Item List CSIS-1410

One of the main purposes of this assignment is to review and refresh many of the concepts covered in CSIS-1400.
It also is an opportunity to demonstrate that you understood how to use doc comments and how to create jar files.

Learning Objectives :

- Design and implement a class
- access class members of another class
- use an ArrayList
- use a static field
- use doc comments to provide useful documentation for your code
- read user input from keyboard
- provide user choices with a menu
- use repetition statements
- use selection statements
- override toString
- create a jar file that includes the Java source code

Description:

In this assignment I encourage you to work with a partner so you can help each other refresh concepts that have been covered in CSIS-1400. It is important to be responsive and that each partner contributes his/her fair share. If you have concerns about your partner bring it up as early as possible. First try to resolve the issue with your partner. If that is difficult talk to your instructure.
Each code file should include a comment on top that ***lists both students*** and the name of the assignment.

Together write a program that keeps track of a list of items and that provides the user with a menu that allows the user to add, remove, list items, etc. For more details read the instructions below.

Instructions:

- Decide which items you would like to store in the list. (e.g. books, bikes, gemstones, etc.)
  It can by any thing but **not** cars (I used that for the example) and **not** people.
- Create a class that represents the item you chose.
  Here are some requirements your class needs to fulfill:
  - It can **not** be cars or people (see above)
  - The class needs to have at least 3 attributes you are keeping track of (e.g. year, make, model)
  - It needs to have two additional fields:
    - a unique id that cannot be changed once created (like a primary key in a database)
    - a static count that is used to initialize the id with a unique number

    Notice: At this point, we have a total of at least 5 attributes
  - It needs a parameterized constructor.
    It allows the user to provide values for all the attributes you are keeping track of but not for the id nor for the count. (in our case that would be 3 parameters: year, make, and model)
    The constructor creates a unique id – e.g. an 8 digit number for each item based on the static field count. (e.g. 12345678 + count++; In this example the smallest id would be 12345678)
  - It needs a getter for each of the fields except for the static count (in our case that would be 4 getters)
    The static count should not be exposed to another class. It is only used to initialize the unique id in the

constructor.

Hint: http://stackoverflow.com/questions/7221691/is-there-a-way-to-automatically-generate-getters-and-setters-in-eclipse

- It needs to override the method toString.
  The method toString has no parameters and returns a string that represents an instance of the class.
  This string needs to include the values of all fields except for count (e.g. 2015 Honda Accord id: 12345679)
  Labels should be added when the value itself is not self-explanatory. (e.g. id)

Sample Output :

```
1. Display all items
2. Add an item
3. Find an item
4. Delete an item
5. Number of items in list
6. Exit
Enter your selection: 1

2011 Toyota Corolla  id:12345678
2015 Honda Accord  id:12345679
2008 Audi A4  id:12345680
2017 Tesla Model 3  id:12345681

1. Display all items
2. Add an item
3. Find an item
4. Delete an item
5. Number of items in list
6. Exit
Enter your selection: 5

Number of cars: 4

1. Display all items
2. Add an item
3. Find an item
4. Delete an item
5. Number of items in list
6. Exit
Enter your selection: 3

Id: 12345680
2008 Audi A4  id:12345680

1. Display all items
2. Add an item
3. Find an item
4. Delete an item
5. Number of items in list
6. Exit
Enter your selection: 4

Id: 12345680
2008 Audi A4 has been deleted

1. Display all items
2. Add an item
3. Find an item
4. Delete an item
5. Number of items in list
6. Exit
Enter your selection: 2

Year: 2013
Make: Ford
Model: Mustang
```

Instructions continued:

At this point you have one class that represents one item (e.g. Book, Bike, Gemstone, etc.). This class does not contain the list of items nor should it include any print statements.
You still need at least one more class (optionally two) that include(s) the list of items, the menu with options, and the user communication (input / output)

Create one or two additional classes to do the following:

- Declare an ArrayList of items (each element in the list should be an instance of the new Java type you just wrote)
- Initialize the list with four different elements (hard coded).
- Display a menu with the following choices until the user chooses to exit:
  ```
  1. Display all items
  2. Add an item
  3. Find an item
  4. Delete an item
  5. Number of items in list
  6. Exit
  Enter your selection:
  ```
  Important: replace the word *item* and *items* with the actual type you store in the list. (e.g. Display all *cars*)
- Provide the functionality chosen by the user.
  Use private methods to print the menu and to display, find, add, or delete items. Using private methods in such a way structures the code and to makes the code more clear and easier to read.
- When the user wants to find or delete an item s/he should be prompted for the id number.
  If the user provides and id number that doesn't exist, an appropriate message should be displayed.
  (e.g. The id … could not be found.)
  If the id number could be found the item information should be displayed as part of the response
  (e.g. 2015 Honda Accord id:12345679 if the item was found or 2015 Honda Accord has been deleted if it was deleted)
- If the user enters an invalid choice from the menu (e.g. 9) an appropriate message should be printed (e.g. `Enter a selection 1 - 6`)
- If the user enters 6 a message should be printed before terminating the program (e.g. `Good bye`)

Below you can find a sample output. This sample is about cars. Your output needs show elements of the item type you chose above.

```
1. Display all items
2. Add an item
3. Find an item
4. Delete an item
5. Number of items in list
6. Exit
Enter your selection: 1

2011 Toyota Corolla  id:12345678
2015 Honda Accord  id:12345679
2017 Tesla Model 3  id:12345681
2013 Ford Mustang  id:12345682

1. Display all items
2. Add an item
3. Find an item
4. Delete an item
5. Number of items in list
6. Exit
Enter your selection: 4

Id: 12345680
The id 12345680 could not be found

1. Display all items
2. Add an item
3. Find an item
4. Delete an item
5. Number of items in list
6. Exit
Enter your selection: 4

Id: 12345682
2013 Ford Mustang has been deleted

1. Display all items
2. Add an item
3. Find an item
4. Delete an item
5. Number of items in list
6. Exit
Enter your selection: 5

Number of cars: 3

1. Display all items
2. Add an item
3. Find an item
4. Delete an item
5. Number of items in list
6. Exit
Enter your selection: 7

Enter a selection 1 - 6

1. Display all items
2. Add an item
3. Find an item
4. Delete an item
5. Number of items in list
6. Exit
Enter your selection: 6

Good bye
```

Javadoc:

Ensure that you have a doc comment above each of the classes, fields, and methods

Jar file:

Create a **jar file** that includes the source code.

It is optional but not required to make it runnable as well.

Video:

Create a screen recording in one of the following formats: `mp4`, `swf`, or `WebM`

If you have already a software that you like to use for screen casts that's great. If not you can check out https://screencast-o-matic.com/screen_recorder. It is free and easy to use.

The recording should be no more than 5 minutes long and it should include the following:

- Your names (both partners)
- When you start recording Eclipse should already be open and the windows should be adjusted so that the code can be easily seen.
- Show all the code by slowly scrolling down the classes from top to bottom.
  If you have any missing parts or known errors in your code it should be pointed out here. Apart from that no explanation is needed. I will pause the video whenever I need a closer look.
- Run the code and demonstrate the functionality as shown in the sample output on the left.
  Make sure all options from the menu are used.
- Briefly reflect on the experience. Was there a difficulty you were able to overcome? What did you learn from this experience?
- Briefly describe the "pebble distribution".
  Assume your team has 100 pebbles. They represent 100% of the work done. If both partners worked well together, were responsive, contributed their fair share etc. then the pebble distribution should be 50 / 50. However, sometimes the team work is less balanced. In either case discuss the pebbles distribution with your partner before submitting it.

Turning in :

- ONE student submits the jar file AND the video
- The other student writes a brief submission note listing the name of the partner