06-26956/26955 *Software Workshop*
Spring Semester 2015-16

The University of Birmingham
School of Computer Science
© Uday Reddy2015-16

Work Sheet 1: Recursion and Lists

(Assessed Exercise: 20% of the Spring Term continous assessment)

Assigned: Thursday, 14 January **Due**: Thursday, 21 January, 11:59pm.

(5% late submission penalty for up to 24 hours. No submission after 24 hours.)
(Junit testing and documentation mandatory.)

You will be using the class `List` used in Lecture. This class as well as templates for your solution are included in the "starter pack" which you should download from the module web page.

In this Worksheet, *you must not use assignment statements to change the value of any variables.* Recursion is the main vehicle by which you will get computations done. You can use local variable declarations with initialisations. But once initialized, the variables keep those values till the end of their scope. They do not change. You cannot use any loop statements either. Any solution containing an assignment statement or a loop statement will receive a mark of 0.

Exercise 1: Powers of integers (10%)

```
static int power(int m, int n)
static int fastPower(int m, int n)
```

Code the two recursive algorithms discussed in class for raising integer m to the power of integer n (both non-negative). You can test them by checking to see if both the algorithms produce the same results.

Exercise 2: Negate a list (5%)

```
static List negateAll(List a)
```

Given a list of integers a, write a method that returns a new list with all the elements of a with sign negated, i.e., positive integers become negatives and negative integers become positives. Example:

```
[2, -5, 8, 0] ==> [-2, 5, -8, 0]
```

Exercise 3: Searching for an element (10%)

```
static int find(int x, List a)
```

Given an integer x which is assumed to be in the list a, write a method that returns the position of the first occurrence of x in a. Positions are counted as $0, 1, 2, \ldots$. If x does not appear in the list, you should throw an `IllegalStateException`. Examples:

```
x: 3   a: [7, 5, 3, 8]   ==>   2
x: 2   a: [7, 5, 3, 8]   ==> IllegalStateException
```

Exercise 4: Check for positive (10%)

```
static boolean allPositive(List a)
```

Given a list of integers a, return a boolean value indicating whether *all* its elements are positive, i.e., ≥ 0.
Think: What should be returned if the list is empty?

Exercise 5: Find the positives (15%)

```
static List positives(List a)
```

Given a list of integers a, return a new list which contains all the positive elements of a. The elements should appear in the result in the same relative order as in a. Example:

```
[2, 3, -5, 8, -2]   ==>   [2, 3, 8]
```

Exercise 6: Sortedness (15%)

```
static boolean sorted(List a)
```

Given a list of integers a, this method must return a boolean value indicating whether a is sorted in increasing order. (There can be duplicate copies of elements. But, sortedness would require that all the duplicate copies would appear together.)

Exercise 7: Merging (20%)

```
static List merge(List a, List b)
```

Given two *sorted* lists a and b, your method must return a new *sorted* list that contains all the elements of a and all the elements of b. Any duplicate copies of elements in a or b or their combination are retained. Examples:

```
a: [2, 5, 5, 8]   b:[5, 7, 8, 9]   ==>   [2, 5, 5, 5, 7, 8, 8, 9]
a: [2, 5, 5, 8]   b:[9]   ==>   [2, 5, 5, 8, 9]
```

Exercise 8: Remove duplicates (25%)

```
static List removeDuplictes(List a)
```

Given a *sorted* list a, this method must return a copy of the list a with all duplicate copies removed. Example:

```
[2, 5, 5, 5, 7, 8, 8, 9] ==> [2, 5, 7, 8, 9]
```

(Hint: Please feel free to define helper functions!)
The naturally immediate solution for this problem requires $O(n^2)$ time for producing its result. But, we are really only interested in using this function for the case of lists sorted in increasing order. For sorted lists, the problem can be solved more efficiently, in $O(n)$ time. To receive full credit, you should produce an $O(n)$ program which works for sorted lists. We will only test the function for sorted lists.

End Notes:

- You should download the "starter pack" for Worksheet 1 from the module web site. The starter pack includes the class `List` and the following files:
  - `Worksheet1Interface.java` — interface
  - `Worksheet1.java` — template for your solution
  - `Worksheet1Test.java` — template for your test cases
- For this Worksheet (as well as all other exercises this semester), you need to write your own test cases. It is not necessary to run a large number of test cases, but you should pick your test cases *intelligently*. You should make sure you test for the "border cases" and cases that involve special treatment in the code (if any).
- When a method expects arguments satisfying some preconditions, e.g., the sortedness condition for `merge`, your method does not need check to see if the arguments actually satisfy those conditions. It is the responsibility of the calling programs to ensure that they pass the right kind of arguments. While testing it, you should only provide test cases that satisfy the preconditions.
- If the method receives arguments that do not satisfy the precondition, you can throw an exception. Use `IllegalStateException` which is predefined in Java.
- You must start your work, at the latest, in the scheduled lab session on Friday afternoon, when you will have lab demonstrators available for seeking help. In a two-hour lab session, you should be able to complete roughly half of this Worksheet.