ENPM808X: Software Development for Robotics

Srinidhi Sreenath UID: **115526723**

September 11, 2018

1. **Software Engineering 3.13**
   What is inheritance in object-oriented technology? Give an example

   Inheritance is a relation between two classes. It is the mechanism of basing an object or class upon another object or class, retaining similar implementation.

   Example: Consider a class named robot defined as follows:

```
class Robot{
    private:
    int degreesOfFreedom;

    public:
    double calculatePowerToRun();
    }
```

   This class defines a generic robot. A derived class can be a wheeled robot defined as follows:

```
class WheeledRobot : public Robot {
    private:
    int numberOfWheels;

    public:
    double calculatePowerForEachWheel();
    }
```

   class WheeledRobot inherits from class Robot the variable degreesOfFreedom, and the method move calculatePowerToRun. Moreover, the WheeledRobot class contains its own specific parameter numberOfWheels and a method calculatePowerForEachWheel. .

2. **Software Engineering 3.14**
   What is the difference between an object and a class in OO technology?

   Class is a blueprint or template from which objects are created. Object is an instance of a class. A class defines object properties including a valid range of values, and a default value.

3. **Software Engineering 3.15**
   Describe the role of polymorphism in object-oriented technology. Give an example

   Polymorphism is a construct of object-oriented technology that allows the use of a derived type in the place of the base type. It is the ability to present the same interface for differing underlying forms (data types).

Example: Consider a class named robot defined as follows:

```
class Robot{
    private:
    int degreesOfFreedom;

    public:
    double calculatePowerToRun();
    virtual void printTypeofRobot() {}
    }
```

Consider two derived classes WheeledRobot,

```
class WheeledRobot : public Robot {
    private:
    int numberOfWheels;

    public:
    void printTypeofRobot() {std::cout << "Wheeled Robot\n" << std:endl;}
    double calculatePowerForEachWheel();
    }
```

and FlyingRobot

```
class FlyingRobot : public Robot {
    private:
    int numberOfMotors;

    public:
    void printTypeofRobot() {std::cout << "Flying Robot\n" << std:endl;}
    double calculatePowerForEachMotor();
    }
```

This is an example of dynamic linking type Polymorphism using virtual methods, where when virtual methods are called and the object that calls it is of the derived type, then the derived type's method is called instead of the base type's method.

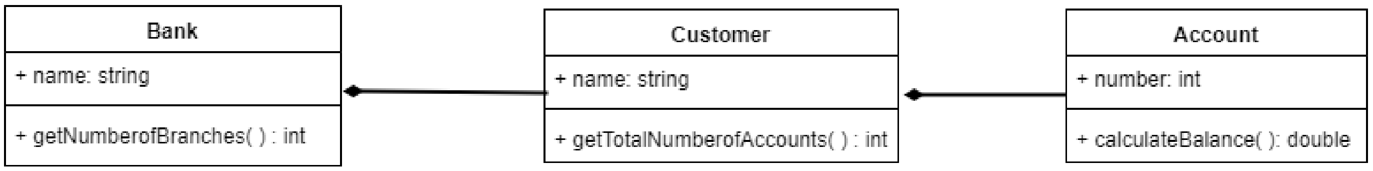
When the following code is executed,

```
Robot* myRobot[3];
myRobot[0] = new WheeledRobot;
myRobot[1] = new FlyingRobot;
myRobot[2] = new WheeledRobot;
for(size_t i = 0; i < 3; i++){
ourAnimal[i] -> printTypeofRobot();
}
```

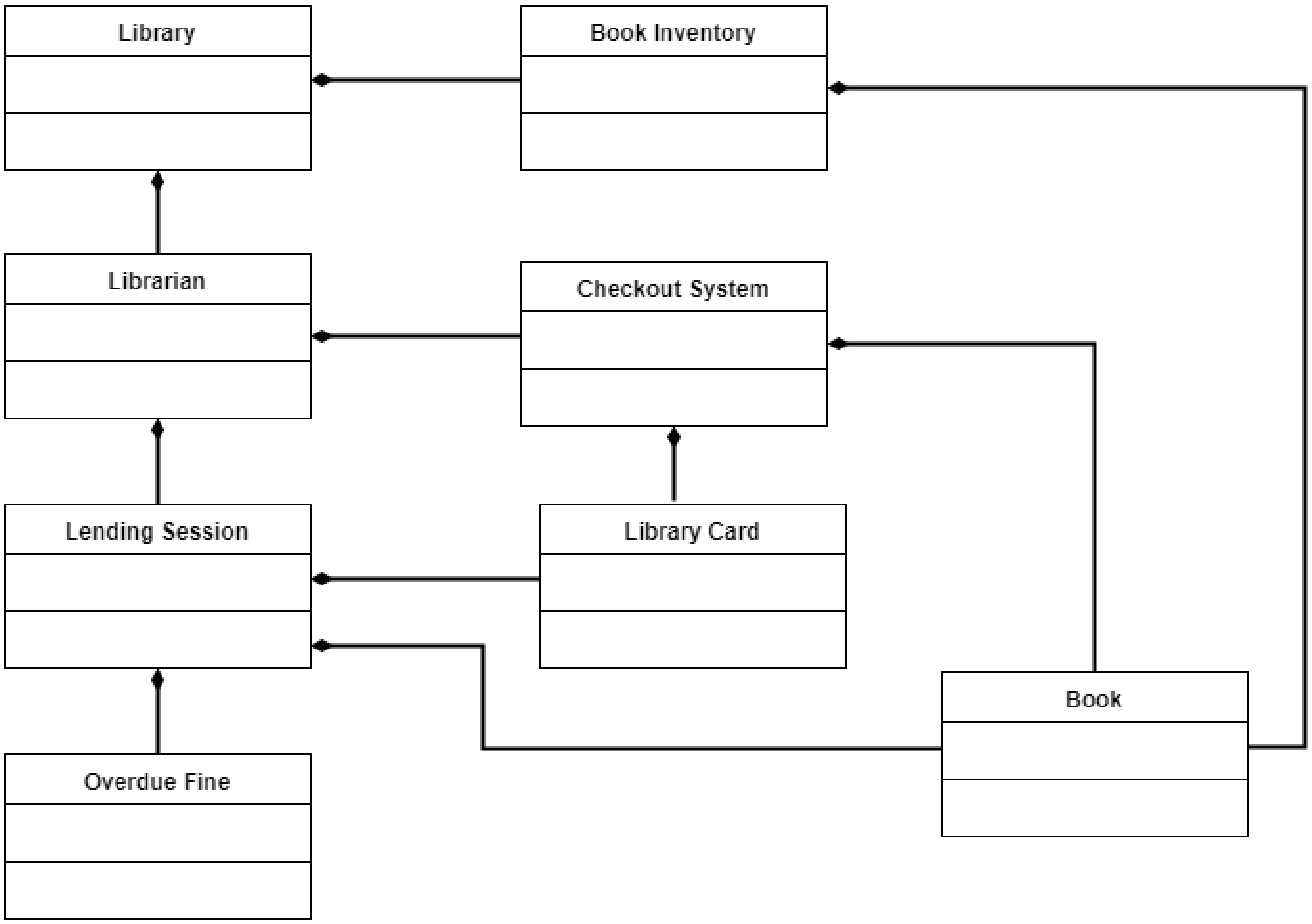
The output will be

```
Wheeled Robot
Flying Robot
Wheeled Robot
```

4. **Software Engineering 4.1**
Draw a class diagram of a small banking system showing the associations between three classes: the bank, customer, and the account.


Bank
+ name: string
+ getNumberofBranches( ) : int
Customer
+ name: string
+ getTotalNumberofAccounts( ) : int
Account
+ number: int
+ calculateBalance( ): double

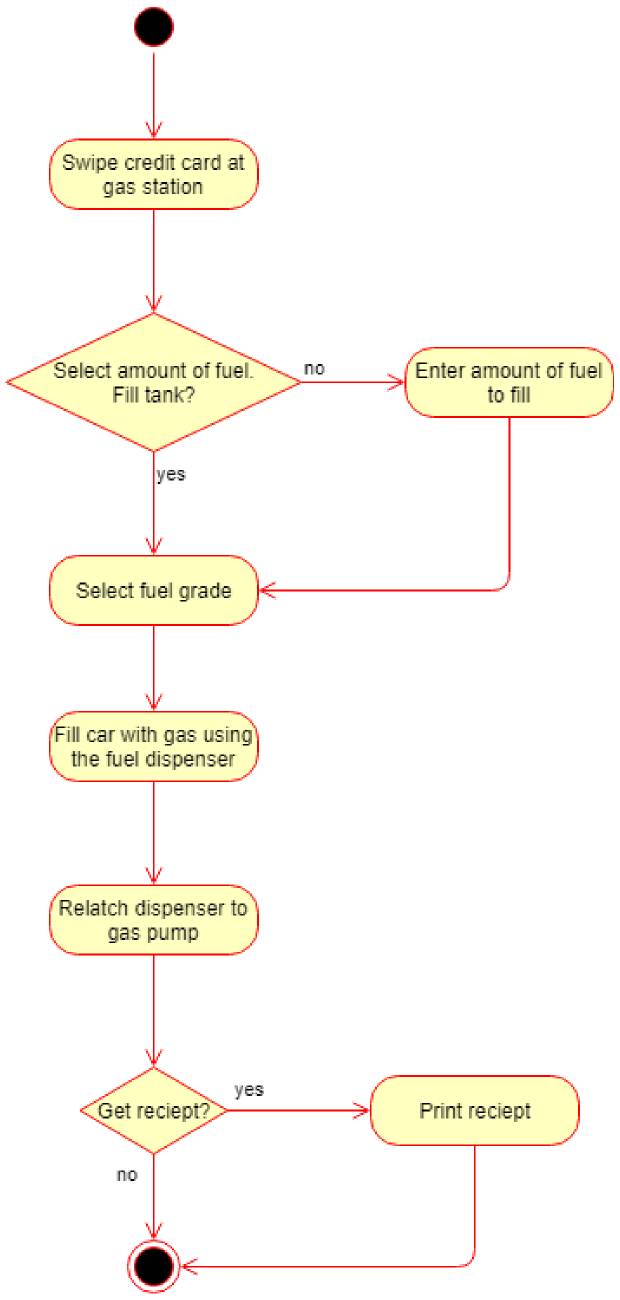
5. **Software Engineering 4.2**
Draw a class diagram of a library lending books using the following classes: Librarian, Lending Session, Overdue Fine, Book Inventory, Book, Library, Checkout System, and Library Card.



- **Librarian**: Contains information about librarian and his/her functions which include handling lending sessions and managing checkout system. Part of library.
- **Lending Session**: Contains information regarding a lending session like person, duration, book, fine etc. Part of librarian's function.
- **Overdue Fine**: Contains information on the fine amount and its calculation. Part of lending session.

- **Book Inventory**: Contains all books' information. Part of library's information.
- **Book**: Contains information on the book like name and genre etc. Part of Lending session, checkout system as well as book inventory.
- **Library**: Contains information about library.
- **Checkout System**: Contains information about the book being checked out, type of checkout system, generating receipt. Part of librarian's functions.
- **Library Card**: Contains information on the person lending the book. Part of lending session and checkout system.

6. **Software Engineering 4.3**
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?"



Swipe credit card at gas station
Select amount of fuel. Fill tank?
no
Enter amount of fuel to fill
yes
Select fuel grade
Fill car with gas using the fuel dispenser
Relatch dispenser to gas pump
Get reciept?
yes
Print reciept
no

7. **Software Engineering 4.5**
Explain how a class dependency graph differs from a UML class diagram

- Class dependency graph emphasizes relations between classes as **supplier** and **client**, whereas as UML defines relationship between classes as **part-of** and **is-a**.
- *Responsibilities* of classes are easier to determine and track in class dependency graphs. A class can have **local responsibility** or *supplier slice* and *customer slice* can also be determined to define **combined responsibility** or **contracts**. Such relation cannot be easily determined from an UML diagram.
- Class dependency graph is similar in graphical notation to UML diagrams but represented as a directed graph.
- UML diagrams are more popular than class dependency graphs although they may be less accurate/ appropriate in some cases.