Dictionary Creation

I created a dictionary from several different lists, each list being formatted (apart from most common passwords). Here is what the dictionary contained:

- **100,000 Most used words in the English language**
  - https://gist.github.com/h3xx/1976236
- **Swearwords in the English language**
  - http://urbanoalvarez.es/blog/2008/04/04/bad-words-list/
- **All known surnames**
  - http://www2.census.gov/topics/genealogy/1990surnames/dist.all.last
- **All known male and female first names**
  - http://www2.census.gov/topics/genealogy/1990surnames/dist.female.first
  - http://www2.census.gov/topics/genealogy/1990surnames/dist.male.first
- **Names of all animals**
  - https://github.com/hzlzh/Domain-Name-List/blob/master/Animal-words.txt
- **All Urban Dictionary words**
  - http://www.urbandictionary.com/
- **All UK/US used Brand Names**
  - http://www.namedevelopment.com/brand-names.html
- **Every Country in the world**
  - http://www.state.gov/misc/list/
- **Every City and Town in the world**
  - https://en.wikipedia.org/wiki/List_of_towns_and_cities_with_100,000_or_more_inhabitants/cityname:_B
- **List of most common passwords**
  - https://raw.githubusercontent.com/neo/discourse_heroku/master/lib/common_passwords/10k-common-passwords.txt

Available Threads

Here is a list of all of the transformations my code does on the characters:

After each transformation each transformed word is hashed and tested against the hashed password list, writing an output if a match occurs. Each bullet point is a method within the worker that is run on a separate thread.

No Transformation:

- noTrans
    - This thread hashed each dictionary entry before checking them against hashed password list directly. No transformation was done on the dictionary.

Capitals:

- Capitals
    - This thread first changed the case of each letter of the word to a capital. (E.g. **One**, **oNe**, **onE**).
    - The thread then changed the case of two letters in the word to capitals. (E.g. **ONe**, **OnE**, **oNE**).
    - The thread then changed all the letters in the word to capital and hashed it. (E.g. **ONE**).
- CapitalNumber
    - Add capitals and number on end (one number and two numbers). (E.g. **One0** … **One9** … **onE0** … **onE9**).

Symbols:

- Symbols
    - This thread first adds a symbol (from a symbol array containing all symbols) to the end of the word. (E.g. **one!**, **one£**, **one$**, etc.).
    - This thread then adds a symbol (from a symbol array containing all symbols) to the front of the word. (E.g. **!one**, **£one**, **$one** etc.).
- symbolsBoth
    - This thread adds a symbol (from a symbol array containing all symbols) to both ends of a word. (E.g. **!one!**, **£one£**, **one**, etc.).
    - The thread then tests the word encapsulated in brackets. (E.g. **(one)**).

L33T:

- l33t
    - This thread first changes every possible character in a word that can be changed to l33t characters. (E.g. **password** → p@**55w0rd**).
    - The thread then changes one symbol in the word, iterating through all possible changes. (E.g. **password** → **p@ssword** → **pa5sword** → **pas5word** → **passw0rd**).
    - The thread then changes two symbols in the word iterating through all possible changes. (E.g. **password** → **p@5sword** → **p@s5word** → **p@ssw0rd** → **pa5sw0rd**).
    - The thread then changed three symbols in the word iterating through all possible changes. (E.g. **password** → **p@55word** → **p@5sw0rd** → **p@s5w0rd** → **pa55w0rd**).

Numbers Solo:

- oneNumberSolo
    - Tests single numbers. (E.g. **1**, **2** … **9**, **0**).
- twoNumberSolo
    - Tests all combinations of two numbers. (E.g. **00**, **01** … **98**, **99**).
- threeNumberSolo
    - Tests all combinations of three numbers. (E.g. **000**, **001** … **998**, **999**).
- fourNumberSolo
    - Tests all combinations of four numbers. (E.g. **0000**, **0001** … **9998**, **9999**).
- fiveNumberSolo
    - Tests all combinations of five numbers. (E.g. **00000**, **00001** … **99998**, **99999**).
- sixNumberSolo
    - Tests all combinations of six numbers. (E.g. **000000**, **000001** … **999998**, **999999**).
- sevenNumberSolo
    - Tests all combinations of seven numbers. (E.g. **0000000**, **0000001** … **9999998**, **9999999**).
- eightNumbersSolo
    - Tests all combinations of eight numbers. (E.g. **00000000** … **99999999**).
- nineNumberSolo
    - Tests all combinations of nine numbers. (E.g. **000000000** … **999999999**).
- tenNumberSolo
    - Tests all combinations of ten numbers. (E.g. **0000000000** … **9999999999**).
- elevenNumberSolo
    - Tests all combinations of eleven numbers. (E.g. **00000000000** … **99999999999**).

Numbers Beg:

- oneNumberBeg
    - Tests word adding single number at beginning. (E.g. **0one**, **1one**, **2one** etc.).
- twoNumberBeg
    - Tests word adding all combinations of two numbers at beginning. (E.g. **00one** … **99one**).
- threeNumberBeg
    - Tests word adding all combinations of three numbers at beginning. (E.g. **000one** … **999one**).
- fourNumberBeg
    - Tests word adding all combinations of four numbers at beginning. (E.g. **0000one** … **9999one**).

Numbers Both:

- oneNumberBoth
    - Tests word adding numbers at beginning and end. (E.g. **0one0** … **9one9**).
- twoNumberBoth
    - Tests word adding all combinations of two numbers at beginning and end. (E.g. **00one00** … **99one99**).

NumbersEnd:

- oneNumberEnd
    - Tests word adding numbers at end. (E.g. **one0** … **one9**).
- twoNumberEnd
    - Tests word adding all combinations of two numbers at end. (E.g. **one00** … **one99**).
- threeNumberEnd
    - Tests word adding all combinations of three numbers at end. (E.g. **one000** … **one999**).
- fourNumberEnd
    - Tests word adding all combinations of four numbers at end. (E.g. **one0000** … **one9999**).

Combine:

Each combination thread (apart from combineSame) processes one quarter of the list with the whole list.

- combineSame
  - Combines the same word together. (E.g. **one** → **oneone**).
- combineTwoWordsOne
  - Combines the first quarter of the words with all other words. (E.g. **one** → **onetwo** → **onethree** → etc.) (Starts with word at 0% and stops at 25%).
- combineTwoWordsTwo
  - Combines the second quarter of the words with all other words. (Same as above but starts with word at 25% and stops at 50%).
- combineTwoWordsThree
  - Combines the third quarter of the words with all other words. (Same as above but starts with word at 50% and stops at 75%).
- combineTwoWordsFour
  - Combines the fourth quarter of the words with all other words. (Same as above but starts with word at 75% and stops at 100%).

To ensure that the threads were run in the most efficient way, I timed each thread in milliseconds and ran them from shortest running time to longest running time taking into account the average number of password matches received. (Those without number values were not able to be completed on my computer, and it was estimated in which order to place them). See run order below in figure 1:

```
commands = {"oneNumberSolo",//1,189
            "twoNumberSolo",//2,341
            "threeNumberSolo",//3,717
            "fourNumberSolo",//6,051
            "fiveNumberSolo",//8,657
            "combineSame",//14,662
            "noTrans",//46,452
            "sixNumberSolo",//49,398
            "oneNumberBeg",//62,842
            "oneNumberBoth",//64,991
            "oneNumberEnd",//74,485
            "capitals",//238,091
            "symbols",//358,327
            "sevenNumberSolo",//386,987
            "twoNumberBeg",//619,021
            "twoNumberEnd",//630,009
            "twoNumberBoth",//649,637
            "l33t",//1,270,990
            "eightNumberSolo",//3,788,402
            "threeNumberBeg",//6,157,056
            "threeNumberEnd",//6,166,286

            "capitalNumber",//

            "fourNumberEnd", //
            "fourNumberBeg", //

            "nineNumberSolo", //
            "tenNumberSolo", //
            "elevenNumberSolo", //

            "combineTwoWordsOne",//
            "combineTwoWordsTwo", //
            "combineTwoWordsThree",//
            "combineTwoWordsFour",

            "symbolsBoth" };//
```

Figure 1: Operations List.

As some i7 quad-core processor can run up to 9 threads at once (depending on their setup) I have opted to run 9 threads (where if only 8 are available then only 8 will be run at once) based on a thread pool of differing operations (figure 2), the operations list is seen above in figure 1.

```
ExecutorService executor = Executors.newFixedThreadPool(9);
for (int i = 0; i < commands.length; i++) {
    Runnable worker = new WorkerThread(commands[i], dict, pwds);
    executor.execute(worker);
  }
```

Figure 2: Thread pool creation.