ENPM808X Assignment-1

Aditya Khopkar | UID 116911627

1. **What is inheritance in object-oriented technology? Give an example.**

   Inheritance in object-oriented technology is a mechanism of basing an object or a class (derived) on another object or class (parent), while retaining some of the attributes and properties. For example, a Person class is defined which has the attributes such as age, name and id. Classes such as Student, Professor may be inherited from the base class which has the similar attributes as that of the base class along with some additional individual properties/attributes such as name of program, etc.

2. **What is the difference between an object and a class in OO technology?**

   A class is a container of the object, the class contains the object properties and details about the behaviour of the object. An object is an instance of the class, i.e., the object is a variable that represents the class.

3. **Describe the role of polymorphism in object-oriented technology. Give an example.**

   Polymorphism simply means having multiple forms. Polymorphism is used to re-use the same class/function/operator in different forms. There are two types of polymorphism- Static Polymorphism (method overloading, constructor overloading, etc) :

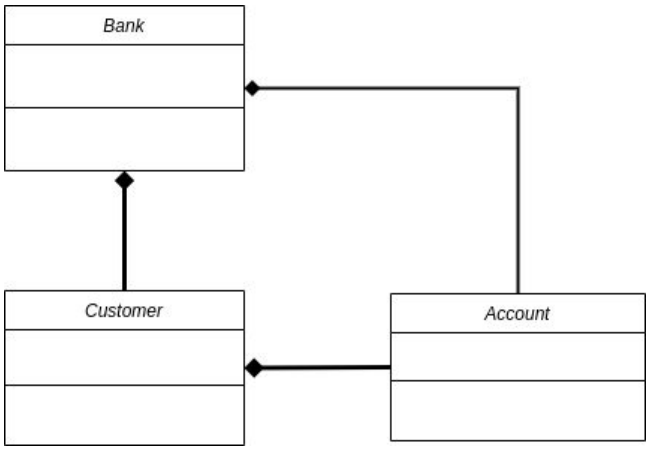
```
void add(int& a, int& b);
void add(int& a, int& b, int& c); // add used to add two numbers as well as 3 numbers
```

   and Dynamic Polymorphism (Runtime polymorphism): In a hierarchical structure (child-parent relationship) when there are methods in different forms, the correct reference to the method may not be called during run-time, to avoid this virtual overriding is performed which is a type of Dynamic Polymorphism to call the correct method during run-time. Let's say a Parent class Animal has a method make_sound() which prints "bark!", a child class Cow has a method make_sound() which should print "mooh", this is achieved by virtual overriding.

```
class Animal{
public:
        virtual void make_sound(){ std::cout << "bark!" << std:: endl; }
}
class Cow:public Animal{
public:
```

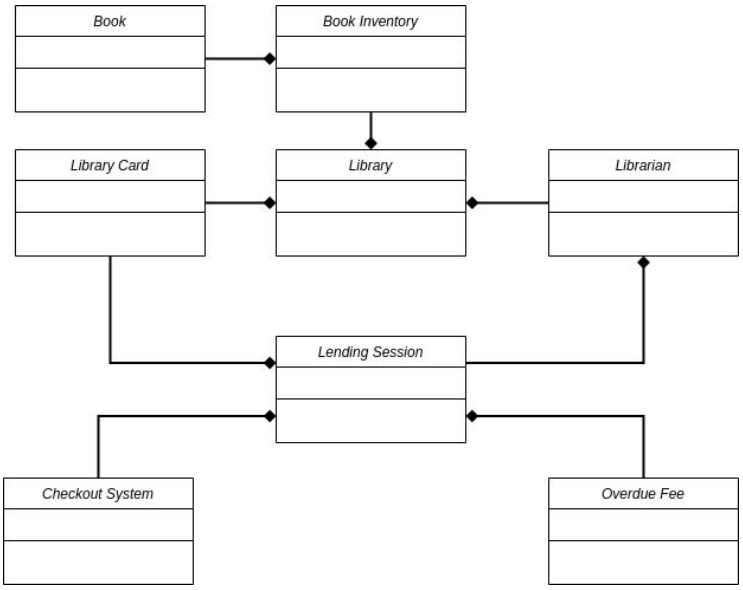
```
                    void make_sound(){ std::cout << “mooh”<<std::endl; }
}
```

4. **Draw a class diagram of a small banking system showing the associations between three classes: the bank, customer, and the account.**



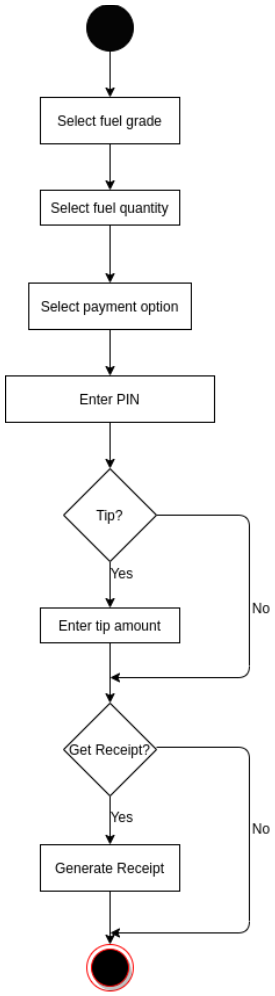
Bank*:* This is the central class, contains information about the customers and their accounts along with transaction details.
Customer*:* Each customer is associated with the bank and contains a unique account number.
Account*:* Account is made by the bank and thus has association with the bank and is unique for a customer and thus, is associated to the customer as well.

5. **Draw a class diagram of a library lending books using the following classes: Librarian, Lending Session, Overdue Fine, Book Inventory, Book, Library, Checkout System, and Library Card.**



***Library**:* Central Unit of the organization, contains the book inventory, a librarian and library card.
***Librarian**:* Directly associated with the library and responsible for keeping track of the lending session.
***Book Inventory**:* Contains a database of all the books in the library.
***Library Card**:* Issued from the library to each customer and is used in the lending session.
***Book**:* An item in the book inventory.
***Lending Session**:* The central action unit of the system. Lending the books and payment of the book is done in this action and is administered by the librarian.
***Overdue Fee**:* A fee charged through the lending session for overdue in returning a book.
Checkout System: Charge when a book is checked out from the library.

6. **Draw an activity diagram of pumping gas and paying by credit card at the pump. Include at least five activities, such as "Select fuel grade" and at least two decisions, such as "Get receipt?"**



7. **Explain how class dependency graph differs from a UML class diagram.**

   A class dependency graph is a directed graph where classes are the nodes and the edges are the dependencies. Class dependency graphs shows all the possible relationships of classes with each other in the form of dependencies, for example, if there is an ‘is a’ relationship between class A and B, the dependency is established as <A,B>, similarly a polymorphic relationship is also represented between class <X,Y>. Such relational information is not well represented in UML class diagrams. The UML diagrams may give an idea of what the class is and how the classes are interacting among each other through the program, but little does it tell about the dependencies of these classes with other classes.