StatsLibrary, SetOperations, and Dependencies Documentation

By Jake Cubernot

Table Of Contents

StatsLibrary: **2**
SetOperations: **4**
Dependencies: **6**
Testing: **8**

StatsLibrary:

- The StatsLibrary class contains multiple functions associated with statistics. Functions included are central tendencies (mean, median, and mode), standard deviation and variance, permutation, combination, and other functions involving probability and distribution.

- To receive the best result, some functions use the class BigInteger to ensure that formulas that may use large values, like permutations and combinations, will not compile incorrectly for the user:

```
 * @param n The integer value that the user wants to find the factorial result.
 * @return Factorial result as BigInteger.
 */
private BigInteger findFactorial(int n) {
    BigInteger factorialResult = BigInteger.ONE;
    for(int i = n; i > 0; i--) {
        factorialResult = factorialResult.multiply(BigInteger.valueOf(i));
    }
    return factorialResult;
}
```

```
 * @param n The integer value of the set's size.
 * @param r The integer value of the total number of choosen values in the set.
 * @return Permutation result as BigInteger.
 */
public BigInteger findPermutation(int n, int r) {
    BigInteger permutationResult = BigInteger.ONE;
    permutationResult = permutationResult.multiply(findFactorial(n).divide(findFactorial(n - r)));
    return permutationResult;
}
```

```
 * @param n The integer value of the set's size.
 * @param r The integer value of the total number of choosen values in the set.
 * @return Combination result as BigInteger.
 */
public BigInteger findCombination(int n, int r) {
    BigInteger combinationResult = BigInteger.ONE;
    combinationResult = combinationResult.multiply(findFactorial(n).divide(findFactorial(r)
    .multiply(findFactorial(n - r))));
    return combinationResult;
}
```

- Functions like *listToArrayList* and *getList* also may help the user as needed to either convert any int[] to an ArrayList<Integer> or help visually represent their list[]:

```
 * @param userList The list that the user wants to convert to an Arraylist<Integer>.
 * @return The converted int[] to ArrayList<Integer>.
 */
private ArrayList<Integer> listToArrayList(int[] userList) {
    ArrayList<Integer> newArrayList = new ArrayList<Integer>();
    for(int i = 0; i < userList.length; i++) {
        newArrayList.add(userList[i]);
    }
    return newArrayList;
}
```

```
 * @param userList The int[] that the user wants to convert to a String.
 * @return The String that shows a visual representation of the list, int[].
 */
public String getList(int[] userList) {
    String userListToString = "{";
    for (int i = 0; i < userList.length; i++) {
        if (i == userList.length - 1) {
            userListToString += (String.valueOf(userList[i]) + "}");
        } else {
            userListToString += (String.valueOf(userList[i]) + ", ");
        }
    }
    return userListToString;
}
```

SetOperations:

- The SetOperations class contains some basic set operations for mainly integer lists:

```
 * @param userSet1 User's Set A as an int[].
 * @param userSet2 User's Set B as an int[].
 * @return String displaying the result of A union B.
 */
public String findUnion(int[] userSet1, int[] userSet2) {
    ArrayList<Integer> userCombinedSet = listToArrayList(userSet1);
    ArrayList<Integer> userArrayListTemp = listToArrayList(userSet2);
    ArrayList<Integer> unionResult = new ArrayList<Integer>();
    userCombinedSet.addAll(userArrayListTemp);
    userCombinedSet.sort(c:null);
    unionResult.add(userCombinedSet.get(index:0));
    for (int i = 1; i < userCombinedSet.size(); i++) {
        if (userCombinedSet.get(i) != userCombinedSet.get(i - 1)) {
            unionResult.add(userCombinedSet.get(i));
        }
    }
    return getList(arrayListToList(unionResult));
}
```

```
 * @param userSet1 User's Set A as an int[].
 * @param userSet2 User's Set B as an int[].
 * @return String displaying the result of A intersects B.
 */
public String findIntersection(int[] userSet1, int[] userSet2) {
    ArrayList<Integer> intersectionResult = new ArrayList<Integer>();
    for (int i = 0; i < userSet1.length; i++) {
        for (int j = 0; j < userSet2.length; j++) {
            if (userSet1[i] == userSet2[j]) {
                intersectionResult.add(userSet1[i]);
            }
        }
    }
    return getList(arrayListToList(intersectionResult));
}
```

```java
 * @param userSet User's set as an int[].
 * @param minCount The minimum value of the universal set.
 * @param maxCount The maximum value of the universal set.
 * @return String displaying the result of the complement of the user's set
 * where the universal set is a specified set of integer values.
 */
public String findComplementOfNumber(int[] userSet, int minCount, int maxCount) {
    ArrayList<Integer> userArrayList = listToArrayList(userSet);
    ArrayList<Integer> complementResult = new ArrayList<Integer>();
    for (int i = minCount; i <= maxCount; i++) {
        boolean valueInSet = false;
        for (int j = 0; j < userArrayList.size(); j++) {
            if (i == userArrayList.get(j)) {
                valueInSet = true;
                break;
            }
        }
        if (valueInSet == false) {
            complementResult.add(i);
        }
    }
    return getList(arrayListToList(complementResult));
}
```

- In addition, there is also a simple function, *findComplementOfDaysOfWeek,* that finds the complement set of a set where the universal set is the days of the week.

Dependencies:

- The dependencies class is used to find the boolean of whether the given values are an independent event, a dependent event, mutually exclusive, or mutually inclusive:

```
 * @param pA Probability of A
 * @param pB Probability of B
 * @param pAIntersectsB Probability of A intersects B
 * @return Boolean of if is an independent event
 */
public boolean isIndependentEvent(double pA, double pB, double pAIntersectsB) {
    if (pAIntersectsB == pA * pB) {
        return true;
    } else {
        return false;
    }
}
```

```
 * @param pA Probability of A
 * @param pAIntersectsB Probability of A intersects B
 * @param pBGivenA Probability of A given B
 * @return Boolean of if is a dependent event
 */
public boolean isDependentEvent(double pA, double pAIntersectsB, double pBGivenA) {
    if (pAIntersectsB == pA * pBGivenA) {
        return true;
    } else {
        return false;
    }
}
```

```
 * @param pA Probability of A
 * @param pB Probability of B
 * @param pAUnionsB Probability of A unions B
 * @return Boolean of if is mutually exclusive
 */
public boolean isMutuallyExclusive(double pA, double pB, double pAUnionsB) {
    if (pAUnionsB == pA + pB) {
        return true;
    } else {
        return false;
    }
}
```

```java
 * @param pA Probability of A
 * @param pB Probability of B
 * @param pAUnionsB Probability of A unions B
 * @param pAIntersectsB Probability of A intersects B
 * @return Boolean of if is mutually inclusive
 */
public boolean isMutuallyInclusive(double pA, double pB, double pAUnionsB, double pAIntersectsB) {
    if (pAUnionsB == pA + pB - pAIntersectsB) {
        return true;
    } else {
        return false;
    }
}
```

Testing:

- The *TestStatsLibrary* class offers a simple test displaying all of the *StatsLibrary* class's features:

```
Sample List: {3, 9, 1, 4, 5, 7, 4, 6, 8, 2}

The mean is: 4.9
The median is: 4.5
The mode is: 4.0

The standard deviation is: 2.4677925358506134
The variance is: 6.090000000000001

P(5,3) = 60
C(5,3) = 10

Binomial Distribution of n=6, x=2, p=80%: 0.015359999999999988
Conditional Probability of A?B=50% and B=20%: 2.5
Geometric Distribution of k=3 and p=20%: 0.12800000000000003
Hypergeometric Distribution of N=10, n=4, k=3, x=2: 0
```

- The *TestSetOperations* class offers a simple test displaying all of the *SetOperations* class's features:

```
Sample Set A: {1, 2, 3, 4, 5}
Sample Set B: {2, 4, 5, 6, 7, 2}
A union B: {1, 2, 3, 4, 5, 6, 7}
A interect B: {2, 2, 4, 5}
Complement of A where A={Tuesday, Thursday}: [Sunday, Monday, Wednesday, Friday, Saturday]
Complement of A: {-2, -1, 0, 6, 7, 8, 9, 10}
```

- The *TestDependencies* class offers a simple test displaying all of the *Dependencies* class's features:

```
double pA = 0.4;
double pB = 0.3;
double pAIntersectsB = 0.12;
double pBGivenA = 0.6;
double pAUnionsB = 0.58;
```

```
Events A and B are independent: true
Events A and B are dependent: false
Events A and B are mutually exclusive: false
Events A and B are mutually inclusive: true
```