Pour les exercices d'affichage, vous devez impérativement répondre sur l'énoncé et le rendre en fin d'examen (dans une copie cachetée permettant de vous identifier après correction). Exceptée cette partie cachetée, rien ne doit pouvoir identifier votre copie et votre énoncé.

```
public class Exo11 { // Affichage (1)
  public static void main(String[] args) {
    String s = "";
    int i = 2;
    int j = 3;
    for (int k = 0; k < 5; k++) {
      s = s + k;
      System.out.println(i + ", " + j);
      i += 2;
      for (int u = 0; u < s.length(); u++) {
        j = j + u % 2;
      }
      System.out.println(s);
      if (j > 4) {
        i = i - 1;
      }
    }
  }
}
```

Affichage produit

```
import java.util.Arrays;

public class Exo21 { // Affichage (2)
  public static void main(String[] args) {
    int[] u = new int[4];
    for (int i = 0; i < u.length; i++) {
      u[i] = 2 * i + 1;
    }
    System.out.println(u);
    boolean[] v = new boolean[u.length + 1];
    v[0] = true;
    int j = 1;
    for (int i = 0; i < u.length; i++) {
      v[i + 1] = u[i] > j;
      if (v[i]) {
        j = j + 2;
      }
    }
    System.out.println(j);
    System.out.println(Arrays.toString(v));
    int m = 4;
    for(int k: u) {
      k = k - 1;
      System.out.println(k);
      v[m] = !v[m];
      m--;
    }
    System.out.println(Arrays.toString(u));
    System.out.println(Arrays.toString(v));
  }
}
```

Affichage produit

Programmation (1)
Écrire un programme **complet** qui réalise les opérations suivantes :

1. demande à l'utilisateur une valeur entière k ;
2. affiche la valeur de $u_j = \sum_{i=1}^{j}(2i+1)$ pour j allant de k à 1 dans l'ordre décroissant de j. Pour $k = 2$, on affichera ainsi :

```
u_2 = 8
u_1 = 3
```

Affichage produit

```
import java.util.Arrays;

public class Exo31 { // Affichage (3)

  public static void main(String[] args) {
    int[][] bla = new int[2][];
    int[][] foo = new int[2][];
    for (int i = 0; i < bla.length; i++) {
      int[] sub = new int[i + 2];
      for (int j = 0; j < sub.length; j++) {
        sub[j] = j + i + 1;
      }
      System.out.println(Arrays.toString(sub));
      bla[i] = sub;
      foo[bla.length - i - 1] = sub;
    }
    System.out.println(Arrays.deepToString(bla));
    System.out.println(Arrays.deepToString(foo));
    bla[1][1] = bla[1][1] + 2;
    int[] pouic = foo[1];
    foo[1] = new int[] { 0, 1, 6 };
    foo[1][0] = -1;
    bla[0][1] = -2;
    System.out.println(Arrays.deepToString(bla));
    System.out.println(Arrays.deepToString(foo));
    System.out.println(Arrays.toString(pouic));
  }

}
```

Programmation (2)
Écrire un programme qui réalise les opérations suivantes (on pourra se contenter de donner le contenu de la méthode `main`) :

1. demande à l'utilisateur deux valeurs entières k et j ;
2. crée un tableau à deux dimensions de taille $k \times j$;
3. place dans la case $[k][j]$ du tableau le nombre réel $\sqrt{k^2 + j^2}$ (on rappelle que la méthode `Math.sqrt` permet le calcul d'une racine carrée) ;
4. affiche les lignes du tableau une par une, sans utiliser les méthodes `Arrays.toString` et `Arrays.deepToString`. Attention, on souhaite un passage à la ligne seulement à la fin d'une ligne du tableau, pas après chaque nombre. Les nombres d'une ligne seront séparés par le signe de ponctuation deux points (:).

```java
import java.math.BigInteger;

public class Exo41 { // Affichage (4)
  private BigInteger x;

  private StringBuilder y;

  public Exo41(int z) {
    x = BigInteger.valueOf(z);
    y = new StringBuilder(x.toString());
  }

  public void plop(int z) {
    x = x.add(BigInteger.valueOf(z));
    y = y.append(z);
  }

  public void foo(Exo41 that) {
    x = that.x.add(x);
    y = y.append(that.y);
  }

  public long getX() {
    return x.longValue();
  }

  @Override
  public String toString() {
    return y.toString();
  }
}

public class TestExo41 {
  public static void main(String[] args) {
    Exo41 a = new Exo41(3);
    System.out.println(a);
    Exo41 b = new Exo41(3);
    Exo41 c = a;
    String s = c.toString();
    a.plop(2);
    System.out.println(a);
    System.out.println(b + " " + c + " " + s);
    Exo41 d = new Exo41(45);
    Exo41 e = d;
    b.foo(d);
    System.out.println(b + " " + d + " " + e);
    Exo41 f = new Exo41(0);
    while (f.getX() < 4) {
      f.plop(1);
      System.out.println(f);
    }
  }
}
```

Affichage produit

Programmation (3)

1. Modifiez la classe `Exo41` pour rendre ses instances immuables en gardant le même comportement général. Vous pouvez vous contenter d'écrire les parties modifiées, sans recopier ce qui ne change pas.
2. Modifiez la méthode main de la classe `TestExo41` pour qu'elle donne exactement le même affichage que la version actuelle mais en utilisant la version immuable de `Exo41`.

Programmation (4)

Un objet de la classe `Compteur` représente un compteur qui peut prendre des valeurs entières entre 0 et `max`. Les objets de cette classe sont **immuables**. Compléter le programme suivant en fonction des commentaires :

```
public class Compteur {
    // ajouter les variables nécessaires

    public Compteur(int max) {
        // le compteur est initialisé à zéro
        // sa valeur maximale possible est max
    }

    public int valeur() {
        // renvoie la valeur du Compteur appelant
    }

    public boolean estMax() {
        // renvoie true si et seulement si la valeur maximale est atteinte
    }

    public Compteur incrémente() {
        // renvoie un nouveau Compteur dont la valeur est n+1 si la valeur du
        // Compteur appelant est n, sauf si n vaut max. Si n vaut max, la valeur
        // reste la même
    }

    public Compteur reset() {
        // renvoie un nouveau Compteur dont la valeur est 0
    }

    @Override
    public String toString() {
        // représentation du Compteur sous la forme "valeur [max]"
    }
}
```

Proposer une version **modifiable** de cette classe en adaptant les définitions des méthodes à cette situation. On se contentera de donner les méthodes différentes de celles de la version immuable.

```
public class A1 { // Affichage (5)
  private int x;

  public A1(int x) {
    this.x = x;
  }

  public int foo() {
    return 2 * x;
  }

  public B1 bar(B1 that) {
    x = x - 2 * that.bar();
    B1 res = new B1(x);
    return res;
  }

  @Override
  public String toString() {
    return "A1@" + x;
  }
}

public class B1 {
  private int x;

  public B1(int x) {
    this.x = x + 1;
  }

  public void foo(A1 that) {
    x = x - that.foo();
  }

  public int bar() {
    return -x;
  }

  @Override
  public String toString() {
    return "B1@" + x;
  }
}

public class TestAB1 {

  public static void main(String[] args) {
    A1 a = new A1(2);
    B1 b = new B1(4);
    System.out.println(a + " " + b);
    b.foo(a);
    System.out.println(a + " " + b);
    a = new A1(1);
    b = new B1(3);
    a.bar(b);
    System.out.println(a + " " + b);
    B1 c = a.bar(b);
    System.out.println(a + " " + b + " " + c);
    a = new A1(3);
    b = new B1(1);
    b = a.bar(a.bar(b));
    System.out.println(a + " " + b);
  }

}
```

Affichage produit

```
public class A1 {
  protected int x;
  public A1(int x) {
    this.x = x;
  }
  public int f() {
    return x + 1;
  }
  @Override
  public String toString() {
    return "[" + x + "]";
  }
}

public class B1 extends A1 {
  public B1(int x) {
    super(x);
  }
  @Override
  public int f() {
    return x + 2;
  }
}

public class C1 extends A1 {
  private int y;
  public C1(int y, int x) {
    super(x);
    this.y = y;
  }
  @Override
  public String toString() {
    return y + "<>" + x;
  }
}

public class D1 extends B1 {
  public D1(int x) {
    super(x + 1);
  }
  public int g() {
    return x - 2;
  }
  @Override
  public String toString() {
    return "{" + x + "}";
  }
}

public class E1 extends C1 {
  public E1(int y, int x) {
    super(y - 1, x + 1);
  }
  public int g(int z) {
    return x + z;
  }
}
```

Pour chaque ligne du programme suivant, indiquez l'affichage produit si la ligne compile (il peut ne pas y avoir d'affichage pour une ligne qui compile, ce qu'il faut indiquer), ou indiquez que la ligne ne compile pas.

```
A1 a = new A1(4);                                          1
System.out.println(a + " " + a.f());                       2
B1 b = new B1(5);                                          3
System.out.println(b + " " + b.f());                       4
Object o = b;                                              5
System.out.println(o);                                     6
C1 c = new C1(1, 2);                                       7
System.out.println(c + " " + c.f());                       8
a = c;                                                     9
System.out.println(a + " " + a.f());                       10
D1 d = new D1(3);                                          11
System.out.println(d + " " + d.f() + " " + d.g());         12
b = d;                                                     13
System.out.println(b + " " + b.f());                       14
E1 e = new E1(0, 3);                                       15
System.out.println(e + " " + e.f() + " " + e.g(3));        16
a = e;                                                     17
System.out.println(a + " " + a.f() + " " + a.g());         18
b = e;                                                     19
System.out.println(b + " " + b.f() + " " + b.g(3));        20
```