Link to JavaDoc 7: http://docs.oracle.com/javase/7/docs/api/

Problem 1 `EvenDigits.java (25%)`

Write a method *evenDigits* that accepts an integer parameter n and that returns the integer formed by removing the odd digits from n. The following table shows several calls and their expected return values:

Call	Valued Returned
`evenDigits(8342116);`	`8426`
`evenDigits(4109);`	`40`
`evenDigits(8);`	`8`
`evenDigits(-34512);`	`-42`
`evenDigits(-163505);`	`-60`
`evenDigits(3052);`	`2`
`evenDigits(7010496);`	`46`
`evenDigits(35179);`	`0`
`evenDigits(5307);`	`0`
`evenDigits(7);`	`0`

If a negative number with even digits other than 0 is passed to the method, the result should also be negative, as shown above when -34512 is passed. Leading zeros in the result should be ignored and if there are no even digits other than 0 in the number, the method should return 0, as shown in the last three outputs.

Write a main method to test your code.

Problem 2 Palindromes.java (20%)

A palindrome is a string that is spelled the same way forwards and backwards. Some examples of palindromes are "radar," "able was i ere i saw elba" and (if spaces are ignored) "a man a plan a canal panama." Write a recursive method testPalindrome that returns boolean value true if the string stored in the array is a palindrome and false otherwise. The method should ignore spaces and punctuation in the string.

Write a main method to test your code.

Problem 3 `Bears.java (25%)`

This question involves a game with teddy bears. The game starts when I give you some bears. You can then give back some bears, but each time that you give back some bears you must give back half of all your bears. (If you have n bears, and n is even, then you will give back n/2. If n is odd, then you may round down to give back (n-1)/2 bears, or you may round up to give back (n+1)/2 bears.) The goal of the game is to end up with EXACTLY 42 bears.

For example, suppose that you start with 337 bears. Then you could make these moves:

- Start with 337 bears.
- Give back 168 (which is half of 337, rounding down) to leave 169 bears.

- You now have 169 bears.
- Give back 85 (which is half of 169, rounding up) to leave 84 bears.
- You now have 84 bears.
- Give back 42 (which is half of 42), to leave 42 bears.
- You have reached the goal!

Write a function (recursive or iterative) to meet the following specifications:

```
bool bears(int n) {
// Postcondition: A true return value means that it is
// possible to win the bear game by starting with n bears. A
// false return value means that it is not possible to win the
// bear game by starting with n bears.
// Examples:
//   bear(337) is true (as shown above)
//   bear(42) is true
//   bear(83), bear(84) and bear(85) are all true
//   bear(52) is false
//   bear(41) is false

}
```

Write a main method to test your code.

Problem 4 `Grammar.java (30%)`

Write a recursive program to generate random sentences from a given BNF grammar. A BNF grammar is a recursively defined file that defines rules for creating sentences from tokens of text. Rules can be recursively self-similar. The following grammar can generate sentences such as "Fred honored the green wonderful child":

```
<s>::=<np> <vp>
<np>::=<dp> <adjp> <n>|<pn>
<dp>::=the|a
<adjp>::=<adj>|<adj> <adjp>
<adj>::=big|fat|green|wonderful|faulty|subliminal|pretentious
<n>::=dog|cat|man|university|father|mother|child|television
<pn>::=John|Jane|Sally|Spot|Fred|Elmo
<vp>::=<tv> <np>|<iv>
<tv>::=hit|honored|kissed|helped
<iv>::=died|collapsed|laughed|wept
```

Programming Logic:

1- Read the text file containing the grammar rules (one per line), each rule consists of a symbol on the left-hand side (LHS) of the ::= sequence and related tokens on the right hand-side (RHS). The delimiter is a pipe bar |, if there is only one token in the RHS then there will be no pipe bar

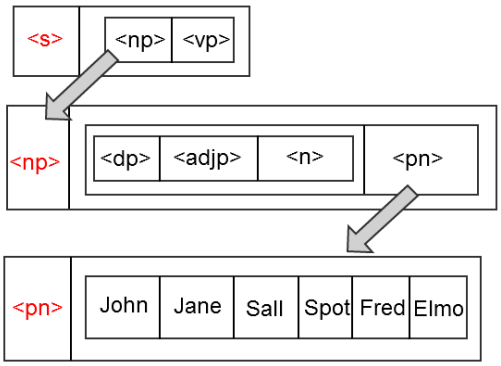
2- A grammar rule consists of terminal and non-terminal symbols, ex:
   a. Terminal symbols: died, collapsed, hit, big, Jane
   b. Non-terminal symbols: <np> <tv> <iv> (they are surrounded by <>)

3- Use a Map data structure to load all the BNF rules above
   a. Parse each rule creating one entry in the map:
      i. The key is the LHS, ex: the first rule's key is <s>
      ii. The value the RHS, ex: The first rule's value is the string <np> <vp>

4- To generate a sentence always start with the <s> key in the map
   a. Get the value corresponding to <s>, i.e. the string <np> <vp>
   b. First split on |, select one of the value at random, if there are not | then continue
   c. split the different tokens <np> and <vp> for each one assign
   d. If the value has at least one | then split that string on | and choose at random a single value
   e. Take the value you just obtained and decide if it is terminal or non-terminal
      i. If you have a terminal symbol, print it

ii. Otherwise use it as a key and ask the map for the corresponding value and repeat the process (recursively)

Example:



Hints:

1- To generate random numbers in Java:

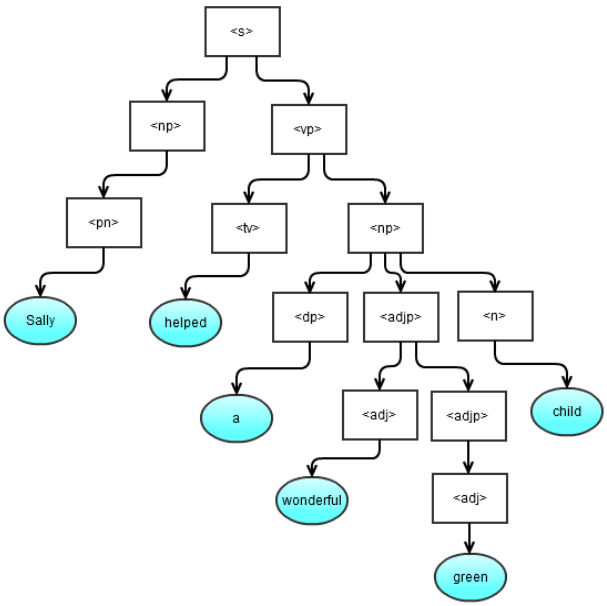
```
Random randomGenerator = new Random();
randomGenerator.nextInt(100);
// Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.
```

2- To split a string based on a delimiter use the `java.util.StringTokenizer` class:

```
StringTokenizer stringTokenizer = new StringTokenizer(line, "|");
while (stringTokenizer.hasMoreElements()) { // do something }
```

Example Run:

For example, starting at <S>, you can end up with the sentence "Sally helped a wonderful green child", following this trace:



Submission Instructions

- Your exam submission must consist of a single zip archive named **s#_exam3_netid** that contains your properly commented source files (.java files). The source files that must be submitted are: **EvenDigits.java**, **Palindromes.java**, **Bears.java**, and **Grammar.java**. No other files will be accepted. We will compile and run your programs.