Al-Mustaqbal University
College of Sciences
Intelligent Medical System Department

جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلوم
قسم الانظمة الطبية الذكية

Lecture: (4)

Software Analysis and Design using OOP Techniques

Subject: Object oriented programming I

Class: Second

Lecturer: Dr. Maytham N. Meqdad

Software Analysis and Design using OOP Techniques

Object-Oriented Analysis and Design(OOAD)

Object-Oriented Analysis and Design (OOAD) is a software engineering methodology that employs object-oriented principles to model and design complex systems. It involves analyzing the problem domain, representing it using objects and their interactions, and then designing a modular and scalable solution. It helps create systems that are easier to understand, maintain, and extend by organizing functionality into reusable and interconnected components.

Important Aspects of OOAD

Here are some important aspects of OOAD:

- **Object-Oriented Programming:** Object-oriented programming involves modeling real-world objects as software objects, with properties and methods that represent the behavior of those objects. OOAD uses this approach to design and implement software systems.

- **Design Patterns:** Design patterns are reusable solutions to common problems in software design. OOAD uses design patterns to help developers create more maintainable and efficient software systems.

- **UML Diagrams:** Unified Modeling Language (UML) is a standardized notation for creating diagrams that represent different aspects of a software system. OOAD uses UML diagrams to represent the different components and interactions of a software system.

- **Use Cases:** Use cases are a way of describing the different ways in which users interact with a software system. OOAD uses use cases to help developers understand the requirements of a system and to design software systems that meet those requirements.

Object-Oriented Analysis

Object-Oriented Analysis (OOA) is the first technical activity performed as part of object-oriented software engineering. OOA introduces new concepts to investigate a problem. It is based on a set of basic principles, which are as follows:

- **The information domain is modeled:**
    - Lets say you're building a game. OOA helps you figure out all the things you need to know about the game world – the characters, their features, and how they interact. It's like making a map of everything important.

- **Behavior is represented:**
    - OOA also helps you understand what your game characters will do. If a character jumps when you press a button, OOA helps describe that action. It's like writing down a script for each character.

- **The function is described:**
    - Every program has specific tasks or jobs it needs to do. OOA helps you list and describe these jobs. In our game, it could be tasks like moving characters or keeping score. It's like making a to-do list for your software.

- **Data, functional, and behavioral models are divided to uncover greater detail:**
    - OOA is smart about breaking things into different parts. It splits the job into three categories: things your game knows (like scores), things your game does (like jumping), and how things in your game behave (like characters moving around). This makes it easier to understand.

- **Starting Simple, Getting Detailed:**
    - OOA knows that at first, you just want to understand the big picture. So, it starts with a simple version of your game or program. Later on, you add more details to make it work perfectly. It's like sketching a quick drawing before adding all the colors and details.

Object-Oriented Design

In the object-oriented software development process, the analysis model, which is initially formed through object-oriented analysis (OOA), undergoes a transformation during object-oriented design (OOD). This evolution is crucial because it shapes the analysis model into a detailed design model, essentially serving as a blueprint for constructing the software.

The outcome of object-oriented design, or OOD, manifests in a design model characterized by multiple levels of modularity. This modularity is expressed in two key ways:

- **Subsystem Partitioning:**
    - At a higher level, major components of the system are organized into subsystems.
    - This practice is similar to creating modules at the system level, providing a structured and organized approach to managing the complexity of the software.
- **Object Encapsulation:**
    - A more granular form of modularity is achieved through the encapsulation of data manipulation operations into objects. " It's like putting specific tasks (or operations) and the data they need into little boxes called "objects."
    - Each object does its job neatly and keeps things organized. So, if our game has a character jumping, we put all the jumping stuff neatly inside an object.
    - It's like having a box for each task, making everything easier to handle and understand.

Furthermore, as part of the object-oriented design process, it is essential to define specific aspects:

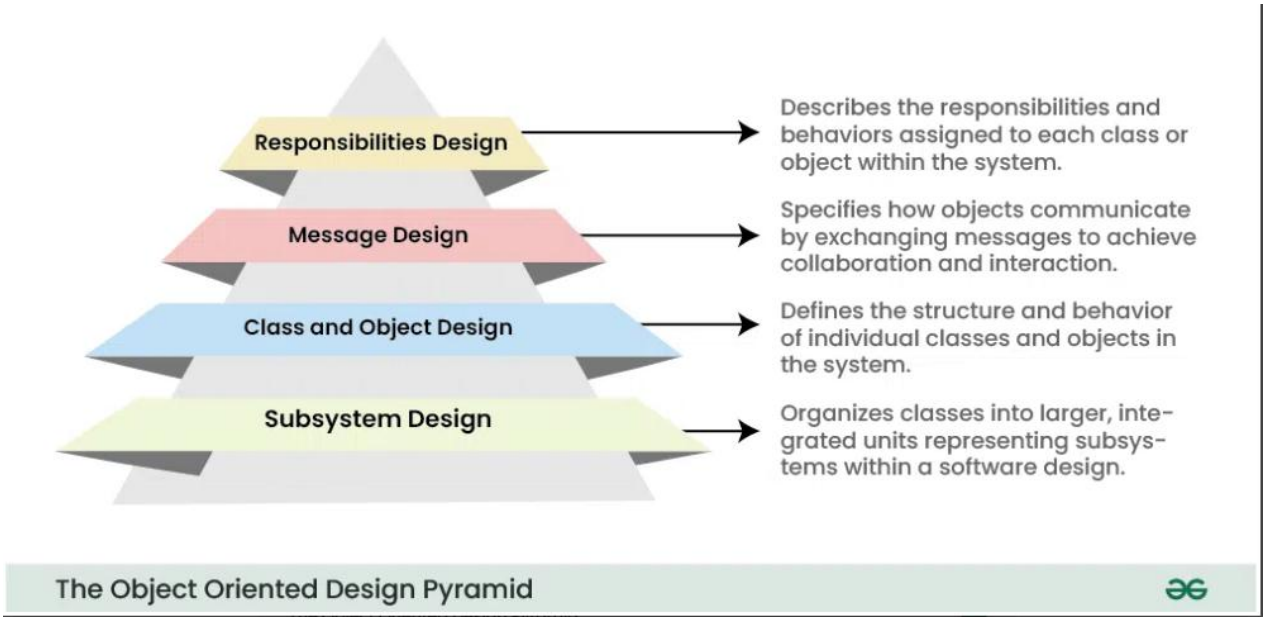
- **Data Organization of Attributes:**

- OOD involves specifying how data attributes are organized within the objects. This includes determining the types of data each object will hold and how they relate to one another, ensuring a coherent and efficient data structure.

- **Procedural Description of Operations:**

  - OOD requires a procedural description for each operation that an object can perform. This involves detailing the steps or processes involved in carrying out specific tasks, ensuring clarity and precision in the implementation of functionality.

Below diagram shows a design pyramid for object-oriented systems. It is having the following four layers.



The Object Oriented Design Pyramid

1. **The Subsystem Layer:** It represents the subsystem that enables software to achieve user requirements and implement technical frameworks that meet user needs.

2. **The Class and Object Layer:** It represents the class hierarchies that enable the system to develop using generalization and specialization. This layer also represents each object.

3. **The Message Layer:** This layer deals with how objects interact with each other. It includes messages sent between objects, method calls, and the flow of control within the system.

4. **The Responsibilities Layer:** It focuses on the responsibilities of individual objects. This includes defining the behavior of each class, specifying what each object is responsible for, and how it responds to messages.

Benefits of Object-Oriented Analysis and Design(OOAD)

- **Improved modularity:** OOAD encourages the creation of small, reusable objects that can be combined to create more complex systems, improving the modularity and maintainability of the software.
- **Better abstraction:** OOAD provides a high-level, abstract representation of a software system, making it easier to understand and maintain.
- **Improved reuse:** OOAD encourages the reuse of objects and object-oriented design patterns, reducing the amount of code that needs to be written and improving the quality and consistency of the software.
- **Improved communication:** OOAD provides a common vocabulary and methodology for software developers, improving communication and collaboration within teams.
- **Reusability:** OOAD emphasizes the use of reusable components and design patterns, which can save time and effort in software development by reducing the need to create new code from scratch.
- **Scalability:** OOAD can help developers design software systems that are scalable and can handle changes in user demand and business requirements over time.
- **Maintainability:** OOAD emphasizes modular design and can help developers create software systems that are easier to maintain and update over time.
- **Flexibility:** OOAD can help developers design software systems that are flexible and can adapt to changing business requirements over time.
- **Improved software quality:** OOAD emphasizes the use of encapsulation, inheritance, and polymorphism, which can lead to software systems that are more reliable, secure, and efficient.

Challenges of Object-Oriented Analysis and Design(OOAD)

- **Complexity:** OOAD can add complexity to a software system, as objects and their relationships must be carefully modeled and managed.

- **Overhead:** OOAD can result in additional overhead, as objects must be instantiated, managed, and interacted with, which can slow down the performance of the software.
- **Steep learning curve:** OOAD can have a steep learning curve for new software developers, as it requires a strong understanding of OOP concepts and techniques.
- **Complexity:** OOAD can be complex and may require significant expertise to implement effectively. It may be difficult for novice developers to understand and apply OOAD principles.
- **Time-consuming:** OOAD can be a time-consuming process that involves significant upfront planning and documentation. This can lead to longer development times and higher costs.
- **Rigidity:** Once a software system has been designed using OOAD, it can be difficult to make changes without significant time and expense. This can be a disadvantage in rapidly changing environments where new technologies or business requirements may require frequent changes to the system.
- **Cost:** OOAD can be more expensive than other software engineering methodologies due to the upfront planning and documentation required.

Real world applications of Object-Oriented Analysis and Design(OOAD)

Object-Oriented Analysis and Design (OOAD) has been widely applied across various industries to improve software development processes, enhance maintainability, and promote code reusability. Here are some real-world applications of OOAD:

1. **Financial Systems:** Banking Software: OOAD is often employed in banking systems to model complex financial structures, transactions, and customer interactions. The modular and scalable nature of OOAD helps in designing flexible and robust banking applications.
2. **Healthcare Systems:** Electronic Health Record (EHR) Systems: OOAD is utilized to model patient data, medical records, and healthcare workflows. Object-oriented principles enable the creation of modular and adaptable healthcare applications that can evolve with changing requirements.
3. **Aerospace and Defense:** Flight Control Systems: OOAD is crucial in designing flight control systems for aircraft. It helps model the interactions between different components such as navigation systems, sensors, and control surfaces, ensuring safety and reliability.

4. **Telecommunications:** Telecom Billing Systems: OOAD is applied to model and design billing systems in the telecommunications industry. It allows for the representation of complex billing rules, subscription plans, and customer data in a modular and scalable way.

5. **E-commerce:** Online Shopping Platforms: OOAD is commonly used in the development of e-commerce systems. It helps model product catalogs, user profiles, shopping carts, and payment processes, making it easier to maintain and extend the functionality of the platform.

Here is the rephrased Python code along with an explanation in English:

```
### Python Code (Object-Oriented Design Example)
# Define the Student class
class Student:
    def __init__(self, student_id, name, age):
        self.student_id = student_id
        self.name = name
        self.age = age
        self.courses = []  # List to store the courses the student is
enrolled in

    # Method to enroll the student in a course
    def enroll(self, course):
        self.courses.append(course)
        print(f"{self.name} has enrolled in {course.course_name}")

    # Method to list all courses the student is enrolled in
    def list_courses(self):
        print(f"{self.name} is enrolled in the following courses:")
        for course in self.courses:
            print(f"- {course.course_name}")

# Define the Course class
class Course:
    def __init__(self, course_id, course_name, instructor):
        self.course_id = course_id
        self.course_name = course_name
        self.instructor = instructor

    # Method to get information about the course
    def get_info(self):
        return f"Course: {self.course_name}, Instructor: {self.instructor}"
```

```
# Define the Instructor class
class Instructor:
    def __init__(self, instructor_id, name):
        self.instructor_id = instructor_id
        self.name = name

    # Method to return the instructor's name
    def get_name(self):
        return self.name

# Create some instructors
instructor1 = Instructor(1, "Dr. Ali")
instructor2 = Instructor(2, "Dr. Fatima")

# Create some courses
course1 = Course(101, "Math", instructor1.get_name())
course2 = Course(102, "Physics", instructor2.get_name())

# Create some students
student1 = Student(1, "Ahmed", 20)
student2 = Student(2, "Sara", 22)

# Enroll students in courses
student1.enroll(course1)
student1.enroll(course2)

student2.enroll(course2)

# List the courses each student is enrolled in
student1.list_courses()
student2.list_courses()

# Display information about the courses
print(course1.get_info())
print(course2.get_info())
```

https://www.programiz.com/online-compiler/2LhqrWDu2S6tQ

- Code Explanation

1. **Class `Student`**:
   - The `Student` class represents a student with attributes such as `student_id`, `name`, and `age`. It also contains a list to store the courses that the student is enrolled in.
   - The `enroll()` method allows a student to enroll in a course, adding it to their list of courses.
   - The `list_courses()` method prints all the courses the student is enrolled in.

2. **Class `Course`**:
   - The `Course` class represents a course with attributes like `course_id`, `course_name`, and the name of the `instructor`.

- The `get_info()` method returns information about the course, including its name and instructor.

3. **Class `Instructor`**:
   - The `Instructor` class represents an instructor with attributes like `instructor_id` and `name`.
   - The `get_name()` method returns the instructor's name.

4. **Creating Objects**:
   - Two instructors are created, `Dr. Ali` and `Dr. Fatima`.
   - Two courses are created: "Math" taught by `Dr. Ali` and "Physics" taught by `Dr. Fatima`.
   - Two students, `Ahmed` and `Sara`, are created with their respective details.

5. **Enrollment**:
   - `Ahmed` is enrolled in both the "Math" and "Physics" courses.
   - `Sara` is enrolled in the "Physics" course.

6. **Displaying Information**:
   - The courses each student is enrolled in are listed.
   - The course information (course name and instructor) is displayed.

Expected Output:

Ahmed has enrolled in Math
Ahmed has enrolled in Physics
Sara has enrolled in Physics
Ahmed is enrolled in the following courses:
- Math
- Physics
Sara is enrolled in the following courses:
- Physics
Course: Math, Instructor: Dr. Ali
Course: Physics, Instructor: Dr. Fatima

This example demonstrates how Object-Oriented Analysis and Design (OOAD) principles can be applied to create a basic system for managing students, courses, and instructors. Each class is responsible for specific tasks, making the code modular and easy to maintain.

- object-oriented Python program **Library Management System**. This program will manage books, library members, and the borrowing process.

```
### Library Management System Example in Python

# Define the Book class
class Book:
    def __init__(self, title, author, isbn, available=True):
        self.title = title
        self.author = author
```

```
        self.isbn = isbn
        self.available = available  # Book availability status

    # Method to mark a book as borrowed
    def borrow(self):
        if self.available:
            self.available = False
            return True
        return False

    # Method to mark a book as returned
    def return_book(self):
        self.available = True

    # Method to get book info
    def get_info(self):
        status = "Available" if self.available else "Not Available"
        return f"Title: {self.title}, Author: {self.author}, ISBN:
{self.isbn}, Status: {status}"

# Define the Member class
class Member:
    def __init__(self, member_id, name):
        self.member_id = member_id
        self.name = name
        self.borrowed_books = []  # List to store borrowed books

    # Method to borrow a book
    def borrow_book(self, book):
        if book.borrow():
            self.borrowed_books.append(book)
            print(f"{self.name} has borrowed the book '{book.title}'.")
        else:
            print(f"Sorry, the book '{book.title}' is not available.")

    # Method to return a book
    def return_book(self, book):
        if book in self.borrowed_books:
            book.return_book()
            self.borrowed_books.remove(book)
            print(f"{self.name} has returned the book '{book.title}'.")
        else:
            print(f"{self.name} hasn't borrowed the book '{book.title}'.")

    # Method to list all borrowed books
    def list_borrowed_books(self):
        if self.borrowed_books:
            print(f"{self.name} has borrowed the following books:")
            for book in self.borrowed_books:
                print(f"- {book.title}")
        else:
            print(f"{self.name} has not borrowed any books.")
```

```
# Define the Library class
class Library:
    def __init__(self, name):
        self.name = name
        self.books = []  # List to store all books in the library

    # Method to add a book to the library
    def add_book(self, book):
        self.books.append(book)
        print(f"The book '{book.title}' has been added to the library.")

    # Method to list all books in the library
    def list_books(self):
        print(f"Books available in {self.name} library:")
        for book in self.books:
            print(book.get_info())

# Creating the library
library = Library("Central Library")

# Adding books to the library
book1 = Book("The Great Gatsby", "F. Scott Fitzgerald", "1234567890")
book2 = Book("To Kill a Mockingbird", "Harper Lee", "2345678901")
book3 = Book("1984", "George Orwell", "3456789012")
library.add_book(book1)
library.add_book(book2)
library.add_book(book3)

# Creating library members
member1 = Member(1, "Alice")
member2 = Member(2, "Bob")

# Members borrowing books
member1.borrow_book(book1)
member1.borrow_book(book2)
member2.borrow_book(book2)  # This book is already borrowed by Alice

# Listing borrowed books for each member
member1.list_borrowed_books()
member2.list_borrowed_books()

# Returning a book
member1.return_book(book1)
member2.borrow_book(book1)  # Now Bob can borrow it

# Listing all books in the library
library.list_books()
```

https://www.programiz.com/online-compiler/1jRCukOBHv9OC

Code Explanation

1. **Class `Book`**:
   - This class represents a book with attributes like `title`, `author`, and `isbn`. It also has a `available` attribute to track whether the book is available for borrowing.
   - The `borrow()` method marks the book as borrowed, and `return_book()` marks it as returned.
   - The `get_info()` method returns information about the book including its availability status.

2. **Class `Member`**:
   - This class represents a library member with a `member_id` and `name`.
   - It has a list `borrowed_books` to track the books borrowed by the member.
   - The `borrow_book()` method allows the member to borrow a book, while `return_book()` allows the member to return a book.
   - The `list_borrowed_books()` method prints all books borrowed by the member.

3. **Class `Library`**:
   - The `Library` class represents the library itself, with a name and a list of books.
   - It has methods to `add_book()` to add books to the library and `list_books()` to list all books with their status.

4. **Creating Objects**:
   - A library named "Central Library" is created, and books such as "The Great Gatsby", "To Kill a Mockingbird", and "1984" are added to it.
   - Members "Alice" and "Bob" are created.

5. **Borrowing and Returning**:
   - Alice borrows two books, and Bob tries to borrow a book that Alice already has, which isn't allowed.
   - Alice returns one book, and Bob successfully borrows it afterward.

6. **Displaying Library Status**:
   - The library lists all the books and their availability status.

Expected Output:

The book 'The Great Gatsby' has been added to the library.
The book 'To Kill a Mockingbird' has been added to the library.
The book '1984' has been added to the library.
Alice has borrowed the book 'The Great Gatsby'.
Alice has borrowed the book 'To Kill a Mockingbird'.
Sorry, the book 'To Kill a Mockingbird' is not available.
Alice has borrowed the following books:
- The Great Gatsby
- To Kill a Mockingbird
Bob has not borrowed any books.
Alice has returned the book 'The Great Gatsby'.
Bob has borrowed the book 'The Great Gatsby'.
Books available in Central Library library:
Title: The Great Gatsby, Author: F. Scott Fitzgerald, ISBN: 1234567890, Status: Not Available
Title: To Kill a Mockingbird, Author: Harper Lee, ISBN: 2345678901, Status: Not Available
Title: 1984, Author: George Orwell, ISBN: 3456789012, Status: Available

This program demonstrates another practical example of Object-Oriented Design where classes and methods are used to simulate real-world operations of a library system.