# Fiche d'auto-évaluation du projet
# Problème du sac à dos

## Note préliminaire

- Cette fiche est a remplir par le binôme ayant participé au projet.
- Le code source sera vérifié méticuleusement par l'enseignant pour s'assurer de la cohérence de vos réponses d'auto-évaluation vis-à-vis du code développé. En cas d'incohérence (exemple : 1 algorithme marqué comme « fonctionnel » dans cette fiche, mais non opérationnel dans le code rendu), le binôme sera pénalisé par des points négatifs.
- L'ensemble des codes sources de la promotion seront passés dans un détecteur d'anti-plagiat pour rechercher des projets trop similaires ou des fraudes éventuelles.

## Informations générales

| | **Nom** | **Prénom** | **Groupe** |
|---|---|---|---|
| **Étudiant 1** | YE | Fangyuan Lisa | 204 |
| **Étudiant 2** | MILLO | Chelsey | 207 |

| **(cocher la case X)** | **CK** | **JAB** | **JFB** |
|---|---|---|---|
| **Enseignant TD/TP** | X | | |

# Grille d'auto-évaluation

| (cocher la case X) | Non fonctionnel | Partiellement fonctionnel | Fonctionnel |
|---|---|---|---|
| **1- Algorithme Glouton** | | | X |

**Commenter vos réponses par quelques lignes**

| | |
|---|---|
| **Décrire en quelques lignes l'algorithme mis en œuvre** | **L'algorithme Glouton** que nous avons écrit consiste à trier (par ordre décroissant) la liste d'objets en fonction du rapport entre la valeur et le poids de l'objet. Par la suite, nous sélectionnons les objets de la liste un par un pour ensuite les déposer dans une autre liste (dans le sac) tout en vérifiant que le poids du sac, comportant les objets mit, ne dépasse pas le poids maximal tapé par l'utilisateur. |
| **Fonction de tri utilisée (nom de l'algorithme)** | Nous avons utilisé **le tri rapide** pour sa très bonne complexité en moyenne et essentiellement pour trier notre liste d'objets* (présent dans le fichier.txt) dans l'ordre décroissant<br><br>* La liste d'objets est obtenue à partir du chemin que l'utilisateur à communiquer. |
| **Copier / coller la trace d'exécution de l'algorithme (terminal) obtenue avec « ItemEval.txt »** | Vous allez commencer à resoudre le problème du sac à dos<br>Veuillez entrer le chemin(en type .txt), le poid maximal de sac et le methode choisir.<br>(Pour les méthodes vous avez le choix entre : Glutonne(1), Dynamique(2) et PSE(3))<br><br>D:\iut paris decartes\2020-2021\AAV\Projet\itemsEval.txt<br>20<br>1<br>Objets choisi :<br>Lingot d'or a pour poids 10.0kg possedant une valeur égale à 1000.0<br>iPhone a pour poids 2.0kg possedant une valeur égale à 200.0<br>Couteau suisse a pour poids 1.0kg possedant une valeur égale à 50.0<br>Camping gaz a pour poids 1.0kg possedant une valeur égale à 40.0<br>Chaussures a pour poids 2.0kg possedant une valeur égale à 80.0<br>Shorts a pour poids 1.0kg possedant une valeur égale à 24.0<br><br>La valeur totale du sac est égale à 1394.0 et le poid final est égale à 17.0 kg<br>Le temps utilise pour resoudre le probleme de sac = 4ms |

| (cocher la case X) | Non fonctionnel | Partiellement fonctionnel | Fonctionnel |
|---|---|---|---|
| **2 - Programmation Dynamique** | | | X<br>( ce code est fonctionnel si les valeurs ne dispose un chiffre après la virgule ) |

**Commenter vos réponses par quelques lignes**

| | |
|---|---|
| **Décrire en quelques lignes l'algorithme mis en œuvre** | L'algorithme que nous avons écrit a pour but de trouver la solution la plus optimale au problème du sac à dos.<br><br>Tout d'abord, nous commençons par créer un tableau de taille nb objet * poids_max. Afin d'éviter tout erreur liée aux chiffres à virgule, nous multiplions poids_max et les poids_obj par 10 pour obtenir un entier.<br>Nous avons ensuite rempli le tableau où « chaque case représente le bénéfice maximum possible pour le i premiers objets avec un poids. »<br>Ensuite, commence par la dernière ligne du tableau, nous récupérons un par un les objets de combinaisons optimales |
| **Une fois la matrice calculée, est-ce que votre code permet de lister l'ensemble des items conduisant à la solution optimale ? (OUI / NON, si oui décrire comment)** | A la fin du programme, le code nous renvoie la meilleure solution de liste d'objets dans le sac respectant le poids_max.<br><br>On commence par la dernière case de la table, à chaque ligne, on récupère le poids minimal nécessaire pour faire le bénéfice optimal et le comparer à l'objet précédant.<br>Dans le cas où le bénéfice est identique on abandonne cet objet pour ensuite se rediriger vers l'objet avant.<br>Dans le cas contraire, on récupère l'objet pour le mettre dans la liste des objets à mettre dans le sac. Par la suite, avec un calcul simple on chercher le poids restant du sac.<br>On recommence ensuite le même processus jusqu'à ce que tout le tableau ait été traité.<br><br>Nous obtiendrons à la fin une liste d'objets à mettre dans le sac |

| | |
|---|---|
| **Copier / coller la trace d'exécution du terminal obtenue avec « ItemEval.txt »** | Vous allez commencer à resoudre le problème du sac à dos<br>Veuillez entrer le chemin(en type .txt), le poid maximal de sac et le methode choisir.<br>(Pour les méthodes vous avez le choix entre : Glutonne(1), Dynamique(2) et PSE(3))<br><br>D:\iut paris decartes\2020-2021\AAV\Projet\itemsEval.txt<br>20<br>2<br>Objets choisi :<br>Lingot d'or a pour poids 10.0kg possedant une valeur égale à 1000.0<br>iPhone a pour poids 2.0kg possedant une valeur égale à 200.0<br>Couteau suisse a pour poids 1.0kg possedant une valeur égale à 50.0<br>Camping gaz a pour poids 1.0kg possedant une valeur égale à 40.0<br>Sac de couchage a pour poids 4.0kg possedant une valeur égale à 60.0<br>Chaussures a pour poids 2.0kg possedant une valeur égale à 80.0<br><br>La valeur totale du sac est égale à 1430.0 et le poid final est égale à 20.0 kg<br>Le temps utilise pour resoudre le probleme de sac = 5ms |

| (cocher la case X) | Non fonctionnel | Partiellement fonctionnel | Fonctionnel |
|---|---|---|---|
| **3 - Algorithme PSE** | | | X |

**Commenter vos réponses par quelques lignes**

| | |
|---|---|
| **Décrire en quelques lignes l'algorithme mis en œuvre** | On a utilisé un algorithme de procédures par séparation et évaluation (PSE) et pour l'optimiser nous avons élaguer cet arbre en utilisant des bornes inférieures et supérieures de la fonction objective.<br>Tout d'abord, on a créé une class Nœud et ensuite créé un nœud racine qui fait appelle à la fonction qui créé ensuite deux nœuds fils (un avec « ajouter d'objet » et un autre sans « ajouter d'objet »). Pour déterminer la propriété de nœud, selon leurs bornes supérieures, on créé deux nœuds fils pour le nœud dont la borne supérieure est plus grande. On répètera cet algorithme récursif jusqu'à atteindre la feuille de l'arbre. On compare la borne supérieure à borne inferieure, si elle est plus petite, on va développer les autres nœuds, sinon, on prend ce nœud et mettre les objets de ce nœud dans la liste (de sac). |
| **Décrire la structure de données utilisée pour modéliser l'arbre des solutions** | Nous avons créé une classe Nœud qui est composée par le poidsActuelle, le borneInf, le borneSup, ListeObjetsParNoeud et intégrer le listeObjetPresent, et le poids_maximal. Nous avons utilisé une structure qui ressemble à celle de l'arbre binaire par chaînage. Mais au lieu de mettre deux nœuds fils, on met un nœud père pour récupère les données du nœud parent.<br>Cette création d'arbre dépend des bornes supérieures de chaque nœud (la borne supérieure possédant la plus grande valeur sera la première à être développé).<br>On a aussi utilisé l'ArrayListe pour récupérer les objets présents et pour récupérer les objets de chaque nœud. Nous avons aussi utilisé les LinkedListe pour récupérer les nœuds qui n'ont pas encore développé. |
| **Avez-vous pu réduire la taille de l'arbre via la stratégie d' élagage ? (OUI / NON, si oui décrire comment)** | Oui<br>Nous avons pu réduire la taille de l'arbre en utilisant la contrainte de borne supérieure et borne inferieure.<br>En effet, avant de créer un nœud qui va ajouter l'objet, on vérifie d'abord si les ajouts des poids de ce nouvel objet vont dépasser la capacité du sac. Après la création de deux fils, on trie par ordre décroissant les nœuds créés selon leurs bornes supérieures. On crée ensuite les fils pour ce nœud, possédant une plus grande borne supérieure et on supprimer ce nœud dans la liste de nœuds qui n'ont pas encore été traiter.<br>Lorsque on atteint la feuille de l'arbre du nœud, on compare la borne inférieure de cette feuille aux bornes supérieurs des autres nœuds qui sont présentes dans la liste de nœuds qui n'ont pas encore été traités, pour vérifier s'il est possible d'avoir une meilleure solution que cette feuille.<br>Ainsi, on trouve la meilleure solution que l'on mettra dans la liste (de |

| | |
|---|---|
| | sac). |
| **Copier / coller la trace d'exécution du terminal obtenue avec « ItemEval.txt »** | Vous allez commencer à resoudre le problème du sac à dos<br>Veuillez entrer le chemin(en type .txt), le poid maximal de sac et le methode choisir.<br>(Pour les méthodes vous avez le choix entre : Glutonne(1), Dynamique(2) et PSE(3))<br><br>D:\iut paris decartes\2020-2021\AAV\Projet\itemsEval.txt<br>20<br>3<br>Objets choisi :<br>Lingot d'or a pour poids 10.0kg possedant une valeur égale à 1000.0<br>iPhone a pour poids 2.0kg possedant une valeur égale à 200.0<br>Couteau suisse a pour poids 1.0kg possedant une valeur égale à 50.0<br>Camping gaz a pour poids 1.0kg possedant une valeur égale à 40.0<br>Chaussures a pour poids 2.0kg possedant une valeur égale à 80.0<br>Sac de couchage a pour poids 4.0kg possedant une valeur égale à 60.0<br><br>La valeur totale du sac est égale à 1430.0 et le poid final est égale à 20.0 kg<br>Le temps utilise pour resoudre le probleme de sac = 4ms |

| **4 - ANNEXE** | |
|---|---|
| **Indiquer à droite si vous avez implémenté des éléments non demandés dans le sujet (interface graphique, menu, algorithmes supplémentaires, etc.)** | |

**COPIER / COLLER L'INTÉGRALITÉ DES CODES SOURCES DE VOTRE PROJET**

```
package sacADos;


import java.io.IOException;
import java.util.Scanner;

public class Main {

	public static void resoudre_sac_a_dos() throws IOException {
	String chemin = null;
	int poids_maximal = 0;
	int méthode = 0;

	@SuppressWarnings("resource")
	Scanner scan = new Scanner(System.in);
	if(scan.hasNextLine())
		chemin = scan.nextLine(); // Enregistrer le chemin
	if(scan.hasNextInt())
		poids_maximal = scan.nextInt(); // Enregistrer le poid maximum de sac
	if(scan.hasNextInt())
		méthode = scan.nextInt();

	SacADos sac = new SacADos(chemin, poids_maximal);
	sac.resoudre(méthode);
	}

	public static void main(String[] args) throws IOException {

		System.out.println("Vous allez commencer à resoudre le problème du sac à dos");
		System.out.println("Veuillez entrer le chemin(en type .txt), le poid maximal de sac et le methode choisir.");
		System.out.println("(Pour les méthodes vous avez le choix entre : Glutonne(1), Dynamique(2) et PSE(3))\n");

		resoudre_sac_a_dos();
	}



}
```

---

```
package sacADos;

import java.io.BufferedReader;
import java.io.FileReader;
```

```
import java.io.IOException;
import java.util.ArrayList;

import méthodes.Exacte_Dynamique;
import méthodes.Exacte_PSE;
import méthodes.Gloutonne;

public class SacADos {

	@SuppressWarnings("unused")
	private String chemin;
	private float poids_maximal;
	private float valeurs_total;
	private float poids_total;
	ArrayList<Objets> objetsSelectionnée = new ArrayList<Objets>(); // Objet metter dans le
sac
	ArrayList<Objets> objetsPresents;// Objets presents
	long upTime; // Le temps utilise en ms
	long startTime, endTime; // Le temps réelle debut et fin d'un methode

	//Sac à dos
	public SacADos(String chemin, float poids_maximal) throws IOException {
		this.chemin = chemin;
		this.poids_maximal = poids_maximal;
		this.valeurs_total = 0;
		this.poids_total = 0;

		//Lire les fichier et mettre les objets dans le Liste de objets Presente
		@SuppressWarnings("resource")
		BufferedReader text = new BufferedReader(new FileReader(chemin));
		objetsPresents = new ArrayList<Objets>();
		String line;
	   while((line = text.readLine()) != null) {
		String[] strArray = line.split(";"); //Les elements de la ligne est separer par symbole
';'
		Objets obj = new
Objets(strArray[0],Float.parseFloat(strArray[1]),Float.parseFloat(strArray[2]));
		objetsPresents.add(obj); // mettre les elements dans la liste de l'objet presente
	   }
	}

	//Mettre les objet chosi par les méthodes dans le liste ObjetSelectionne
	public void getElementOfList() {
		for (Objets objs : objetsSelectionnée) {
			poids_total += objs.getPoids();
			valeurs_total += objs.getValeur();
			}
	}
```

```
	//Affichier les Liste des objets mettre dans le sac
	public String toString() {
		StringBuilder ObjetInsere = new StringBuilder();
		if(this.objetsSelectionnée.size() == 0) {
			ObjetInsere.append("Aucun element present");
			return ObjetInsere.toString();
		}
		else {
			ObjetInsere.append("Objets choisi : \n");
			for(Objets objs : objetsSelectionnée) {
				ObjetInsere.append("\t" + objs.toString()+ "\n");
			}
		getElementOfList();
		ObjetInsere.append( "\nLa valeur totale du sac est égale à " + valeurs_total + " et le poid final est égale à " + poids_total + " kg");
		upTime = endTime - startTime;
		ObjetInsere.append("\nLe temps utilise pour resoudre le probleme de sac = " + upTime +"ms");
		return ObjetInsere.toString();
		}
	}


	@SuppressWarnings("unused")
	public void resoudre(int a) {
		switch(a){
		case 1:
			//methode Glutonne
			startTime = System.currentTimeMillis();
			Gloutonne méthoGlutonne = new Gloutonne(objetsPresents,objetsSelectionnée,poids_maximal);
			méthoGlutonne.methodeGlutonne();
			endTime = System.currentTimeMillis();
			System.out.println(toString());
			break;
		case 2:
			//methode Dynamique
			startTime = System.currentTimeMillis();
			Exacte_Dynamique méthoDynamique = new Exacte_Dynamique(objetsPresents,objetsSelectionnée,poids_maximal);
			méthoDynamique.methodeDynamique();
			endTime = System.currentTimeMillis();
			System.out.println(toString());
			break;
		case 3:
			//methode PSE
			startTime = System.currentTimeMillis();
			Exacte_PSE méthodePSE = new Exacte_PSE(objetsPresents,objetsSelectionnée,poids_maximal);
```

```
                        méthodePSE.methodePSE();
                        endTime = System.currentTimeMillis();
                        System.out.println(toString());
                        break;
                default :
                        System.out.println("valeur inconnue");
                }
        }


        public ArrayList<Objets> getObjetsPresents(){
                return objetsPresents;
        }

        public ArrayList<Objets> getObjetsSelectionnée(){
                return objetsSelectionnée;
        }


}
```

---

```
package sacADos;

public class Objets {

        private String nom;
        private float poids;
        private float valeur;
        private float rapport;
        private int poidsFause;

        public Objets(String nom, float poids, float valeur) {
                this.nom = nom;
                this.poids = poids;
                this.valeur = valeur;
        }

        public String getNom() {
                return this.nom;
        }

        public float getPoids() {
                return this.poids;
        }

        public int getPoidsFause() {
                poidsFause = (int)(poids*10);
                return poidsFause;
        }

        public float getValeur() {
                return this.valeur;
        }
```

```
	public float getRapport() {
		rapport = valeur/poids;
		return rapport;
	}

	public String toString() {
		String line;
		line = nom + "a pour poids " + poids + "kg possedant une valeur égale à "
+ valeur;
		return line;
	}




}
```

---

```
package méthodes;

import java.util.ArrayList;

import sacADos.Objets;

public class Gloutonne {
	float poids_maximal;
	ArrayList<Objets> ListObjet;
	ArrayList<Objets> ListObjetChoisi;
	TriRapide tri;

	public Gloutonne(ArrayList<Objets> ListObjet, ArrayList<Objets> ListObjetChoisi,float
poids_maximal) {
		this.ListObjet = ListObjet;
		this.ListObjetChoisi = ListObjetChoisi;
		this.poids_maximal = poids_maximal;
		tri = new TriRapide(ListObjet);
	}

	//Mehode glutone
	public void methodeGlutonne() {
		tri.triRapide();
		ajouterObjetsDansSac();
	}


	public void ajouterObjetsDansSac() {
		float calcul = 0;
		float a;
		for (int i = 0; i < ListObjet.size(); i++) {
```

```
                    a = ListObjet.get(i).getPoids();
                    if ((calcul += a) <= poids_maximal)
                    ListObjetChoisi.add(ListObjet.get(i));
                }
        }
}
```

---

```
package méthodes;

import java.util.ArrayList;
import sacADos.Objets;

//Tri rapide
public class TriRapide {
        ArrayList<Objets> ListObjet;
        public TriRapide(ArrayList<Objets> ListObjet) {
                this.ListObjet = ListObjet;
        }

        public void echanger(Objets x, Objets y) {
                int x1 = ListObjet.indexOf(x);
                int y1 = ListObjet.indexOf(y);
                ListObjet.set(x1, y);
                ListObjet.set(y1, x);
        }


        public int repartition(int premierObj, int dernierObj, int positionPivot) {
                echanger(ListObjet.get(positionPivot), ListObjet.get(dernierObj));
                int compt = premierObj;
            for(int i = premierObj; i <= dernierObj - 1; i++){
                if (ListObjet.get(i).getRapport() > ListObjet.get(dernierObj).getRapport()){
                    echanger(ListObjet.get(i), ListObjet.get(compt));
                    compt++;
                }
            }
            echanger(ListObjet.get(dernierObj), ListObjet.get(compt));
            return compt;
      }

        public void triRapide(int premierObj, int dernierObj){
            if(premierObj < dernierObj){
                int positionPivot = choixPivot(premierObj, dernierObj);
                positionPivot = repartition(premierObj, dernierObj, positionPivot);
                triRapide(premierObj, positionPivot-1);
                triRapide(positionPivot+1, dernierObj);
            }
```

```
    }

        public int choixPivot(int premierObj, int dernierObj) {
                if((dernierObj - premierObj) % 2 == 0) {
                        int p1 = (dernierObj - premierObj)/2;
                        return p1+premierObj;
                }
                else {
                        int p2 = (dernierObj - 1 - premierObj)/2;
                        return p2+premierObj;
                        }
        }

        public void triRapide() {
                int longueur = ListObjet.size();
                triRapide(0,longueur-1);
        }


}
```

---

```
package méthodes;

import java.util.ArrayList;

import sacADos.Objets;

public class Exacte_Dynamique {

        float poids_maximal;
        ArrayList<Objets> ListObjet;
        ArrayList<Objets> ListObjetChoisi;

        public Exacte_Dynamique(ArrayList<Objets> objetsPresents, ArrayList<Objets>
objetsSelectionnée, float poids_maximal) {
                this.ListObjet = objetsPresents;
                this.ListObjetChoisi = objetsSelectionnée;
                this.poids_maximal = poids_maximal;
        }

        //méthode dynamique
        public void methodeDynamique() {
                int a = (int)(poids_maximal*10);//mettre les poids_maximal en int (on le multiplie
par 10 pour eviter d'avoir le chiffre apres virgule )
                int b = ListObjet.size();
                float[][] tableDynamique = new float[b][a+1];// créer un table de dimension nb
objet*(poids_maximal*10)
```

```
		//on remplir la premier ligne de table
		for(int j = 0; j < tableDynamique[0].length; j++) {
			if(ListObjet.get(0).getPoidsFause() > j)

				tableDynamique[0][j] = 0;
			else
				tableDynamique[0][j] = ListObjet.get(0).getValeur();

			}

		//remplir les cases restes
		int i= 1, j = 0;
		for(; i < tableDynamique.length; i++) {
			for(j = 0; j < tableDynamique[0].length; j++) {
				if (ListObjet.get(i).getPoidsFause() > j)
					tableDynamique[i][j] = tableDynamique[i - 1][j];
				else
					tableDynamique[i][j] = max(tableDynamique[i - 1][j],tableDynamique[i - 1][j - ListObjet.get(i).getPoidsFause()] + ListObjet.get(i).getValeur());
			}
		}

		i--;
		j--;

		// Trouver le meilleur benefice
		while(tableDynamique[i][j] == tableDynamique[i][j-1]) {
			j--;
		}

		//Mettre les objets dans les sac selon leur benefices
		while(j > 0) {
			while(i > 0 && tableDynamique[i][j] == tableDynamique[i - 1][j])
				i--;
			j = j - ListObjet.get(i).getPoidsFause();
			if (j >= 0) {
				ListObjetChoisi.add(ListObjet.get(i));
				i--;
			}
		}
	}

	//Trouver le max
	public float max(float a, float b) {
		if(a >= b)
			return a;
		else
			return b;
	}
```

```
}
```

---

```
package méthodes;

import java.util.ArrayList;
import java.util.LinkedList;

import sacADos.Objets;

public class Exacte_PSE {
	float poids_maximal;
	ArrayList<Objets> ListObjet;
	ArrayList<Objets> ListObjetChoisi;
	TriRapide tri;
	LinkedList<Noeud> ListNoeud; //Liste de Noeud traite

	public Exacte_PSE(ArrayList<Objets> objetsPresents, ArrayList<Objets> objetsSelectionnée, float poids_maximal) {
		this.ListObjet = objetsPresents;
		this.ListObjetChoisi = objetsSelectionnée;
		this.poids_maximal = poids_maximal;
		tri = new TriRapide(ListObjet);
	}

	//methode PSE
	public void methodePSE() {
		tri.triRapide();
		Noeud racine = new Noeud(ListObjet, poids_maximal); // créer le le racine de l'arbre
		racine.setBorneSupRacine(); //mis à jour son borne superieur
		ListNoeud = new LinkedList<Noeud>(); // creer le liste de noeud pour traite son prioritaire
		ListNoeud.add(racine); // ajouter le Racine dans le liste
		NoeudPrioritaire(); //determiner le prioritaire de noeud selon leurs borne superieur
	}

	//Tri à bulle le Liste de noeud dans l'ordre decroissant selon leur borne superieur
	public void NoeudPrioritaire() {
		for (int i = 0; i < ListNoeud.size() - 1; i++) {
			for(int j = i + 1; j < ListNoeud.size(); j++) {
				if(ListNoeud.get(j).getBorneSup() > ListNoeud.get(i).getBorneSup())
					echanger(ListNoeud.get(j), ListNoeud.get(i));
			}
		}
		NouveauNoeud(); // Créer des nouveaux noeuds pour celle qui a un plus grand borne sup.
```

```
	}

	//
	public void NouveauNoeud() {
		//verifier si la profondeur de noeud a depassée le nombre d'objet presente
		if(ListNoeud.getFirst().getProfondeur() < ListObjet.size()) {
			if(ListNoeud.getFirst().verifierPoids()) { // Verifier s'il a depasse le poids
				Noeud noeudAjouter = new Noeud(ListNoeud.getFirst()); // Créer un nouveau noeud qui va ajouter l'objet
				noeudAjouter.noeudAjouterObjet(ListNoeud); // Mis à jour infos du noeud
			}

			Noeud noeudSansAjouter = new Noeud(ListNoeud.getFirst()); // Créer un nouveau noeud qui n'ajoute pas l'objet
			noeudSansAjouter.noeudSansAjouterObjet(ListNoeud); // Mis à jour infos du noeud
			ListNoeud.removeFirst(); // Supprimer la noeud qui ont deja créer les fils
			NoeudPrioritaire(); // Recommencer l'etape
		}else {
			for(Objets o : ListNoeud.getFirst().getObjetParNoeud()) // Obtient les objets de noeud qui a un meilleur borne sup.
			ListObjetChoisi.add(o);
		}
	}



	public void echanger(Noeud x, Noeud y) {
		int x1 = ListNoeud.indexOf(x);
		int y1 = ListNoeud.indexOf(y);
		ListNoeud.set(x1, y);
		ListNoeud.set(y1, x);
	}


}
```

---

```
package méthodes;

import java.util.ArrayList;
import java.util.LinkedList;
import sacADos.Objets;

public class Noeud {

	private float poidsActuelle;
	private int profondeur;
```

```
	private float borneSup;
	private float borneInf;
	float poids_maximal;
	private ArrayList<Objets> ListObjet;
	private ArrayList<Objets> ListObjetParNoeud; //Liste d'objet qui presente dans chaque noeud

	// Noeud pour racine
	public Noeud(ArrayList<Objets> ListObjet, float poids_maximal) {
		this.ListObjet = ListObjet;
		this.poidsActuelle = 0;
		this.profondeur = 0;
		this.borneInf = 0;
		this.borneSup = 0;
		this.poids_maximal = poids_maximal;
		this.ListObjetParNoeud = new ArrayList<Objets>();
	}

	//tous les autres noeuds
	public Noeud(Noeud pere) {
		this.poidsActuelle = pere.poidsActuelle;
		this.profondeur = pere.profondeur + 1;
		this.borneInf = pere.borneInf;
		this.borneSup = pere.borneSup;
		this.ListObjet = pere.ListObjet;
		this.poids_maximal = pere.poids_maximal;
		nouvelleListeObjetParNoeud(pere.ListObjetParNoeud);
	}

	//Créer un nouvelle Liste d'objet pour chaque noeud
	private void nouvelleListeObjetParNoeud(ArrayList<Objets> list){
		this.ListObjetParNoeud = new ArrayList<Objets>();
		for(Objets o : list)this.ListObjetParNoeud.add(o);
	}

	//Mis à jour les infos pour le noeud qui va ajouter l'objet et le mettre dans le Liste de
	//noeud presente actuelle(non traite)
	public void noeudAjouterObjet(LinkedList<Noeud> ListNoeud) {
		borneInf += ListObjet.get(profondeur-1).getValeur();
		poidsActuelle += ListObjet.get(profondeur-1).getPoids();
		setBorneSup();
		ListObjetParNoeud.add(ListObjet.get(profondeur-1));
		ListNoeud.add(this);
	}

	//Mis à jour les infos pour le noeud qui n'ajoute pas l'objet et le mettre dans le Liste de
	//noeud presente actuelle(non traite)
	public void noeudSansAjouterObjet(LinkedList<Noeud> ListNoeud) {
		setBorneSup();
```

```
                ListNoeud.add(this);
        }


        //Verifier si le poid de nouvelle noeud est depassé va depasse le poid maximal (poids de
sac)
        public boolean verifierPoids() {
                return this.getPoidsActuelle() + ListObjet.get(profondeur).getPoids() <=
poids_maximal;
        }

        //Mis à jour le borne superieur du Racine
        public void setBorneSupRacine() { //mis à jour le borne sup de racine
                borneSup = poids_maximal*ListObjet.get(profondeur).getRapport();
        }

        //Mis a jour le borne superieur du noeud
        public void setBorneSup() {
                if(profondeur < ListObjet.size())borneSup = borneInf + ((poids_maximal -
poidsActuelle)*ListObjet.get(profondeur).getRapport());
                else borneSup = borneInf + ((poids_maximal - poidsActuelle)*0);
        }

        public float getPoidsActuelle() {
                return this.poidsActuelle;
        }

        public float getBorneSup() {
                return this.borneSup;
        }

        public int getProfondeur() {
                return this.profondeur;
        }

        //Obtien la liste d'objet par noeud
        public ArrayList<Objets> getObjetParNoeud() {
                return ListObjetParNoeud;
        }

}
```