Exercises and Homework

java.util Methods for Arrays

fill(A, x)

```
int[] array = new int[5];
Arrays.fill(array, 10);
System.out.println(Arrays.toString(array)); // Output: [10, 10, 10, 10, 10]
```

copyOf(A, n)

```
int[] array = {1, 2, 3, 4, 5};
int[] newArray = Arrays.copyOf(array, 7);
System.out.println(Arrays.toString(newArray)); // Output: [1, 2, 3, 4, 5, 0, 0]
```

copyOfRange(A, s, t):

```
int[] array = {1, 2, 3, 4, 5};
int[] newArray = Arrays.copyOfRange(array, 1, 4);
System.out.println(Arrays.toString(newArray)); // Output: [2, 3, 4]
```

toString(A)

```
int[] array = {1, 2, 3, 4, 5};
String arrayString = Arrays.toString(array);
System.out.println(arrayString); // Output: [1, 2, 3, 4, 5]
```

sort(A):

```
int[] array = {5, 3, 1, 4, 2};
Arrays.sort(array);
System.out.println(Arrays.toString(array)); // Output: [1, 2, 3, 4, 5]
```

binarySearch(A, x)

```
int[] array = {1, 2, 3, 4, 5};

int index = Arrays.binarySearch(array, 3);

System.out.println(index);
```

R-3.1	Give the next five pseudorandom numbers generated by the process described on page 113, with a = 12, b = 5, and n = 100, and 92 as the seed for cur. See page 113 On page 113 of the book, "Data Structures and Algorithms in Java" by Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser, the process described is a random number generator using the linear congruential method. The formula for generating the next pseudorandom number is as follows: cur = (a * cur + b) % n Given the values a = 12, b = 5, n = 100, and seed for cur = 92, we can calculate the next five pseudorandom numbers using the above formula: 1. cur = (12 * 92 + 5) % 100 = 1109 % 100 = 9 2. cur = (12 * 9 + 5) % 100 = 113 % 100 = 13 3. cur = (12 * 13 + 5) % 100 = 161 % 100 = 61 4. cur = (12 * 61 + 5) % 100 = 737 % 100 = 37 5. cur = (12 * 37 + 5) % 100 = 449 % 100 = 49 Therefore, the next five pseudorandom numbers in this sequence are: 9, 13, 61, 37, 49.

R-3.2	Write a Java method that repeatedly selects and removes a random entry from an array until the array holds no more entries.
	`import java.util.Random;`
	
	`public class RandomArrayRemoval {`
	`  public static void main(String[] args) {`
	`  String[] array = {"A", "B", "C", "D", "E"};`
	`  shuffleArray(array);`
	`  removeRandomEntries(array);`
	`  }`
	
	`  public static void shuffleArray(Object[] array) {`

	Random random = new Random();
	for (int i = array.length - 1; i > 0; i--) {
	int index = random.nextInt(i + 1);
	Object temp = array[index];
	array[index] = array[i];
	array[i] = temp;
	}
	}
	
	public static void removeRandomEntries(Object[] array) {
	Random random = new Random();
	int numEntries = array.length;
	
	for (int i = 0; i < numEntries; i++) {
	int randomIndex = random.nextInt(numEntries - i);
	Object removedEntry = array[randomIndex];
	
	// Move the last element to the randomly selected position
	array[randomIndex] = array[numEntries - i - 1];
	array[numEntries - i - 1] = removedEntry;
	
	System.out.println("Removed entry: " + removedEntry);
	}
	}
	}

R-3.3	Explain the changes that would have to be made to the program of Code Fragment 3.8 so that it could perform the Caesar cipher for messages that are written in an alphabet-based language other than English, such as Greek, Russian, or Hebrew. Alphabet: Modify the ALPHABET string to include the characters specific to the chosen language. For example, for

	the Greek alphabet, you would update the ALPHABET string to include the Greek letters. Array Size: Update the size of the shiftedAlphabet array to match the number of characters in the chosen language's alphabet. The array should accommodate all the characters used in the language. Shifting: Adjust the shifting logic to match the specific alphabet. In the original code, the shifting is based on the English alphabet's index positions. You would need to update the shifting logic to work with the indices of the chosen language's alphabet. For example, if the Greek alphabet is used, the shifting logic should consider the Greek letters' indices.

R-3.4	The TicTacToe class of Code Fragments 3.9 and 3.10 has a flaw, in that it allows a player to place a mark even after the game has already been won by someone. Modify the class so that the putMark method throws an IllegalStateException in that case
	`public class TicTacToe {`
	`  private static final int BOARD_SIZE = 3;`
	`  private String[][] board;`
	`  private boolean gameOver;`
	
	`  public TicTacToe() {`
	`  board = new String[BOARD_SIZE][BOARD_SIZE];`
	`  gameOver = false;`
	`  }`
	
	`  public void putMark(int row, int col, String mark) {`
	`  if (gameOver) {`
	`  throw new IllegalStateException("Game has already been won");`
	`  }`

```
        if (row < 0 || row >= BOARD_SIZE || col < 0 || col >=
BOARD_SIZE || board[row][col] != null) {
            throw new IllegalArgumentException("Invalid
position");
        }

        board[row][col] = mark;
        gameOver = checkForWin(mark);
    }

    public boolean checkForWin(String mark) {
        // Check rows and columns for a win
        for (int i = 0; i < BOARD_SIZE; i++) {
            boolean rowWin = true;
            boolean colWin = true;

            for (int j = 0; j < BOARD_SIZE; j++) {
                if (!mark.equals(board[i][j])) {
                    rowWin = false;
                }
                if (!mark.equals(board[j][i])) {
                    colWin = false;
                }
            }

            if (rowWin || colWin) {
                return true;
            }
        }

        // Check diagonals for a win
        if (mark.equals(board[0][0]) &&
mark.equals(board[1][1]) && mark.equals(board[2][2])) {
            return true;
        }
        if (mark.equals(board[0][2]) &&
mark.equals(board[1][1]) && mark.equals(board[2][0])) {
            return true;
        }
```

	return false;
	}
	
	public boolean isGameOver() {
	return gameOver;
	}
	}

R-3.13	What is the difference between a shallow equality test and a deep equality test between two Java arrays, A and B, if they are one-dimensional arrays of type int? What if the arrays are two-dimensional arrays of type int?
	
	Here are examples of shallow and deep equality tests for one-dimensional int arrays:
	
	int[] A = {1, 2, 3};
	int[] B = {1, 2, 3};
	int[] C = A;
	
	System.out.println(A == B); // Output: false (shallow equality test)
	System.out.println(Arrays.equals(A, B)); // Output: true (deep equality test)
	System.out.println(A == C); // Output: true (shallow equality test)
	
	As for two-dimensional int arrays, the same concept applies:
	
	int[][] A = {{1, 2}, {3, 4}};
	int[][] B = {{1, 2}, {3, 4}};
	int[][] C = A;
	
	System.out.println(A == B); // Output: false (shallow equality test)

	System.out.println(Arrays.deepEquals(A, B)); // Output: true (deep equality test)
	System.out.println(A == C); // Output: true (shallow equality test)

R-3.14	Give three different examples of a single Java statement that assigns variable, backup, to a new array with copies of all int entries of an existing array, original.
	
	Using Arrays.copyOf() method:
	`int[] backup = Arrays.copyOf(original, original.length);`
	Using a for loop to copy each element individually:
	`int[] backup = new int[original.length];`
	`for (int i = 0; i < original.length; i++) {`
	`  backup[i] = original[i];`
	`}`
	Using Arrays.copyOf() with two-dimensional arrays:
	`int[][] backup = Arrays.copyOf(original, original.length);`
	`for (int i = 0; i < original.length; i++) {`
	`  backup[i] = Arrays.copyOf(original[i], original[i].length);`
	`}`

R-3.15	What is the difference between System.arraycopy() and Arrays.copyOf()/Arrays.copyOfRange() methods in Java?
	
	One way to fill a one-dimensional int array in Java with the value 5 is by using the static initializer block in the array declaration:
	`int[] arr = new int[10] {{1, 2, 3, 4, 5, 5, 5, 5, 5, 5}};`
	Another way is by using Java's Stream API introduced in Java 8:
	`int[] arr = IntStream.generate(() -> 5).limit(10).toArray();`

C-3.17	Let A be an array of size $n \geq 2$ containing integers from 1 to $n-1$ inclusive, one of which is repeated. Describe an algorithm for finding the integer in A that is repeated.

	public static int findDuplicate(int[] A) {
	int tortoise = A[0];
	int hare = A[0];
	
	while (true) {
	tortoise = A[tortoise];
	hare = A[A[hare]];
	
	if (tortoise == hare) {
	break;
	}
	}
	
	tortoise = A[0];
	while (tortoise != hare) {
	tortoise = A[tortoise];
	hare = A[hare];
	}
	
	return hare;
	}

C-3.18	Let B be an array of size n ≥ 6 containing integers from 1 to n−5 inclusive, five of which are repeated. Describe an algorithm for finding the five integers in B that are repeated.
	
	import java.util.ArrayList;
	import java.util.HashMap;
	import java.util.List;
	import java.util.Map;
	
	public class RepeatedIntegersFinder {
	public static List<Integer> findRepeatedIntegers(int[] B) {
	Map<Integer, Integer> frequencyMap = new HashMap<>();

```
        for (int num : B) {
            frequencyMap.put(num,
frequencyMap.getOrDefault(num, 0) + 1);
        }

        List<Integer> repeatedIntegers = new ArrayList<>();

        for (Map.Entry<Integer, Integer> entry :
frequencyMap.entrySet()) {
            if (entry.getValue() > 1) {
                repeatedIntegers.add(entry.getKey());
            }
            if (repeatedIntegers.size() == 5) {
                break;
            }
        }

        return repeatedIntegers;
    }

    public static void main(String[] args) {
        int[] B = {1, 2, 3, 4, 5, 2, 3, 4, 6, 1, 5};

        List<Integer> repeatedIntegers =
findRepeatedIntegers(B);

        System.out.println("Repeated Integers: " +
repeatedIntegers);
    }
}
```

C-3.19	Give Java code for performing add(e) and remove(i) methods for the Scoreboard class, as in Code Fragments 3.3 and 3.4, except this time, don't maintain the game entries in order. Assume that we still need to keep n entries stored in indices 0 to n−1. You should be able to implement the methods without using any loops, so that the number of steps they perform does not depend on n.
	public class Scoreboard {
	private String[] scoreboard;
	private int n;
	
	public Scoreboard(int n) {
	this.n = n;
	scoreboard = new String[n];
	}
	
	public void add(String e) {
	scoreboard[n - 1] = e;
	n = n > 0 ? n - 1 : 0;
	}
	
	public String remove(int i) {
	String removedElement = scoreboard[i];
	scoreboard[i] = scoreboard[n - 1];
	scoreboard[n - 1] = null;
	n = n < scoreboard.length ? n + 1 : scoreboard.length;
	return removedElement;
	}
	}

C-3.20	Give examples of values for a and b in the pseudorandom generator given on page 113 of this chapter such that the result is not very random looking, for n = 1000.
	
	r = (a * r + b) % n
	int a = 214013;
	int b = 2531011;

C-3.21	Suppose you are given an array, A, containing 100 integers that were generated using the method r.nextInt(10), where r is an object of type java.util.Random. Let x denote the product of the integers in A. There is a single number that x will equal with probability at least 0.99. What is that number and what is a formula describing the probability that x is equal to that number?

The question is asking for the probability that the product of 100 integers, each randomly chosen between 0 and 9 (inclusive), is equal to a specific number.

Let's first calculate the total number of possible outcomes. Since each integer can take on 10 possible values, the total number of possible outcomes is 10^100.

Next, let's consider a specific number, say N. We want to find the number of ways to write N as a product of 100 integers between 0 and 9. Note that if N is 0, then there is only one way to write it as a product (all 100 integers are 0).

For N > 0, we can write N as a product of 100 integers by choosing how many times each integer from 1 to 9 appears in the product. Let x_i be the number of times the integer i appears in the product, for i from 1 to 9. Then we have:

N = 1^x1 * 2^x2 * ... * 9^x9

where x_1 + x_2 + ... + x_9 = 100.

The number of ways to write N in this form is equal to the number of solutions to the equation:

x_1 + x_2 + ... + x_9 = 100

where x_i is a non-negative integer for all i.

This is a classic combinatorial problem, and the number of solutions is given by the formula:

C(9+100-1, 100) = C(108, 100) = 108! / (100! * 8!)

Therefore, the probability that the product of 100 integers is equal to N is:

	P(N) = (number of ways to write N as a product) / (total number of possible outcomes) For N = 0, we have: P(0) = 1 / 10^100 For N > 0, we have: P(N) = C(108, 100) / 10^100 Therefore, the formula describing the probability that the product of 100 integers is equal to a specific number N is: P(N) = { 1 / 10^100, if N = 0 C(108, 100) / 10^100, if N > 0 }

C-3.22	Write a method, shuffle(A), that rearranges the elements of array A so that every possible ordering is equally likely. You may rely on the nextInt(n) method of the java.util.Random class, which returns a random number between 0 and n−1 inclusive.
	import java.util.Random;
	
	public class ArrayShuffler {
	public static void shuffle(int[] A) {
	Random rand = new Random();
	
	for (int i = A.length - 1; i > 0; i--) {
	int j = rand.nextInt(i + 1); // Random index between 0 and i (inclusive)
	
	// Swap elements at indices i and j
	int temp = A[i];
	A[i] = A[j];
	A[j] = temp;
	}
	}
	
	public static void main(String[] args) {
	int[] A = {1, 2, 3, 4, 5};
	
	System.out.println("Original array: ");

```
        for (int num : A) {
            System.out.print(num + " ");
        }

        shuffle(A);

        System.out.println("\nShuffled array: ");
        for (int num : A) {
            System.out.print(num + " ");
        }
    }
}
```

C-3.23

Suppose you are designing a multiplayer game that has n ≥ 1000 players, numbered 1 to n, interacting in an enchanted forest. The winner of this game is the first player who can meet all the other players at least once (ties are allowed). Assuming that there is a method meet(i, j), which is called each time a player i meets a player j (with i 6= j), describe a way to keep track of the pairs of meeting players and who is the winner.

```
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class MultiplayerGame {
    private int n; // number of players
    private List<Set<Integer>> graph; // adjacency list representation of the graph
    private int[] meetingCounts; // count of unique meetings for each player
    private Set<Integer> winners; // set of winners

    public MultiplayerGame(int n) {
        this.n = n;
        graph = new ArrayList<>();
        meetingCounts = new int[n + 1]; // player numbers start from 1
        winners = new HashSet<>();
```

```
        // Initialize graph and meetingCounts
        for (int i = 0; i <= n; i++) {
            graph.add(new HashSet<>());
            meetingCounts[i] = 0;
        }
    }

    public void meet(int i, int j) {
        if (i == j) {
            return; // Players cannot meet themselves
        }

        if (!graph.get(i).contains(j)) {
            // Increment the meeting count for player i
            meetingCounts[i]++;

            // Check if player i has met all other players
            if (meetingCounts[i] == n - 1) {
                winners.add(i);
            }
        }

        if (!graph.get(j).contains(i)) {
            // Increment the meeting count for player j
            meetingCounts[j]++;

            // Check if player j has met all other players
            if (meetingCounts[j] == n - 1) {
                winners.add(j);
            }
        }

        // Update the graph to indicate the meeting between
players i and j
        graph.get(i).add(j);
        graph.get(j).add(i);
    }

    public Set<Integer> getWinners() {
```

	return winners; }}

C-3.24	Write a Java method that takes two three-dimensional integer arrays and adds them componentwise.
	public class Main {
	
	public static void main(String[] args) {
	int[][][] arr1 = {{{1, 2, 3}, {4, 5, 6}}, {{7, 8, 9}, {10, 11, 12}}};
	int[][][] arr2 = {{{13, 14, 15}, {16, 17, 18}}, {{19, 20, 21}, {22, 23, 24}}};
	int[][][] result = add3DArrays(arr1, arr2);
	
	// Print the result
	for (int i = 0; i < result.length; i++) {
	for (int j = 0; j < result[i].length; j++) {
	for (int k = 0; k < result[i][j].length; k++) {
	System.out.print(result[i][j][k] + " ");
	}
	System.out.println();
	}
	System.out.println();
	}
	}
	
	public static int[][][] add3DArrays(int[][][] arr1, int[][][] arr2) {
	int[][][] result = new int[arr1.length][arr1[0].length][arr1[0][0].length];
	for (int i = 0; i < arr1.length; i++) {
	for (int j = 0; j < arr1[i].length; j++) {
	for (int k = 0; k < arr1[i][j].length; k++) {
	result[i][j][k] = arr1[i][j][k] + arr2[i][j][k];
	}
	}
	}
	return result;
	}
	}

الاسم : زهرة خالد

تقنية معلومات عربي
