Programming Exercises

Software Engineering

3.13 → What is inheritance in object-oriented technology? Give an example.

Inheritance is a core concept in the object oriented programming paradigm which provides a mechanism for establishing relationships & building hierarchies. It's a programming procedure that helps programmers reuse code by referencing behaviours & attributes of an object. It provides a method of arranging items (objects) from most general to most specific.

An object that inherits from another object is considered to be a subtype of that object, for example the "Person" class could be considered a base class from which more specific sections like a "Teacher" or "Student" class could be derived.
A class that inherits could be called the sub/derived/child class whereas the other class being inherited from could be called a base/parent/super class..

Example:

```
class Animal {
public:
void eat() {
    cout << "can eat";
    }
void sleep() {
    cout << "can sleep";
    }
};

class Dog : public Animal {

public:
    void bark() {
        cout << "Bark";
    }
};
```

← Base class Animal has methods general to all animals
↳ Could add methods like set name, etc which are common to most dogs.

← Derived class Dog has methods more specific to dogs, such as barking

3.14 → What is the difference between an object & a class in OO technology?

- A class a template for creating objects, like a blueprint for creating objects whereas objects are the instances of a class. All functionalities and data members of a class can only be accessed when an object is created.

- A class is a logical entity, an object is the manifestation of the class in a program.

- class by itself does not hold values as memory isn't allocated when classes are made, only when objects are made and instantiated does the memory get allocated and values are assigned & set.

3.15 → Describe the role of polymorphism in object oriented technology. Give an example.

Polymorphism is a core concepts of the Object Oriented Programming paradigm which allows an object to take on many forms. Inheritance allows us to inherit attributes & methods from another class, and polymorphism uses the methods to perform different tasks.
Polymorphism in C++ allows for classes & functions to perform in different ways, depending on the usage. It allows us to reuse code by creating one function for multiple use.
It's generally categorised into 2 types →
1. Compile time polymorphism
2. Runtime polymorphism

Compile time polymorphism →
1. Function overloading
Considering a function "add" →

```
int add (int x, int y) {
    return x + y
}

double add (double a, double b) {
    return a + b;
}
```

* Both functions are called add but take different parameters and perform different actions based on the input.

2. operator overloading

Defines additional tasks to operators without changing the meaning.

→ Defn :→

```
class complex_numbers {

    private :
        int real , img;
    public :
        complex_number operator + (complex_number const & object ) {
            complex_number res;
                res.real = real + obj_real
                res.img = img + obj.img;
                return res;
        }
}
```

→ Usage → complex_numbers C1(10,5), C2(2,4);

```
    complex_number C3 = C1 + C2;
}
```

(The operator + is overloaded with a new behaviour defined in the class.

Runtime polymorphism:

Virtual functions & pointers make up function overriding & affects the program @ runtime as the program decides @ runtime.

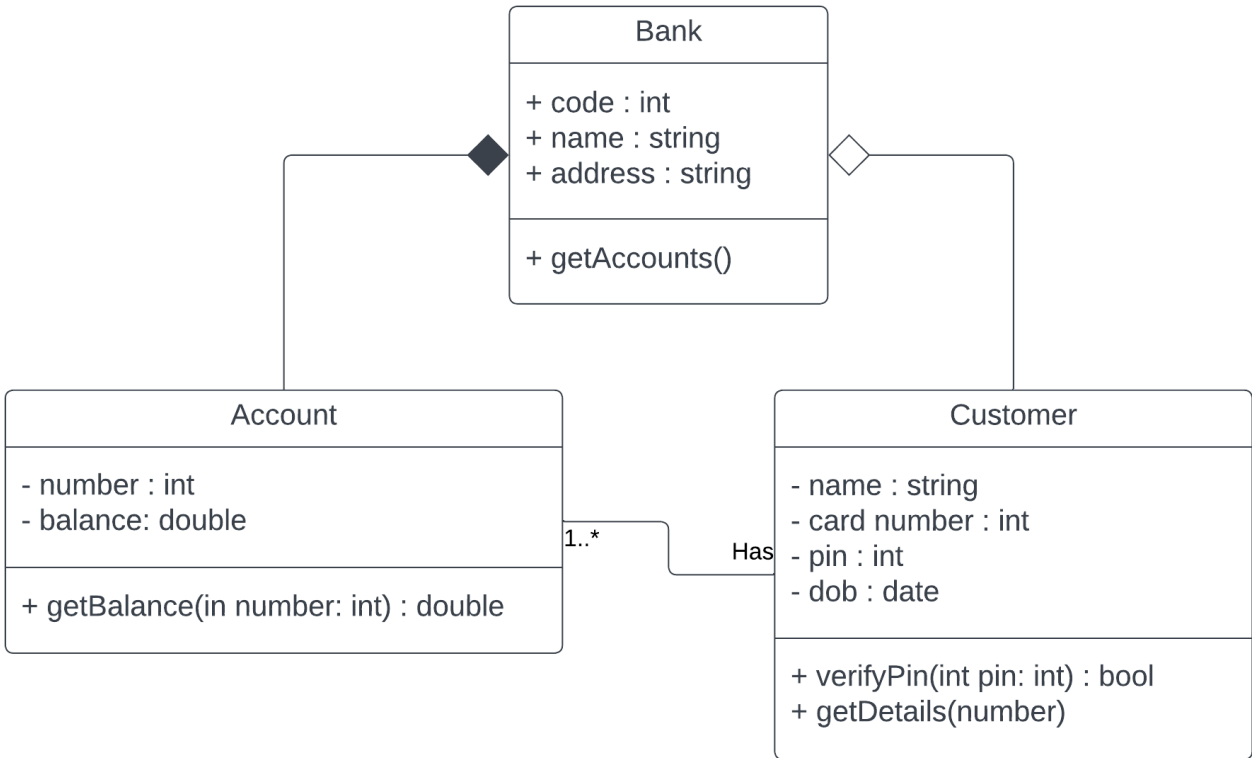
Example:

```
class Animal {
public:
    virtual void sound () = 0;
};

class dog : public Animal {

public
void sound () {
    cout << "bark";
}
};
```

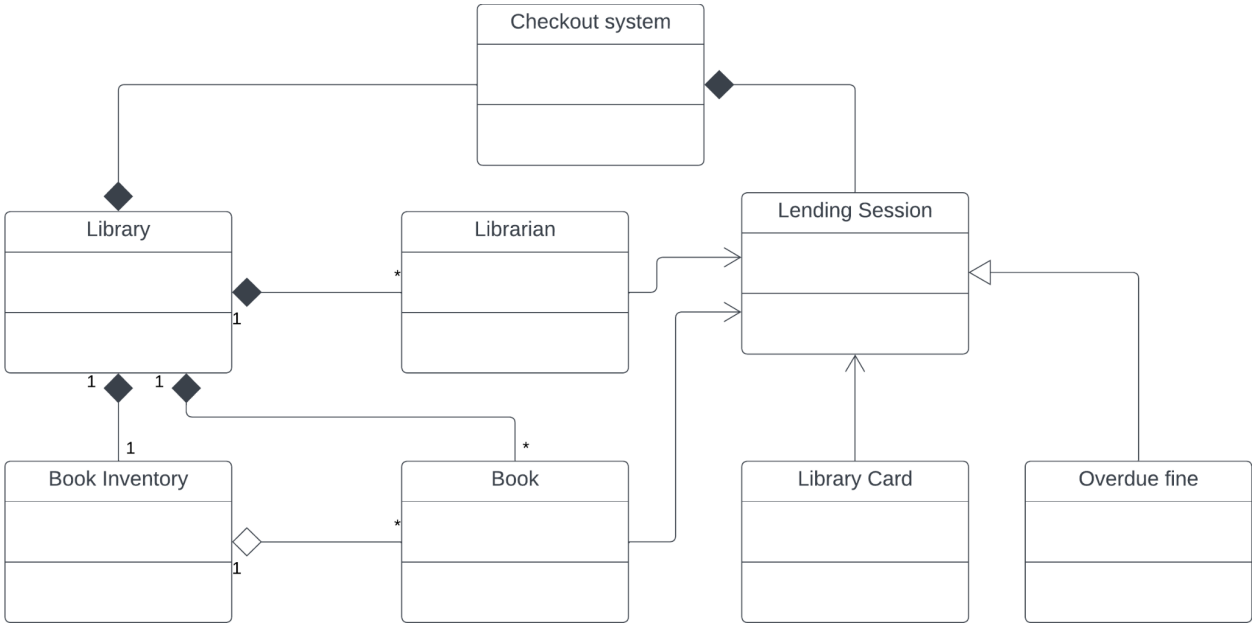
→ Cannot define sound for all animals

→ Function overriden to suit specific needs of an animal.

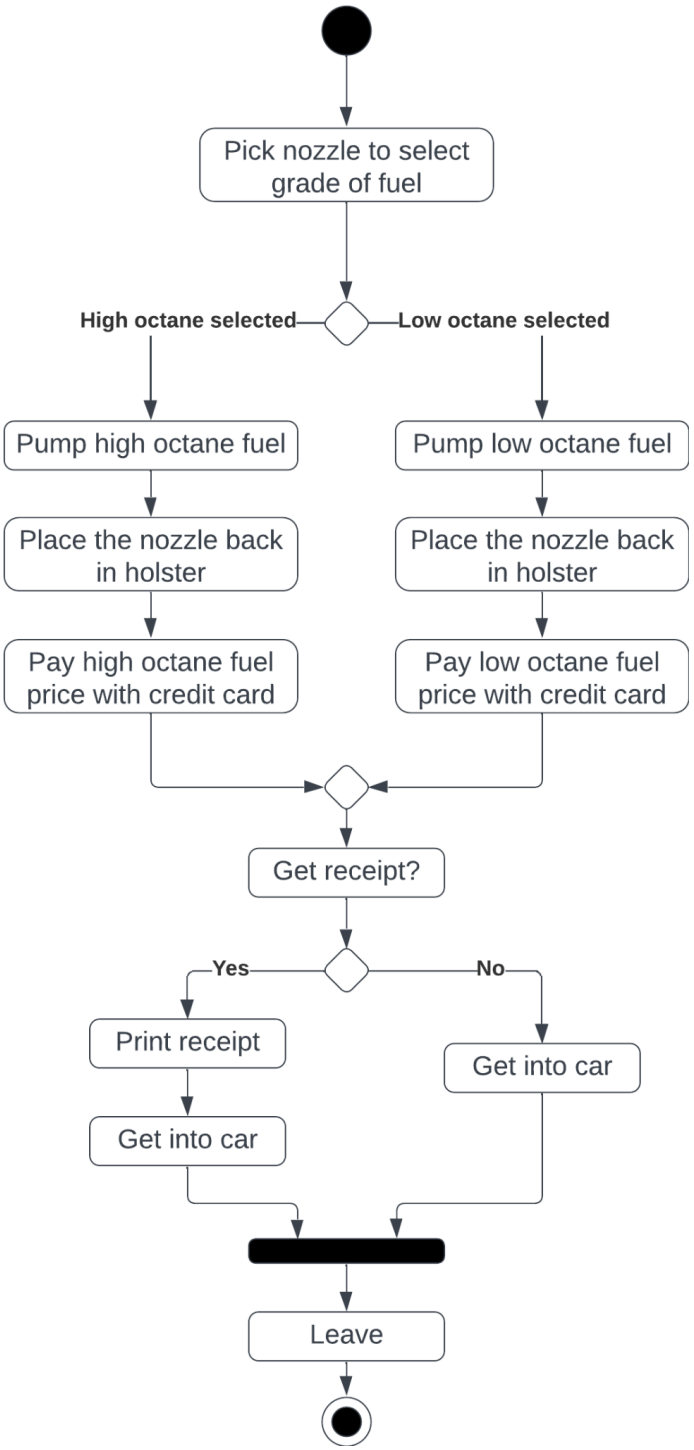
4.1 UML Class Diagram for Bank, Account and Customer



4.2 UML Class Diagram for a library lending books



4.3 UML Activity Diagram for fuel pumping and payment activity



4.5.

A class dependency diagram / graph is a model which highlights responsibility by using a similar structure to the UML class diagram, but differs in highlighting responsibility by dependencies.

It is a directed graph which highlights classes as nodes and dependencies by broken / directed arrows.

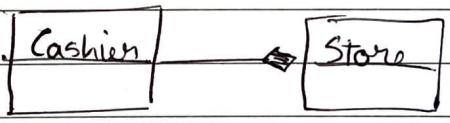
It adds a supplier / client relationship to the basic UML class diagram.

If a class A expects a responsibility from class B for a process, class a is the client and class B is the supplier. Class A depends on class B & is denoted by <A, B>.

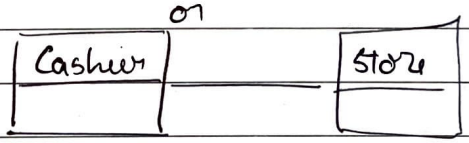
Part of relationships is <A, B> if A has a part of relation with B.

If a non polymorphic relationship exists Then there both <A, B> & <B, A> as dependencies

Class diagram:



or



Class dependency: