A. Java Program to Print an Array

To understand this example, you should have the knowledge of the following Java programming topics:

- Java Arrays
- Java Multidimensional Arrays
- Java for Loop

Example 1: Print an Array using For loop

```
public class Array {
   public static void main(String[] args) {
      int[] array = {1, 2, 3, 4, 5};
      for (int element: array) {
         System.out.println(element);
      }
   }
}
```

Output

```
1
2
3
4
5
```

In the above program, the for-each loop is used to iterate over the given array, array.

It accesses each element in the array and prints using println().

Example 2: Print an Array using standard library Arrays

```
import java.util.Arrays;
public class Array {
   public static void main(String[] args) {
      int[] array = {1, 2, 3, 4, 5};
      System.out.println(Arrays.toString(array));
```

```
    }
}
```

Output

```
[1, 2, 3, 4, 5]
```

In the above program, the for loop has been replaced by a single line of code using Arrays.toString() function.

As you can see, this gives a clean output without any extra lines of code.

Example 3: Print a Multi-dimensional Array

```
import java.util.Arrays;

public class Array {

    public static void main(String[] args) {

        int[][] array = {{1, 2}, {3, 4}, {5, 6, 7}};

        System.out.println(Arrays.deepToString(array));

    }

}
```

Output

```
[[1, 2], [3, 4], [5, 6, 7]]
```

In the above program, since each element in array contains another array, just using Arrays.toString() prints the address of the elements (nested array).

To get the numbers from the inner array, we just another function Arrays.deepToString(). This gets us the numbers 1, 2 and so on, we are looking for.

This function works for 3-dimensional arrays as well.

B. Java Program to Concatenate Two Arrays

To understand this example, you should have the knowledge of the following Java programming topics:

- Java Arrays
- Java for-each Loop

Example 1: Concatenate Two Arrays using arraycopy

```
import java.util.Arrays;

public class Concat {

   public static void main(String[] args) {

      int[] array1 = {1, 2, 3};

      int[] array2 = {4, 5, 6};

      int aLen = array1.length;

      int bLen = array2.length;

      int[] result = new int[aLen + bLen];

      System.arraycopy(array1, 0, result, 0, aLen);

      System.arraycopy(array2, 0, result, aLen, bLen);

      System.out.println(Arrays.toString(result));

   }

}
```

Output

```
[1, 2, 3, 4, 5, 6]
```

In the above program, we've two integer arrays array1 and array2.

In order to combine (concatenate) two arrays, we find its length stored in aLen and bLen respectively. Then, we create a new integer array result with length aLen + bLen.

Now, in order to combine both, we copy each element in both arrays to result by using arraycopy() function.

The arraycopy(array1, 0, result, 0, aLen) function, in simple terms, tells the program to copy array1 starting from index 0 to result from index 0 to aLen.

Likewise, for arraycopy(array2, 0, result, aLen, bLen) tells the program to copy array2 starting from index 0 to result from index aLen to bLen.

Example 2: Concatenate Two Arrays without using arraycopy

```
import java.util.Arrays;

public class Concat {

   public static void main(String[] args) {

      int[] array1 = {1, 2, 3};

      int[] array2 = {4, 5, 6};

      int length = array1.length + array2.length;

      int[] result = new int[length];

      int pos = 0;

      for (int element : array1) {

         result[pos] = element;

         pos++;

      }

      for (int element : array2) {

         result[pos] = element;

         pos++;

      }

      System.out.println(Arrays.toString(result));

   }

}
```

Output

```
[1, 2, 3, 4, 5, 6]
```

In the above program, instead of using arraycopy, we manually copy each element of both arrays array1 and array2 to result.

We store the total length required for result, i.e. array1.length + array2. length. Then, we create a new array result of the length.

Now, we use the for-each loop to iterate through each element of array1 and store it in the result. After assigning it, we increase the position pos by 1, pos++.

Likewise, we do the same for array2 and store each element in result starting from the position after array1.

Before we wrap up, let's put your knowledge of Java Program to Concatenate Two Arrays to the test! Can you solve the following challenge?

Challenge:

Write a function to concatenate two arrays to form a target array.

- Given two arrays arr1 and arr2, concatenate them, and return the resulting array.
- For example, if arr1[] = {1, 2, 3} and arr2[] = {4, 5, 6}, the expected output is {1, 2, 3, 4, 5, 6}.

```
1
2
3
4
5
public class Solution {
    public static int[] concatenateArrays(int[] arr1, int[] arr2)
  {
    }
}
```

C. Java Program to Check if An Array Contains a Given Value

Example 1: Check if Int Array contains a given value

```
class Main {
  public static void main(String[] args) {
    int[] num = {1, 2, 3, 4, 5};
    int toFind = 3;
    boolean found = false;
    for (int n : num) {
      if (n == toFind) {
        found = true;
        break;
      }
    }
    if(found)
      System.out.println(toFind + " is found.");
    else
      System.out.println(toFind + " is not found.");
  }
}
```

Output

3 is found.

In the above program, we have an array of integers stored in variable num. Likewise, the number to be found is stored in toFind.

Now, we use a for-each loop to iterate through all elements of num and check individually if toFind is equal to n or not.

If yes, we set found to true and break from the loop. If not, we move to the next iteration.

Example 2: Check if an array contains the given value using Stream

```
import java.util.stream.IntStream;

class Main {

  public static void main(String[] args) {

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

    int toFind = 7;

    boolean found = IntStream.of(num).anyMatch(n -> n == toFind);

    if(found)

      System.out.println(toFind + " is found.");

    else

      System.out.println(toFind + " is not found.");

  }

}
```

Output

```
7 is not found.
```

In the above program, instead of using a for-each loop, we convert the array to an IntStream and use its anyMatch() method.

anyMatch() method takes a predicate, an expression, or a function that returns a boolean value. In our case, the predicate compares each element n in the stream to toFind and returns true or false.

If any of the element n returns true, found is set to true as well.

Example 3: Check if an array contains a given value for non-primitive types

```
import java.util.Arrays;
class Main {
  public static void main(String[] args){
    String[] strings = {"One","Two","Three","Four","Five"};
    String toFind= "Four";
    boolean found = Arrays.stream(strings).anyMatch(t -> t.equals(toFind));
    if(found)
      System.out.println(toFind + " is found.");
    else
      System.out.println(toFind + " is not found.");
  }
}
```

Output

Four is found.

In the above program, we've used a non-primitive data type String and used Arrays's stream() method to first convert it to a stream and anyMatch() to check if the array contains the given value toFind.