1. L'exécution du `main` met en évidence un problème de sécurité pour la classe `A`. Quel est-il ? Quelle est la solution à ce problème (on ne demande pas de code) ?

L'adresse du tableau `FINAL_D` est recopié en l'état sans copie défensive dans le constructeur de la classe A. Donc l'appelant peut à loisir modifier les éléments stockés dans ce tableau sans qu'une instance de A ne puisse le contrôler. On retrouve la même faille de sécurité dans le getter `getFINAL_D()` qui retourne l'adresse du tableau et non l'adresse d'une copie de ce tableau. Dans les deux cas (constructeur et getter) il faudrait réaliser une copie défensive superficielle (car le type stocké est primitif, int) du tableau respectivement reçu et émis).

2. Donnez un nouveau code pour la méthode `getFINAL_D` qui corrige pour celle-ci le problème de sécurité

```
public int[] getFINAL_D(){
      int[] ret=null;
      if (this.FINAL_D!=null){
            ret=new int[this.FINAL_D.length];
            for (int i=0;i<ret.length;i++){
                  ret[i]=this.FINAL_D[i]; // copie superficielle car type primitive int
            }
      }
      return ret;
}
```

3. Donnez le code java d'une méthode `public int getMode()` de la classe `A`. Elle doit retourner la valeur du tableau FINAL_D qui a le plus d'occurrences (la première de ces valeurs en cas d'égalité). Exemples : {5,6,4} retourne 5 (la première car les 3 valeurs n'ont qu'une seule occurrence), {4,2,2,4,7} retourne 4, {2,3,6,5,6} retourne 6.

Voici un algorithme trivial en $O(n^2)$. Pour chaque élément, on cherche son nombre d'occurrence. On mémorise le max des occurrences réalisés et on retient la valeur correspondante. Il n'a pas été compté complètement faux de passer par un algo de comptage usant du principe que les valeurs stockées sont positives et en nombre limité (algo en O(n)). Mais ici aucune hypothèse n'était faite sur les valeurs positives ou négatives stockées !

```
public int getMode() {
      int ret =-1;
      int nbMax=0;
      int nbOcc=0;
      if ((this.FINAL_D!=null)&&(this.FINAL_D.length>0)){
            ret=this.FINAL_D[0];
            for (int i=0;i<this.FINAL_D.length;i++){
                  nbOcc=0; // encore aucune occurence
                  for (int j=i;j<this.FINAL_D.length;j++){
                        if(this.FINAL_D[i]==this.FINAL_D[j]){ //une occurrence
                                    nbOcc++;
                        }
                  }
                  if (nbOcc>nbMax){ // on a battu le precedent score
```

```
                    nbMax=nbOcc;
                    ret=this.FINAL_D[i];
                }
        }

}
else{
            System.out.println("Ce calcul est impossible");
        }
return ret;
}
```

```
public class A{
    private String a;
    private final char FINAL_B;
    private final String FINAL_C;
    private final int[] FINAL_D;

    public A(String paramA, char paramB, String paramC, int[] paramD){
        this.a = paramA;
        this.FINAL_B = paramB;
        this.FINAL_C = paramC;
        this.FINAL_D = paramD;
    }
    public String getA(){return this.a;}
    public char getFINAL_B(){return this.FINAL_B;}
    public String getFINAL_C(){return this.FINAL_C;}
    public int[] getFINAL_D(){return this.FINAL_D;}

    public void setA(String newA){
        this.a = newA;
    }
    public String toString(){
        String str = a + "\n" + FINAL_B + "\n" + FINAL_C +"\n";
        for(int i = 0; i < FINAL_D.length;i++){
            str += FINAL_D[i] + " ";
        }
        return str;
    }
    public static void main(String[] args) {
        String nom = "mon instance";
        char monChar = 'Z';
        String str = new String ("chaine1");
        int[] tab = {3,4,5};
        A monA = new A(nom,monChar,str,tab);
        tab[0] = 100;
        str = "chaine2";
        monChar = 'B';
        monA.setA("chaine2");
        int[] tab2 = monA.getFINAL_D();
        tab2[1] = 200;
        System.out.println(monA);
    }
}
```