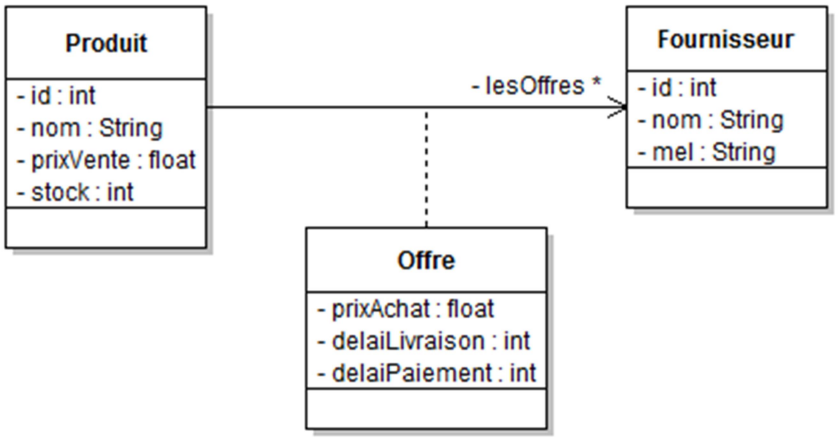
16) Les liens de 1 à plusieurs : classe association

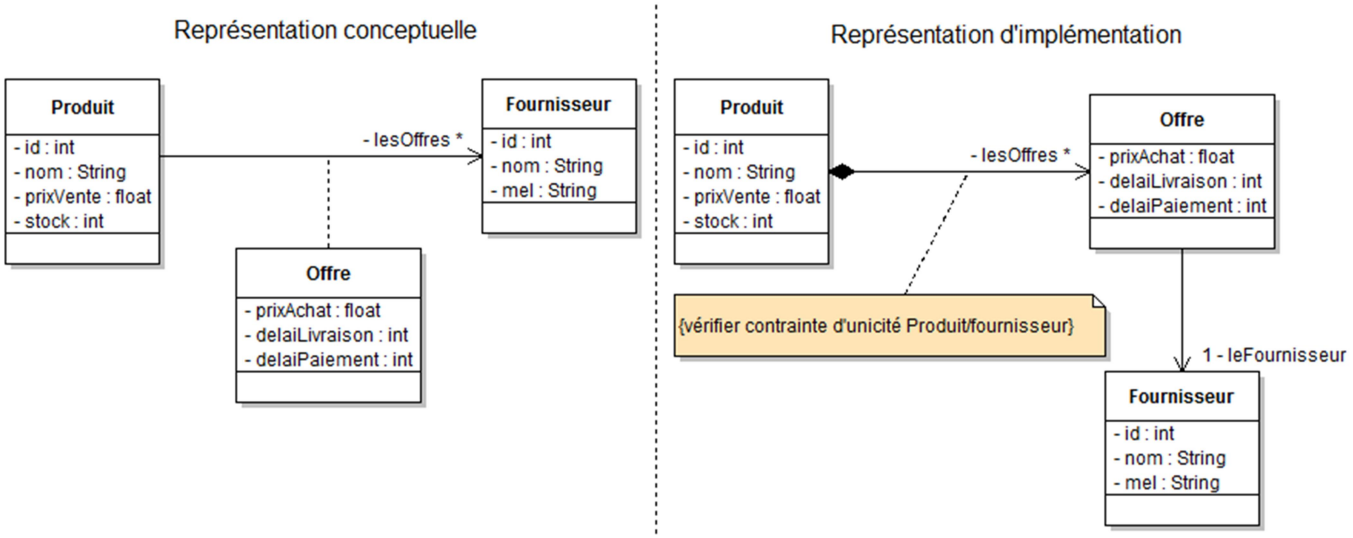
On veut traduire le fait qu'un produit est distribué par plusieurs fournisseurs chacun ayant son prix, son délai de livraison et son délai de paiement. Voici la représentation conceptuelle du diagramme de classes :



Offre est une **classe association**, c'est-à-dire que le couple Produit/Fournisseur concerne une et une seule Offre.

Ce qui est important c'est de **vérifier la contrainte d'unicité** d'un fournisseur, c'est à dire de vérifier que pour un produit on ne va pas stocker plusieurs offres du même fournisseur.

Cette situation peut se traduire de plusieurs façons en java. En voici une qui respecte la contrainte d'unicité et qui correspond à la transformation suivante :



Implémentation java

La classe Fournisseur

```
public class Fournisseur {
    //attribut privés
    private int id;
    private String nom;
    private String mel;

    //Constructeur
    public Fournisseur(int i, String n, String m){
        this.id = i;
        this.nom = n;
        this.mel = m;
    }

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

La classe Offre

```
public class Offre {
    //Attribut privés
    private Fournisseur leFournisseur;
    private float prixAchat;
    private int delaiLivraison;
    private int delaiPaiement;

    //Constructeur
    public Offre(Fournisseur f, float pa, int dl, int dp){
        this.leFournisseur = f;
        this.prixAchat = pa;
        this.delaiLivraison = dl;
        this.delaiPaiement = dp;
    }

    @Override
    public String toString(){
        String res = "\n\tFournisseur : " + this.leFournisseur.getNom();
        res = res + "\n\tPrix d'achat : " + this.prixAchat + " €";
        res = res + "\n\tDélai de livraison : " + this.delaiLivraison + " jours";
        res = res + "\n\tDélai de paiement : " + this.delaiPaiement + " jours";
        res = res + "\n\t----------------------------------------\n";
        return res;
    }

    //Accesseur
    public Fournisseur getLeFournisseur(){
        return this.leFournisseur;
    }
}
```

La classe Produit

```
public class Produit {
    //attribut privés
    private int id;
    private String nom;
    private float prixVente;
    private int stock;
    private ArrayList<Offre> lesOffres;

    //Constructeur
    public Produit(int i, String n, float pv, int s){
        this.id = i;
        this.nom = n;
        this.prixVente = pv;
        this.stock = s;
        this.lesOffres = new ArrayList<Offre>();
    }
    //Ajouter Offre
    public boolean ajouterOffre(Fournisseur f, float pa, int dl, int dp){
        //Vérifier unicité fournisseur
        boolean ok = true;
        for(Offre o : this.lesOffres){
            if(o.getLeFournisseur() == f){
                ok = false;
            }
        }

        //Si le fournisseur n'existe pas
        if(ok){
            Offre uneOffre = new Offre(f, pa, dl, dp);
            this.lesOffres.add(uneOffre);
        }
        return ok;
    }

    @Override
    public String toString(){
        String res = "\n---------------------------------------";
        res = res + "\nProduit : " + this.nom;
        res = res + "\nPrix de vente : " + this.prixVente + " €";
        res = res + "\nStock actuel : " + this.stock;
        res = res + "\nFournisseurs : ";
        for(Offre o : this.lesOffres){
            res = res + "\t" + o.toString();
        }
        res = res + "\n---------------------------------------";
        return res;
    }
}
```

la méthode renvoit vrai uniquement si l'ajout peut se faire

Composition

Vérification de la contrainte d'unicité