R1.01 – TP2(B)
Fonctions

L'objectif de ce TP est :
- *d'étudier la définition de fonctions et leur utilisation dans une classe main() d'une classe java*
- *de perfectionner l'emploi des structures de contrôle (if / loop)*

1. Avant de commencer...

✓ Lisez entièrement ce sujet

✓ Ouvrez un terminal et placez-vous dans votre répertoire `R1.01` et lancez `IJ` en tapant la commande `idea`

✓ Suivez la même procédure que celle indiquée dans le TP2(A) pour créer un nouveau projet de nom `TP2B`

NOTE Les parties sont indépendantes les unes des autres, commencez par celle qui vous inspire le plus…

RAPPEL : le fichier `UTILISATION_IJ` accessible à partir de ce parcours, peut vous être utile…

2. Jeu de Nim

*Le jeu de **Nim** peut être pratiqué par deux joueurs ou plus, chaque joueur jouant à son tour.*

RÈGLE DU JEU adaptée pour ce TP :
- Les joueurs s'accordent sur un nombre d'allumettes à disposer sur une table (ce nombre doit être supérieur ou égal à 3 fois le nombre de joueurs)
- Les joueurs jouent à tour de rôle tant qu'il reste des allumettes sur la table
- Lorsque c'est son tour, un joueur choisit le nombre d'allumettes qu'il veut retirer (minimum 1 allumette, maximum 4 allumettes)
- Le gagnant est celui qui a retiré les dernières allumettes

2.1. Créez une nouvelle classe java `Nim`

2.2. Dans la classe `Nim` :

- Créez une fonction `saisieEntPosMin` dont la spécification est la suivante :

```
private static int saisieEntPosMin(int min) {
//{min > 0} => {résultat = un entier supérieur ou égal à min
//                saisi par l'utilisateur}
```

NOTE : un message invitera l'utilisateur à recommencer tant qu'il n'a pas tapé un entier supérieur ou égal à min

- Créez une procédure `main` où vous devrez :
  - ✓ Déclarer une constante `MINJOUEURS` initialisée à 2
  - ✓ Déclarer des variables entières :
    - `nbJoueurs` pour le nombre de joueurs
    - `nbAllumettes` pour le nombre d'allumettes sur le plateau
  - ✓ Ajouter les instructions permettant d'**initialise**r le nombre de joueurs et le nombre initial d'allumettes sur le plateau (cf. Règles du jeu)

    NOTE : vous devrez impérativement utiliser la fonction `saisieEntPosMin`
- Compilez et testez plusieurs fois en entrant volontairement des valeurs invalides

2.3. Dans la classe `Nim` :

- Créez la fonction `prise` dont la spécification est la suivante :

```
private static int prise(int reste, int max) {
// { reste > 0, max > 0} => {résultat = entier > 0 et inférieur ou égal
//                           à la plus petite valeur parmi reste et max
//                           après saisie par l'utilisateur}
```

- Dans l'enveloppe de la classe `main` :
  - ✓ Déclarez une constante `MAXPRISE` initialisée à 3
  - ✓ Déclarez des variables entières :
    - `numJoueur` pour le numéro du joueur courant
    - `nbCoups` pour le nombre de fois où des allumettes ont été retirées du plateau jusqu'à la victoire d'un des joueurs
    - `reste` pour comptabiliser au fil du jeu le nombre d'allumettes restantes
  - ✓ Ajoutez les instructions d'initialisation des variables `numJoueur` et `nbCoups`
  - ✓ Ajoutez les instructions permettant de simuler le jeu, de façon à obtenir une trace d'exécution de ce type (*où les saisies de l'utilisateur sont en **vert***) :

```
*** Combien de joueurs ?
Entrez un entier supérieur ou égal à 2 : 2
 *** Combien d'allumettes ?
Entrez un entier supérieur ou égal à 6 : 10
*******************************
Nombre de joueurs : 2
Nombre d'allumettes : 10
*******************************
Joueur n°1, combien prenez-vous d'allumettes ? Entrez un entier compris entre 1 et 3 : 1
>>> Il reste 9 allumettes
Joueur n°2, combien prenez-vous d'allumettes ? Entrez un entier compris entre 1 et 3 : 2
>>> Il reste 7 allumettes
Joueur n°1, combien prenez-vous d'allumettes ? Entrez un entier compris entre 1 et 3 : 3
>>> Il reste 4 allumettes
Joueur n°2, combien prenez-vous d'allumettes ? Entrez un entier compris entre 1 et 3 : 1
>>> Il reste 3 allumettes
Joueur n°1, combien prenez-vous d'allumettes ? Entrez un entier compris entre 1 et 3 : 3
>>> Il reste 0 allumettes

Victoire du joueur n°1 après 5 coups !
```

3. Un peu de combinatoire…

Cette partie vous conduit à définir et utiliser des fonctions mathématiques utilisées en dénombrement et calcul probabiliste.

3.1. Créez une nouvelle classe `Calculs`

3.2. Dans la classe `Calculs` :

- Importez la bibliothèque `java.util.Scanner`
- Créez les fonctions dont les spécifications vous sont données ci-dessous :

```
private static int saisieEntPos() {
//{} => {résultat = un entier positif ou nul saisi par l'utilisateur}
```

```
private static int saisieEntPosInfEgal(int x) {
//{x > 0} => {résultat = entier compris entre 0 et x,
//            saisi par l'utilisateur}
```

```
private static int factorielle(int val) {
// {val >= 0} => {résultat = factorielle de n (n!) }
```

```
private static int combinaison(int n, int p) {
// {n >= 0, 0 <= p <= n}
// => {résultat = nombre de sous-ensembles de p éléments que
//     l'on peut obtenir à partir d'un ensemble de n éléments
//     formule : n!/(p! * (n-p)!)
```

```
private static int arrangement(int n, int p) {
// {n >= 0, 0 <= p <= n}
// => {résultat = nombre de n-uplets ordonnés de p éléments que
//     l'on peut obtenir à partir d'un ensemble de n éléments
//     formule : n!/(n-p!)}
```

- Créez une classe `main` (RAPPEL raccourci : *tapez le mot-clé* `main`*, puis appuyez sur la touche* ENTER)

- Dans l'enveloppe de la classe `main` :
  - ✓ Déclarez des variables entières n et p :
  - ✓ Ajoutez les instructions permettant d'initialiser ces variables par saisie (n doit être positif ou nul et p doit être inférieur ou égal à n)
  - ✓ Ajoutez les instructions permettant d'obtenir la trace d'exécution donnée ci-dessous (*où les saisies de l'utilisateur apparaissent en* ***vert***) :

```
Entrez un entier positif ou nul : 10
Entrez un entier inférieur ou égal à 10 : 3
-----------------
Factorielle de 10 = 3628800
-----------------
Nombre de sous-ensembles de 3 éléments parmi 10 éléments : 120
-----------------
Nombre de n-uplets ordonnés 3 éléments parmi 10 éléments : 720
```

NOTE : vous devez impérativement utiliser les fonctions de la classe `Calculs`

- Compilez et testez plusieurs fois en tentant d'initialiser volontairement les entiers n et p avec des valeurs invalides

3.3. Triangle de Pascal

6 PREMIÈRES LIGNES DU TRIANGLE DE PASCAL

	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5	1	5	10	10	5	1

*La **n**ième ligne du triangle de Pascal permet d'obtenir les coefficients de $\boldsymbol{a}$ et $\boldsymbol{b}$ dans le développement de $(\boldsymbol{a}+\boldsymbol{b})^{\boldsymbol{n}}$, n étant un entier positif ou nul.*

EXEMPLE : *La ligne **3** donne les coefficients de a et b dans le développement de $(a+b)^3$*

$$(a+b)^3 = \mathbf{1} \times a^3 + \mathbf{3} \times a^2b + \mathbf{3} \times ab^2 + \mathbf{1} \times b^3$$

Ces coefficients (dits coefficients binomiaux) s'obtiennent par calcul du nombre de sous-ensembles de $\boldsymbol{p}$ éléments ($0 \leq \boldsymbol{p} \leq \boldsymbol{n}$) que l'on peut obtenir à partir d'un ensemble de $\boldsymbol{n}$ éléments.

Ainsi, le coefficient binomial situé dans la colonne $\boldsymbol{j}$ de la ligne $\boldsymbol{i}$, est égal à : $\dfrac{\boldsymbol{i}!}{\boldsymbol{j}!(\boldsymbol{i}-\boldsymbol{j})!}$

- Dans la classe `Calculs` :
  - ✓ Créez la fonction suivante :

```
private static void trianglePascal(int n) {
// {n >= 0} => {les lignes 0 à n du triangle de Pascal ont été affichées
//              de façon à que sur chaque ligne, les coefficients binomiaux
//              soient séparés par un espace}
```

  - ✓ Ajoutez dans l'enveloppe de la classe `main`, les instructions affichant les lignes 0 à n du triangle de Pascal
  - ✓ Compilez et testez