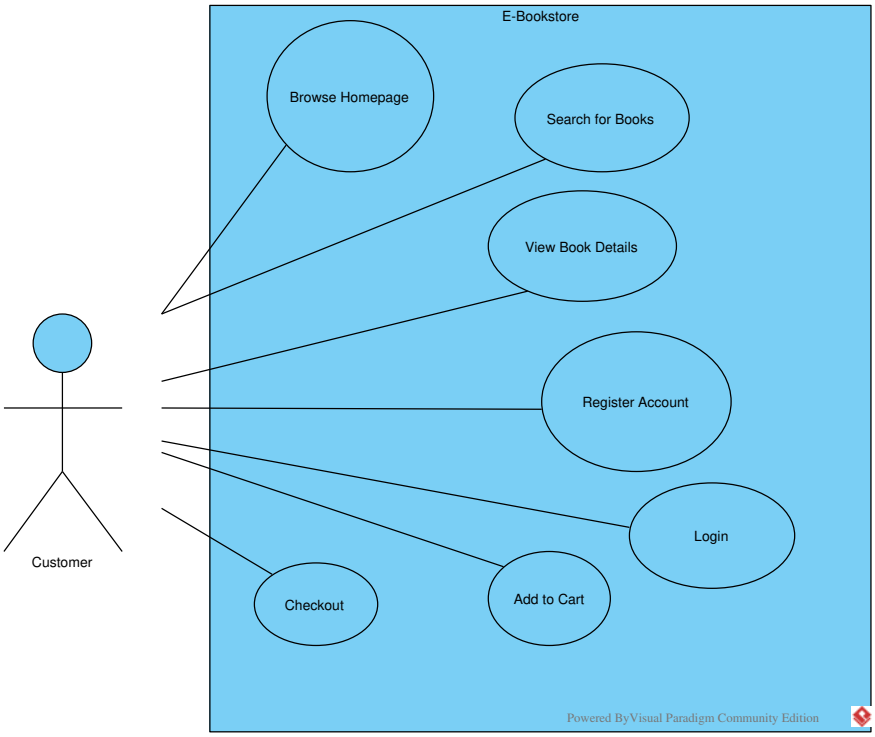


Figure 1.9: Use Case Diagram for E-Bookstore

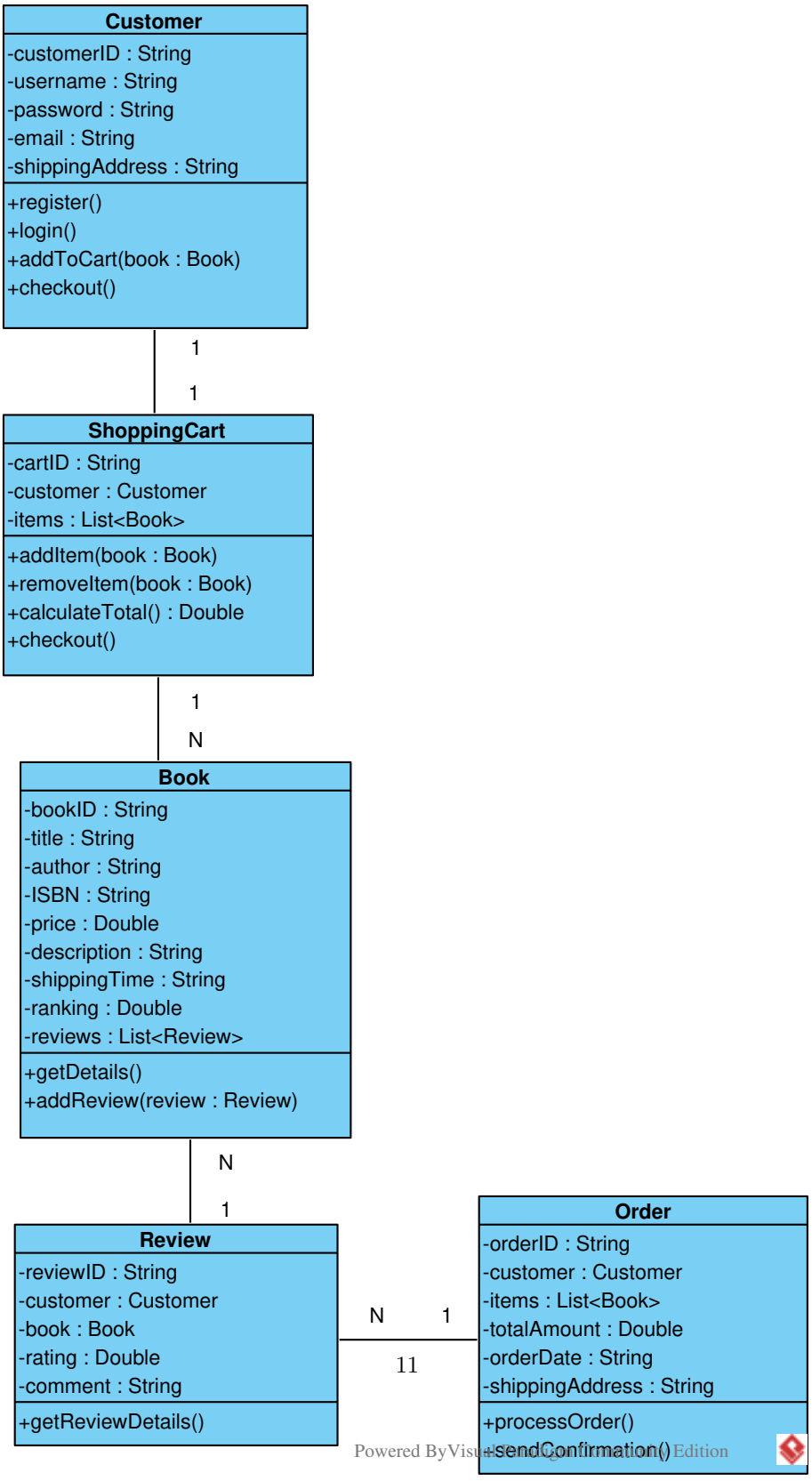


Figure 1.10: Class Diagram for E-Bookstore

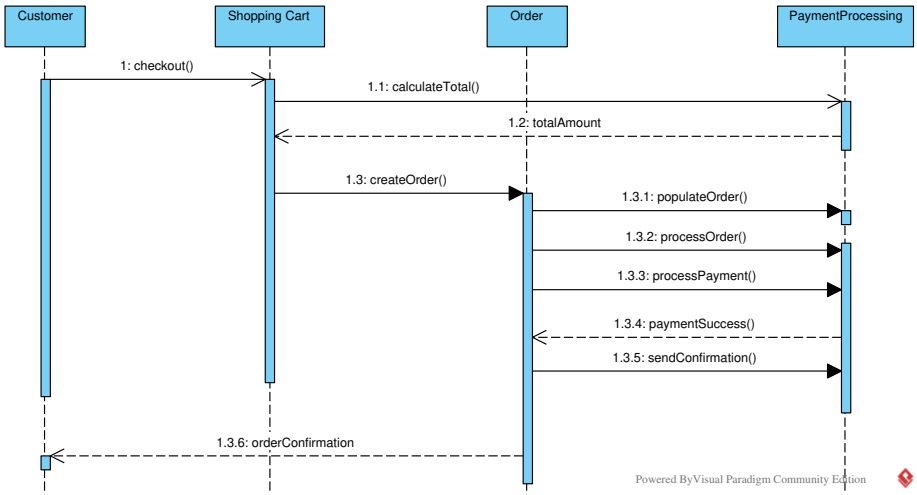


Figure 1.11: Sequence Diagram for Checkout Function

Chapter 2

Simulations with Simulink

GitHub link to lab repository: `https://github.com/ryry91021/ssw345-labs/tree/main/simulationsWithSimulink`

2.1 Exercises

2.1.1 Exercise 3.1

Simulink models

Simulink model: `https://github.com/ryry91021/ssw345-labs/blob/main/simulationsWithSimulink/exercise7_1/model7_1.slx`

Scope output

Scope output: `https://github.com/ryry91021/ssw345-labs/blob/main/simulationsWithSimulink/exercise7_1/output.txt`

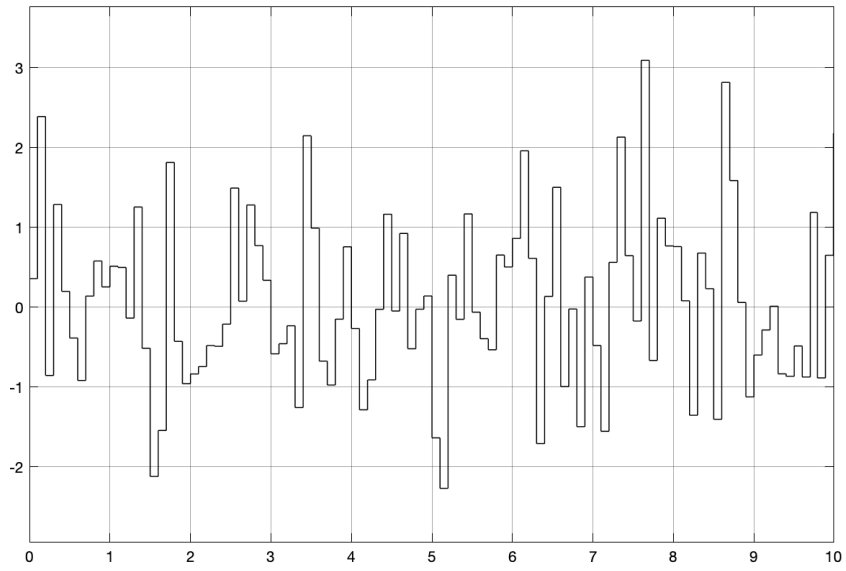
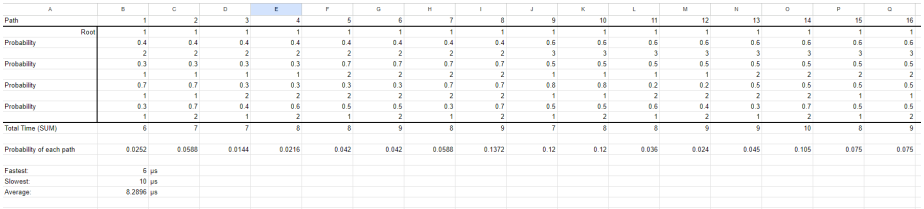


Figure 2.1: Simulink Model Scope

2.1.2 Exercise 3.2

Part 1

Excel sheet: `https://docs.google.com/spreadsheets/d/1AxTKCtFW4Yg4dQDA87gTBRYYrQZAZCoSntrs_2nK9pw/edit?usp=sharing`



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Path	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	Root	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	Probability	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6
4		2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3
5	Probability	0.3	0.3	0.3	0.3	0.7	0.7	0.7	0.7	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
6		1	1	1	1	2	2	2	2	1	1	1	1	2	2	2	2
7	Probability	0.7	0.7	0.3	0.3	0.3	0.3	0.7	0.7	0.8	0.8	0.2	0.2	0.5	0.5	0.5	0.5
8		1	1	2	2	2	2	2	2	1	1	2	2	2	2	1	1
9	Probability	0.3	0.7	0.4	0.6	0.5	0.5	0.3	0.7	0.5	0.5	0.6	0.4	0.3	0.7	0.5	0.5
10		1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
11	Total Time (SUM)	6	7	7	8	8	9	8	9	7	8	8	9	9	10	8	9
12																	
13	Probability of each path	0.0252	0.0588	0.0144	0.0216	0.042	0.042	0.0588	0.1372	0.12	0.12	0.036	0.024	0.045	0.105	0.075	0.075
14																	
15	Fastest:	6	µs														
16	Slowest:	10	µs														
17	Average:	8.2896	µs														

Figure 2.2: Excel results

To calculate the fastest, slowest, and average search times for the binary tree search, we consider the processing times at each node and the total time for various search paths.

Fastest Search Time: The fastest search occurs when the search follows the shallowest path in the binary tree. In this case, the search proceeds through the following nodes: Node $A \to B \to C$. The total search time is the sum of the processing times at each node.

$$t_{\text{fastest}} = 1 + 2 + 1 + 1 + 1 = 6\,\mu\text{s}$$

Slowest Search Time: The slowest search occurs when the search follows the deepest path in the binary tree. This path is: Node $A \to D \to E \to F$. The total search time is the sum of the processing times at each node.

$$t_{\text{slowest}} = 1 + 3 + 2 + 2 + 2 = 10\,\mu\text{s}$$

Average Search Time: The average search time is computed by averaging the total time for all possible search paths, along with the weights/probabilities. The average is calculated as:

$$\text{Weighted Average} = \sum_{i=1}^{n} (\text{Time}_i \times \text{Probability}_i)$$

Simplifying this expression, the average search time is:

$$t_{\text{average}} = 8.2896\,\mu\text{s}$$

In conclusion, the fastest search time is $6\,\mu$s (Path 1), the slowest search time is $10\,\mu$s (Path 14), and the average search time is $8.2896\,\mu$s.

Part 2

Simulink model: `https://github.com/ryry91021/ssw345-labs/blob/main/simulationsWithSimulink/exercise7_2/x7_2Simulink1.slx`

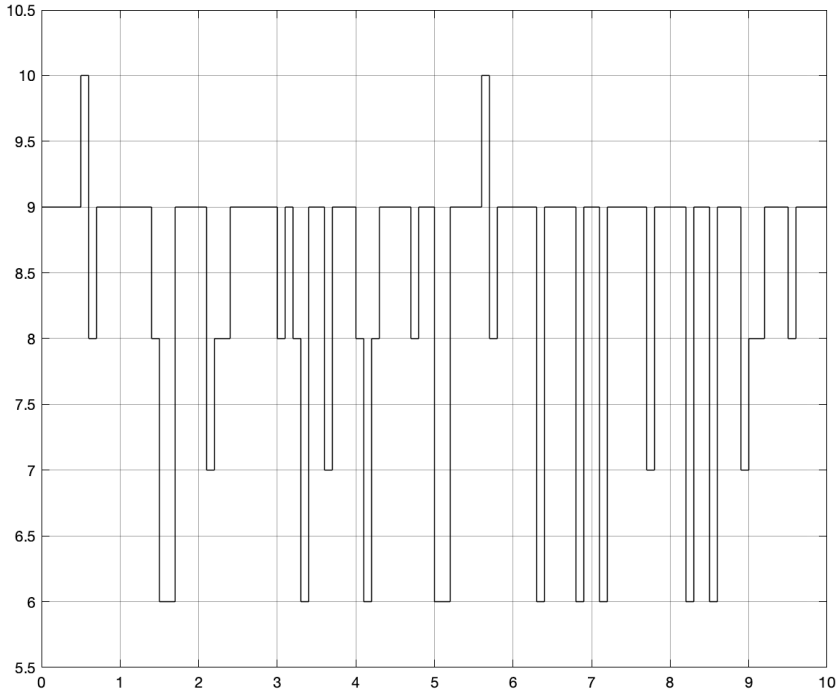


Figure 2.3: Simulink Model Scope

This simulation matches manual calculations as the max value provided in

the scope is 10, and the minimum value provided is 10. Furthermore, from the derived average time of 8.2896, the average time is visually estimated to be approximately 8.2 according to the scope. The instances in the scope match the calculations from the manual excel calculations of the simulated process.