## 

## 

**Experiment 3:**

**Object Oriented Design and Implementation**

CPE106L (Software Design Laboratory)

Group No.: **10**

Section: **B2**

## **PreLab**



|  |
| --- |
| **Readings, Insights, and Reflection**  **<COLLAMAT>**  **Chapter 4 & 5: Systems Analysis and Design: An Object-Oriented Approach with UML**   * Chapters 4 and 5 of the book on system analysis and design cover two essential types of Unified Modeling Language (UML) diagrams – use case diagrams and class diagrams. Chapter 4 delves into use case diagrams, which provide a visual representation of the functionality of a system from the perspective of its users or actors. It explains the key components of use case diagrams, including actors, use cases, and the various types of associations between them. The chapter highlights the significance of use case diagrams in defining user requirements and establishing the scope of a system, making them indispensable tools for modeling system behavior during the software development process. Moving on to Chapter 5, the focus shifts to class diagrams, which illustrate the structural aspects of a system by depicting classes, their attributes, operations, and relationships. The chapter meticulously details the syntax and notation used in class diagrams, enabling readers to model the data requirements and class relationships within a system effectively. Additionally, it explores how class diagrams can be extended to model system behavior through state diagrams, which trace the behavior of objects over time, further enhancing the understanding of system dynamics. Together, these chapters provide a comprehensive exploration of two fundamental UML diagram types – use case diagrams and class diagrams – that are pivotal in the software development lifecycle. By mastering these diagrams, analysts and designers can effectively capture and communicate user requirements, system functionality, and structural design, laying a solid foundation for the successful development of software systems.   **Chapter 9: Fundamentals of Python: First Programs**   * Regarding Chapter 9 of "Fundamentals of Python: Data Structures" by Kenneth Lambert, titled "Design with Classes," it presents an in-depth exploration of object-oriented programming (OOP) concepts in Python. Although the initial introduction to classes may seem complex, the chapter gradually unveils the power and versatility of this programming paradigm for software development. The chapter begins by introducing the fundamental concepts of classes, their significance in software design, and the process of creating classes, defining attributes and methods, and instantiating objects. As readers progress, they delve into advanced OOP principles such as inheritance, which enables code reuse and hierarchical organization of classes; polymorphism, which allows objects of different classes to be treated as objects of a common superclass; and encapsulation, which promotes data hiding and modular design. A significant takeaway from this chapter is the role of classes in enhancing code modularity, maintainability, and scalability by facilitating the sharing of behaviors and properties among objects. This knowledge becomes indispensable when developing complex applications, such as graphical user interfaces (GUIs) and web applications, where Python has gained widespread adoption, particularly in frameworks like Django. While Chapter 9 may initially pose challenges for newcomers to OOP, mastering the concepts and techniques presented lays a crucial foundation for writing structured, maintainable, and scalable Python code. By embracing the principles of object-oriented programming through classes, developers can create robust and extensible software systems that stand the test of time.   **<CANDA>**  **Chapter 4 & 5: Systems Analysis and Design: An Object-Oriented Approach with UML**   * Popular methods for acquiring needs, including surveys, inspections, and interviews, were covered in the preceding chapter. Using use case and sequence diagram, use case model, and use case descriptions, the information acquired using these techniques is grouped and presented as use cases in chapter 4 of “Systems Analysis and Design: An Object-Oriented Approach with UML”. A use case is a formal method of describing how a business system communicates with its surroundings. A use case is basically an executive summary of the business procedures in a business information platform. Use cases serve as the object-oriented system's complete foundation from a pragmatic perspective. Use cases might describe the existing system or the one being created.      * While chapter 5 of the reading, it focuses on class diagrams. A class diagram is a static model that shows the classes and connections between classes consistent during the system. The class diagram illustrates the connections among classes, which comprise both actions and situations. The class diagram's components, several methods for simplifying class diagrams, and an alternative structural diagram called the object diagram are all covered in the parts that follow. Understanding this would advance and organize coding significantly. Overall, Chapter 9 offers a crucial basis for Python object-oriented programming, despite its initial appearance intimidating. You can build a better organized and efficient environment by understanding the topic of classes.     **Chapter 9: Fundamentals of Python: First Programs**     * Reading Chapter 9 of Kenneth Lambert's "Fundamentals of Python: Data Structures" takes a while. "Design with Classes" explores Python's object-oriented programming (OOP) in further detail and introduces the idea of classes. Beginning with a discussion of classes and their significance in software design, the chapter introduces the topic. The creation of classes, defining attributes and methods, and instantiating objects are all covered after that. Also, the section delves into depth into polymorphism, encapsulation, inheritance, and which are crucial concepts in OOP. Users of an object's class's methods are the only ones who can access its data thanks to encapsulation. This lessens the likelihood of data modifications being made without cause. One class can freely adopt the traits and behaviors of another class thanks to inheritance. The subclass may also alter or add to the parent class's existing methods and add new data. One important way to reuse code is through inheritance. The same headers might exist for methods across numerous classes thanks to polymorphism. This lessens the requirement for learning new names for common operations.   **<Estacion>** |

**Chapter 4 & 5: Systems Analysis and Design: An Object-Oriented Approach with UML**

* Chapters 4 and 5 of Systems Analysis and Design: This lecture will be "Object-Oriented Approach with UML," which will teach the use case and class diagrams as graphical tools for system analysis and design. In these chapters, the use case diagrams are described as the representation of interactions between the actor and the system, and the class diagrams are used to maintain the organization of classes, interfaces, and objects within a system. The authors provide specific guidelines on how to draw these diagrams, from identifying actors/classes and defining use cases, attributes, and methods to using UML notation to represent relationships. Besides, they also explore the inheritance and abstract classes that are more complicated. The text mainly emphasizes the vital role of use case and class modeling in system design and provides valuable tips on making compelling diagrams.

**Chapter 9: Fundamentals of Python: First Programs**

* In the ninth chapter of "Fundamentals of Python: In the first section, "Programs and Data Structures," the author introduces OOP language and describes essential notions like inheritance, classes, and objects. The chapter highlights the significance of structured class design by guiding through Python class design and implementation processes. The author gives a comprehensive course about the design of classes, their implementation in Python, and the drawing of class diagrams using UML. This chapter details the concepts of class inheritance, polymorphism, and encapsulation, which are the advanced OOP ideas. Adherence to proper class design principles is the key to creating Python applications that are more robust, efficient, and maintainable.

**<Trindad>**

**Chapter 4 & 5: Systems Analysis and Design: An Object-Oriented Approach with UML**

* In the assigned reading, chapter 4, the topics of business process and functional modeling were covered. The chapter discussed two types of models: activity diagrams and use cases. Overall, these two contribute to the simplification of an information system's functionality. Additionally, this chapter included examples and illustrations on how to use the provided diagrams. The representation of the various elements of a diagram was displayed by symbols. Actors and activities are a few of these. A use case diagram for an appointment system was displayed in one of the instances. In the book's fifth chapter, which addresses structural modeling and class diagrams, another diagram is covered. The class diagram emphasizes the links between classes and objects, as opposed to the use case diagram, which concentrates on the functionality. Together with their syntax, these classes' many purposes and behaviors were covered in Chapters 4 and 5. As I read these chapters, it became clear that both diagrams would be helpful for generating and streamlining during the design stages. Gaining knowledge of this would be very beneficial since it would simplify most of the design process.

**Chapter 9: Fundamentals of Python: First Programs**

* The ninth chapter of Kenneth Lambert's "Fundamentals of Python: Data Structures" was, in my opinion, both illuminating and challenging to learn at first. In this chapter, "Design with Classes," object-oriented programming (OOP) in Python is examined in more detail, and the concept of classes is presented. The conversation started out with a look at classes and how important they are to software design. The definition, syntax, behavior, creation, instantiating objects, and further information about the class were provided. I tried to understand the basic structure of the grammar, even if it could seem intimidating at first, and I discovered how useful classes can be for software development. The three essential OOP concepts—inheritance, polymorphism, and encapsulation—are also thoroughly covered in this chapter.

**Answers to Questions**

1. A
2. C
3. B
4. B
5. B
6. B
7. A
8. B
9. B
10. A