# Problem Set 2

The following problem set will be worth 100 points. The code will be submitted electronically via Canvas using the "Problem Set 2" dropbox. The assignment is **due at the start of the class two weeks from the date it was assigned**.

Your code will be graded on both elegance and *user-friendliness*.

## Exercise #1 – Payment Problem (10pts)

Define a class called *Payment* which contains an instance variable of type *double* that stores the amount of a payment. Include appropriate getter/setter methods in your class. Include a method named *paymentDetails* which outputs an English sentence describing the payment.

Next define a class named *CashPayment* which is derived from *Payment*. The class should redefine the *paymentDetails* method to indicate that the payment is cash. Include necessary constructors for this class.

In addition to this, define a third class called *CreditCardPayment* that is derived from *Payment*. This class should contain instance variables for the:

- Cardholder's Name
- Expiration Date
- Credit Card Number

Lastly modify the *paymentDetails* method on this class to include all credit card information in the printout. Include a main method in your submission (either in one of the three classes or in a separate class) which demonstrates two instances of *CashPayment* and at least two instances of *CreditCardPayment* and the different values of *paymentDetails* for each.

Save your submissions in **Payment.java**, **CashPayment.java** and **CreditCardPayment.java**.

## Exercise #2 – Movie Problem (10pts)

Consider a movie rental business. Create a class named *Movie* which can be used for your video rental business. The Movie class should track the following:

- Motion Picture Association of America Rating (e.g. Rated G, PG-13, R)
- ID number
- Movie title

Ensure appropriate getter/setter methods in your class definition. Create an *equals()* method (which overrides *java.lang.Object's* method), where two movies objects are equal if their ID number is identical.

Next, create three additional classes named *Action*, *Comedy* and *Drama* which are each derived from *Movie*. Lastly create an overridden method named *calculateLateFees* which takes input a number of days the movie is late and returns the late fee for that movie. The default fee is $2 a day. Action movies have a fee of $3 per day, comedies are $2.50 per day, and dramas are $2 per day. Test your classes with a main method, either inside one of the classes or inside a class of its own.

Save your submission in **Movie.java**, **Action.java**, **Comedy.java**, and **Drama.java**.

## Exercise #3 – Shape Problem (20pts)

Create an interface named *Shape* with the following method signature:

```
double area();
```

Next define classes which implement this interface as follows:

- *Circle*, with instance data for the radius, a constructor which sets the radius and appropriate getter/setter methods
- *Rectangle*, with instance data for the length and width, a constructor which sets both properties and appropriate getter/setter methods

Use the following code to test your implementations:

```
public static void main(String[] args)
{
        Circle c = new Circle(4); // Radius of 4
        Rectangle r = new Rectangle(4,3); // Height = 4, Width = 3
        ShowArea(c);
        ShowArea(r);
}
public static void ShowArea(Shape s)
{
      double area = s.area();
      System.out.println("The area of the shape is " + area);
}
```

Save your submissions in **Circle.java**, **Shape.java** and **Rectangle.java**.

## Exercise #4 – Rating Problem (20pts)

Consider a list of movie reviews you have compiled, where each movie receives a rating from 1 (bad) to 5 (excellent.) The first line in the file is the number of ratings that are in that particular file. After that each rating consists of two lines:

- the name of the movie
- the numeric rating between 1 to 5

Here is a sample file with four unique movies and seven ratings:

```
7
Harry Potter and the Half-Blood Prince
4
```

```
Harry Potter and the Half-Blood Prince
5
Army of the Dead
1
Harry Potter and the Half-Blood Prince
4
Army of the Dead
2
The Uninvited
4
Pandorium
3
```

For this exercise you are to write classes which will read in a file in the above format, calculate the average rating for each movie, and the output the average along with the number of reviews. Using the input file from above, sample output can be shown as follows:

```
Harry Potter and the Half-Blood Prince: 3 reviews, average of 4.3 / 5
Army of the Dead: 2 reviews, average of 1.5 / 5
The Uninvited: 1 review, average of 4 / 5
Pandorium: 1 review, average of 3 / 5
```

Ensure to use an instance (or instances) of the class *HashMap* in your solution. Your map should index from a string representing each movie's name to integers that store the number of reviews for the movie, as well as the sum of ratings for the movie. Feel free to name the source code files anything to your liking when completing the solution.

## Exercise #5 – Registration problem (20pts)

You have a list of Student ID numbers followed by the course number (separated by a space) that each student is enrolled in. The listing is in no particular order. For instance, if student 1 is in CS100 and CS200 while student 2 is in CS105 and MATH210, then the list might look like this:

1 CS100

2 MATH210

2 CS105

1 CS200

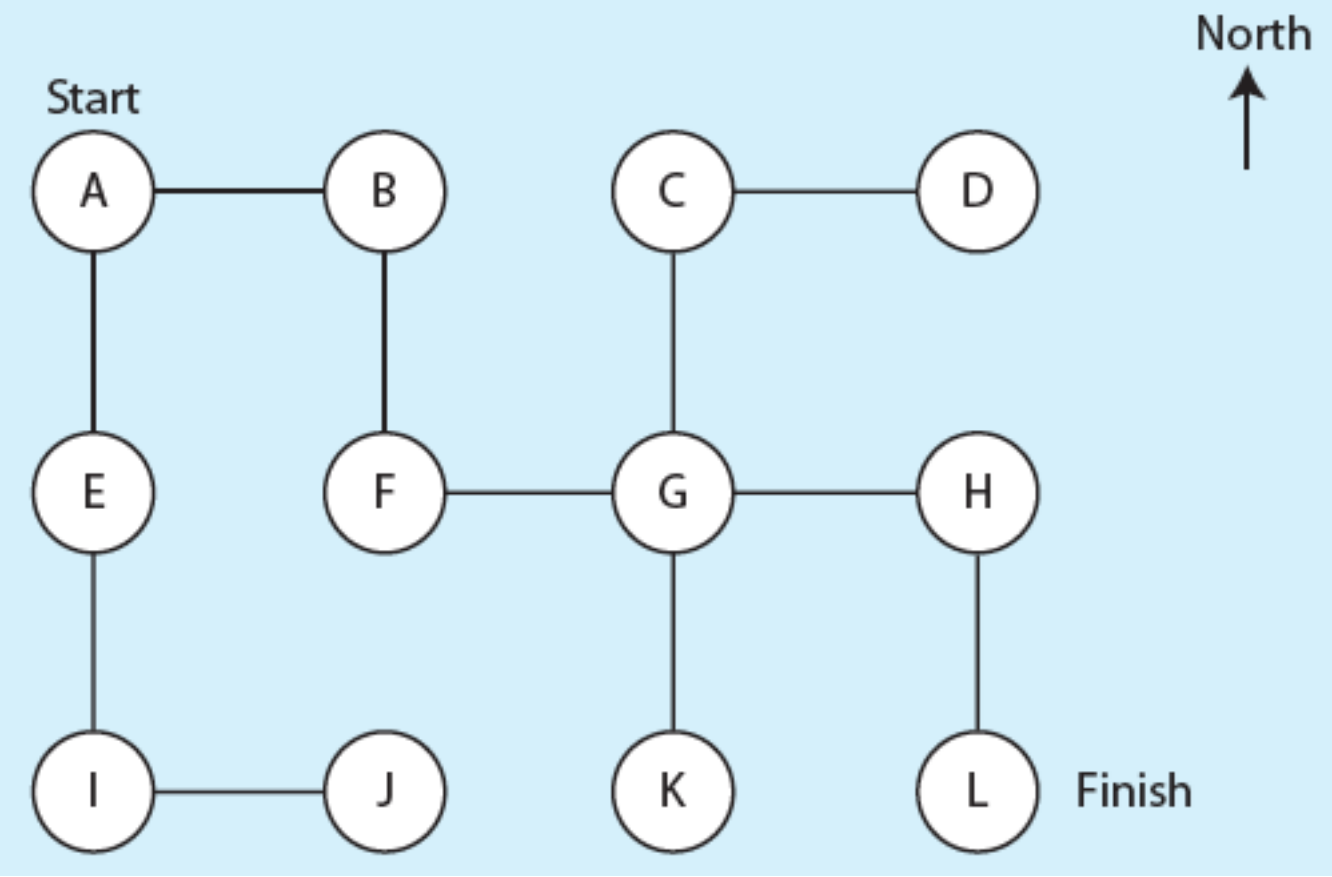
Write a program that reads data in this format from the console. If the ID number is -1, then stop inputting data. Use the **HashMap** class to map from an Integer (the student ID number) to an **ArrayList** of type **String** that holds each course that the student has enrolled in. The declaration should look like the following:

**Map<Integer, ArrayList<String>> students = new HashMap<>();**

After all the data is inputted, iterate through the map and output the student ID number and all courses stored in the vector for the student. The result should be a list of courses organized by the student ID number.

## Exercise #6 – Maze Problem (20pts)

Consider the following graph with edges and vertices defined as follows:




Write an application which implements this graph maze using reference to a *Node* class. Each node in the graph will correspond to an instance of the *Node* class. The edges correspond to the links that connect one node to another and can be represented in *Node* as an instance variable which references another *Node* class. Start the user in node A. The user's goal is to finish in node L. The program should output possible moves in the cardinal directions.

Include a main method in a class called *Maze* which can be used to test this maze. Sample output can be shown as follows:

```
You are in room A of a maze of twisty little passages, all alike. You can go east or south.
E
You are in room B of a maze of twisty little passages, all alike. You can go west or south.
S
You are in room F of a maze of twisty little passages, all alike. You can go north or east.
E
```

Save your submissions in **Maze.java** and **Node.java**.

**Submission Requirements**: Submit the aforementioned files in a zip file with the naming strategy:

Last Modified: 9/17/2018

First initial + last name + PS + problem set number.zip

As an example, I would submit the code in a zip file named **mmeluskyPS2.zip**. Submit your zip file via the "Problem Set 2" Canvas dropbox before the date of the close of the assignment.