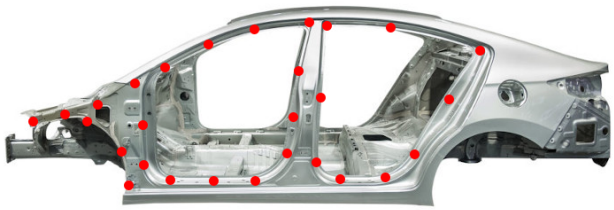
TP (4h)

Problème du voyageur de commerce

Compétences

- construire et utiliser une `ArrayList` (`add`, `remove`, `size`, `contains`, etc.).
- utiliser les méthodes de la classe `Collections` (`sort`, `min`, `max`).
- définir une classe contenant une liste d'objets.
- écrire des algorithmes spécifiques de gestion d'une liste d'objets.
- écrire quelques heuristiques simples sur le problème du voyageur de commerce.



La soudure par point représente une part très importante de l'assemblage d'une caisse automobile. Équipé d'une pince à souder, le robot doit parcourir une liste de points sur le châssis. L'ordre de parcours de ces points est primordial pour le temps de cycle. Si il est mal choisi, le robot perdra beaucoup de temps à se déplacer d'un point à un autre.

Ce problème est équivalent au voyageur de commerce, qui consiste à déterminer, étant donné une liste de villes et les distances entre toutes les paires de villes, le plus court circuit qui passe par chaque ville une et une seule fois.

Ce problème est plus compliqué qu'il n'y paraît. Pour un ensemble de n villes, il existe au total $n!$ chemins possibles. On ne connaît pas de méthode de résolution permettant d'obtenir des solutions exactes en un temps raisonnable pour de grandes instances (grand nombre de villes) du problème. Pour ces grandes instances, on devra donc souvent se contenter de solutions approchées, car on se retrouve face à une explosion combinatoire.

L'objectif de ce TP est d'apprendre l'utilisation de la classe `ArrayList` en Java en réalisant quelques méthodes heuristiques permettant de trouver rapidement un chemin pas trop long !

I Représentation d'un chemin

Pour simplifier, on traitera le problème du voyageur de commerce dans le plan euclidien. Le but est de définir un ordre de parcours de n points 2D en minimisant la somme des distances euclidiennes entre chaque points.

Pour gagner du temps, une classe `Point` similaire à celle que vous avez codée précédemment est fournie. On définit alors un chemin comme une liste de points. L'ordre des points dans la liste définit le sens du parcours.

1 *Créer une classe `Chemin` dans le package `chemin` à l'aide de la commande `File->New File->Java Class` puis définir ses attributs et les méthodes suivantes :*

   a. *un constructeur qui définit un chemin vide ;*
   b. *une méthode permettant d'ajouter un point à la fin d'un chemin ;*
   c. *une méthode qui retourne la longueur totale du chemin ;*
   d. *une méthode permettant de savoir le chemin contient un point donné ;*
   e. *une méthode qui mélange aléatoirement le chemin à l'aide de la commande `Collections.shuffle( ... );`*
   f. *une méthode `toString` avec tous les points du chemin.*

2 *Créer une classe `TestChemin` dans le package `chemin` à l'aide de la commande `File->New File->Java Main Class` puis construire et afficher un chemin comportant une dizaine de points que l'on prendra soit de mélanger.*

3 *Ajouter une méthode une méthode permettant de dessiner le chemin dans une fenêtre graphique 2D. Tester.*

II Méthode du balayage angulaire

Dans le cas de points 2D, une méthode heuristique simple consiste à parcourir les points en fonction de leur angle en coordonnées polaires. Autrement dit, la méthode consiste à :

- translater les points pour que le barycentre des points corresponde au centre du repère ;
- ranger les points par ordre croissant de leurs angles en coordonnées polaires (ρ, θ) ;
- translater les points pour les remettre à leurs places initiales.

4 *Ajouter à la classe `Chemin` les méthodes suivantes :*

   a. *une méthode qui retourne le barycentre des points du chemin ;*
   b. *une méthode permettant de translater les points du chemin selon les coordonnées d'un point donné ;*
   c. *une méthode permettant de ranger les points par ordre croissant de leurs angles en coordonnées polaires ;*
   d. *une méthode qui applique la méthode du balayage angulaire au chemin.*

5 *Tester avec les points précédents.*

6 *Tester avec les points du fichier `TestChassis`.*

III Méthode du plus proche voisin

Une autre méthode heuristique est de partir d'un point quelconque puis d'aller toujours au point le plus proche sans repasser par un point déjà parcouru. L'algorithme est résumé ci-dessous.

```
Q ← liste de points vide                                          pseudo-code
P ← point quelconque de la liste initiale H
retirer P de la liste H
ajouter P à la liste Q
tant que H n'est pas vide faire
    P ← le point le plus proche de P dans H
    retirer P de la liste H
    ajouter P à la liste Q
fintantque
C ← Q
```

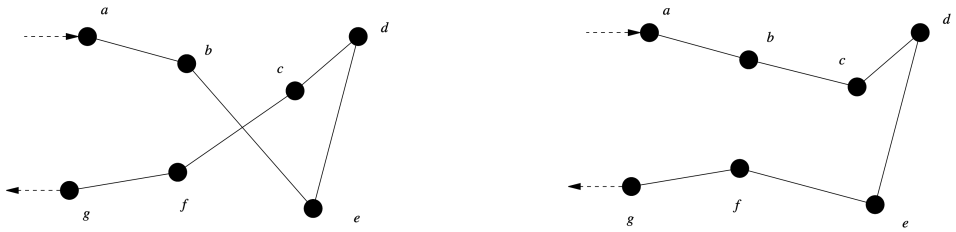
7 *Ajouter à la classe* `Chemin` *les méthodes suivantes :*

a. *une méthode qui retourne l'indice du point du chemin le plus proche d'un point donné ;*
b. *une méthode qui applique la méthode du plus proche voisin au chemin.*

8 *Tester.*

IV Méthode itérative 2-opt

Une heuristique classique, appelée 2-opt est une recherche locale qui consiste à partir d'une solution et à essayer de l'améliorer en échangeant itérativement les sommets de deux arêtes. Dans le cas du problème du voyageur de commerce géométrique, la permutation consiste généralement à remplacer les arêtes par leurs diagonales comme illustré ci-dessous.




```
amélioration ← vrai                                               pseudo-code
tant que amélioration = vrai faire
    amélioration := faux
    pour tout point Pi de H faire
        pour tout point Pj de H, avec j différent de i-1, de i et de i+1 faire
            si distance(Pi, Pi+1) + distance(Pj, Pj+1) > distance(Pi, Pj) +
                distance(Pi+1, Pj+1) alors
                Remplacer les arêtes (xi, xi+1) et (xj, xj+1) par (xi, xj) et (xi
                    +1, xj+1) dans H
                amélioration ← vrai
            finsi
        finpour
    finpour
fintantque
```

9 *Ajouter à la classe* `Chemin` *une méthode qui applique la recherche locale 2-opt au chemin.*

10 *Tester.*