SLAM 4
Ch. 10 – Les dictionnaires approfondissement

3. Un panier en Java
3.1 - *Le contexte : la société Shop'n Bag*

- Shop'n Bag est une entreprise spécialisée dans la maroquinerie grand public.

- site web marchand accessible à tous les internautes.
- application en Java.

3. Un panier en Java
3.1 - *Le contexte : la société Shop'n Bag*

- Système de commande avec panier

Produit	sac à main - 80 €	ceinture - 25 €	étiquette - 3 €
			
Quantité	1	2	5

- Le panier = **dictionnaire** (java.util.**HashMap**) :
  - les **clés** sont les **produits,**
  - les **valeurs** sont les **quantités** commandées.

Rappel : dictionnaire

clés :	orange	citron	pomme	poire
valeurs :	5	1	6	3

Exercice 1. Travail à faire : écrire le code Java qui permet d'instancier et de remplir le panier.

Produit	sac à main - 80 €	ceinture - 25 €	étiquette - 3 €
			
Quantité	1	2	5

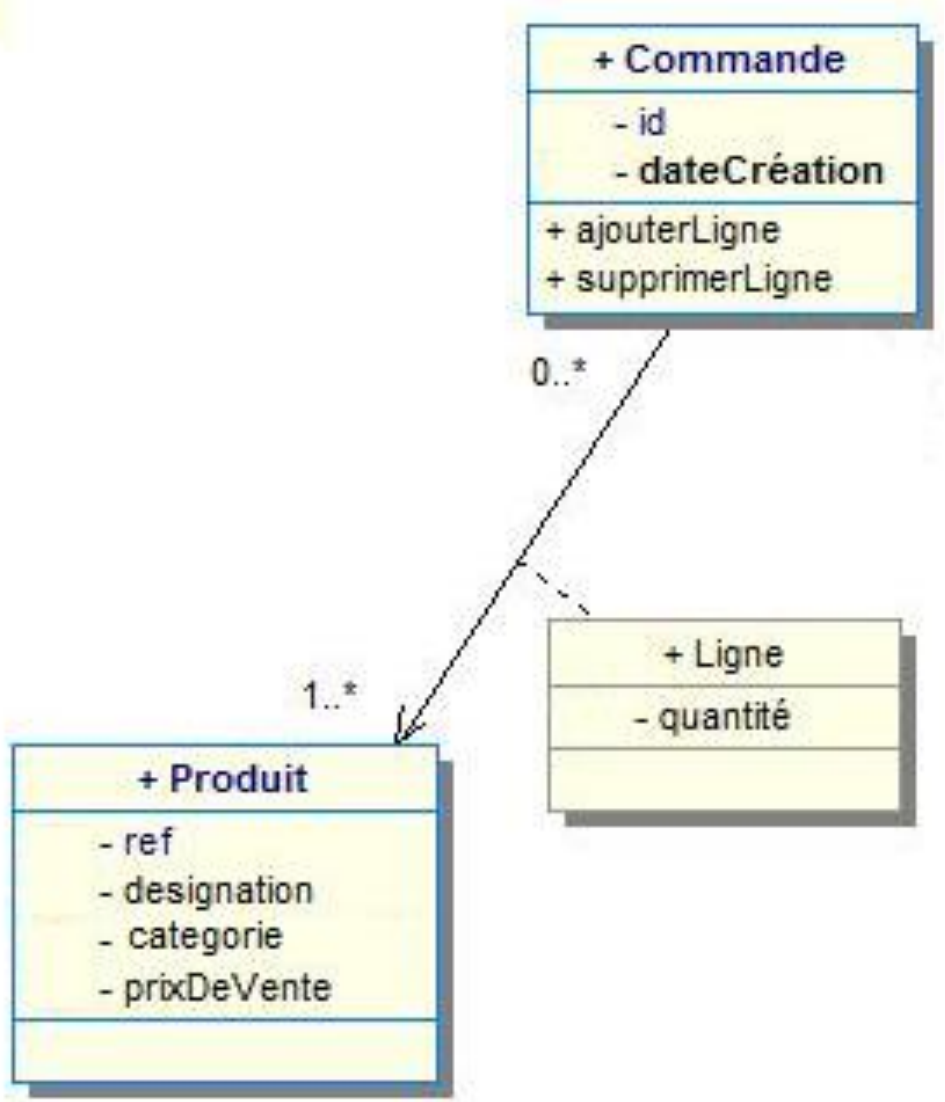
```
Produit p1 = new Produit("sac à main", 80) ;

.  .  .

// classe HashMap<Produit, Integer>

.  .  .
```

Exercice 1. Travail à faire : écrire le code Java qui permet d'instancier et de remplir le panier.

Produit	sac à main - 80 €	ceinture - 25 €	étiquette - 3 €
			
Quantité	1	2	5

```
Produit p1 = new Produit("sac à main", 80) ;
Produit p2 = new Produit("ceinture", 25) ;
Produit p3 = new Produit("étiquette", 3) ;
HashMap<Produit, Integer> panier = new HashMap<Produit, Integer>();
panier.put(p1, 1);
panier.put(p2, 2);
panier.put(p3, 5);
```

3.2 - Le diagramme de classes



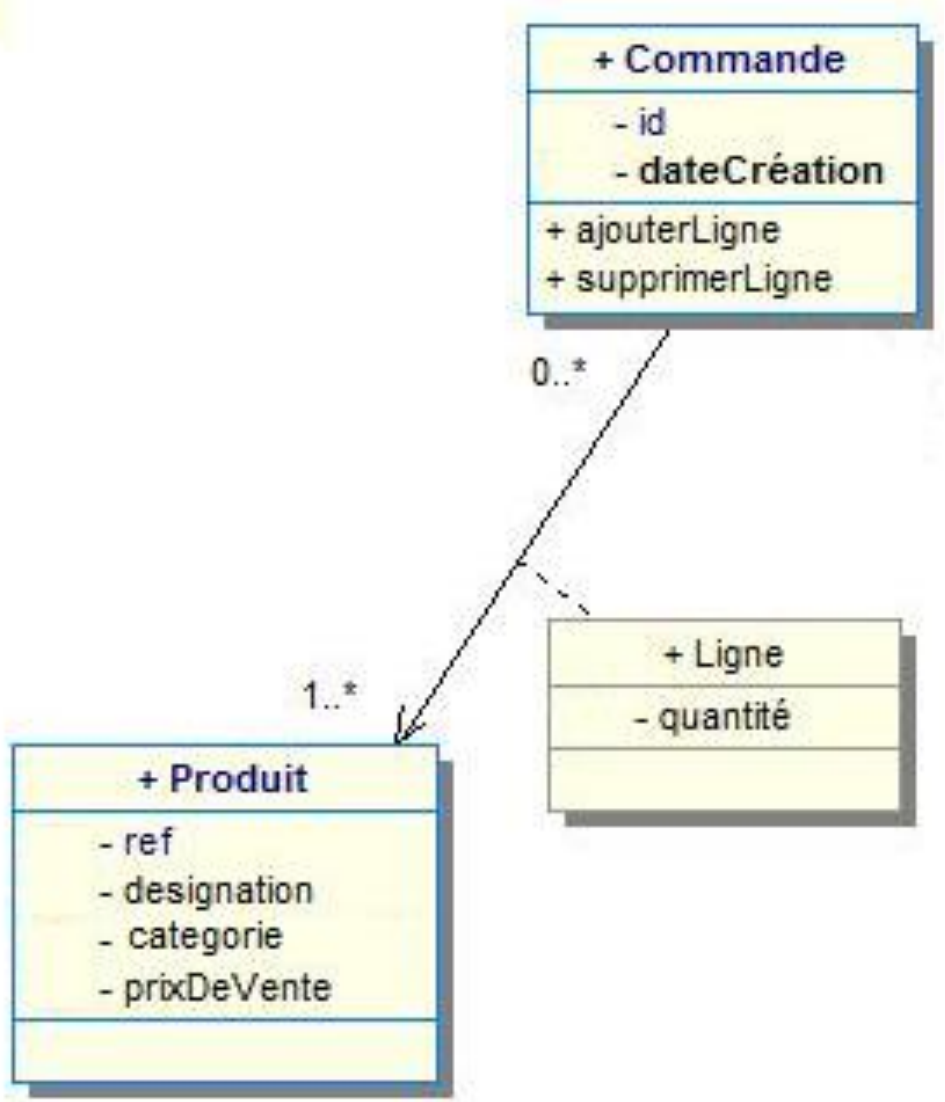
- un produit peut faire partie de plusieurs commandes (ou aucune) :

  0..*

- une commande contient plusieurs produits, et au moins un :

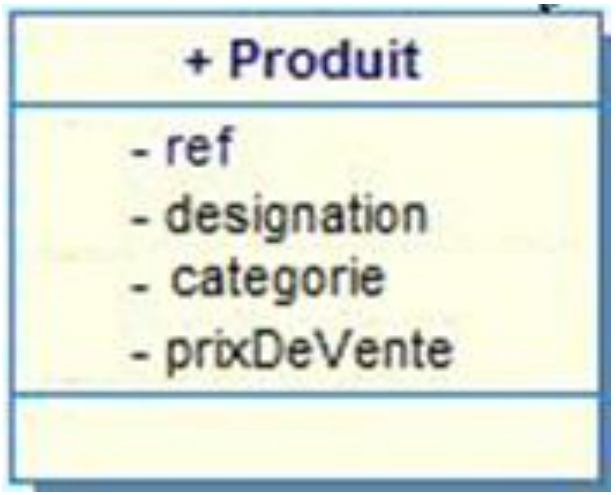
  1..*

3.2 - Le diagramme de classes



- association porteuse d'un attribut : **quantité**
- La quantité concerne un couple d'instances des deux classes **Commande** et **Produit**
- classe **Ligne :** classe-association

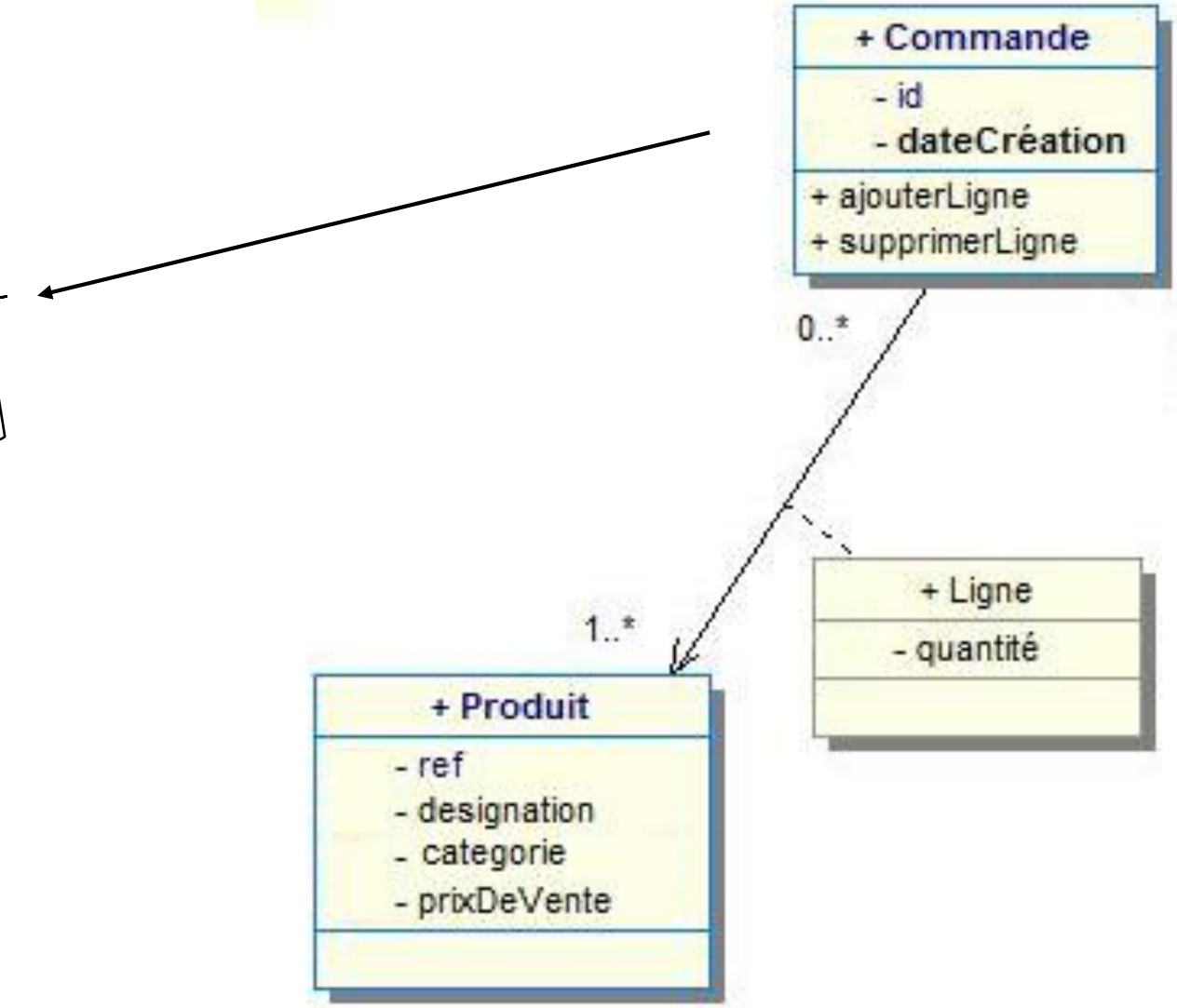
3.3 - *Le code Java*




```
public class Produit
{
    // attributs privés
    private String ref;
    private String designation;
    private String categorie;
    private double prixDeVente;

    // constructeur
    public Produit(String ref, String des, String categ, double pri)
    {   // à compléter…   }

    // et encore :
    // accesseurs, modificateurs, toString.
}
```

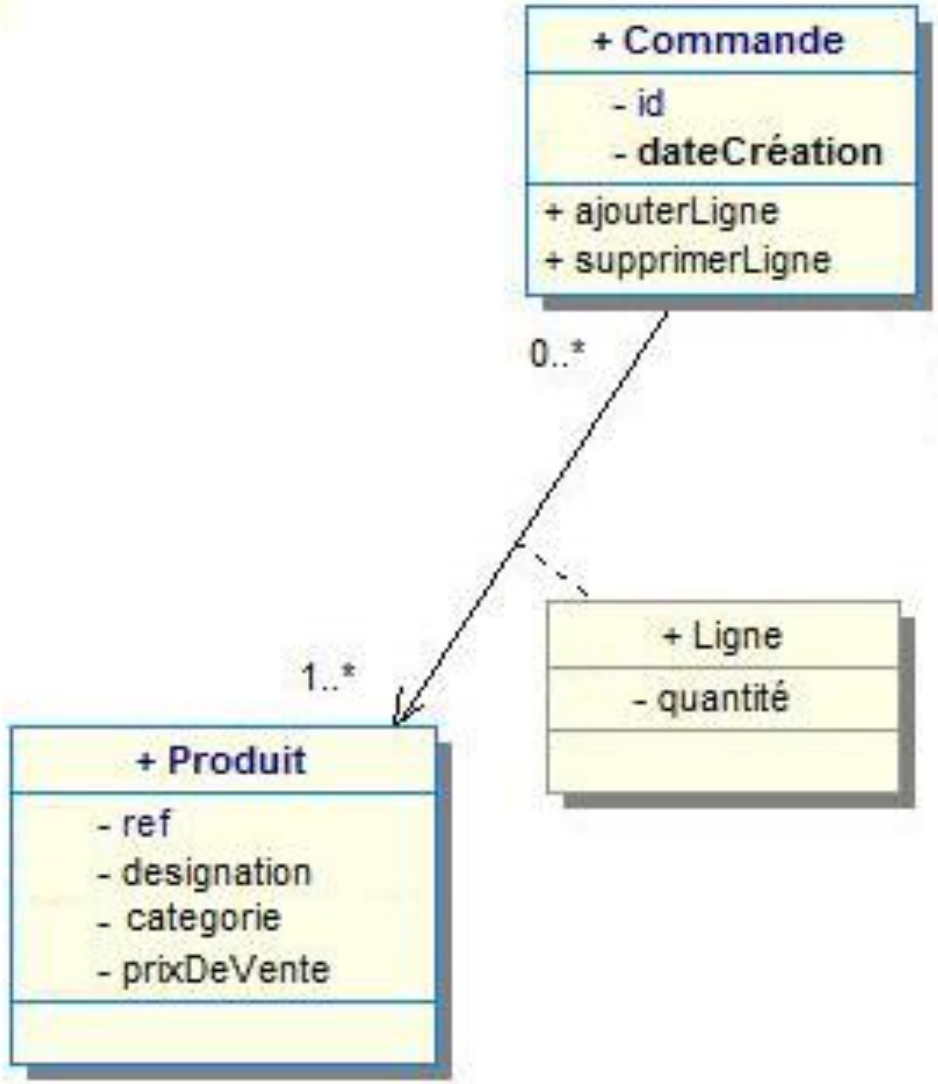
3.3 - *Le code Java*

```
public class Commande
{
    private int id;
    private String dateCreation;
```



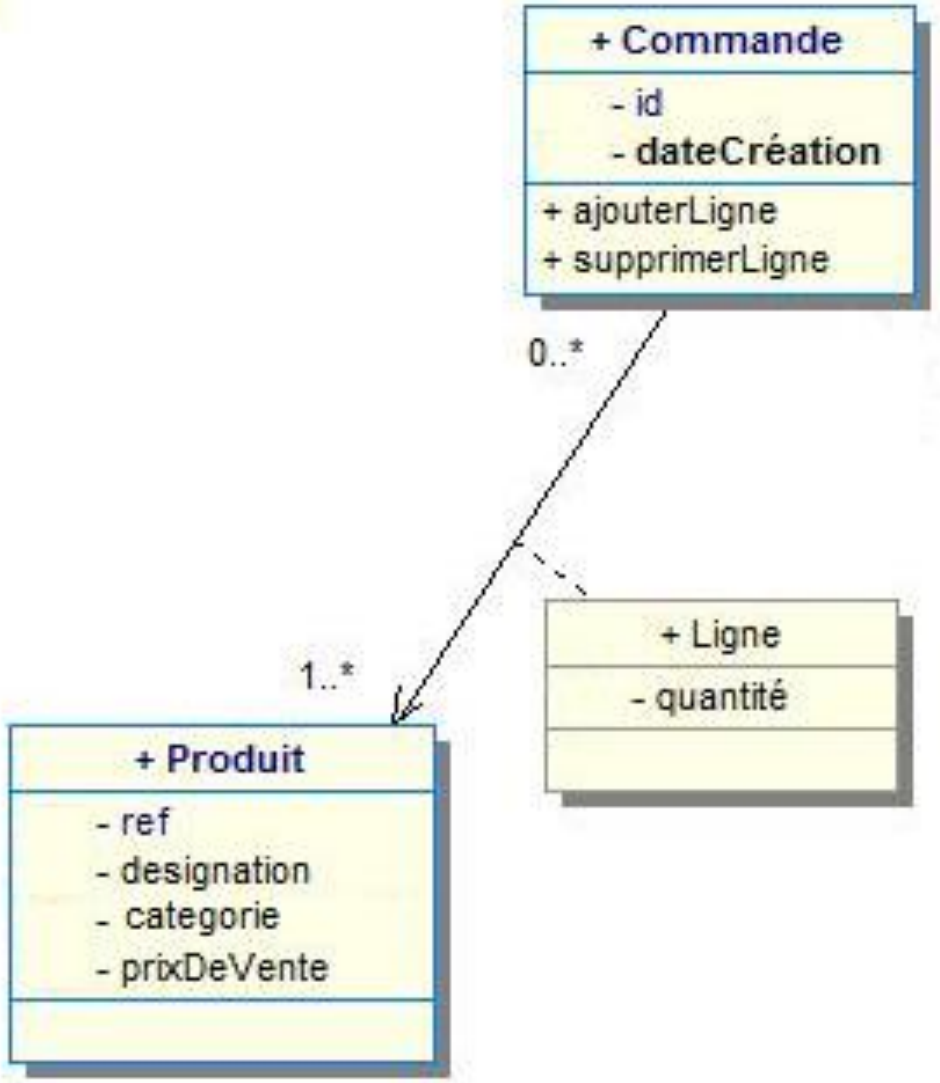
```
public class Commande
{
    private int id;
    private String dateCreation;

    private HashMap<Produit, Integer> lesLignes;

    // dictionnaire :
    // clé = produit, valeur = quantité
```

Produit	quantité
sac à main - 80 € 	1
ceinture - 25 € 	2
étiquette - 3 € 	5



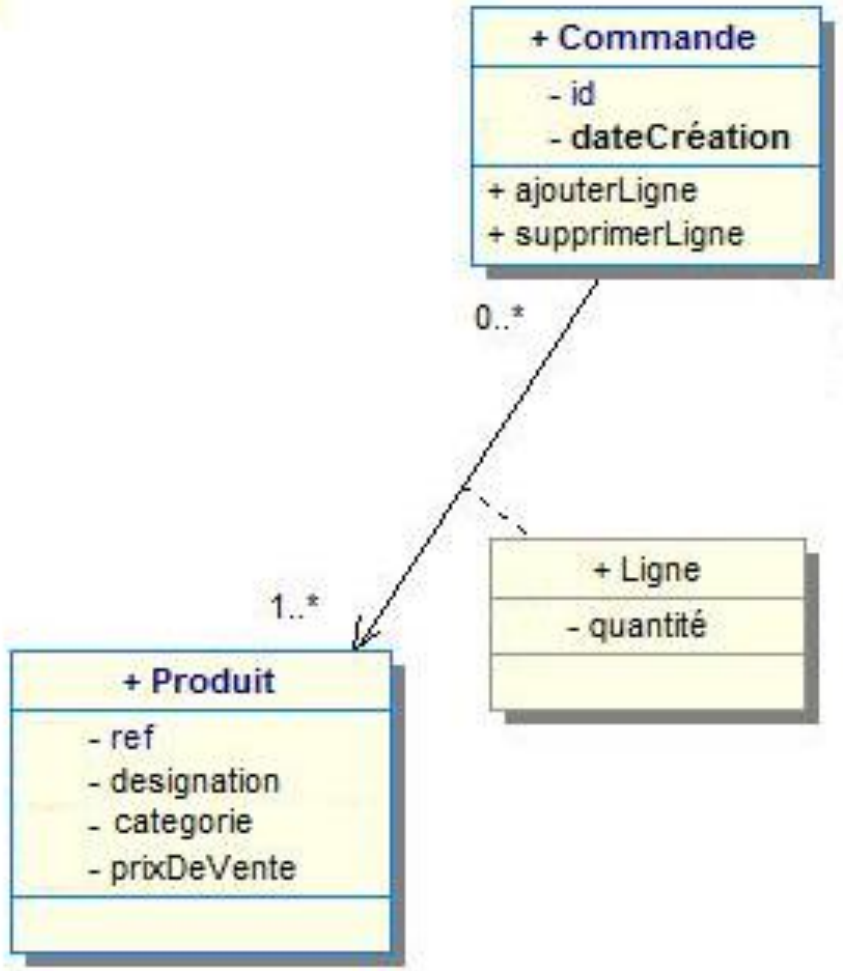
3.3 - *Le code Java*

```
public class Commande
{
    private int id;
    private String dateCreation;

    private HashMap<Produit, Integer> lesLignes;
    // dictionnaire :    clé = produit,
    //                   valeur = quantité

    /**
     * Constructeur
     */
    public Commande(int id, String date)
    {
        // valorisation des attributs
        this.id = id;
        this.dateCreation = date;
        // instanciation du dictionnaire
        this.lesLignes = new HashMap<Produit, Integer>();
    }
```



3.3 - *Le code Java*

```
public class Commande
{
   …
 // accesseur
 public HashMap<Produit, Integer> getLesLignes()
 {
      return this.lesLignes;
 }

/**
* Ajoute une ligne de commande à la commande courante
*/
public void ajouterLigne(Produit unProd, int uneQte)
{
       this.lesLignes.put(unProd, uneQte);
}
```



Exercice 2.

2.a) Ecrire la méthode **supprimerLigne** qui supprime la ligne de commande correspondant au produit passé en paramètre. (ressemble à **ajouterLigne**).

```
public void supprimerLigne(Produit unProduit)
{
    . . .
}
```

// utiliser la méthode de la classe HashMap
// qui retire du dictionnaire l'élément correspondant à la clé
// spécifiée :
remove (KeyType key)

Produit	quantité
sac à main - 80 € 	1
ceinture - 25 € 	2
étiquette - 3 € 	5

Exercice 2. Solution

2.a) Ecrire la méthode **supprimerLigne** qui supprime la ligne de commande correspondant au produit passé en paramètre. (ressemble à **ajouterLigne**).

```
public void supprimerLigne(Produit unProduit)
{
      this.lesLignes.remove(unProduit);
}
```

Produit	quantité
sac à main - 80 € 	1
ceinture - 25 € 	2
étiquette - 3 € 	5

Exercice 2.

2.b) Ecrire la méthode **getQuantite** décrite dans le commentaire **:**

```
/**
 * Retourne la quantité du produit unProduit, contenue dans les lignes de la commande.
 * @param unProduit : le produit dont on veut obtenir la quantité.
 * @return la quantité du produit s'il est présent dans la commande, 0 sinon.
 */
public int getQuantite(Produit unProduit)
{
    .  .  .
}
```

Produit	quantité
sac à main - 80 € 	1
ceinture - 25 € 	2
étiquette - 3 € 	5

// utiliser les méthodes de HashMap :

public boolean containsKey (KeyType key)

// retourne true si l'élément dont la clé est passée en paramètre est présent dans le dictionnaire, false sinon

public ValueType get (KeyType key)

// retourne la valeur correspondant à la clé spécifiée, ou null si la clé spécifiée est inexistante

Exercice 2. solution

2.b) Ecrire la méthode **getQuantite** décrite dans le commentaire **:**

```
/**
 * Retourne la quantité du produit unProduit, contenue dans les lignes de la commande.
 * @param unProduit : le produit dont on veut obtenir la quantité.
 * @return la quantité du produit s'il est présent dans la commande, 0 sinon.
 */
public int getQuantite(Produit unProduit)
{
	if (this.lesLignes.containsKey(unProduit))
		return this.lesLignes.get(unProduit);
	else
		return 0;
}
```

Produit	quantité
sac à main - 80 € 	1
ceinture - 25 € 	2
étiquette - 3 € 	5

Exercice 2.

- **2.c)** Ecrire la méthode **getNombreArticles** décrite dans le commentaire **:**

```
/**
 * Retourne le nombre total d'articles contenus dans les lignes de la commande.
 * @return le nombre total d'articles de la commande.
*/
public int getNombreArticles()
{
    . . .
}
```

Produit	quantité
sac à main - 80 € 	1
ceinture - 25 € 	2
étiquette - 3 € 	5

// utiliser la méthode de HashMap :

public Collection<KeyType> values ()

// retourne une collection des valeurs contenues dans le dictionnaire

// parcourir la collection avec :

```
for ( KeyType uneValeur  :  laCollection )  {  …  }
```

Exercice 2. Solution

- **2.c)** Ecrire la méthode **getNombreArticles** décrite dans le commentaire **:**

```
/**
 * Retourne le nombre total d'articles contenus dans les lignes
de la commande.
 * @return le nombre total d'articles de la commande.
*/
public int getNombreArticles()
{
      int total = 0;
      for (int qte : this.lesLignes.values())
      {
            total += qte;
      }
      return total;
}
```

Produit	quantité
sac à main - 80 € 	1
ceinture - 25 € 	2
étiquette - 3 € 	5

3.4 - Exercice 3 – suite sur PC (ou sur papier)

- Voir support de cours page 4