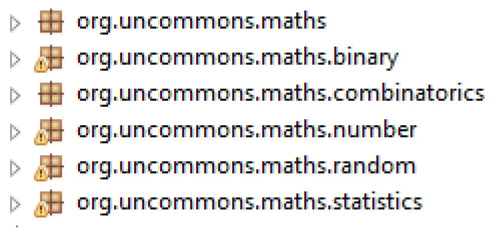
Statistics/Random Number Generator

Overview

We selected the first topic for our group project of this course: Statistics/random number generator library https://github.com/dwdyer/uncommons-maths. According to the introduction of this Java library's author: Daniel W.Dyer, this is a random number generators, probability distributions, combinatorics and statistics library for Java.

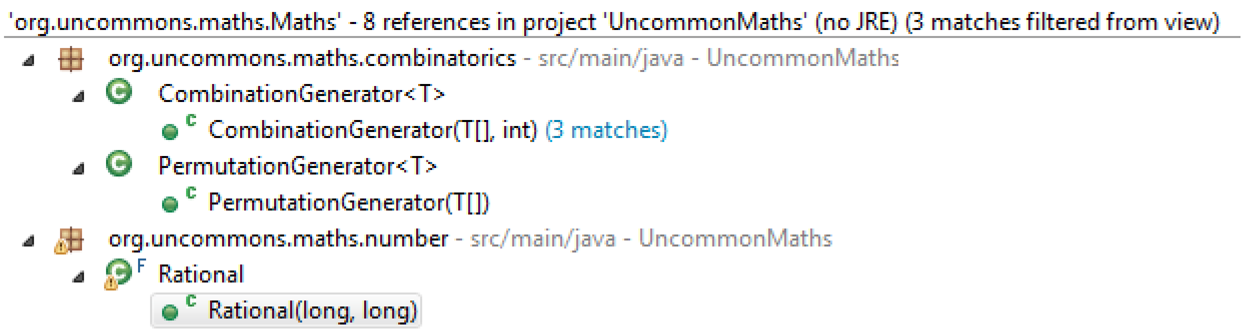
We downloaded the source code from the above web site and the packages are as below:



```
org.uncommons.maths
org.uncommons.maths.binary
org.uncommons.maths.combinatorics
org.uncommons.maths.number
org.uncommons.maths.random
org.uncommons.maths.statistics
```

- The top package is org.uncommons.maths, there is only one Java class under this package directly: Maths.
- The package org.uncommons.maths.binary provides two Java classes for manipulating binary data.
- The package org.uncommons.maths.combinatorics provides two classes: CombinationGenerator and PermutationGenerator.
- The package org.uncommons.maths.number provides some custom numeric data types and classes. For example, there is Raional class which focus on how to perform arithmetic on fractional values without loss of precision.
- The package org.uncommons.maths.random is the core of this library which provides deterministic, repeatable, pseudo-random number generators.
- The package org.uncommons.maths.statistics provides two utility classes for statistical analysis. For example, variance, median, standard deviation and etc.

The Maths class provides some dedicated functions that the standard java.lang.Math does not include, so the usage of this class is similar with java.lang.Math. For example, in combination and permutation operations, to calculate the factorial of one number is used frequently, this Maths provides a function: bigFactorial which can calculate the factorial of n where n is a positive integer. After the analysis, we found a couple of other Java classes use this Maths class as below:



'org.uncommons.maths.Maths' - 8 references in project 'UncommonMaths' (no JRE) (3 matches filtered from view)

- org.uncommons.maths.combinatorics - src/main/java - UncommonMaths
  - CombinationGenerator<T>
    - CombinationGenerator(T[], int) (3 matches)
  - PermutationGenerator<T>
    - PermutationGenerator(T[])
- org.uncommons.maths.number - src/main/java - UncommonMaths
  - Rational
    - Rational(long, long)

Algorithms

In this section, we will focus on the Java class: PermutationGenerator which can generate all permutations for all sets up to 20 elements in size. If the size is 20, the number of permutations will be 2432902008176640000, so there is a huge operation in order to the all of the permutations.

Firstly, in order to analyze how to generate the permutations, we develop a Java class for testing:

```
public static void main(String[] args) {
    String[] elements = {"1", "2", "3"};
    PermutationGenerator<String> generator = new PermutationGenerator<String>(elements);
    long count=generator.getTotalPermutations();
    System.out.println("The number of permutations: "+count);
    for(int i=0;i<count;i++){
        String[] permutation1 = generator.nextPermutationAsArray();
        print(permutation1);
    }

}
```

As expected, the output is as below:

```
The number of permutations: 6
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
```

Secondly, we found that there is an overloaded method:

```
public T[] nextPermutationAsArray(T[] destination) {
    generateNextPermutationIndices();
    for (int i = 0; i < permutationIndices.length; i++) {
        destination[i] = elements[permutationIndices[i]];
    }
    return destination;
}
```

The basic idea of this method is to generate the next permutation and return an array containing the elements in the appropriate order. The algorithm is that with this way, this method allows caller to provide an array that will be used and returned. As a result, the performance will be improved significantly because without this, there will be created a couple of temporary array objects, and waiting to be garbage collected.

The secondly point about algorithm we focus on is how to get the order of permutation with an array. This basic idea is from Kenneth H. Rosen, Discrete Mathematics and itsApplications, 2nd edition (NY: McGraw-Hill, 1991), p. 284). After the analysis, the pseudo code with comments is below:

Step 1: Find largest index j with permutationIndices[j] < permutationIndices[j + 1]

Step 2: Find index k such that permutationIndices[k] is smallest integergreater than permutationIndices[j] to the right of permutationIndices[j].

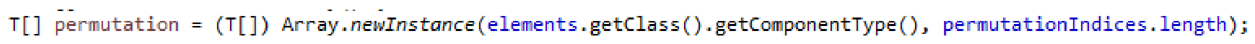
Step 3: swap permutationIndices[k] and permutationIndices[j]

Step 4: Put tail end of permutation after jth position in increasing order.

Finally, get a new order of permutation.

Data structures

In the source code, we found the data structure of Array has been used frequently:



```
T[] permutation = (T[]) Array.newInstance(elements.getClass().getComponentType(), permutationIndices.length);
```

We are familiar with how to array in Java and Generics, according to the introduction at: https://docs.oracle.com/javase/tutorial/java/generics/restrictions.html, we cannot create Arrays of parameterized types. As a result, the following source code is wrong:

T a[] = new T[10];

Alternatively, in order to create a generic array, this library uses Array.newInstance method. We discover the java.lang.reflect.Array at https://docs.oracle.com/javase/7/docs/api/java/lang/reflect/Array.html. It means that this java.lang.reflect.Array class provides static methods to dynamically create and access java arrays.

There are two parameters for newInstance method:

The first one is componentType which means the class object representing the component type of the new array. The second parameter is the length of the new array. This method will return the new array.

With a deep research, we find more useful information about reflection in Java at https://docs.oracle.com/javase/tutorial/reflect/. Generally, reflection is a feature in Java programming language which allows an executing Java program to examine itself and manipulate internal properties of the program. Another useful function about reflection is to find out what methods are defined within a class.

An example about how to use reflection is as below:

```java
private static void printReflection() throws ClassNotFoundException {
    Class cls = Class.forName("TestPermutation");
    Method methlist[] = cls.getDeclaredMethods();
    for (int i = 0; i < methlist.length; i++) {
        Method m = methlist[i];
        System.out.println("name = " + m.getName());
    }
}
```