Faculty of Arts & Sciences
Department of Computer Science
CMPS 200—Introduction to Programming
Assignment 11 – Due December 1 (8:00AM), 2014

1. **Hangman**. Write a program `Hangman.java` that emulates the game Hangman. The game is played as follows: to win the game, the user must guess a word letter-by-letter before exhausting the maximum number of guesses; otherwise the user will lose the game.
   The program must read the words to be guessed from the input file `words.txt`. It must also ask the user to enter the maximum number of guesses N allowed. The program must keep playing the game as long as there are words in the file and the user enters a value N > 0; otherwise the game should stop. The program must also keep track of the guessed and missed letters, and should discard already attempted guesses (and misses). If the user guesses a letter that is repeated in the word, all the (same) letters must be revealed to the user, and not just one. *PS*: when comparing letters, you should ignore case.
   Example:

If the word is "test", the following would be a sample output for *winning:*	If the word is "test", the following would be a sample output for *losing*:

`Enter the maximum number of guesses: 6`
`Word: _ _ _ _`
`Misses:`
`Guess: r`
 	
`Word: _ _ _ _`
`Misses: r`
`Guess: t`
 	
`Word: t _ _ t`
`Misses: r`
`Guess: g`
 	
`Word: t _ _ t`
`Misses: r g`
`Guess: e`
 	
`Word: t e _ t`
`Misses: r g`
`Guess: e`
 	
`Word: t e _ t`
`Misses: r g`
`Guess: s`
 	
`You Won! The word was: test`
	`Enter the maximum number of guesses: 3`
	
	`Word: _ _ _ _`
	`Misses:`
	`Guess: r`
	
	`Word: _ _ _ _`
	`Misses: r`
	`Guess: T`
	
	`Word: t _ _ t`
	`Misses: r`
	`Guess: r`
	
	`Word: t _ _ t`
	`Misses: r`
	`Guess: p`
	
	`Word: t _ _ t`
	`Misses: r p`
	`Guess: g`
	
	`You Lost! The word was: test`

2. **BubbleSort**. Write a program `BubbleSort.java` that implements the Bubble Sort algorithm to sort an array but in the **descending** order. The Bubble Sort algorithm is a simple sorting algorithm that works by repeatedly stepping through the array to be sorted, comparing each pair of adjacent elements and swapping them if they are in the wrong order. The pass through the array is repeated until no swaps are needed, which indicates that the list is sorted.
   You should create an array of size 10, such that the elements are randomly generated (between 0 and 1000). The program should take as command line argument the number of times to perform bubble sort (for new random values in the array). PS: You are NOT allowed and should NOT use the method `Arrays.sort`.

Example:

```
> java BubbleSort 3

before: 45 87 39 32 93 86 12 44 75 50
after : 93 87 86 75 50 45 44 39 32 12

before: 450 871 392 321 932 865 121 441 755 500
after : 932 871 865 755 500 450 441 392 321 121

before: 145 187 139 132 193 186 112 144 175 150
after : 193 187 186 175 150 145 144 139 132 112
```

3. **Set**. A *set* is an unordered collection of objects with no duplicate elements. In this program we consider sets whose elements are decimal digits: 0,1,2,3,4,5,6,7,8, or 9.

   The cardinality of a set of is the number of elements in it. For example, {5, 2, 0} and {1, 2, 5, 9} are sets with cardinality 3 and 4 respectively. The union of two sets (∪) is a set containing elements that appear in either of the two sets. The intersection (∩) of two sets is a set containing elements that appear in both sets. If we refer to the sets above as A and B, then A ∪ B = {5, 2, 0, 1, 9} and A ∩ B = {2, 5}.

   A set can be modeled as an array of ten `boolean`s that indicate whether a digit is in the set or not. For example the set {5, 2, 0} above may be represented by:

true	false	ture	false	false	true	false	false	false	false

   Write a java program, `Set.java`, that implements the following methods:

   1. `public static boolean[] createSet(String s)`
      // returns a set given a string as parameter, e.g., "611" returns the set {6 1}
   2. `public static int cardinality(boolean[] set)`
      // returns number of elements in set
   3. `public static boolean inSet(boolean[] set, int d)`
      // is the digit d in the set?
   4. `public static void addElement(boolean[] set, int d)`
      // add the digit d to the set (if not already in it)
   5. `public static boolean equals(boolean[] set1, boolean[] set2)`
      // does set1 and set2 have the same elements?
   6. `public static boolean[] union(boolean[] set1, boolean[] set2)`
      // returns the union of set1 and set2
   7. `public static boolean[] intersect(boolean[] set1, boolean[] set2)`
      // returns the intersection of set1 and set2
   8. `public static String  toString(boolean[] set)`
      // returns a string representation of set, e.g., {1, 2, 3}
   9. `public static boolean[] multiUnion(boolean[][] sets)`
      // returns the union of all sets, i.e., sets[0] ∪ … ∪ sets[sets.length - 1] .

   Your class should include a main( ) method that tests all the above methods.

4. **SearchPuzzle**. Write a program to check if a word may be found in a two dimensional array of letters. For example, given the following grid:

```
m v j l i x a p e
j h b c e e n p p
h u n o h b s w y
r w a m n u y z h
p p f p r d z k q
t p n u q o y j y
a n h t p f g b g
h x m e h w y l y
u a r r a y s o a
```

your program will find if a word can be spelled out in the grid by starting at any character, then moving in a straight line down or right. For example the grid above contains the word `computer` because it can be spelled out by starting at the character `'c'` in the second row, fourth column and moving down.

Write a program `SearchPuzzle.java` that reads a grid of characters from a file and checks whether or not a word may be found in it. The file contains two integers `n` and `m` on its first line denoting the numbers of rows and columns in the grid, followed by (n x m) characters on a second line. PS: You should fill the grid in a two dimensional array of letters and you are not allowed to use the class `String`.

For example, the file `grid.txt` contains the following input for the grid shown above. The file consists of two lines: the first line has the two integers 9 and 9 specifying the size of the grid as 9-by-9, and the second line has a sequence of 81 consecutive characters containing the entries of the grid in row order:

```
9 9
mvjlixapejhbceenpphunohbswyrwamnuyzhppfprdzkqtpnuqoyjyanhtpfgbghxmehwylyuarraysoa
```

Invocations of the program would produce something like:

```
> java PuzzleSearch grid.txt computer
computer found: row 2, column 4, going down

> java PuzzleSearch grid.txt party
party not found.
```

Submission Instructions

- As usual, submit your commented source code and sample runs in a zip file named s*#*`_asst11_`*netid*, where *#* is your section number and *netid* stands for your AUBnet user name.