# 

# 

# 420-EAB-LI Programmation orientée objets 2

## Vecteurs d’objets

## Les tableaux (ou vecteurs) de primitives (types de base) sont des objets

Il est important de réaliser que la différence fondamentale existant entre les primitives et les objets, existe également entre les primitives et les tableaux de primitives. Les tableaux sont des objets. Lorsque vous manipulez un tableau, vous ne manipulez réellement qu'une référence vers ce tableau. Le programme suivant met cela en évidence :

class TestVecteur1

{

public static void main(String[] args)

{

int[] vec1 = {1, 2, 3, 4, 5};

int[] vec2 = {10, 11, 12};

vec1 = vec2;

vec1[2] = 100;

System.out.println(vec2[2]);

}

}

Nous aurons donc le résultat suivant à l’écran :

100

**class** TestVecteur1

{

**public** **static** **void** main(String[] args)

{

**int**[] vec1 = {1, 2, 3, 4, 5};

**int**[] vec2 = {10, 11, 12};

vec1 = vec2;

vec1[2] = 100;

System.***out***.println(vec2[2]);

**for** (**int** i=0;i<vec1.length;i++)

{

System.***out***.println("Vect1:"+vec1[i]+"\n");

}

**for** (**int** i=0;i<vec2.length;i++)

{

System.***out***.println("Vect2:"+vec2[i]+"\n");

}

}

}

100

Vect1:10

Vect1:11

Vect1:100

Vect2:10

Vect2:11

Vect2:100

A la septième ligne de ce programme, nous affectons la valeur de **y** à **x**. Cela paraît choquant si l'on considère que **x** et **y** sont des tableaux de longueurs différentes. En fait, cela ne l'est pas du tout, car il s'agit simplement de détacher la référence **x** du tableau vers lequel il pointe pour le faire pointer vers le même tableau que la référence **y**. La seule chose nécessaire pour que cela soit possible est que les deux tableaux pointés soient de même type.

A la ligne 8 du programme, nous modifions la valeur de l'élément d'indice 2 du tableau **x**. Comme nous le voyons à la ligne suivante, cette modification concerne également le tableau pointé par **y**, puisqu'il s'agit du même.

## 

12

11

10

5

4

3

2

1

vec2

vec1

## Les tableaux d'objets sont des tableaux de références à des objets

Tout comme les références à des objets ne sont pas des objets, les tableaux d'objets ne contiennent pas des objets, mais des références à des objets. Nous pouvons mettre cela en évidence à l'aide du programme suivant :

class TestVecteur2   
{

public static void main(String[] args)   
{

Cercle c1 = new Cercle(2, "Rouge");

Cercle c2 = new Cercle(3, "Bleu");

Cercle c3 = new Cercle(4, "Vert");

Cercle c4 = new Cercle(5, "Jaune");

Cercle c5 = new Cercle(6, "Orange");

Cercle[] vec1 = {c1, c2, c3};

Cercle[] vec2 = {c4, c5};

vec2[0] = vec1[0];

// Modifier le rayon du cercle1

c1.setRayon(10);

**// ou vec1[0].setRayon(10); puisque les 2**

**// références pointent au même endroit**

//Que sera-t-il affiché ?

System.out.println(vec1[0].getRayon());

System.out.println(vec2[0].getRayon());

}

}

On obtient donc l’affichage suivant à l’écran :

10

10

Ce programme crée deux vecteurs **vec1** et **vec2**, tous deux de type **Cercle[]**. L'élément 0 du tableau **vec1** contient la référence **c1**, pointant vers un objet de type **Cercle** dont l’attribut rayon vaut 2. L'élément 0 du tableau **vec2** contient la référence **c4**, pointant vers un objet de type **Cercle** dont l’attribut rayon vaut 5.

La ligne :

vec2[0] = vec1[0];

fait pointer l'élément 0 de **vec2** vers le même objet que l'élément 0 de **vec1**, c'est-à-dire l'objet pointé par **c1**. Lorsque, à la ligne suivante, nous modifions l’attribut rayon de **c1**, **vec1[0]** et **vec2[0]** sont également modifiés.

La situation de l'exemple **TestVecteur2** peut être représentée graphiquement de la façon suivante:

c5

c4

c3

c2

c1

rayon = 3

couleur = Bleu

rayon = 2

couleur = Rouge

rayon = 5

couleur = Jaune

rayon = 4

couleur = Vert

rayon = 6

couleur = Orange

vec1

vec2

Ici, lorsque nous modifions la valeur de **vec1[0]**, **c1** est également modifié. Encore une fois, répétons que les tableaux permettent de stocker et de manipuler des références à des objets et des primitives, mais en aucun cas des objets.

## Vecteurs et objets anonymes

Les vecteurs peuvent contenir des objets anonymes, comme le montre l'exemple suivant:

class TestVecteur3

{

public static void main(String[] args)

{

Cercle[] vec1 = {new Cercle(2, "Rouge"),

new Cercle(3, "Bleu"),

new Cercle(4, "Vert")};

Cercle[] vec2 = {new Cercle(5, "Jaune"),

new Cercle(6, "Orange")};

// Modifier le rayon du cercle Rouge

// c1.setRayon(10); Ne fonctionne plus puisque

// le cercle rouge n'a plus de nom

// Comment le modifier alors ???

vec1[0].setRayon(10);

System.out.println(vec1[0].getRayon());

}

}

Les objets des 2 vecteurs sont créés de façon anonyme. **Ils ne sont pas perdus pour autant** puisqu'ils sont immédiatement affectés aux éléments des vecteurs vec1 et vec2.

**Exemple**

**1- Classe Voiture**

**package** VoitureP;

**import** javax.swing.JOptionPane;

/\*\*

\* Classe représentant une voiture avec marque, modèle, année, km et prix.

\* Exemple pour les notes de cours POO

\* \*

\*/

**public** **class** Voiture

{

// Constantes de classe

/\*\*

\* Marque par défaut

\*/

**public** **static** **final** String ***MARQUE\_DEFAUT*** = "marque";

/\*\*

\* Modèle par défaut

\*/

**public** **static** **final** String ***MODELE\_DEFAUT*** = "modele";

/\*\*

\* Année par défaut

\*/

**public** **static** **final** **int** ***ANNEE\_DEFAUT*** = 1900;

/\*\*

\* Kilométrage par défaut

\*/

**public** **static** **final** **float** ***KM\_DEFAUT*** = 0f;

/\*\*

\* Prix par défaut

\*/

**public** **static** **final** **float** ***PRIX\_DEFAUT*** = 0f;

/\*\*

\* Nombre de caractères minimum pour la chaîne marque et modèle

\*/

**public** **static** **final** **int** ***NOMBRE\_DE\_CARACTERES\_MIN*** = 2;

/\*\*

\* Nombre de caractères maximum pour la chaîne marque et modèle

\*/

**public** **static** **final** **int** ***NOMBRE\_DE\_CARACTERES\_MAX*** = 25;

/\*\*

\* Année minimum

\*/

**public** **static** **final** **int** ***ANNEE\_MIN*** = 1900;

/\*\*

\* Année maximum

\*/

**public** **static** **final** **int** ***ANNEE\_MAX*** = 2015;

/\*\*

\* Kilométrage minimum

\*/

**public** **static** **final** **float** ***KM\_MIN*** = 0f;

/\*\*

\* Kilométrage maximum

\*/

**public** **static** **final** **float** ***KM\_MAX*** = 500000f;

/\*\*

\* Prix minimum

\*/

**public** **static** **final** **float** ***PRIX\_MIN*** = 0f;

/\*\*

\* Prix maximum

\*/

**public** **static** **final** **float** ***PRIX\_MAX*** = 300000f;

// Attributs de l'objet

/\*\*

\* Le nom de la marque de la voiture

\*/

**private** String marque = **null**;

/\*\*

\* Le nom du modèle de la voiture

\*/

**private** String modele = **null**;

/\*\*

\* L'année de la voiture

\*/

**private** **int** annee = 0;

/\*\*

\* Le kilométrage de la voiture

\*/

**private** **float** km = 0;

/\*\*

\* Le prix de la voiture

\*/

**private** **float** prix = 0;

/\*\*

\* Constructeur voiture par défaut

\*/

**public** Voiture()

{

**this**(***MARQUE\_DEFAUT***, ***MODELE\_DEFAUT***, ***ANNEE\_DEFAUT***, ***KM\_DEFAUT***, ***PRIX\_DEFAUT***);

}

/\*\*

\* Constructeur voiture avec les attributs reçus en entrée.

\*

\* **@param** paramMarque une string, la marque de la voiture

\* **@param** paramModele une string, le modèle de la voiture

\* **@param** paramAnnee un entier, l'année de la voiture

\* **@param** paramKm un réel, le kilométrage de la voiture

\* **@param** paramPrix un réel, le prix de la voiture

\*/

**public** Voiture(String paramMarque, String paramModele, **int** paramAnnee,

**float** paramKm, **float** paramPrix)

{

// Valider les paramètres reçus et les assigner aux attributs de

// l'objet, s'ils sont tous valides.

**boolean** ok = setMarque(paramMarque) && setModele(paramModele)

&& setAnnee(paramAnnee) && setKm(paramKm) && setPrix(paramPrix);

// Sinon, attribuer les valeurs par défaut aux attributs de l'objet

**if** (!ok)

{

marque = ***MARQUE\_DEFAUT***;

modele = ***MODELE\_DEFAUT***;

annee = ***ANNEE\_DEFAUT***;

km = ***KM\_DEFAUT***;

prix = ***PRIX\_DEFAUT***;

}

}

/\*\*

\* Retourner la marque de la voiture.

\*

\* **@return** string, la marque de la voiture

\*/

**public** String getMarque()

{

**return** marque;

}

/\*\*

\* Modifier la marque de la voiture à partir de celle reçue en entrée.

\*

\* **@param** paramMarque une string, la marque de la voiture

\*

\* **@return** boolean, indiquant vrai si la marque a été modifiée

\*/

**public** **boolean** setMarque(String paramMarque)

{

// Valider la marque reçue en entrée

**boolean** ok = *validerMarque*(paramMarque);

// Si elle est valide, attribuer sa valeur à l'attribut marque

**if** (ok)

{

marque = paramMarque;

}

**return** ok;

}

/\*\*

\* Vérifier si la marque reçue en entrée est valide.

\*

\* **@param** paramMarque une string, la marque de la voiture

\*

\* **@return** boolean, indiquant vrai si c’est une marque valide.

\*/

**public** **static** **boolean** validerMarque(String paramMarque)

{

**return** ((paramMarque.length() >= ***NOMBRE\_DE\_CARACTERES\_MIN***) && (paramMarque

.length() <= ***NOMBRE\_DE\_CARACTERES\_MAX***));

}

/\*\*

\* Retourner le modèle de la voiture.

\*

\* **@return** string, le modèle de la voiture

\*/

**public** String getModele()

{

**return** modele;

}

/\*\*

\* Modifier le modèle de la voiture à partir de celui reçu en entrée.

\*

\* **@param** paramModele une string, le modèle de la voiture

\*

\* **@return** boolean, indiquant vrai si le modèle a été modifié.

\*/

**public** **boolean** setModele(String paramModele)

{

// Valider le modèle reçu en entrée

**boolean** ok = *validerModele*(paramModele);

// S'il est valide, attribuer sa valeur à l'attribut modele

**if** (ok)

{

modele = paramModele;

}

**return** ok;

}

/\*\*

\* Vérifier si le modèle reçu en entrée est valide.

\*

\* **@param** paramModele une string, le modèle de la voiture

\*

\* **@return** boolean, indiquant vrai si c’est un modèle valide.

\*/

**public** **static** **boolean** validerModele(String paramModele)

{

**return** ((paramModele.length() >= ***NOMBRE\_DE\_CARACTERES\_MIN***) && (paramModele

.length() <= ***NOMBRE\_DE\_CARACTERES\_MAX***));

}

/\*\*

\* Retourner l’année de la voiture.

\*

\* **@return** int, l'année de la voiture

\*/

**public** **int** getAnnee()

{

**return** annee;

}

/\*\*

\* Modifier l’année de la voiture à partir de celle reçue en entrée.

\*

\* **@param** paramAnnee un entier, l'année de la voiture

\*

\* **@return** boolean, indiquant vrai si l’année a été modifiée.

\*/

**public** **boolean** setAnnee(**int** paramAnnee)

{

// Valider l’année reçue en entrée

**boolean** ok = *validerAnnee*(paramAnnee);

// Si elle est valide, attribuer sa valeur à l'attribut annee

**if** (ok)

{

annee = paramAnnee;

}

**return** ok;

}

/\*\*

\* Vérifier si l’année reçue en entrée est valide.

\*

\* **@param** paramAnnee un entier, l'année de la voiture

\*

\* **@return** boolean, indiquant vrai si c’est une année valide

\*/

**public** **static** **boolean** validerAnnee(**int** paramAnnee)

{

**return** ((paramAnnee >= ***ANNEE\_MIN***) && (paramAnnee <= ***ANNEE\_MAX***));

}

/\*\*

\* Retourner le kilométrage de la voiture.

\*

\* **@return** float, le kilométrage actuel de la voiture.

\*/

**public** **float** getKm()

{

**return** km;

}

/\*\*

\* Modifier le kilométrage de la voiture à partir de celui reçu en entrée.

\*

\* **@param** paramKm un réel, le kilométrage de la voiture

\*

\* **@return** boolean, indiquant vrai si le kilométrage a été modifié.

\*/

**public** **boolean** setKm(**float** paramKm)

{

// Valider le kilométrage reçu en entrée

**boolean** ok = *validerKm*(paramKm);

// S'il est valide, attribuer sa valeur à l'attribut km

**if** (ok)

{

km = paramKm;

}

**return** ok;

}

/\*\*

\* Vérifier si le kilométrage reçu en entrée est valide.

\*

\* **@param** paramKm un réel, le kilométrage de la voiture

\*

\* **@return** boolean, indiquant vrai si c’est un kilométrage valide.

\*/

**public** **static** **boolean** validerKm(**float** paramKm)

{

**return** ((paramKm >= ***KM\_MIN***) && (paramKm <= ***KM\_MAX***));

}

/\*\*

\* Retourner le prix de la voiture.

\*

\* **@return** float, le prix de la voiture

\*/

**public** **float** getPrix()

{

**return** prix;

}

/\*\*

\* Modifier le prix de la voiture à partir de celui reçu en entrée.

\*

\* **@param** paramPrix un réel, le prix de la voiture

\*

\* **@return** boolean, indiquant vrai si le prix a été modifié.

\*/

**public** **boolean** setPrix(**float** paramPrix)

{

// Valider le prix reçu en entrée

**boolean** ok = *validerPrix*(paramPrix);

// S'il est valide, attribuer sa valeur à l'attribut prix

**if** (ok)

{

prix = paramPrix;

}

**return** ok;

}

/\*\*

\* Vérifier si le prix reçu en entrée est valide.

\*

\* **@param** paramPrix un réel, le prix de la voiture.

\*

\* **@return** boolean, indiquant vrai si c’est un prix valide.

\*/

**public** **static** **boolean** validerPrix(**float** paramPrix)

{

**return** ((paramPrix >= ***PRIX\_MIN***) && (paramPrix <= ***PRIX\_MAX***));

}

/\*\*

\* Afficher les attributs d’une voiture

\*/

**public** **void** afficherVoiture()

{

System.***out***

.println(**this**.getMarque() + "\t" + **this**.getModele() + "\t"

+ **this**.getAnnee() + "\t" + **this**.getKm() + "\t"

+ **this**.getPrix());

}

/\*\*

\* Permet d'afficher le contenue de l'objet Voiture

\*

\* **@return** string, une chaîne présentant les valeurs des attributs d'un

\* objet Voiture

\*

\* **@see** java.lang.Object#toString()

\*/

**public** String toString()

{

**return** **this**.getMarque() + "\t" + **this**.getModele() + "\t"

+ **this**.getAnnee() + "\t" + **this**.getKm() + "\t" + **this**.getPrix();

}

}

**2- Classe TabObjetVoiture**

**package** VoitureP;

**public** **class** TabObjetVoiture {

/\*\*

\* Faire des essais unitaires sur la classe

\*

\*

\* \*/

**public** **static** **void** main(String[] args)

{

// Exemple d'initialisation et d'utilisation d'objets dans une vecteur

Voiture v0 = **new** Voiture("Honda", "Civic", 2008, 10000, 30200);

Voiture v1 = **new** Voiture("Nisan", "Sentra", 2000, 1000, 30000);

Voiture v2 = **new** Voiture("Dodge", "Dart", 2011, 20000, 12000);

Voiture[] vecV = **new** Voiture[3];

// Il n'y a pas encore d'objet voiture dans le vecteur de voitures

**for** (**int** i = 0; i < vecV.length; i++)

{

System.***out***.println("vecV à l'indice : " + i + " = " + vecV[i]);

}

System.***out***.println();

// On place une voiture dans le vecteur de voitures

vecV[2] = v2;

// On regarde le résultat de l'affectation

**for** (**int** i = 0; i < vecV.length; i++)

{

System.***out***.println("vecV à l'indice : " + i + " = " + vecV[i]);

}

System.***out***.println();

// On place des voitures dans le vecteur de voitures

vecV[2] = v0;

vecV[1] = v2;

vecV[0] = v1;

// On regarde le résultat de l'affectation

**for** (**int** i = 0; i < vecV.length; i++)

{

System.***out***.println("vecV à l'indice : " + i + " = " + vecV[i]);

}

System.***out***.println();

// On ajoute 1000Km à toutes les voitures

**for** (**int** i = 0; i < vecV.length; i++)

{

vecV[i].setKm(vecV[i].getKm() + 1000);

}

// On regarde le résultat de l'affectation

**for** (**int** i = 0; i < vecV.length; i++)

{

System.***out***.println("vecV à l'indice : " + i + " = " + vecV[i]);

}

System.***out***.println();

// On vide le vecteur d'objets

**for** (**int** i = 0; i < vecV.length; i++)

{

vecV[i] = **null**;

}

// On regarde le résultat de l'affectation

**for** (**int** i = 0; i < vecV.length; i++)

{

System.***out***.println("vecV à l'indice : " + i + " = " + vecV[i]);

}

System.***out***.println();

}

}

vecV à l'indice : 0 = null

vecV à l'indice : 1 = null

vecV à l'indice : 2 = null

vecV à l'indice : 0 = null

vecV à l'indice : 1 = null

vecV à l'indice : 2 = Dodge Dart 2011 20000.0 12000.0

vecV à l'indice : 0 = Nisan Sentra 2000 1000.0 30000.0

vecV à l'indice : 1 = Dodge Dart 2011 20000.0 12000.0

vecV à l'indice : 2 = Honda Civic 2008 10000.0 30200.0

vecV à l'indice : 0 = Nisan Sentra 2000 2000.0 30000.0

vecV à l'indice : 1 = Dodge Dart 2011 21000.0 12000.0

vecV à l'indice : 2 = Honda Civic 2008 11000.0 30200.0

vecV à l'indice : 0 = null

vecV à l'indice : 1 = null

vecV à l'indice : 2 = null