1. **Animals Class Diagram**

Create a Class Diagram involving Animals. All animals have name and age. All animals can make sound, but its implementation is only made on the specific subclass.

All Mammals can provide milk. Specific mammals include Cat, Dog, Whale, Bat, Platypus.
All Fishes can swim.
All Birds can fly. Birds include Penguin.

Have interfaces to accommodate Flyer that can fly (returns nothing), Swimmer that can swim (returns nothing), and EggLayer that can lay eggs which will return the average number of eggs.
Flyers include Birds and Bat.
Swimmers include Fishes, Penguin, Whale, and Platypus.
EggLayers include Birds, Fishes, and Platypus.

2. **Employees Class Diagram**

All kinds of people have two fields in common, name and age. The name can only be set once, but the age can only be updated when they have their birthdays. These fields are required when creating an instance of a person. These fields should not be accessed outside their class, but the methods that can change and access them can be used anywhere.

Employees have their salary and ID number. ID numbers can only be set once. There are restrictions for setting the salary such that it should be above the minimum wage, that is PHP501.25. Upon construction of the employee, it may have both ID and salary, or only ID. If only ID, set its salary to the minimum wage. The methods that change and access them can be used anywhere.

Customers can purchase an item, and when it does, output “[name] buys [item] worth [amount]” if the given parameters are a String and a double type. Otherwise, it outputs “[name] buys [number] pieces of [item]” if the given parameters are a String and an integer type. These methods are named “purchase” and will not return anything.

Both the developer and a manager can work. However, when a manager works, it instructs a given developer to work by outputting, “Hi! I am [manager’s name]. Can you program this for me, [developer’s name]?“ and invokes the developer’s work method and raises the developer’s salary by PHP25.60. When the developer works, it simply outputs “I am [name] and I am programming.

3. **Library System**
    a. Use Case

They will begin by identifying key **actors** such as **Students, Librarians, and Administrators**, and then define essential **use cases** that represent system functionality, including **Search for Books, Borrow Book, Return Book, Reserve Book, Pay Fines, Manage Student Accounts, and Generate Reports**. Students will determine the relationships between actors and use cases, such as how **Students interact with borrowing and returning books**, while **Librarians manage inventory and student records**.

b. Class Diagram

The diagram includes key classes such as **User, Student, Librarian, Book, Transaction, and LibrarySystem**.

- The **User** class acts as a superclass for **Student** and **Librarian**, containing attributes like `userID`, `name`, and `email`, with methods such as `login()` and `logout()`.
- The **Book** class has attributes like `bookID`, `title`, `author`, and `availabilityStatus`, along with methods such as `checkout()`, `returnBook()`, and `reserve()`.
- The **Transaction** class records borrowing and returning actions, linking a **Student** to a **Book** with attributes like `transactionID`, `borrowDate`, and `dueDate`.
- The **LibrarySystem** class manages overall operations, including `searchBook()`, `addBook()`, and `generateReport()`.

Relationships between classes include **inheritance (User → Student, Librarian)**, **associations (Student borrows Book, Librarian manages Books)**, and **aggregation (LibrarySystem contains Books and Users)**.

c. Sequence Diagram

A **Sequence Diagram** for a **School Library System** illustrates the step-by-step interaction between different entities to complete a specific process, such as **borrowing a book**. It shows the **actors** (e.g., Student, Librarian) and **system components** (e.g., Library System, Book Database) exchanging messages in a time-ordered sequence. The process begins when a **Student searches for a book**, and the **Library System** queries the **Book Database** for availability. If the book is available, the student requests to borrow it, and the system checks their borrowing eligibility (e.g., no overdue books or borrowing limits exceeded). Once verified, the **Library System** updates the book's status, records the transaction, and notifies the student of a successful loan.