Exercises and Homework

java.util Methods for Arrays

fill(A, x)

copyOf(A, n)

copyOfRange(A, s, t):

toString(A)

sort(A):

binarySearch(A, x)

|  |  |  |
| --- | --- | --- |
| 1 | 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  9  13  61  37  49 |
| 2 | 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.  public static void removeRandomEntries(int[] array) {  Random random = new Random();  while (array.length > 0) {  // Generate a random index within the array's current bounds  int index = random.nextInt(array.length);  // Remove the entry at the randomly selected index  int removedEntry = array[index];  // Shift the remaining elements to fill the gap created by the removal  for (int i = index; i < array.length - 1; i++) {  array[i] = array[i + 1];  }  // Decrement the array's effective length  array[array.length - 1] = 0;  array = Arrays.copyOf(array, array.length - 1);  }  }  كيف يعمل الكود؟  تحويل المصفوفة إلى قائمة: يتم تحويل المصفوفة إلى قائمة ArrayList لأن المصفوفة في Java ثابتة الحجم، بينما القائمة تسمح بالإزالة بسهولة.  اختيار الفهرس العشوائي: يتم استخدام random.nextInt(list.size()) لتحديد فهرس عشوائي ضمن حدود القائمة الحالية.  الإزالة والطباعة:  يتم طباعة العنصر الموجود عند الفهرس العشوائي.  يتم إزالة العنصر باستخدام list.remove(randomIndex).  تكرار العملية:  تستمر العملية داخل حلقة while حتى تصبح القائمة فارغة (list.isEmpty()). |
| 3 | 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.  def caesar\_cipher(message, shift, alphabet):  """  شيفرة قيصرية تعمل مع أبجدية مخصصة  """  # تعريف متغير لتخزين النص المشفر  encrypted\_message = []  alphabet\_size = len(alphabet)    # تحويل الرسالة إلى قائمة للحروف  for char in message:  if char in alphabet:  # الحصول على الفهرس الحالي للحرف  original\_index = alphabet.index(char)  # حساب الفهرس الجديد بعد الإزاحة  new\_index = (original\_index + shift) % alphabet\_size  # إضافة الحرف المشفر  encrypted\_message.append(alphabet[new\_index])  else:  # إذا كان الحرف غير موجود في الأبجدية، يُترك كما هو  encrypted\_message.append(char)    # تحويل القائمة إلى نص وإرجاعه  return ''.join(encrypted\_message)  # أمثلة على الأبجديات المختلفة  greek\_alphabet = list("αβγδεζηθικλμνξοπρστυφχψω")  russian\_alphabet = list("абвгдежзийклмнопрстуфхцчшщъыьэюя")  hebrew\_alphabet = list("אבגדהוזחטיכלמנסעפצקרשת")  # مثال على النصوص والتجريب  message = "καλημέρα"  shift = 3  result\_greek = caesar\_cipher(message, shift, greek\_alphabet)  print("Greek Ciphered Message:", result\_greek)  message\_russian = "привет"  shift = 5  result\_russian = caesar\_cipher(message\_russian, shift, russian\_alphabet)  print("Russian Ciphered Message:", result\_russian)  message\_hebrew = "שלום"  shift = 4  result\_hebrew = caesar\_cipher(message\_hebrew, shift, hebrew\_alphabet)  print("Hebrew Ciphered Message:", result\_hebrew)  الأبجدية المخصصة: قمنا بتعريف الأبجديات كقوائم تحتوي على الحروف الخاصة بكل لغة (اليونانية، الروسية، العبرية). يمكنك تعديلها لأي لغة أخرى.  التشفير: إذا كان الحرف موجودًا في الأبجدية، يتم:  إيجاد فهرسه.  حساب الفهرس الجديد باستخدام الإزاحة مع الالتفاف (%).  إضافة الحرف المشفر إلى النص الجديد.  الحروف غير الموجودة في الأبجدية: إذا كان الحرف غير أبجدي (مثل الأرقام أو الرموز)، يُترك كما هو.  الإزاحة والالتفاف: استخدام modulus (%) لضمان الالتفاف عند نهاية الأبجدية. |
| 4 | 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 X = 1, O = -1; // قيم اللاعبين  private static final int EMPTY = 0; // خانة فارغة  private int[][] board = new int[3][3]; // لوحة اللعبة  private int currentPlayer; // اللاعب الحالي  private boolean gameWon = false; // حالة الفوز  // إنشاء اللعبة  public TicTacToe() {  currentPlayer = X; // يبدأ اللاعب X  }  // وضع العلامة  public void putMark(int row, int col) {  // تحقق إذا انتهت اللعبة  if (isGameOver()) {  throw new IllegalStateException("The game is already over. No more moves are allowed.");  }  if (row < 0 || row >= 3 || col < 0 || col >= 3) {  throw new IllegalArgumentException("Invalid board position.");  }  if (board[row][col] != EMPTY) {  throw new IllegalArgumentException("Board position is already occupied.");  }  // وضع العلامة  board[row][col] = currentPlayer;  // تحقق من حالة الفوز  if (isWinner(row, col)) {  gameWon = true;  }  // تغيير اللاعب  currentPlayer = -currentPlayer;  }  // التحقق إذا انتهت اللعبة  public boolean isGameOver() {  return gameWon || isBoardFull();  }  // تحقق إذا امتلأ الجدول  private boolean isBoardFull() {  for (int row = 0; row < 3; row++) {  for (int col = 0; col < 3; col++) {  if (board[row][col] == EMPTY) {  return false;  }  }  }  return true;  }  // تحقق من حالة الفوز  private boolean isWinner(int row, int col) {  int sumRow = 0, sumCol = 0, sumDiag = 0, sumAntiDiag = 0;  for (int i = 0; i < 3; i++) {  sumRow += board[row][i];  sumCol += board[i][col];  sumDiag += board[i][i];  sumAntiDiag += board[i][2 - i];  }  return Math.abs(sumRow) == 3 || Math.abs(sumCol) == 3 || Math.abs(sumDiag) == 3 || Math.abs(sumAntiDiag) == 3;  }  // طباعة اللوحة  public String toString() {  StringBuilder sb = new StringBuilder();  for (int row = 0; row < 3; row++) {  for (int col = 0; col < 3; col++) {  switch (board[row][col]) {  case X: sb.append(" X "); break;  case O: sb.append(" O "); break;  case EMPTY: sb.append(" "); break;  }  if (col < 2) sb.append("|");  }  sb.append("\n");  if (row < 2) sb.append("---+---+---\n");  }  return sb.toString();  }  // فائز اللعبة  public String winner() {  if (gameWon) {  return currentPlayer == -X ? "X" : "O";  }  return "No winner";  }  }  شرح الكود:  isGameOver:  تتحقق من حالتين:  إذا كان هناك فائز (gameWon = true).  إذا امتلأت كل الخانات (isBoardFull).  putMark:  تضيف شرطًا في البداية لرمي استثناء إذا انتهت اللعبة بالفعل.  تتأكد من صحة الإحداثيات وعدم شغل الخانة قبل وضع العلامة.  isWinner:  تتحقق مما إذا كان اللاعب الحالي قد فاز عن طريق جمع القيم في:  الصف.  العمود.  القطر الرئيسي.  القطر العكسي.  إذا كان المجموع مطلقًا (absolute value) يساوي 3، فهذا يعني الفوز.  toString:  لطباعة لوحة اللعبة بشكل جميل.  winner:  تعيد اسم الفائز إذا انتهت اللعبة، أو "No winner" إذا لم تنتهِ بعد. |
| 5 | 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?  import java.util.Arrays;  public class ArrayEquality {  public static void main(String[] args) {  // One-dimensional arrays  int[] arrayA = {1, 2, 3};  int[] arrayB = {1, 2, 3};  int[] arrayC = arrayA; // Reference to the same array as arrayA  // Shallow equality using ==  System.out.println("Shallow equality between A and B: " + (arrayA == arrayB)); // false  System.out.println("Shallow equality between A and C: " + (arrayA == arrayC)); // true  // Deep equality using Arrays.equals  System.out.println("Deep equality between A and B: " + Arrays.equals(arrayA, arrayB)); // true  System.out.println("Deep equality between A and C: " + Arrays.equals(arrayA, arrayC)); // true  }  } |
| 6 | 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.  backup = original.clone( );  int[ ] temp = Arrays.copyOf(original, n);  **public** **static** **void** arraycopy(      Object src\_array, **int** src\_Pos,Object dest\_array, **int** dest\_Pos, **int** length )  System.arraycopy(src\_array, 0, dest\_array, 0,19); |
| 7 | C-3.17 | Let A be an array of size n ≥ 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.  def find\_repeated\_element(B):  distinct\_elements = set()  for b in B:  if b in distinct\_elements:  return b  else:  distinct\_elements.add(b)  return None |
| 8 | 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.  Algorithm:   * Create a set S to store the distinct elements encountered so far. Initialize S to an empty set. * Iterate through the array B: a. For each element b in B: i. If b is not in S, add b to S. This indicates that the element b has been seen once. ii. If b is already in S, then b is a repeated element. Add b to a list of repeated elements. * Since there are five repeated elements, continue iterating through B until you find five distinct elements that are repeated. * The list of repeated elements contains the five repeated integers in B.   Analysis:  Time Complexity: O(n), where n is the size of the array B. This is because the algorithm iterates through the array B only once, and each operation takes constant time.  Space Complexity: O(n-5), where n-5 is the size of the set S. This is because the set S stores a maximum of n-5 distinct elements. |
| 9 | 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 int n = 0; // عدد الإدخالات الحالية  private GameEntry[] board; // مصفوفة لتخزين الإدخالات  // الباني (Constructor)  public Scoreboard(int capacity) {  board = new GameEntry[capacity]; // إنشاء المصفوفة بحجم محدد  }  // طريقة لإضافة إدخال جديد  public void add(GameEntry e) {  if (n < board.length) { // إذا كانت هناك مساحة متاحة  board[n] = e; // أضف الإدخال في الموقع التالي  n++; // زيادة العدد الإجمالي  } else {  board[n - 1] = e; // استبدل الإدخال الأخير  }  }  // طريقة لإزالة إدخال عند الفهرس i  public GameEntry remove(int i) throws IndexOutOfBoundsException {  if (i < 0 || i >= n) { // تحقق من صحة الفهرس  throw new IndexOutOfBoundsException("الفهرس غير صالح: " + i);  }  GameEntry removed = board[i]; // احفظ الإدخال المراد إزالته  board[i] = board[n - 1]; // استبدله بالإدخال الأخير  board[n - 1] = null; // احذف الإدخال الأخير  n--; // قلل العدد الإجمالي  return removed; // أعد الإدخال الذي تمت إزالته  }  // طريقة اختيارية لعرض محتويات المصفوفة  public void display() {  for (int i = 0; i < n; i++) {  System.out.println(board[i]);  }  }  // فئة لتمثيل الإدخالات  public static class GameEntry {  private String name;  private int score;  public GameEntry(String name, int score) {  this.name = name;  this.score = score;  }  public String toString() {  return "(" + name + ", " + score + ")";  }  }  // طريقة لاختبار البرنامج  public static void main(String[] args) {  Scoreboard scoreboard = new Scoreboard(5);  scoreboard.add(new GameEntry("Alice", 100));  scoreboard.add(new GameEntry("Bob", 200));  scoreboard.add(new GameEntry("Charlie", 150));  scoreboard.add(new GameEntry("Diana", 180));  scoreboard.add(new GameEntry("Eve", 130));  System.out.println("اللوحة بعد إضافة الإدخالات:");  scoreboard.display();  scoreboard.add(new GameEntry("Frank", 250)); // استبدال الإدخال الأخير  System.out.println("\nاللوحة بعد إضافة Frank:");  scoreboard.display();  scoreboard.remove(2); // إزالة Charlie (الفهرس 2)  System.out.println("\nاللوحة بعد إزالة Charlie:");  scoreboard.display();  }  } |
| 10 | 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.  import java.util.HashSet;  public class PseudoRandomGenerator {  public static void main(String[] args) {  // إعداد القيم  int n = 1000; // الحد  int a = 10; // معامل الضرب  int b = 0; // معامل الإضافة  int x0 = 1; // القيمة الابتدائية    int iterations = 20; // عدد القيم لتوليدها  int[] results = new int[iterations];    // التوليد  results[0] = x0;  for (int i = 1; i < iterations; i++) {  results[i] = (a \* results[i - 1] + b) % n;  }  // طباعة النتائج  System.out.println("Generated sequence:");  for (int i = 0; i < iterations; i++) {  System.out.print(results[i] + " ");  }  // التحقق من التكرار  HashSet<Integer> uniqueValues = new HashSet<>();  for (int value : results) {  uniqueValues.add(value);  }  System.out.println("\nUnique values: " + uniqueValues.size());  System.out.println("Number of repeats: " + (iterations - uniqueValues.size()));  }  } |
| 11 | 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? |
| 12 | 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.  public static void shuffle(int[] A) {  Random rnd = new Random();  for (int i = A.length - 1; i > 0; i--) {  // Swap the current element with a randomly chosen element from the remaining array  int j = rnd.nextInt(i + 1);  int temp = A[i];  A[i] = A[j];  A[j] = temp;  }  } |
| 13 | 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.  Here's a strategy to track pairs of meeting players and determine the winner(s):  1. Data Structure for Tracking Meetings:   * Bit Array: Employ a 2D boolean array meetings of size n x n. Set meetings[i][j] = true when players i and j meet, indicating a meeting has occurred. * Alternative: For extremely large n, consider a Bit Set for memory efficiency.   2. Tracking Meetings within meet(i, j):   * When meet(i, j) is called: * Set both meetings[i][j] and meetings[j][i] to true (meetings are bidirectional). * Increment counters for both players: * meetingCount[i]++ * meetingCount[j]++   3. Determining Winners:   * After each meet(i, j) call: * Check if either player i or j has met all other players: * If meetingCount[i] == n-1, player i has met everyone and is a winner. * Similarly, if meetingCount[j] == n-1, player j is a winner.   4. Handling Ties:   * Maintain a Set<Integer> called winners to track distinct winners. * Whenever a potential winner is found, add their ID to winners.   5. Code Structure (Example in Java):  Java  // Data structures  boolean[][] meetings = new boolean[n][n];  int[] meetingCount = new int[n];  Set<Integer> winners = new HashSet<>();  void meet(int i, int j) {  meetings[i][j] = meetings[j][i] = true;  meetingCount[i]++;  meetingCount[j]++;  if (meetingCount[i] == n - 1) {  winners.add(i);  // Handle player i's victory (e.g., announce win, mark as finished)  }  if (meetingCount[j] == n - 1) {  winners.add(j);  // Handle player j's victory (e.g., announce win, mark as finished)  }  }  Use code with caution. [Learn more](https://bard.google.com/faq)  content\_copy  Additional Considerations:   * Game Ending: Decide when the game ends (e.g., when a certain number of winners emerge or after a time limit). * Winner Announcement: Appropriately announce the winner(s) based on the game's design. * Memory Optimization: For extremely large n, consider memory-optimized data structures like Bit Sets for storing meeting information. |
| 14 | C-3.24 | Write a Java method that takes two three-dimensional integer arrays and adds them componentwise.  public static int[][] addThreeDimensionalArrays(int[][][] array1, int[][][] array2) {  // Check if the arrays have the same dimensions  if (array1.length != array2.length || array1[0].length != array2[0].length || array1[0][0].length != array2[0][0].length) {  throw new IllegalArgumentException("Arrays must have the same dimensions");  }  // Create a new array to store the result  int[][][] result = new int[array1.length][array1[0].length][array1[0][0].length];  // Add the corresponding elements of the two arrays  for (int i = 0; i < result.length; i++) {  for (int j = 0; j < result[0].length; j++) {  for (int k = 0; k < result[0][0].length; k++) {  result[i][j][k] = array1[i][j][k] + array2[i][j][k];  }  }  }  return result;  } |