**Cemetery**

The ArrayList class (in the java.util package) provides the implementation for a convenient, **mutable** list. Ordinary arrays are *immutable* - their sizes can't be changed after they're instantiated (i.e. you can't add or remove an element from an array). ArrayLists, on the other hand, can grow/shrink as necessary.

However, they can only store objects! Primitive data types (like int, double, char, etc.) must be added as an object (referred to as **boxing/unboxing**). Try [this](https://www.youtube.com/watch?v=s8ju-dZJJ50) video and our [website](bit.ly/mrbunnCS) for help with the syntax.

1. Make a class called ArrayListProbs (with an empty, no-parameter constructor). The methods below will go in this class.
2. Make a class called ArrayListProbsRunner with a main() method, that will be used to test ArrayListProb's methods.
   1. **Unlike most labs, test cases can be found in the "ArrayListProbs test cases" document in the lab folder (too much code to include in the lab document).**
3. Complete the method public void makeListAndPrint(int num, int limit) that creates a new ArrayList<Integer> object, adds num random integers from 1 to limit to it, and prints the resulting list.

/\* You can print an ArrayList "directly"! System.out.println(list); will print the contents of an ArrayList object called *list*. You don't have to use Arrays.toString() as with regular arrays! (They have an overridden, implicitly called toString() method, more on what that means in the inheritance unit.) \*/

1. Complete the method public ArrayList<Integer> minToFront(ArrayList<Integer> list), that adds a *copy* of the smallest value in list at index 0 (shifting the rest "down" one position) and returns the modified list.
2. Complete the method public ArrayList<Integer> addOne(ArrayList<Integer> list), that adds one to every element in listand returns the modified list.
3. Complete the method public ArrayList<String> removeDupes(ArrayList<String> list), that removes all (back-to-back) duplicate values from list*.* Remember, the size of an ArrayList is *dynamic* – if you remove an element, the size of the list will shrink!
4. Complete the method public ArrayList<Integer> swapPairs(ArrayList<Integer> list), that swaps every number in listwith the element immediately after it.
5. Complete the method public ArrayList<String> removeLenN(ArrayList<String> list, int n), that removes all Strings with a length of n*.* Remember, the size of an ArrayList is dynamic – if you remove an element, the size of the list will shrink!
6. (Riddle) I am the beginning of everything, the end of time and space, the beginning of every end, and the end of every place. What am I?
7. Make a class called Person that has the following:
   1. String name and int IQ instance variables (all private)
   2. Constructor / getter methods
8. Complete the method public int dumbestPerson(ArrayList<Person> list) that returns the index of the dumbest Person in the list (this method should go in the ArrayListProbs class).
9. Make a class called Book that has the following:
   1. String title, String author, and double price instance variables (all private)
   2. Parameterized constructor / getter methods
   3. A method public String toString() that returns the info for a Book as a String in printable form, like this:

"<title>, by <author>. Cost: $<price>"

1. Complete the method public Book highestPricedBook(ArrayList<Book> list) that returns a reference to the Book object with the highest price, given a list of Books.
2. Complete the method public ArrayList<Book> banBook(ArrayList<Book> list, Book book) that removes all Books with the same title as the book parameter from list and returns the modified list.
3. Make a class called Bookstore that has the following:
   1. ArrayList<Book> inventory – a list of all the books in the store's inventory
   2. No-parameter (default) constructor that initializes inventory

//*inventory*'s value should be a reference to a new ArrayList<Book> object

* 1. A method public void addBook(Book b) that will add a Book to the store's inventory.
  2. A method public int numBooks() that returns the number of Books in inventory.
  3. A method public Book getBook(int index) that returns the book at index in inventory.
     1. This method should return null if the supplied index is less than 0 or greater than or equal to the number of books in the inventory.

1. Complete the method public double bookstoreValue(Bookstore store) that returns the total value (prices) of all the books in the Bookstore's inventory.
2. (Riddle) I used to fill rooms but now I'm a tower, I can stay put or I can travel far. What am I?

**Cemetery**

Write a program to find the average life span of people buried at St. Mary Magdalene Cemetery, Old Fish Street, London, England during the period January 5, 1813 to July 10, 1853.

Below is the start of the **"cemetery.txt"** data file (the **bold** lines represent the actual contents of the file):

*ST MARY MAGDALENE OLD FISH STREET CITY OF LONDON*

*Burials 5th Jan 1813 - 10th July 1853*

*NAME BURIAL DATE AGE RESIDENTIAL ADDRESS*

*------------------------ ----------- --- ----------------------------*

**John William ALLARDYCE 17 Mar 1844 2.9 Little Knight Ryder Street**

**Frederic Alex. ALLARDYCE 21 Apr 1844 0.17 Little Knight Ryder Street**

**Philip AMIS 03 Aug 1848 1 18 1/2 Knight Rider Street**

**Thomas ANDERSON 06 Jul 1845 27 2, Bennet's Hill**

.....

Note that the AGE field in the file could be in any of the following formats:

**AGE column  Interpretation**

39   39 years

11.5   11.5 years

15w   15 weeks

22d 22 days

1. Create a Tombstone class that has instance variables to store a person's name, burialDate, age (as an integer, more info to follow), and address, as well as applicable constructor and getter methods.
2. Create a Cemetery class with one instance variable, an ArrayList of Tombstone objects. Cemetery's constructor should have a parameter String fileName, which represents a text file containing the information on the interred for this cemetery.

The Cemetery class will parse the information in the text file to create and store Tombstone objects. Each line in the file represents one Tombstone object.

* 1. Use a Scanner object to get a single line from the input file at a time, until the end of the file. For each line, make *another* Scanner object that will scan the "current" line String.

Be careful when parsing each line! As you can see from the data above, a person can have two or three names – it is suggested that you use Scanner's next() method until the input source has a next integer (the day at the beginning of the burial date) to parse the names. Parsing the input will require some hard-coding; this is inevitable given the format of the data.

* 1. A method parseAge(String ageString) has been written for you (copy/paste from the text file in the lab folder into your Cemetery class), that will take the String version of a person's age (e.g. "11.5" or "22d") from the input file and return the (rounded) total number of ***days*** that person was alive when they were interred. Note that static methods can be called in a constructor.

1. Finally, in a Runner class with a main() method, instantiate a Cemetery object, supplying the file name of **"cemetery.txt"** to Cemetery's constructor\*. The input file should be in the root project folder (the main folder in BlueJ, the project folder [NOT the ***src*** folder] in Eclipse). **Print the average age, in years, rounded to one decimal place.**

*\*Don't forget the throws IOException line after every method header that reads from a file OR calls a method that reads from a file. You can also use a try / catch block – more info in the "Exceptions and errors" PPT (or just google it)*

**(Advanced) Sorted Tombstones**

Add the ability to sort Tombstones, based on their burial date. Implementing the Comparable interface would be useful for this, more info can be found in the powerpoints from later in the year.

*Adapted from the* ***Cemetery*** *project*

[*rfrank.net/projects/cslabs-final/1390-cemetery/1390.html*](http://rfrank.net/projects/cslabs-final/1390-cemetery/1390.html)