Aucun document. Aucune machine. Les sept exercices sont indépendants.
Les morceaux de code Java devront être clairement présentés, indentés et commentés.
Les dix-neuf *méthodes* demandées sont des *méthodes statiques*, également appelées *fonctions*.

Exercice 1 On considère l'expression `z%3-z%5<7/z/2.0-z`.

1. Dessiner l'arbre syntaxique associé à cette expression.
2. Déterminer le type des expressions associées à chacun des nœuds de l'arbre si `z` est de type `short`.
3. Déterminer la valeur des expressions associées à chacun des nœuds de l'arbre si `z` vaut `-4`.
4. Calculer le nombre minimal de registres nécessaires à l'évaluation de cette expression.

Exercice 2 On considère la méthode `funk` suivante :

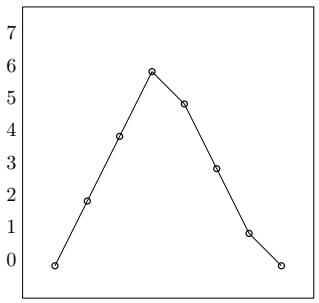
```
static String funk(String s, char c, int n){
    String res="";
    int comp=0;
    int i=0;
    while(comp<n && i<s.length()){
        if(Deug.charAt(s,i)!=c){
            res=res+Deug.charAt(s,i);
            comp++;
        }
        i++;
    }
    return res;
}
```

1. Préciser ce que renvoie `funk("malabar",'a',3)`.
2. Plus généralement, expliquer en une phrase ce que renvoie la méthode `funk(s,c,n)` avec `s`, `c` et `n` des arguments quelconques de types respectifs `String`, `char` et `int`.

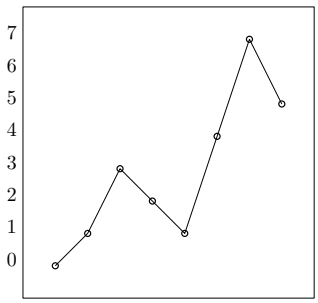
Exercice 3 Soient p un réel positif et r sa racine carrée. On a $r^2 = p$, d'où $r^2 + r = r + p$ puis $r = \dfrac{r+p}{r+1}$. Une approximation de r est alors obtenue par un calcul itératif : $r_{i+1} = \dfrac{r_i + p}{r_i + 1}$ avec r_0 que l'on choisit ici égal à $\dfrac{p}{2}$.

1. Écrire une méthode `termeN` qui prend en arguments un réel p et un entier n et qui renvoie le terme r_n.
2. Écrire une méthode `termeE` qui prend en arguments deux réels p et e et qui renvoie le terme r_i de plus petit indice vérifiant $-e < r_{i+1} - r_i < e$.

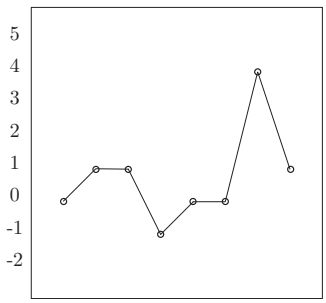
Exercice 4 On considère un tableau d'entiers relatifs et on le voit comme une courbe montante et descendante de la façon suivante. On part de l'altitude `0`, puis on lit le tableau de gauche à droite, et pour chaque entier `x`, s'il est positif on monte de `x`, et s'il est négatif on descend de `-x`.



(a) courbe | 2 | 2 | 2 | -1 | -2 | -2 | -1 |



(b) courbe | 1 | 2 | -1 | -1 | 3 | 3 | -2 |



(c) courbe | 1 | 0 | -2 | 1 | 0 | 4 | -3 |

1. Écrire une méthode `altitudeMin` qui renvoie l'altitude du point le plus bas d'une courbe donnée.
2. Pour une telle courbe, on appelle *pic* un point de la courbe qui est un maximum local strict, c'est-à-dire un point dont l'altitude est strictement plus grande que celle des deux points qui l'entourent. Écrire une méthode `nbPics` qui renvoie le nombre de pics d'une courbe donnée (les extrémités d'une courbe ne comptent jamais comme des pics). Par exemple, les courbes (a) et (c) ont un pic et la courbe (b) en a deux.

3. Pour un tableau d'entiers relatifs, on appelle *montée* une suite d'éléments consécutifs strictement positifs. Sur la courbe, cela correspond bien à une portion où on ne fait que monter. Écrire une méthode `monteeMax` qui renvoie la valeur de la plus grande montée d'une courbe donnée. Par exemple, la valeur de la montée maximale des courbes (a) et (b) est 6 et celle de la courbe (c) est 4.

Exercice 5 Le numéro d'immatriculation d'un véhicule est constitué de 7 caractères (il peut être représenté par un tableau de 7 caractères de type `char`) organisés en un premier bloc (deux lettres), un deuxième bloc (trois chiffres) et un troisième bloc (deux lettres). Le deuxième bloc évolue le plus vite, le premier bloc le moins vite :

de `AA-001-AA` à `AA-999-AA` (deuxième bloc) ;
puis de `AA-001-AB` à `AA-999-AZ` (troisième bloc, lettre de droite) ;
puis de `AA-001-BA` à `AA-999-ZZ` (troisième bloc, lettre de gauche) ;
puis de `AB-001-AA` à `AZ-999-ZZ` (premier bloc, lettre de droite) ;
puis de `BA-001-AA` à `ZZ-999-ZZ` (premier bloc, lettre de gauche).

Pour le deuxième bloc, les dix chiffres sont autorisés. Pour les premier et troisième blocs, les lettres `I`, `O`, `U` sont interdites (pour éviter qu'elles ne soient confondues avec les chiffres `1`, `0` et la lettre `V`).

Les cinq méthodes ont pour argument un tableau de 7 caractères représentant un numéro d'immatriculation.

1. Écrire une méthode `toString` qui renvoie la chaîne de caractères représentant le numéro d'immatriculation associé avec des `"-"` entre les blocs.
2. Écrire une méthode `incremBloc2` qui incrémente le deuxième bloc.
3. Écrire une méthode `incremBloc3` qui incrémente le troisième bloc.
4. Écrire une méthode `incremBloc1` qui incrémente le premier bloc en évitant la série `WW` (réservée à l'immatriculation temporaire des véhicules neufs).
5. En déduire une méthode `incrementation` qui l'incrémente.

Exercice 6 On représente une image en niveaux de gris par une matrice d'entiers courts (type `short`) et on définit le *voisinage* d'un point de la matrice par les huit possibles voisins autour du point plus le point lui-même.

1. Écrire une méthode `nbVoisins` qui prend en arguments quatre entiers `i,j,h,w` et renvoie le nombre de voisins du point de coordonnées `(i,j)` dans une image de hauteur `h` et de largeur `w`.
2. Écrire une méthode `coordVoisins` qui prend en arguments quatre entiers `i,j,h,w` et renvoie les coordonnées des voisins du point de coordonnées `(i,j)` dans une image de hauteur `h` et de largeur `w`.
3. Écrire une méthode `dilatation` qui prend en argument une image et renvoie l'image obtenue par *dilatation*, opération qui consiste à placer en chaque point le maximum des valeurs prises par son voisinage.

Exercice 7 On considère une élection avec `n` candidats numérotés de `1` jusqu'à `n` et `m` électeurs numérotés de `0` jusqu'à `m-1`. Chaque électeur doit voter en donnant un classement complet des candidats dans l'ordre de sa préférence. Par exemple, pour `4` candidats, un vote correct d'un électeur est la suite `3,1,2,4` (on dit que `3` est premier, `1` est deuxième, etc), alors que `1,2,2,4` ou `1,3,2` ne sont pas corrects. Un vote d'un électeur est un tableau de `n` entiers. Une élection est un tableau de `m` votes.

1. Écrire une méthode `estUnVoteCorrect` qui teste si un vote est correct.
2. Écrire une méthode `estUneElectionCorrecte` qui teste si tous les votes d'une élection sont corrects.

Étant donné un vote, le score d'un candidat est `n` s'il est premier, `n-1` s'il est deuxième, etc. Le score d'un candidat pour une élection est la somme de tous les scores de chaque vote de l'élection. Le candidat avec le plus grand score gagne l'élection. En cas d'égalité entre deux ou plusieurs candidats, on regarde celui arrivant en premier le plus souvent. S'il y a toujours égalité, l'élection n'a pas de vainqueur.

3. Écrire une méthode `score` qui renvoie le score d'un candidat pour une élection.
4. Écrire une méthode `scores` qui renvoie le score de *tous* les candidats pour une élection (il s'agira ici de créer un tableau et d'y mettre les scores en parcourant le tableau d'élection qu'une seule fois).
5. Écrire une méthode `vainqueur` qui renvoie le candidat vainqueur s'il existe.
6. Écrire une méthode `main` qui permette de tester les méthodes précédentes.