Nom / Prénom : _____

1. Déclarez un tableau d'entiers de 10 éléments et remplissez le 3ème et 6ème élément. Déclarez ensuite un tableau de caractères dont vous initialiserez tout de suite l'intégralité du contenu.

Rappel de cours :

Pour déclarer un tableau, on utilise la syntaxe suivante :

```
typeTableau[] nomTableau;
```

On remplace *typeTableau* par un des types connus : **int, long, float, double, boolean, char, String**
Attention ! Il est **obligatoire** d'initialiser le tableau avant de pouvoir l'utiliser.
On utilise pour cela la commande **new** en spécifiant la taille de la manière suivante :

```
typeTab[] nomTab = new typeTab[taille];
```

On remplace évidemment *taille* par la taille voulue pour le tableau.
Ici on initialise un tableau de 10 chaînes de caractères

```
String[] tabChaine = new String[10];
```

Attention ! Après cette initialisation, les cases du tableau sont ***vides !***
On peut alors stocker des valeurs dans les cases du tableau en utilisant l'*affectation* de la manière suivante :

```
tabChaine[2] = "Peace Love & Unity";
tabChaine[4] = fonctionQuiRetourneString();
```

Il est évidemment possible d'utiliser des *variables* ***entières*** pour accéder aux éléments du tableau:

```
int index = 2;
tabChaine[index + 2] = "5ème élément";
```

On peut récupérer la taille d'un tableau *nomTableau* en utilisant *nomTableau*.***length***
Attention ! Rappellez-vous que les éléments d'un tableau vont de **0 à *nomTableau.length - 1***

```
tabChaine[tabChaine.length - 1] = "Dernier";
```

Enfin il est possible d'initialiser l'intégralité du contenu d'un tableau dès le départ. On écrit alors simplement :

```
int[] tabEnt = {1, 1, 2, 3, 5, 8, 13, 21, 34,
55};
```

```
int[] tabEntiers = new int[10];
tabEntiers[2] = 42;
tabEntiers[5] = 23;

char[] tabChars = {'w', 'a', 'o', 'u', 'h', ' ', '!'}
```

2. Ecrivez une boucle permettant d'afficher les entiers pairs de 2 à 500

Rappel de cours :

Les boucles permettent de répéter des opérations suivant certains critères et donc d'écrire (encore) moins d'instructions. On distingue deux types de boucles principaux. La boucle **while** et la boucle **for.**

```
while (e)
{
  B1;
}
```

L'expression **e** doit être un test booléen.

```
int sommeI = 0, j = 0;
while (j <= 100)
{
    sommeI = sommeI + j;
    j++;
}
System.out.println("Somme de 0 à 100 : " + sommeI);
```

La boucle **for** permet de réduire les instructions :

```
for (e1; e2; e3)
{
  B1;
}
```

- L'expression ***e1*** initialise l'itérateur et le déclare
- L'expression ***e1*** teste si la condition d'achèvement
- L'expression ***e1*** modifie l'itérateur.

Les expressions e1 et e3 sont optionnelles.
On peut remarquer que la structure **for** telle que définie au dessus est équivalente au while :

```
e1;
while (e2)
{
  B1;
  e3;
}
```

On peut ainsi effectuer facilement des calculs complexes :

```
int n[] = new int[100];
// Calculer la somme des entiers pour indice du vecteur
for (int i = 1; i < 100; i++)
    n[i] = n[i - 1] + i;
```

Attention à l'instruction *return* qui <u>signifie la fin de la fonction</u> !!!!

```
int index;
for (index = 2; index <= 500; index = index + 2)
{
   System.out.println(index);
}
```

3. Déclarez une fonction *sommationTableau()* qui prend en argument un tableau de flottants et retourne la somme de tous les éléments de ce tableau.

```
public static float sommationTableau(float[] tabArgument)
{
    int cpt;
    float resultat = 0.0;
    for (cpt = 0; cpt < tabArgument.length; cpt++)
    {
        resultat = resultat + tabArgument[cpt];
    }
    return resultat;
}
```

4. Déclarez une fonction *power2()* qui prend en argument un entier *max* et retourne un tableau d'entiers contenant les puissances de 2 allant de 2^0 à 2^{max}.

```
public static int[] power2(int max)
{
    int cpt, currentPow = 1;
    int[] resultat = new int[max + 1];
    for (cpt = 0; cpt <= max; cpt++)
    {
        resultat[cpt] = currentPow;
        currentPow = currentPow * 2;
    }
    return resultat;
}
```