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.

```
public class PseudorandomNumbers {
    public static void main(String[] args) {
        int a = 12;
        int b = 5;
        int n = 100;
        int seed = 92;
        int cur = seed;

        for (int i = 0; i < 5; i++) {
            cur = (a * cur + b) % n;
            System.out.println(cur);
        }
    }
}
```

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 RandomEntryRemoval {
   public static void main(String[] args) {
      String[] array = {"Apple", "Banana", "Orange", "Grapes", "Mango"};
      removeRandomEntries(array);
   }

   public static void removeRandomEntries(Object[] array) {
      Random rand = new Random();

      while (array.length > 0) {
         int randomIndex = rand.nextInt(array.length); // Generate a random index
         Object removedEntry = array[randomIndex]; // Get the entry at the random
index
         System.out.println("Removed: " + removedEntry);

         // Shift the elements to the left to fill the gap
         for (int i = randomIndex; i < array.length - 1; i++) {
            array[i] = array[i + 1];
         }

         // Decrease the array size by 1
```

```
            Object[] newArray = new Object[array.length - 1];
            System.arraycopy(array, 0, newArray, 0, array.length - 1);
            array = newArray;
        }
    }
}
```

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.

```
public class CaesarCipherGreek {
    public static void main(String[] args) {
        String message = "Γειά σας!"; // Greek message
        int shift = 3; // Shift by 3 positions

        String encryptedMessage = caesarCipher(message, shift);
        System.out.println("Encrypted Message: " + encryptedMessage);

        String decryptedMessage = caesarCipher(encryptedMessage, -shift); //
Decrypt by shifting back
        System.out.println("Decrypted Message: " + decryptedMessage);
    }

    public static String caesarCipher(String message, int shift) {
        StringBuilder result = new StringBuilder();

        for (int i = 0; i < message.length(); i++) {
            char character = message.charAt(i);

            if (Character.isLetter(character)) {
                int originalAlphabetPosition = character - 'A'; // Calculate the original
alphabet position
                int newAlphabetPosition = (originalAlphabetPosition + shift) % 24; //
Greek alphabet has 24 characters
                char newCharacter = (char) ('A' + newAlphabetPosition); // Convert back
to the Greek character
```

```
            result.append(newCharacter);
        } else {
            result.append(character); // Non-alphabetic character, leave it unchanged
        }
    }

    return result.toString();
  }
}
```

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?

```
int[] A = {1, 2, 3};
int[] B = {1, 2, 3};

int[][] C = {{1, 2, 3}, {4, 5, 6}};
int[][] D = {{1, 2, 3}, {4, 5, 6}};

// Shallow Equality Test
System.out.println(A == B); // false (Different memory references for one-dimensional arrays)
System.out.println(C == D); // false (Different memory references for two-dimensional arrays)

// Deep Equality Test
System.out.println(Arrays.equals(A, B)); // true (Same content for one-dimensional arrays)
System.out.println(Arrays.deepEquals(C, D)); // true (Same content for two-dimensional arrays)
```

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.

```
int[] backup = new int[original.length];
for (int i = 0; i < original.length; i++) {
   backup[i] = original[i];
}
```

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.

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

public class FindRepeatedInteger {
   public static int findRepeatedNumber(int[] A) {
      Set<Integer> visited = new HashSet<>();

      for (int num : A) {
         if (visited.contains(num)) {
            return num; // Found the repeated integer
         }
         visited.add(num);
      }

      return -1; // No repeated integer found
   }

   public static void main(String[] args) {
      int[] A = {1, 2, 3, 4, 2}; // Example input array
      int repeatedNumber = findRepeatedNumber(A);
      System.out.println("Repeated Number: " + repeatedNumber);
   }
}
```

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.HashSet;
import java.util.Set;

public class FindRepeatedIntegers {
    public static Set<Integer> findRepeatedNumbers(int[] B) {
        Set<Integer> visited = new HashSet<>();
        Set<Integer> repeated = new HashSet<>();

        for (int num : B) {
            if (visited.contains(num)) {
                repeated.add(num); // Found a repeated integer
            }
            visited.add(num);
        }

        return repeated;
    }

    public static void main(String[] args) {
        int[] B = {1, 2, 3, 4, 2, 3, 4, 5, 1, 5}; // Example input array
        Set<Integer> repeatedNumbers = findRepeatedNumbers(B);
        System.out.println("Repeated Numbers: " + repeatedNumbers);
    }
}
```

Write a Java method that takes two three-dimensional integer arrays and adds them componentwise.

```
public class ArrayAddition {
    public static int[][][] addArrays(int[][][] array1, int[][][] array2) {
        int size1 = array1.length;
        int size2 = array1[0].length;
        int size3 = array1[0][0].length;

        int[][][] result = new int[size1][size2][size3];

        for (int i = 0; i < size1; i++) {
            for (int j = 0; j < size2; j++) {
```

```
            for (int k = 0; k < size3; k++) {
                result[i][j][k] = array1[i][j][k] + array2[i][j][k];
            }
        }
    }

    return result;
}

public static void main(String[] args) {
    int[][][] array1 = {
        {{1, 2, 3}, {4, 5, 6}},
        {{7, 8, 9}, {10, 11, 12}}
    };

    int[][][] array2 = {
        {{10, 20, 30}, {40, 50, 60}},
        {{70, 80, 90}, {100, 110, 120}}
    };

    int[][][] result = addArrays(array1, array2);

    // Printing 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();
    }
}
}
```

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); // Generate a random index between 0 and i
(inclusive)

         // Swap elements at index 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};
      shuffle(A);

      // Print the shuffled array
      for (int num : A) {
         System.out.print(num + " ");
      }
      System.out.println();
   }
}
```