Partie 1 / Question 1 : analyse

- Lire en début les deux tableaux COM (contenant les commandes) et DIST (contenant les distances entre les cinémas et les trois dépôts)
- Prendre une variable qui va contenir la distance parcourue et l'initialiser à zéro (Distance =0)
- parcourir le tableau (COM) des commandes, colonne par colonne, (i.e cinéma par cinéma) et pour chaque colonne
    - si au moins une commande est passée, (il suffit d'ignorer les colonnes dont TOUS les éléments sont nuls, grâce au module ColVal), alors :
        - on rajoute dans Distance , la distance entre le cinéma concerné et le dépôt le plus proche (utilisation du module PetitCol)
- on écrit Distance

Partie 1 / Question 2 : algorithme

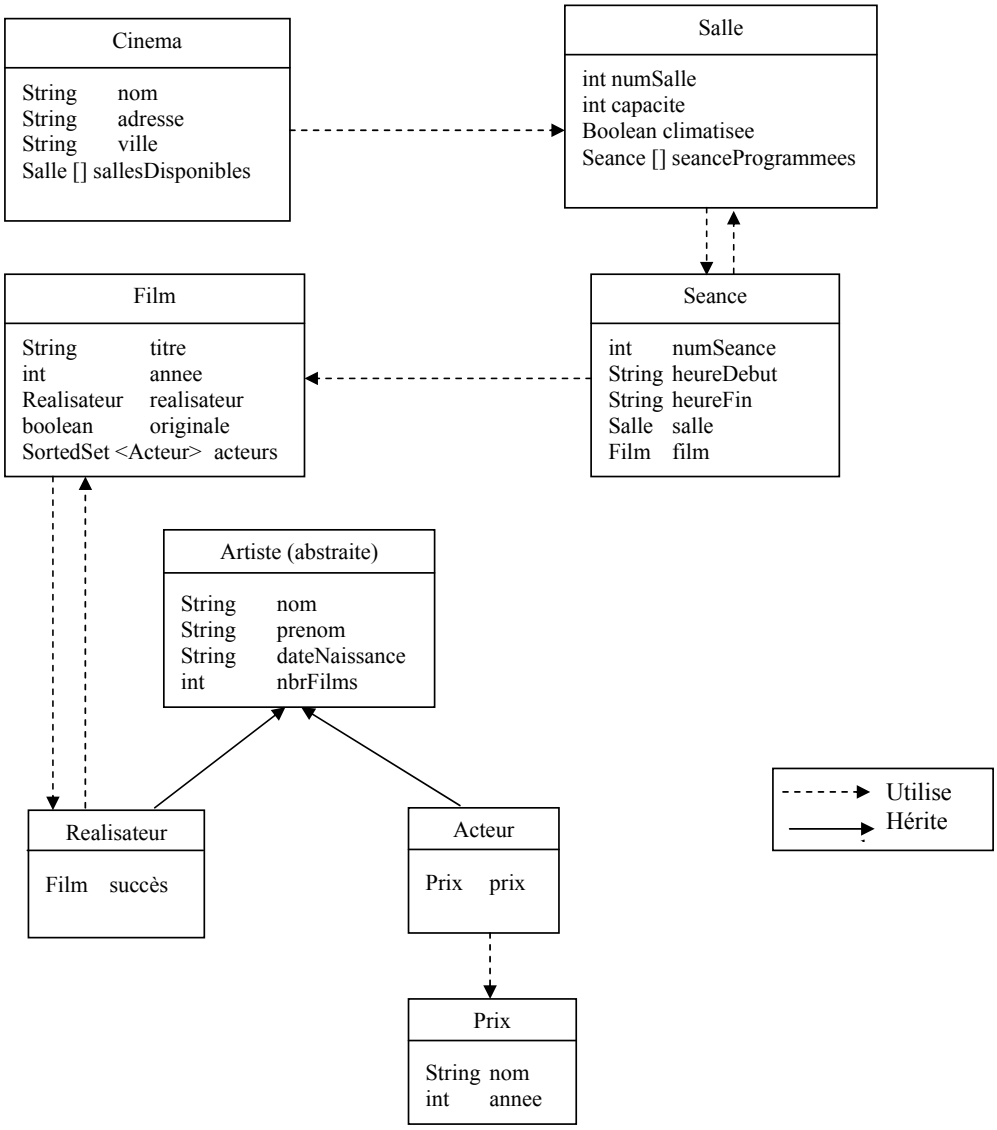
```
ALGORITHME   cp1_2012
Type    tab2 = Tableau [1..100,1..100] d'entiers
variables       Com,dist : tab2
                m,n,k,l,distance,i: Entier
procédure lect2d,
fonctions ColVal , PetitCol

DEBUT
lect2d (com, m, n)
lect2d (dist, k, l)
distance ← 0
Pour  I allant de 1 à  n Faire
      si  colVal(com, m ,i,0) = Faux  Alors
                distance ← distance + Petitcol(dist,k,i)
Ecrire  ( 'Distance := ', distance)
FIN
```

Partie 1 / Question 3 : programme

```
program cp1_2012;
uses crt;
Type    tab2=array[1..100,1..100] of integer;
        tab1= array[1..100] of integer;
var     Com,dist :tab2;
        m,n,k,l,distance,i:integer;
{$i E:\algo\modules\lect2d.pro}
{$i E:\algo\modules\ColVal.fon}
{$i E:\algo\modules\PetitCol.Fon}
BEGIN
clrscr;
lect2d(com,m,n);
lect2d(dist,k,l);
distance:=0;
for i:=1 to n do
        if colVal(com,m,i,0) = false then
                        distance := distance + petitcol(dist,k,i);
writeln ( 'Distance := ', distance);
readln;
END.
```

Partie 1 / Question 4



Partie 1 / Question 5 : La structure de données adéquate est un ensemble (Set) donc HashSet ou TreeSet
HashSet <Acteur> acteurs = new HashSet<Acteur>();

Partie 1 / Question 6 : Pour gérer cette erreur, nous pouvons utiliser le mécanisme d'exceptions offert par Java. Pour cela, nous devons créer une classe AjoutActeurErreur héritant de la classe Exception. Ainsi, la méthode ajouterActeur lancera une exception du type AjoutActeurErreur dans le cas où la date de naissance de l'acteur est supérieure à la date de réalisation du film.

```
public void ajouterActeur (Acteur a) throws AjoutActeurErreur {
     if (a.anneeNaissance > this.anneeRealisation) throw new
     AjoutActeurErreur();
     else acteurs.add(a);
   }
}
class AjoutActeurErreur extends Exception {       }
```

Commentaire [N1] : il peuvent aussi comparer toute la date de daissance s'il ont choisi un attribut de type date

Partie 1 / Question 7 : Ci-dessous la classe « Role » qu'il est demandé de créer :

Role	
Acteur	acteur
int	importance

Pour répondre à la question, nous pouvons :

1. Implémentant l'interface Comparable à la classe Role et implémenter la méthode CompareTo
2. Transformer la structure de données HashSet en TreeSet pour garder les éléments triés
3. Les éléments du TreeSet deviennent des de type « Role » au lieu de « Acteur »

SortedSet <Role> roles = new TreeSet<Role>();

4. Implémenter la méthode permettant d'afficher les acteurs qui ont joué dans un film par ordre décroissant de l'importance de leurs rôles.

Les fragments de code java sont les suivants :

a. La classe Role

```
public class Role implements Comparable <Role>{
   Acteur acteur;
   int importance;

   public Role (Acteur a, int i){
      acteur = a;
      importance = i;
   }
   public int compareTo(Role anotherRole){
      return ((this.importance) - (anotherRole.importance));
   }
}
```

Commentaire [N2] : ok, mais il faut qu'il lise bien l'énoncé ☺

b. La méthode afficherActeurs de la classe Film

Commentaire [N3] : on a aussi une autre méthode de parcours celle du for

```
public void afficherActeurs(){

      Iterator<Role> it = roles.iterator();
        while (it.hasNext()){

          System.out.println((((Role)it.next()).acteur).getNomPrenom());

      }
   }
```

Ou bien

```
public void afficherActeurs(){

     for (Role r: roles)

          System.out.println((r.acteur).getNomPrenom());

      }
   }
```

Partie 2 : Structures de données et de fichiers dynamiques

Liste linéaire chaînée de tableaux

Un maillon de la liste contient un tableau Tab et un champ Nb donnant le nombre d'éléments contenu dans le tableau.

Suivant d'une donnée d'adresse (P, n), P est l'adresse du maillon et n le déplacement dans le maillon

```
n ← n + 1
Si n ≤ Valeur(P).Nb
   Suivant ← Valeur(P) . Tab[n]
Sinon
   Si Suivant(P) ≠ Nil
      Suivant ← Valeur(Suivant(P)).Tab[1]
   Sinon
      Suivant ← Néant
   Fsi
Fsi
```

Arbre de recherche binaire

```
Si Fd(n) <> Nil
   P ← Fd(n) ;
   Tq fg(P) ≠ Nil P ← Fg(P) Ftq
   Suivant ← Info(P)
Sinon
   P ← Père(n) ;
   Remonté ← Vrai
   Tq Remonté
      Si P=Nil : Remonté ← Faux
      Sinon
         Si Fg(P) = n : Remonté ← Faux
         Sinon
            n ← P
            P ← Père(P)
         Fsi
      Fsi
   Ftq
   Si P <> Nil Suivant ← Info(P) Sinon Suivant ← Néant Fsi
Fsi
```

B-Arbre

Structure d'un bloc

```
D : Tableau[1..B] de Typeqq
Pt : Tableau[1..B] of Entier // Adresse logique de bloc
Nb : Entier // Nombre de données dans le bloc
Père : Entier // Adresse logique du bloc Père
```

Point de départ
On est sur le nœud n et la i-ème donnée (Di). Buf contient le nœud n

```
Si Buf.Pt[i+1] ≠ Nil // Non feuille
   P ← Buf .Pt[i+1]
   Tq P ≠ Nil
      Liredir(F, P, Buf)
      P ← Buf.Pt[1]
   Ftq
   Suivant ← Buf.D[1]
```

```
Sinon
   Si (i+1) < B
      Suivant ← Buf.D[i+1]
   Sinon
      Cont ← vrai
      Tq cont
         Père ← Buf.pere
         Si père= Nil
            Suivant ← Néant ; Cont ← faux
         Sinon
            Liredir(F, Buf, Père)
            // Retrouver i' tel que Pi' = P
            Trouv ← faux ; i' ← 1
            Tq i'≤ Buf.nb & Non Trouv
               Si Buf.Pt[i'] = P
                  Trouv ← Vrai
               Sinon
                  i' ← i' + 1
               Fsi
            Ftq
            Si i' < B
               Suivant ← Buf.D[i'] ; cont ← faux
            Sinon
               P ← Père
            Fsi
         Fsi
      Ftq // Cont
   Fsi
Fsi
```

Espace occupé par les structures et Coût de l'opération "Suivant"

	Espace mémoire	Coût
Listes linéaires chaînées	(n Div m) * E (Champ Tableau de m éléments + Champ Suivant)	O(1) visite en RAM
Arbre de recherche binaire	n * E(Champ Val + Champ Fils Gauche + Champ Fils Droit)	O(Log2(N)) visites en RAM
B-arbre	2 * m * E(Donnée + Pointeur) [2 buffers sont utilisés]	O(Log(N)) visites en Disque

E(x) désigne espace mémoire occupé par x.