Melusky – Fall 2016
Penn State Harrisburg

Problem Set 1

The following problem set will be worth 100 points. The code will be submitted electronically via Canvas using the "Problem Set 1" dropbox. The assignment is **due at the start of the class three weeks from the date it was assigned**.

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

Exercise #1 – Account Management Problem (20pts)

Define a Java class named `Account`. Your class should have instance data corresponding to the account number and account type, as well as the individual's name and account balance. In addition to the standard accessor and mutator methods, your class should also contain two methods:

- deposit – deposits an amount into the account
- withdraw – withdraws an amount from the account

Ensure that validation exists in those two methods, ensuring that the user does not deposit a negative amount into the account, or withdraws an amount over the available balance. Write a `main` method which simply tests a series of withdraws and deposits.

Save your solution in the file named **Account.java**.

Exercise #2 – Seating Problem (20pts)

Assume an airplane seats passengers with seat numberings as follows:

1	A	B	C	D
2	A	B	C	D
3	A	B	C	D
4	A	B	C	D
5	A	B	C	D
6	A	B	C	D
7	A	B	C	D

Write a class named **Airplane** which holds this seating arrangement in some sort of data structure (two dimensional array or Java collection class or similar.) Write a `main` method in this class which will loop, allowing the user to enter a row number and column letter for each seat occupied until no more seats need to be added. Print out the grid after each pass of the loop, marking each assigned seat with an **X**. Show the user an error message if the user attempts to set an already assigned seat.

Save your solution in a file named **Airplane.java**.

Exercise #3 – Queueing Problem (20pts)

In data structures, a *queue* is a data structure that is "first in, first out". It contains two methods:

- enqueue – adds an item to the queue
- dequeue – removes an item from the queue

For this assignment you are going to implement a variation on this called a **priority queue**, where an integer corresponding to the priority is passed into the *enqueue* method upon invocation. As an example, we can have three successive calls to add items to the queue:

- q.enqueue("X", 10)
- q.enqueue("Y", 1)
- q.enqueue("Z", 3)

Invoking *dequeue* on the object will then remove the item "X" from the queue, since it has the highest priority. A second invocation of the method returns "Z" while the third invocation returns "Y". Use an **ArrayList** as a representation of the queue. Include a main method which demonstrates successive adds and removals from your queue.

Save your solution in a file named **PriorityQueue.java**.

Exercise #4 – Playing Card Problem (20pts)

A deck of playing cards consists of 52 cards divided into four suits: **hearts, clubs, diamonds, spades**. Within each suit there are thirteen cards, labelled **2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A**. For this assignment you'll need to create two classes, **Deck** and **Card**. The deck should use an **ArrayList** to hold the list of cards.

Your Deck class needs methods which performs the following:

- Prints every card in the deck
- Shuffle the cards in the deck (randomly swapping every card in the deck)
- Add a new card to the deck
- Remove a card from the deck (the first card stored in the ArrayList object)

- Sort the cards in the deck ordered by name

Include a **main** method in your Deck class which validates the methods you made.

Save your solution in files named **Deck.java** and **Card.java**.

Exercise #5 – Number List Problem (20pts)

Write a program that reads numbers from the keyboard into an array of type **int[]**. You can assume that there won't be more than 50 elements added to the list. Your output should be a two column list. The first is the set of distinct array elements, the second column is the number of occurrences of each element. The list should be sorted on entries in the first column, largest to smallest. As an example for the array itself:

{-12, 3, -12, 4, 1, 1, -12, 1, -1, 1, 2, 3, 4, 2, 3, -12}

Your output should look like the following:

N	Count
4	2
3	3
2	2
1	4
-1	1
-12	4

Save your solution in a file named **IntList.java**.

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

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

As an example, I would submit the code in a zip file named **mmeluskyPS1.zip**. Submit your zip file via the "Problem Set 1" Canvas dropbox before the date of the close of the assignment.