Programmation orientée objets : contrôle continu

Prénom :
Nom :

Pour chaque programme, vous devez indiquer l'affichage produit en répondant sur l'énoncé, à droite du programme. **Soyez très précis dans vos réponses.**

```
import java.util.Arrays;

public class Exo11 {
  public static void main(String[] args) {
    int[] tab = { 2, 3, 4 };
    int[] plop = tab;
    int[] foo = { 2, 3, 4 };
    int x = foo[2];
    foo[2] = foo[1] + 4;
    System.out.println(x + " " + foo[2] + " " + plop[2]);
    int j = 0;
    for (int i : tab) {
      i = i - 2;
      j = j + i;
    }
    System.out.println(j);
    System.out.println(Arrays.toString(tab));
    for (int k = 0; k < plop.length; k++) {
      plop[k] = plop[k] + 1;
      tab[k] = tab[k] + 1;
      foo[k] = foo[k] - 1;
    }
    System.out.println(Arrays.toString(tab));
    System.out.println(Arrays.toString(plop));
    System.out.println(Arrays.toString(foo));
  }
}
```

Affichage produit

```
import java.util.Arrays;

public class Exo21 {
  public static void main(String[] args) {
    double[][] x = new double[3][];
    for (int i = 0; i < x.length; i++) {
      x[i] = new double[i + 1];
      for (int j = 0; j < i + 1; j++) {
        x[i][j] = i + j;
      }
    }
    System.out.println(Arrays.deepToString(x));
    double[][] y = x;
    double[] z = x[1];
    y[0][0] = 2;
    y[1] = new double[] {-2.0};
    System.out.println(Arrays.deepToString(x));
    System.out.println(Arrays.deepToString(y));
    System.out.println(Arrays.toString(z));
  }
}
```

Affichage produit
La méthode `Arrays.deepToString` *permet d'afficher un tableau à plusieurs dimensions.*

```
import java.util.Arrays;

public class Exo31 {
  public static void main(String[] args) {
    String[] foo = { "V", "W", "X", "Y", "Z" };
    String[] ploc = foo;
    String jhe = ploc[2];
    String bar = "AbTu";
    for (int i = 1; i < foo.length; i++) {
      foo[i] = foo[i - 1] + bar.charAt(i - 1);
    }
    System.out.println(Arrays.toString(foo));
    System.out.println(Arrays.toString(ploc));
    System.out.println(jhe);
    StringBuilder[] arf = new StringBuilder[5];
    for (int i = 0; i < arf.length; i++) {
      arf[i] = new StringBuilder();
      arf[i].append(i);
    }
    System.out.println(Arrays.toString(arf));
    StringBuilder[] rbt = arf;
    StringBuilder oza = rbt[2];
    for (int i = 0; i < arf.length; i++) {
      rbt[i] = arf[i].append(arf.length-i);
    }
    System.out.println(Arrays.toString(rbt));
    System.out.println(Arrays.toString(arf));
    System.out.println(oza);
  }
}
```

Affichage produit

```
import java.math.BigInteger;
import java.util.Scanner;

public class Exo41 {
    public static void main(String[] args) {
        Scanner scan = // À COMPLÉTER
        int nb = // À COMPLÉTER
        BigInteger res = // À COMPLÉTER
        // À COMPLÉTER
        System.out.println(res);
    }
}
```

Compléter le programme ci dessus pour qu'il calcule puis affiche le produit des entiers impairs strictement inférieurs à la valeur entrée par l'utilisateur. On devra obligatoirement utiliser les `BigInteger`.

```
import java.util.Scanner;

public class Exo51 {
    public static void main(String[] args) {
        Scanner scan = // À COMPLÉTER
        int base = // À COMPLÉTER
        int nb = // À COMPLÉTER
        StringBuilder motif1 = // À COMPLÉTER
        StringBuilder motif2 = // À COMPLÉTER
        // À COMPLÉTER (construction des motifs)
        StringBuilder result = // À COMPLÉTER
        for (int i = 0; i < nb; i++) {
            // À COMPLÉTER
        }
        System.out.println(result);
    }
}
```

Compléter le programme ci dessus pour qu'il un texte de la forme `++===++` obtenu en mettant bout à bout deux motifs. Le premier motif est constitué de `base` fois le symbole `+`. Le second motif est constitué de `base+1` fois le symbole `=`. Le résultat est obtenu en enchaînant `nb` motifs en alternance, en commençant par le premier motif. Dans l'exemple, on avait donc `base=2` et `nb=3`.