Lab 11: More Inheritance and Recursion
CSE-021-01, Spr 2019 **Points: 20**

Overview

We will work more with inheritance, extending the `ArrayList` class (which is a class that works like an array, but provides more flexibility). Additionally, you will learn the basics of using recursion in methods to solve typical problems in CSE elegantly.

Before you get started, read chapters 7.13 – 7.15, 10.5 – 10.7 and 12.1 – 12.7. Answer the Assessment questions as you encounter them in the next section. The prompts for answering assessment questions are placed immediately following the material to which the listed questions relate.

Getting Started

After following the import instructions in the assignment page, you should have a Java project in Eclipse titled Lab 21_11. This PDF document is included in the project in the **doc** directory. The Java files you will use in this lab are in the **src/arraylist** and **src/recursion** directories.

Part 1: ArrayList and Inheritance

The `java.util` class library contains a variety of useful classes. One is the `ArrayList` class, which represents a list of elements that can be accessed by position as with an array. An `ArrayList`, however, can only contain objects from a specific class. To specify the class that an `ArrayList` holds, you use `<` and `>` to enclose the class name, like:

```
ArrayList<String>     //can contain a list of strings
```

Can an `ArrayList` contain `int` data? Not directly, no. An `ArrayList` can only contain objects from a particular class, and an `int` is not an object but a *primitive*: `int`s don't have methods, can't be extended through inheritance, etc. So,

```
ArrayList<int>        //error!
```

will cause a compile error in Java.

Can one `ArrayList` contain both `ModNCounter` and `SeasonCounter` objects? Remember, an `ArrayList` contains only objects from a single class. However, because of inheritance both objects are members of `Counter`, and can be contained in

```
ArrayList<Counter>    //OK
```

The `ArrayList` class has numerous methods; we will use several in this lab.

- `boolean add(Object obj)`: Appends the specified element to the end of this list. You don't have to worry about the current size of the list, it will expand as you add elements, as necessary. Returns `true`, since an `ArrayList` always adds when asked to.
- `boolean remove(Object o)`: Removes a single instance of the specified element from this list, if it is present. Returns `true` if an element was removed.
- `int size()`: Returns the number of elements in this list.

The `Object` class whose object `obj` is a parameter to `add()` should be the class that you defined the `ArrayList` with. For instance, an `ArrayList`declared with

```
ArrayList<String> myList = new ArrayList<String>();
```

would be able to

```
myList.add("a string")        // OK
```

but not (for instance)

```
myList.add(new Counter())     // this will fail with myList declaration above
myList.add(5)                 // this will fail with myList declaration above
```

Task 1

Consider the `ArrayListRunner` class declared in `ArrayListRunner.java`, where an `ArrayList` initialized as follows:

```
ArrayList<String> words = new ArrayList<String>();
```

What is the result of executing the code below? Show all the results after every line of code in a Word or text file named `Part1`.

```
words.add("a");
words.add("b");
words.add("c");
words.remove("b");
words.add("d");
words.remove("a");
words.add("e");
words.remove("b");
words.add("d");
words.remove("c");
words.remove("d");
```

Task 2

Using inheritance, define a class `TrackedArrayList` that behaves just like the `ArrayList` class except that it has an extra method:

```
public int maxSizeSoFar();
```

which returns the maximum number of elements in this list at any time since the list was constructed. For example, if `maxSizeSoFar` were called immediately after each call in the code sequence in Task 1 above, it would return the following values:

```
1
2
3
3
3
3
3
4
4
4
```

Use the framework code in `arraylist/TrackedArrayList.java` to get started. You will see the special format to use when extending `ArrayList`, in order to refer to the class of the elements that it can contain.

Use the class in `ArrayListRunner.java` to test your work as you go, if you need to. You may find that the EduRide feedback tool gives you enough information to solve this task; if so, you'll need to comment on the specific feedback that you found useful.

Take advantage of the `ArrayList` methods and variables as much as possible, using calls to `super`, rather than reinventing the wheel.

[Answer assessment question 1]

Part 2: Introduction to Recursion

We will learn about recursion by stepping through the code that we have supplied for you. Then you will fill in code to complete the behavior of other recursion programs.

Task 3: Modify `SimpleRecursion.java`

There are 6 different versions of `recur` in `SimpleRecursion`. Your first task is to run them and figure out how each of them behave.

[Answer assessment question 2]

Modify `SimpleRecursion.java` so all 6 versions can be run inside `main` (hint: name overloading does not work). Also make sure that it is obvious which version is running by differentiating the outputs.

Part 3: Recursion Using Helper Methods

We are now going to design a recursive version of a method called `reverse` that returns a copy of the argument String with its characters reversed. Here are some examples:

Call to `reverse`	Returned value
`reverse("ABC")`	`"CBA"`
`reverse("120ESC")`	`"CSE021"`

One can easily write `reverse` iteratively, with a loop instead of recursion. In many problems, however, this is not so simple. Our intent here is to help you add another tool to your toolbox, so that when a problem shows up that requires recursion, you'll be ready.

We first will explore writing *separate* methods for different string length – `reverse1`, `reverse2`, `reverse3`, and so on. For some programmers, this is a useful intermediate design step towards the goal of coding a general recursive `reverse`.

Task 4: Fill-in `NaiveReverse.java`

Fill in the code inside `reverse6` method.

[Answer assessment question 3]

A fellow student thinks that using a helper method will make it easier to write `reverse9`, a method to reverse Strings of length 9. Your instructor suggests using `reverse8` as the helper (you can assume that `reverse8` works correctly; that is, it reverses a String of length 8 correctly). The student comes up with the following solution:

```
public String reverse9 (String s) {
   String partial = reverse8(s.substring(1, 9));
   String firstLetter = s.substring(0, 1);
```

```
    return (firstLetter + partial);
}
```

[Answer assessment question 4]

Task 5: Fill-in `Reverse17.java`
Fill in the code inside `reverse17` method.

Task 6: Fill-in `Reverse.java`
The usual way to write recursion is using a base case to return a default value, and using recursion otherwise (Zyante 12.1 and 12.5). Fill in the code inside `reverse` method to check for a base case and recursion case while returning the proper values or expressions.

Part 4: Working With Numbers

Task 7: Fill-in `Halved.java`
Fill in `Halved.java` so it uses recursion to return the result of a given number halved. It does not use any division or multiplication, but uses subtraction instead. The method ignores the fractional parts of halved numbers. So, half of `10` is `5` but it is also the same answer for `11`. Your task is to figure out the base case where the half of a small number is `0` (returning `0`). Figure out the range of small numbers where this is true and place your answer into the condition check of the `if`-statement.

Task 8: Fix `DigitCount.java`
The `digitCount` method is intended to return the number of digits in its (non-negative) integer argument. For example, with `314159` or `700000` as argument, the method should return `6`. Leading zeroes are not counted except in the case where the argument is `0`. The code for the method is as follows:

```
public static int digitCount(int value) {
    if(value == 0) {
        return 1;
    } else {
        return 1 + digitCount(value/10);
    }
}
```

[Answer assessment question 5]

Fix `digitCount` with the answers to the questions so it works properly.

Part 5: (Assessment) Logic Check and Level of Understanding

Create a Word document or text file named `Part5` that contains answers to the following:

1) Consider the following classes in a Java program, with the methods defined in each.

- `Pet`
  - `eat()`
  - `sleep()`
  - `Dog` *`extends`* `Pet`
    - `goForAWalk()`
    - `bark()`
      - `Pomeranian` *`extends`* `Dog`
        - `yap()`
      - `GreatDane` *`extends`* `Dog`
        - `woof()`

- Cat *extends* Pet
  meow()
  - Siamese *extends* Cat
    ignoreYou()

For each of the questions below, answer with an `ArrayList` declaration with the most specific type, if any, that can work. For instance, the `ArrayList` that could contain `Cat` objects is `ArrayList<Cat>`

a. contain `Pomeranian` objects.
b. contain `Pomeranian` and `Cat` objects.
c. contain objects of any of the above types that you can call the `eat()` method on.
d. contain `GreatDane` objects that you will only call the `sleep()` method on.
e. contain `Pomeranian` and `Siamese` objects.
f. contain `Pomeranian` and `Siamese` objects that you will call `yap()` and `ignoreYou()`, as appropriate.

2) For the `recur` method in the `SimpleRecursion` class, what are the outputs of versions 1, 2, 3, 4, 5 and 6 called with argument `0`.
3) After filling-in the `reverse6` method in the `NaiveReverse` class, can you think of how to extend this algorithm for `reverse7` method? Explain.
4) Answer the following questions concerning the `reverse9` method given in **Task 4**:
   a. A student tests the `reverse9` method on the String `"confident"`. What will it return?
   b. What should be the proper reversed string for `"confident"`?
   c. Suggest how to fix the code so the method does it correctly.
5) Answer the following questions concerning the `digitCount` method in the `DigitCount` class:
   a. What is returned from the call `digitCount(0)`?
   b. What is returned from the call `digitCount(10)`?
   c. What is returned from the call `digitCount(314159)`?
   d. For what values of `value` will digit count return `1`?

What to hand in

When you are done with this lab assignment, submit all your work through CatCourses.

Before you submit, make sure you have done the following:

- Attached the file named `Part1` containing the output for code in Lines 19 – 29 of `ArrayListRunner.java`.
- Attached the file named `Part5` containing answers to Assessment questions (1 – 5).
- Attached filled in `ArrayListRunner.java`, if you used it to test. If you didn't, describe what parts of the EduRide feedback tool was useful to you in writing `TrackedArrayList.java` in the file `Part1` you created for Task1.
- Attached filled-in `TrackedArrayList.java`, modified `SimpleRecursion.java`, filled in `NaiveReverse.java`, filled in `Reverse17.java`, filled in `Reverse.java`, filled in `Halved.java`, and fixed `DigitCount.java` files.
- Filled in your collaborator's name (if any) in the "Comments…" text-box at the submission page.

Also, remember to demonstrate your code to the TA or instructor before the end of the grace period.