

CS1020: Data Structures and Algorithms I

Tutorial 2 – Advanced Object Oriented Concepts

(3rd February 2012)

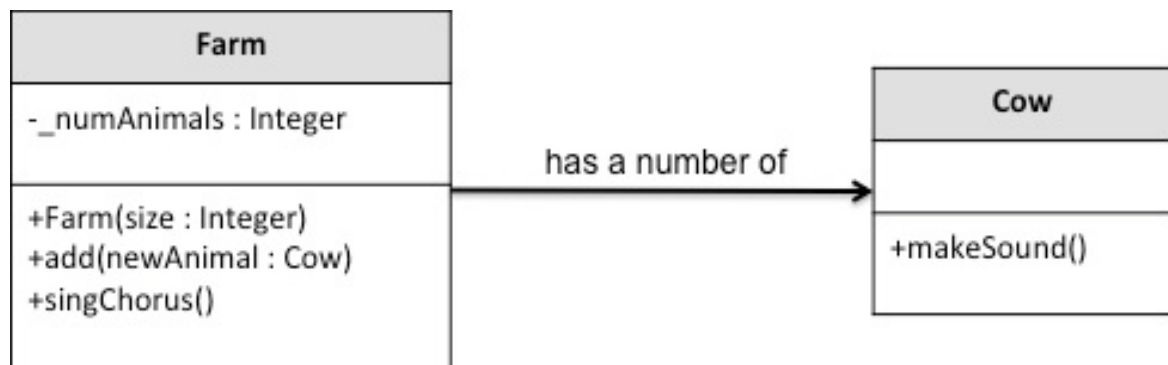
1. **[Polymorphism]** There is a nursery rhyme "Old McDonald" to teach young kids about animals and the sound they make:

```
Old McDonald had a farm, E-Ya-E-Ya-O
And, on his farm he has a Cow, E-Ya-E-Ya-O
There is a "Moo Moo" here, and a "Moo Moo" there
.....
```

Mr. Kidd attempts to model the nursery rhyme using the UML Class Diagram as shown below:

Notes:

- The symbols to the left of the variables and methods denote their visibility. A '-' sign denotes Private, a '#' sign denotes Protected, and a '+' sign denotes Public.
- You may follow this link to read up more on access control:
<http://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html>
- The 'has a number of' arrow between the two classes is an association, not an inheritance.
- You may also read more about UML Class Diagram in the following link:
<http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/>



His idea is to allow a Farm object to store a number of Cow objects. Whenever the **singChorus()** method is invoked, all cows on the farm will make the "Moo" sound. His implementation can be found in **OldMcDonaldCow.java**. Below is a sample run (user input in **bold**):

```
Choose 1. Add more cow, 2. Farm chorus, 3. exit -> 1
Cow added!

Choose 1. Add more cow, 2. Farm chorus, 3. exit -> 2
All farm animals start to sing:                (Only 1 cow on farm)
Mooo Mooo

Choose 1. Add more cow, 2. Farm chorus, 3. exit -> 1
Cow added!

Choose 1. Add more cow, 2. Farm chorus, 3. exit -> 1
Cow added!

Choose 1. Add more cow, 2. Farm chorus, 3. exit -> 2
All farm animals start to sing:                (A total of 3 cows now)
Mooo Mooo
Mooo Mooo
Mooo Mooo
```

CS1020: Data Structures and Algorithms I

```
Choose 1. Add more cow, 2. Farm chorus, 3. exit -> 3
Bye bye!
```

Mr. Kidd realized later that the farm is supposed to have several types of animals instead of just cows! He would like to expand his program to handle at least two more types of animals, the chicken (which goes "Pook Pook") and the pig (which goes "Oink Oink"). Here's a possible sample run of the updated program (user input in **bold**):

```
Choose 1. Add more animal, 2. Farm chorus, 3. exit -> 1
    Add: 1. Cow, 2. Chicken, 3. Pig -> 2
Animal added!

Choose 1. Add more animal, 2. Farm chorus, 3. exit -> 1
    Add: 1. Cow, 2. Chicken, 3. Pig -> 3
Animal added!

Choose 1. Add more animal, 2. Farm chorus, 3. exit -> 1
    Add: 1. Cow, 2. Chicken, 3. Pig -> 1
Animal added!

Choose 1. Add more animal, 2. Farm chorus, 3. exit -> 1
    Add: 1. Cow, 2. Chicken, 3. Pig -> 2
Animal added!

Choose 1. Add more animal, 2. Farm chorus, 3. exit -> 2
All farm animals start to sing:
Pook Pook
Oink Oink
Mooo Mooo
Pook Pook

Choose 1. Add more animal, 2. Farm chorus, 3. exit -> 1
    Add: 1. Cow, 2. Chicken, 3. Pig -> 3
Animal added!

Choose 1. Add more animal, 2. Farm chorus, 3. exit -> 2
All farm animals start to sing:
Pook Pook
Oink Oink
Mooo Mooo
Pook Pook
Oink Oink

Choose 1. Add more animal, 2. Farm chorus, 3. exit -> 3
Bye bye!
```

Your task:

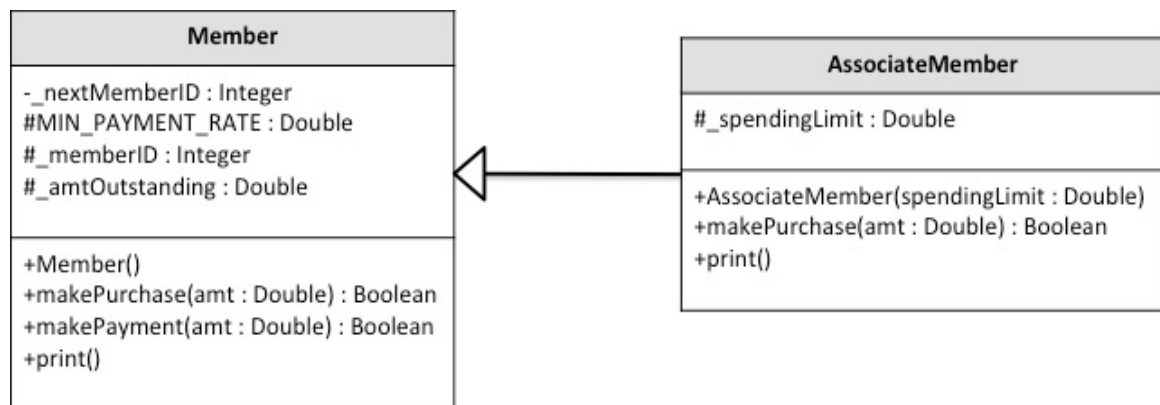
- Modify the UML class diagram to include 3 more classes. The first class is the **Animal** class which consists of the generic *makeSound* method. The next two classes are a **Chicken** class and **Pig** class that have their own unique *makeSound* method. Please take note that the Cow, Chicken and Pig classes should be subclasses of the Animal class.
- Write a Java source code to accommodate the new types of animals as shown. You are encouraged to apply inheritance and polymorphism in your solution.

CS1020: Data Structures and Algorithms I

2. **[Inheritance]** ABC Departmental Store is introducing a new Membership System to reward their customers. Similar to a credit card, a *member* is allowed to make purchases with this membership card, and the total outstanding amount will be billed to the member at the end of the month. All members have to pay a minimum of 20% of the amount outstanding to maintain their membership.

There are three types of memberships. The first type is an *Ordinary Membership*, which is the most basic membership. The second type is an *Associate Membership*, which has a spending limit. The third type is a *Platinum Membership* that does not have such restriction. In addition, a *Platinum Member* will enjoy a discount off the outstanding balance.

The following is part of a UML Class Diagram for the Membership System.



An ordinary member, simply known as **Member**, has the following information:

- Member Identification Number (generated by Member class)
- Amount Outstanding

The functionalities provided by a **Member** class are:

- Make Purchase
 - Ensure amount is positive.
 - Increases the amount outstanding.
- Make Payment
 - Minimum payment is 20% of amount outstanding.
 - Decreases the amount outstanding.
- Print
 - Print out the Member ID and Amount Outstanding.

CS1020: Data Structures and Algorithms I

The second type of member, an **Associate Member**, has the following information:

- Member Identification Number
- Amount Outstanding
- Spending Limit (Different Associate Members may have different limits)

The functionalities provided by an **Associate Member** class are:

- Make Purchase
 - Ensure amount is positive.
 - Ensure that the amount outstanding doesn't exceed the spending limit.
 - Increases the amount outstanding.
- Make Payment
 - Similar to its super class.
- Print
 - Print out the Member ID, Amount Outstanding and Spending Limit.

Your task:

Suppose we want to model the third type of membership: a **Platinum Member** that has the following information:

- Member Identification Number
- Amount Outstanding
- Fixed Discount Rate (It will be the same for all Platinum Members)

The functionalities provided by a **Platinum Member** class are similar to a Member class, with minor differences in the behavior:

- Make Purchase
 - Similar to its super class.
- Make Payment
 - Discount will be applied before calculating the minimum 20% payoff.
 - Decreases the amount outstanding.
- Print
 - Print out the Member ID, Amount Outstanding and Discount Rate.

- a) Modify the UML class diagram to include the **PlatinumMember** class.
- b) Use inheritance to implement a **PlatinumMember** class. You are free to write a main method to test the new class.

CS1020: Data Structures and Algorithms I

3. **[Object Oriented Modeling]** In *Question 2*, we have designed the UML class diagram for the Membership System. With these user-defined data types, we are now ready to build a *simplified version* of the Membership System Application for ABC Departmental Store which contains only the Member class from *Question 2*. We need to write a program to serve as a “Membership System” which provides the following services:
- Create new members.
 - Make Purchase/Payment.
 - Print information of all existing members.

Assumptions:

- The user will create no more than 100 members in a single test run.
- The account number will start from 1000, incremented by 10 for each subsequent new member.

An example test run session is given below (user input in **bold**):

```
*****
*      Membership System of CS1020      *
*****
Service available:
1. New Member, 2. Make Purchase, 3. Make Payment, 4. Print All, 5. Exit

Your choice: 1

Result: Member added successfully!
Member's Identification Number: 1000
Amount Outstanding: $0.00

*****
*      Membership System of CS1020      *
*****
Service available:
1. New Member, 2. Make Purchase, 3. Make Payment, 4. Print All, 5. Exit

Your choice: 1

Result: Member added successfully!
Member's Identification Number: 1010
Amount Outstanding: $0.00

*****
*      Membership System of CS1020      *
*****
Service available:
1. New Member, 2. Make Purchase, 3. Make Payment, 4. Print All, 5. Exit

Your choice: 2

Member's Identification Number: 1010
Purchase Amount: 52.7

Result: Purchase successful!
Member's Identification Number: 1010
Amount Outstanding: $52.70
```

CS1020: Data Structures and Algorithms I

```
*****
*      Membership System of CS1020      *
*****
Service available:
1. New Member, 2. Make Purchase, 3. Make Payment, 4. Print All, 5. Exit

Your choice: 3

Member's Identification Number: 1010
Payment Amount: 2.7

Result: Payment failed because the amount paid is less than the minimum amount
required!
Member's Identification Number: 1010
Amount Outstanding: $52.70

*****
*      Membership System of CS1020      *
*****
Service available:
1. New Member, 2. Make Purchase, 3. Make Payment, 4. Print All, 5. Exit

Your choice: 3

Member's Identification Number: 1010
Payment Amount: 12.7

Result: Payment successful!
Member's Identification Number: 1010
Amount Outstanding: $40.00

*****
*      Membership System of CS1020      *
*****
Service available:
1. New Member, 2. Make Purchase, 3. Make Payment, 4. Print All, 5. Exit

Your choice: 4

Result: 2 existing members.

Member's Identification Number: 1000
Amount Outstanding: $0.00

Member's Identification Number: 1010
Amount Outstanding: $40.00

*****
*      Membership System of CS1020      *
*****
Service available:
1. New Member, 2. Make Purchase, 3. Make Payment, 4. Print All, 5. Exit

Your choice: 5

Bye bye!
```

You are free to decide what message to print if an operation is unsuccessful, e.g. member number not found, payment amount is less than 20% of the amount outstanding and trying to create more than 100 members, etc.

CS1020: Data Structures and Algorithms I

The program implementation should be split into two parts:

Part A. (Simple Class) Skeleton of **Member** class:

```
class Member {  
  
    //Constant Variable  
    protected static final double MIN_PAYMENT_RATE = 0.2;  
  
    //For a controlled memberID  
    private static int _nextMemberID = 1000;  
  
    //Instance Variables  
    protected int _memberID;  
    protected double _amtOutstanding;  
  
    //Constructor  
    public Member() {  
        /* Your Code Here */  
    }  
  
    //Public Methods  
    public boolean makePurchase(double amt) {  
        /* Your Code Here */  
    }  
  
    public boolean makePayment(double amt) {  
        /* Your Code Here */  
    }  
  
    //Tutorial 2, Q3, Part A  
    public int getMemberID() {  
        /* Your Code Here */  
    }  
  
    public void print() {  
        /* Your Code Here */  
        // You may choose to write a toString() method instead  
    }  
}
```

Question to ponder: How do we decide what public methods to provide for a class?

CS1020: Data Structures and Algorithms I

Part B. (Driver Program) With the **Member** class in Part A, we can now write the driver code to provide the services.

Suggestion: Use an array to store the **Member** object references.

Let us modularize the program to make it more manageable. Write the following methods in a driver class **MembershipSystemApp**:

- i. Write a static method

```
int findMemberID(Member[] arr_Members, int size, int targetMemberID);
```

Search through the members in *arr_Member* array to locate the **index** of Member object with the target member identification number, where the parameter *size* is the number of members in the system.

- ii. Write a static method

```
void printAllMembers(Member[] arr_Members, int size);
```

Print out the members' information in the *arr_Member* array.

- iii. Write a main method to:

- Print available services. Read user input.
- Perform the appropriate service from the user input.

The main method will mostly make use of the instance and static method specified in Parts A and B.