CSA 参考 Answer key

Mcq：

EECDB
DEBDB
DEADC
CAAED
BECCE
BCECC
EEACE
BECAA

FRQ:

1(a).

```
public static int ballThrow() {
    int randomNum = (int)(Math.random() * 5) + 1;
    return randomNum * 10;
}
```

1(b).

```
public static double averageThrow(int numThrows, int minScore) {
    int sum = 0;
    int count = 0; // This will count the number of throws greater than minScore

    for (int i = 0; i < numThrows; i++) {
        int score = ballThrow(); // Simulate a throw using the ballThrow method

        // Check if the score is greater than minScore
        if (score > minScore) {
            sum += score; // Add the score to the sum
            count++; // Increment the count
        }
    }

    // Calculate the average if at least one throw is greater than minScore
    if (count > 0) {
        return (double)sum / count; // Cast sum to double to ensure floating-point division
    } else {
        // If no throws are greater than minScore, return 0.0
        return 0.0;
    }
}
```

2.

```
public class TopSecretWord extends SecretWord {

    // 构造函数，调用超类的构造函数
    public TopSecretWord(String word) {
        super(word);
    }

    // 重写 transformWord 方法
    @Override
    public String transformWord() {
        // 直接使用 getOriginal()方法获取原始单词，而不是使用 transformWord()方法
        String originalWord = getOriginal();
        String transformedWord;

        // 判断原字符串长度的奇偶性以确定替换规则
        if (originalWord.length() % 2 == 0) {
            // 如果长度是偶数，替换字符串的前半部分为"***"
            transformedWord = "***" + originalWord.substring(originalWord.length() / 2);
        } else {
            // 如果长度是奇数，替换字符串的后半部分为"***"
            transformedWord = originalWord.substring(0, (originalWord.length() / 2) + 1) + "***";
        }
        return transformedWord; // 返回转换后的字符串
    }

    // 判断转换后的字符串长度是否大于 5
public boolean checkLength() {
    String transformedWord = transformWord(); // 获取转换后的字符串以供长度测量
    // 使用 if...else...结构来判定长度是否大于 5
    if (transformedWord.length() > 5) {
        return true; // 如果长度大于 5，则返回 true
    } else {
        return false; // 否则，返回 false
    }
}
```

3(a)

```
public static ArrayList<Integer> allInversions(int[] numbers) {
    ArrayList<Integer> result = new ArrayList<>();
    for (int i = 0; i < numbers.length; i++) {
        for (int j = i + 1; j < numbers.length; j++) {
            if (numbers[i] > numbers[j]) {
                // 这是一个倒置对，我们将较大的数字先添加到结果中，然后添加较小
的数字
                result.add(numbers[i]);
                result.add(numbers[j]);
            }
        }
    }
    return result;
}
```

3(b)

```
public static int valueWithMostInversions(int[] numbers) {
    // Get all inversions for the given sequence.
    ArrayList<Integer> inversions = allInversions(numbers);

    int valueWithMostInversions = numbers[0];
    int maxOccurrences = 0;

    // Check each number in the array for the number of occurrences in the list of
inversions.
    for (int number : numbers) {
        int occurrences = countOccur(inversions, number);

        // If the current number has more occurrences than the current max, update
valueWithMostInversions.
        if (occurrences > maxOccurrences) {
            maxOccurrences = occurrences;
            valueWithMostInversions = number;
        }
    }

    // Return the value that has the most occurrences in inversion pairs.
    return valueWithMostInversions;
}
```

4(a)

```
public class TreasureMap {
    private Treasure[][] map;

    /** Constructs a treasure map, as described in part (a)
      * Precondition: r > 0, c > 0, and the size of locs is at least 1 and less than r * c.
      * All locations in locs are valid on the map, and there are no duplicates.
      * Postcondition: map contains r rows and c columns.
      * map contains Treasure objects at each location in locs.
      * All other map elements are null.
      */
    public TreasureMap(int r, int c, ArrayList<Location> locs) {
        // Initialize the map array with the given dimensions.
        map = new Treasure[r][c];

        // Iterate through the list of locations.
        for (Location loc : locs) {
            // For each location, place a new Treasure object in the corresponding position
in the map.
            // The getRow() and getCol() methods are used to access the row and column
indices of the location.
            map[loc.getRow()][loc.getCol()] = new Treasure();
        }
    }
```

4(b)

```
public int totalGold(Location start, Location end) {
    int totalGold = 0;

    // Iterate over each row from the start location to the end location, inclusive.
    for (int i = start.getRow(); i <= end.getRow(); i++) {

        // Iterate over each column from the start location to the end location, inclusive.
        for (int j = start.getCol(); j <= end.getCol(); j++) {

            // Check if there is a Treasure object at the current map location.
            if (map[i][j] != null) {
                // Add the gold from this Treasure to the total.
                totalGold += map[i][j].getGold();
            }
        }
    }
```

```
        // Return the total gold found in the specified rectangular region.
        return totalGold;
}
```