UML

Baptiste Bauer

Version v0.0.8.sip-221124091855, 2022-11-23 09:32:18

Table des matières

1. L'interface

1.1. Notion d'interface et modélisation UML

Il faut avoir parfaitement compris ce qu'est une relation abstraite pour comprendre la relation d'interface.

Une interface est une **classe dont toutes les méthodes sont publiques et abstraites**.

Cela ressemble à une classe abstraite sauf que contrairement à elle, une interface n'a que des méthodes sans corps. Il n'y a donc que la signature d'une ou plusieurs méthodes.

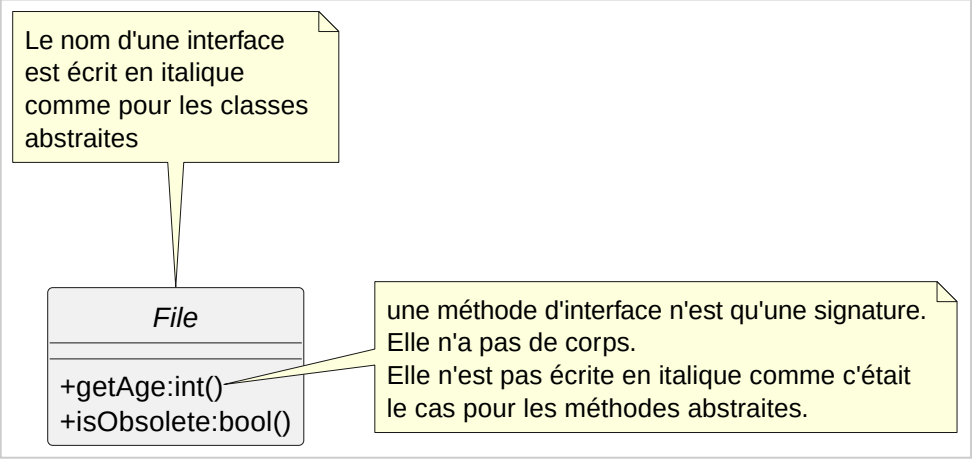
Vous devez vous demander à quoi peut servir une "classe" qui n'a que des méthodes abstraites.

Dans la partie sur la relation abstraite, nous avons vu qu'une classe qui hérite d'une classe abstraite qui contient des méthodes abstraites doit obligatoirement implémenter celles-ci. La classe qui hérite est contrainte, elle n'a pas le choix, elle DOIT implémenter ces méthodes.

Donc, si une classe *hérite* d'une interface, elle devra obligatoirement implémenter les méthodes de celle-ci. C'est très utile pour contraindre les classes à être manipulées avec des méthodes qui sont prévisibles car définies par l'interface.

En programmation, on ne dit pas qu'une classe *hérite* d'une interface, on dit qu'une classe **implémente une interface**.

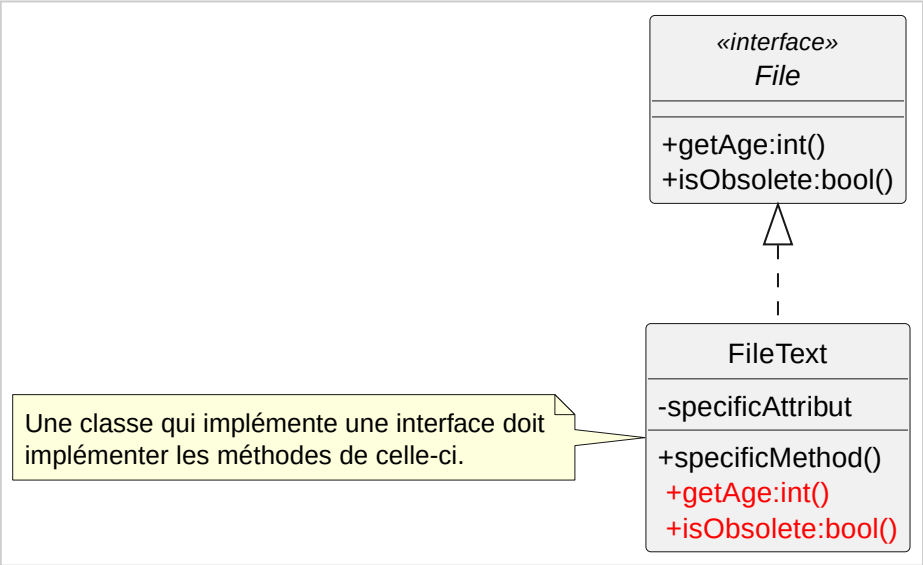
Le mieux est d'illustrer cette logique par un diagramme :



Il est difficile de distinguer une interface d'une classe abstraite (dans les deux cas, le nom est écrit en italique). Il est possible d'utiliser le stéréotype pour une meilleure lisibilité :

Nous avons dans ce diagramme une interface (il ne faut pas utiliser le terme de "classe" dans le cadre d'une interface) qui prévoit deux méthodes. Cela signifie que toute classe implémentant cette interface devra prévoir l'implémentation de ces 2 méthodes.

Ajoutons une classe `FileText` qui **implémente** l'interface `File` :



1.2. Implémentation d'une relation avec une interface

Une interface est déclarée (en PHP) avec le mot `interface` :

```
1 interface File ①
2 {
3     /**
4      * @return int retourne la taille du fichier en Mo
5      */
6     public function getSize():int; ②
7
8     /**
9      * @return int retourne l'âge du fichier en mois
10     */
11    public function getAge():int; ②
12
13 }
```

① Déclaration d'une interface

② Signature des méthodes qui devront être implémentées par les classes qui vont implémenter l'interface

Maintenant que notre interface est en place, nous pouvons indiquer à la classe `FileText` de l'implémenter :

```
1 class FileText implements File{ ①
2
```

```
3     //ici les membres spécifiques de FileText
4
5 }
```

① La classe **implémente** l'interface `File` .

Enfin, il ne reste plus qu'à implémenter les méthodes de l'interface :

```
1 class FileText implements File{ ①
2
3     //ici les membres spécifiques de FileText
4
5
6     public function getAge(): int
7     {
8         // ici du code qui retourne un nombre de mois
9     }
10
11    public function getSize(): int
12    {
13        // ici du code qui retourne un nombre de Mo
14    }
15 }
```

① Notre classe implémente toutes les méthodes de l'interface

Si une classe ne peut pas hériter de plusieurs classes mères, elle peut implémenter plusieurs interfaces :

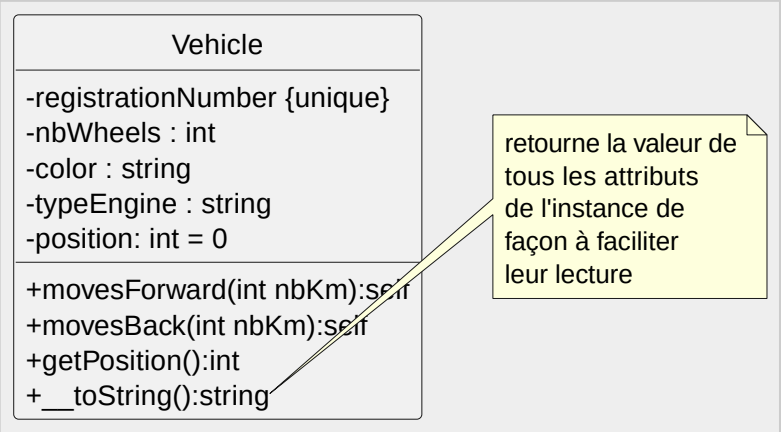
```
1 class classA implements classB, classC, classD {
2     //ici les membres spécifiques de classA
3
4     //ici toutes les méthodes déclarées dans les interfaces classB,
  classC et classD
5 }
```

2. Quelques exercices d'implémentation

2.1. Implémenter une simple classe

Q1) Travail à faire

- Implémentez la classe suivante :



La position du véhicule ne doit pas pouvoir être affectée arbitrairement. Elle doit découler de la position intiale et de ses déplacements.

- Testez votre implémentation en créant deux véhicules différents et en les faisant avancer et reculer différemment, puis affichez les informations de chaque véhicule en mobilisant la méthode `__toString`.

Correction de Q1

```php
<?php

//selon les recommandations PHP, lorsque la méthode __toString est utilisée, il faut implémenter l'interface Stringable.
class Vehicle implements Stringable
{

    //Déclaration des variables d'objet
    /**
     * @var string numéro d'immatriculation du véhicule
     */
    private string $registrationNumber;
    /**
     * @var int nombre de roues
     */
    private int $nbWheels;
    /**
```

```php
     * @var string couleur du véhicule
     */
    private string $color;
    /**
     * @var string type de moteur (essence, diesel, électrique, hydrogène,
...)
     */
    private string $typeEngine;
    /**
     * @var int Position du véhicule
     */
    private int $position = 0;

    //le constructeur aurait pu être omis puisqu'il n'exécute aucun code
    public function __construct()
    {
    }



    //region ****** Mutateurs et accesseurs ******

    /**
     * @return string
     */
    public function getRegistrationNumber(): string
    {
        return $this->registrationNumber;
    }

    /**
     * @return int
     */
    public function getNbWheels(): int
    {
        return $this->nbWheels;
    }

    /**
     * @return string
     */
    public function getColor(): string
    {
        return $this->color;
    }

    /**
     * @return string
     */
    public function getTypeEngine(): string
    {
```

```php
        return $this->typeEngine;
    }

    /**
     * @return int
     */
    public function getPosition(): int
    {
        return $this->position;
    }

    /**
     * @param string $registrationNumber
     *
     * @return Vehicle
     */
    public function setRegistrationNumber(string $registrationNumber): Vehicle
    {
        $this->registrationNumber = $registrationNumber;

        return $this;
    }

    /**
     * @param int $nbWheels
     *
     * @return Vehicule
     */
    public function setNbWheels(int $nbWheels): self
    {
        $this->nbWheels = $nbWheels;

        return $this;
    }

    /**
     * @param string $color
     *
     * @return Vehicule
     */
    public function setColor(string $color): self
    {
        $this->color = $color;

        return $this;
    }

    /**
     * @param string $typeEngine
     *
```

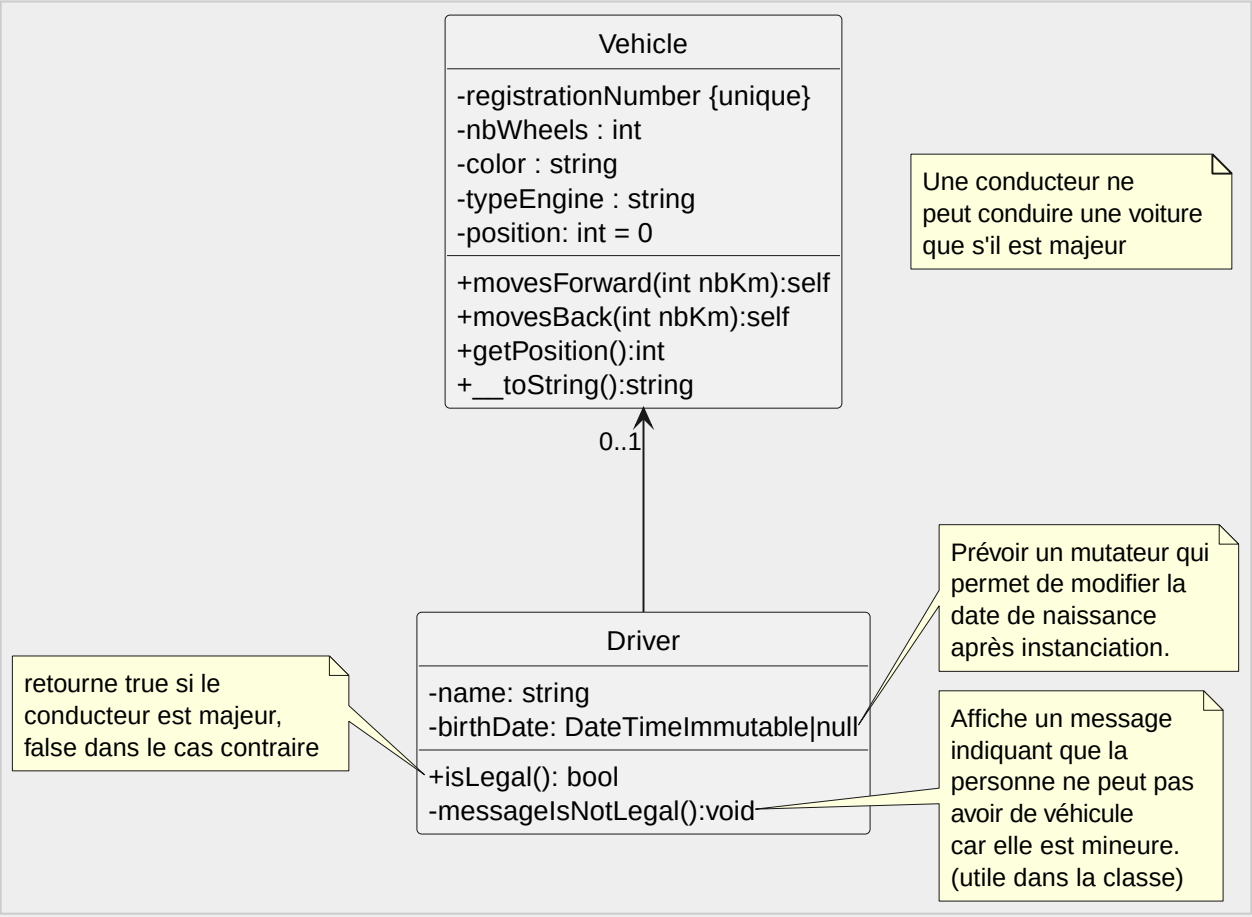
```php
     * @return Vehicule
     */
    public function setTypeEngine(string $typeEngine): self
    {
        $this->typeEngine = $typeEngine;

        return $this;
    }

    //endregion ****** Mutateurs et accesseurs ******


    //region ****** Autres opérations ******

    public function __toString(): string
    {
       return "Immatriculation : {$this->registrationNumber} ; Nombre de roues : {$this->nbWheels} ; couleur : {$this->color} ; type de moteur {$this->typeEngine} ; position : {$this->position}";
    }


    public function moveForward(int $nbKms): self
    {
        $this->position += $nbKms;

        return $this;
    }

    public function moveBack(int $nbKms): self
    {
        $this->position -= $nbKms;

        return $this;
    }

    //endregion ****** Autres opérations ******


}

$v1 = new Vehicle();
$v1->setRegistrationNumber('AAAA')->setNbWheels(4)->setColor('Jaune')->setTypeEngine('diesel');
$v1->moveForward(100)->moveForward(50)->moveBack(30);

$v2 = new Vehicle();
$v2->setRegistrationNumber('BBBB')->setNbWheels(2)->setColor('Rouge')->setTypeEngine('essence');
$v2->moveForward(230)->moveBack(50)->moveBack(10)->moveForward(140);

```

```php
//mobilisation de la méthode toString en affichant directement les objets avec echo (normalement, il n'est pas possible d'afficher un élément non scalaire (une chaîne, un entier) de cette façon)
echo "==== DEBUT DU TEST EXERCICE 1 ==== \n";
echo "Voiture 1 : $v1";
echo "\n";
echo "Voiture 1 : $v2";
echo "\n==== FIN DU TEST EXERCICE 1 ====";
```

2.2. Implémenter une association unidirectionnelle simple

Q2) Travail à faire

- Implémentez le diagramme suivant :



Si un véhicule est associé à un conducteur mineur, l'association ne doit pas se faire. Un message devra s'afficher indiquant que la personne est mineure et qu'il n'est pas possible d'associer un véhicule à un mineur. Attention à la situation qui consisterait à changer la date de naissance après que la personne se soit vue affecter un véhicule.

- Testez votre implémentation en essayant d'affecter un conducteur mineur à un premier

véhicule et un autre conducteur à un second véhicule qui avance et recule selon votre bon vouloir.

- Donnez via le code la position du véhicule du conducteur majeur depuis la variable qui référence celui-ci.

Correction de Q2

```php
<?php

//inclusion de la classe Vehicle
include_once 'exo-1-correction.php';

/**
 * Navigabilité unidirectionnelle de Driver vers Vehicle
 */
class Driver
{
    /**
     * @param string            $name      nom du conducteur
     * @param DateTimeImmutable $birthDate date de naissance du conducteur
     * @param Vehicle|null      $vehicle   instance du véhicule associé au
     *                                     conducteur
     */
    public function __construct(
        private string $name,
        private DateTimeImmutable $birthDate,
        private ?Vehicle $vehicle = null
    ) {

    }

    //region ****** mutateurs et accesseurs ******

    /**
     * @return string
     */
    public function getName(): string
    {
        return $this->name;
    }

    /**
     * @param string $name
     *
     * @return Driver
     */
    public function setName(string $name): self
    {
```

```php
        $this->name = $name;

        return $this;
    }

    /**
     * @return DateTimeImmutable
     */
    public function getBirthDate(): DateTimeImmutable
    {
        return $this->birthDate;
    }

    /**
     * @param DateTimeImmutable $birthDate
     *
     * @return Driver
     */
    public function setBirthDate(DateTimeImmutable $birthDate): self
    {
        $this->birthDate = $birthDate;

        //si jamais la personne change sa date de naissance pour une date qui la conduit à être mineure, alors il faut désassocier le véhicule
        if (!$this->isLegal()) {
            $this->messageIsNotLegal();
            $this->vehicle = null;
        }

        return $this;
    }

    /**
     * @return Vehicle|null
     */
    public function getVehicle(): ?Vehicle
    {
        return $this->vehicle;
    }

    /**
     * @param Vehicle|null $vehicle
     *
     * @return Driver
     */
    public function setVehicle(?Vehicle $vehicle): self
    {
        //si le conducteur est majeur, on peut lui affecter un véhicule, sinon on ne lui affecte pas
        if ($this->isLegal()) {
            $this->vehicle = $vehicle;
```

```
91          } else {
92              $this->messageIsNotLegal();
93              $this->vehicle = null;
94          }
95
96          return $this;
97      }
98
99      private function messageIsNotLegal(): void
100     {
101         echo "\n La personne est mineure, il n'est pas possible de lui
    associer un véhicule.\n";
102
103     }
104
105     //endregion ****** mutateurs et accesseurs ******
106
107     //region ****** Autres opérations ******
108
109     public function isLegal(): bool
110     {
111         //date courante
112         $now = new DateTimeImmutable(
113         ); //j'aurais pu prendre une date de type DateTime
114         //calcul de la différente entre la date du jour et la date de
    naissance
115         //documentation : https://www.php.net/manual/fr/datetime.diff.php
116         $interval = $this->birthDate->diff($now);
117         //cet interval est un objet, il faut retourner l'écart en nombre
    d'années
118         $years = (int)$interval->format('%r%y');
119
120         return $years > 18;
121
122     }
123
124     //endregion ****** Autres opérations ******
125 }
126
127 echo "\n\n==== DEBUT DU TEST EXERCICE 2 ==== \n";
128
129 $driverPaul = new Driver('Paul', new DateTimeImmutable('2017-04-12'));
130 $driverPaul->setVehicle($v1);
131 $driverJuliette = new Driver('Juliette', new DateTimeImmutable('2000-02-
    24'));
132 $driverJuliette->setVehicle($v1);
133 $driverJuliette->getVehicle()->moveForward(100)->moveBack(40);
134 //on donne la position du véhicule depuis l'objet "Juliette"
135 echo "\n La position du véhicule de {$driverJuliette->getName()} est
    {$driverJuliette->getVehicle()->getPosition()}";
```

```
136 echo "\n==== FIN DU TEST EXERCICE 2 ====";
```

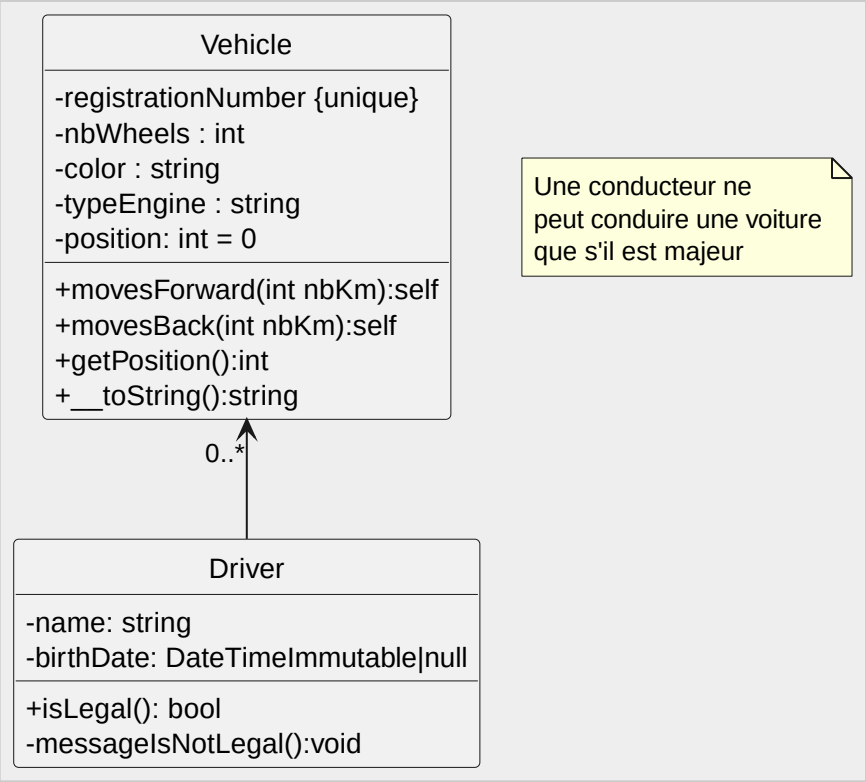
Sortie console :

```
1 ==== DEBUT DU TEST EXERCICE 2 ====
2
3  La personne est mineure, il n'est pas possible de lui associer un véhicule.
4
5  La position du véhicule de Juliette est 180
6 ==== FIN DU TEST EXERCICE 2 ====
```

2.3. Implémenter une association unidirectionnelle multiple avec cardinalité minimum à 0

Q3) Travail à faire

- Implémentez le diagramme suivant :



- Créez le conducteur Paul dont la date de naissance le conduit à être majeur
- Créez la conductrice Juliette dont la date de naissance la conduit à être mineure
- Créez les véhicules A, B, C, D et E avec les caractéristiques de votre choix
- Tentez d'affecter les véhicules A, C et E à Paul
- Tentez d'affecter les véhicules A et B à Juliette

- Faire avancer les véhicules A et C respectivement de 120km et 84km
- Faire reculer le véhicule C de 25km
- Afficher la liste des véhicules affectés à Paul
- Afficher la liste des véhicules affectés à Juliette
- Retirez les véhicules A et E à Paul
- Afficher la liste des véhicules restant à Paul

Correction de Q3

```php
<?php

//inclusion de la classe Vehicle
include_once 'exo-1-correction.php';

/**
 * Navigabilité unidirectionnelle de Driver vers Vehicle
 */
class Driver
{
    /**
     * @param string            $name      nom du conducteur
     * @param DateTimeImmutable $birthDate date de naissance du conducteur
     * @param arrray|Vehicle[]      $vehicles   collection d'objets de type Vehicle
     */
    public function __construct(
        private string $name,
        private DateTimeImmutable $birthDate,
        private array $vehicles = [] ①
    ) {

    }

    //region ****** mutateurs et accesseurs ******

    /**
     * @return string
     */
    public function getName(): string
    {
        return $this->name;
    }

    /**
     * @param string $name
     *
```

```php
     * @return Driver
     */
    public function setName(string $name): self
    {
        $this->name = $name;

        return $this;
    }

    /**
     * @return DateTimeImmutable
     */
    public function getBirthDate(): DateTimeImmutable
    {
        return $this->birthDate;
    }

    /**
     * @param DateTimeImmutable $birthDate
     *
     * @return Driver
     */
    public function setBirthDate(DateTimeImmutable $birthDate): self
    {
        $this->birthDate = $birthDate;

        //si jamais la personne change sa date de naissance pour une date qui la conduit à être mineure, alors il faut désassocier les véhicules
        if (!$this->isLegal()) {
            $this->messageIsNotLegal();
            $this->vehicle = [];
        }

        return $this;
    }

    /**
     * @param Vehicle $vehicle ajoute un item de type Vehicle à la collection
     */
    public function addVehicle(Vehicle $vehicle): bool
    {
        //tester si le conducteur est mineur
        if(!$this->isLegal()){
            $this->messageIsNotLegal();
            return false;
        }
        if (!in_array($vehicle, $this->vehicles, true)) {
            $this->vehicles[] = $vehicle;

            return true;
        }
```

```
87
88          return false;
89      }
90
91      /**
92       * @param Vehicle $vehicle retire l'item de la collection
93       */
94      public function removeVehicle(Vehicle $vehicle): bool
95      {
96          $key = array_search($vehicle, $this->vehicles, true);
97
98          if ($key !== false) {
99              unset($this->vehicles[$key]);
100
101             return true;
102         }
103
104         return false;
105     }
106
107     /**
108      * @return Vehicle[]
109      */
110     public function getVehicles(): array
111     {
112         return $this->vehicles;
113     }
114
115     private function messageIsNotLegal(): void
116     {
117         echo "\n La personne est mineure, il n'est pas possible de lui
    associer un véhicule.\n";
118
119     }
120
121     //endregion ****** mutateurs et accesseurs ******
122
123     //region ****** Autres opérations ******
124
125     public function isLegal(): bool
126     {
127         //date courante
128         $now = new DateTimeImmutable(
129         ); //j'aurais pu prendre une date de type DateTime
130         //calcul de la différente entre la date du jour et la date de
    naissance
131         //documentation : https://www.php.net/manual/fr/datetime.diff.php
132         $interval = $this->birthDate->diff($now); ②
133         //cet intervalle est un objet, il faut retourner l'écart en nombre
    d'années
134         $years = (int)$interval->format('%r%y'); ③
```

```php

        return $years > 18;

    }

    //endregion ****** Autres opérations ******
}

echo "\n\n==== DEBUT DU TEST EXERCICE 3 ==== \n";

//création des conducteurs
$driverPaul = new Driver('Paul', new DateTimeImmutable('2000-04-12'));
$driverJuliette = new Driver('Juliette', new DateTimeImmutable('2019-02-24'));

//création des véhicules
$vA = (new Vehicle())->setRegistrationNumber('AAAA');
$vB = (new Vehicle())->setRegistrationNumber('BBBB');
$vC = (new Vehicle())->setRegistrationNumber('CCCC');
$vD = (new Vehicle())->setRegistrationNumber('DDDD');
$vE = (new Vehicle())->setRegistrationNumber('EEEE');

//affectation des véhicules à Paul
$driverPaul->addVehicle($vA);
$driverPaul->addVehicle($vB);
$driverPaul->addVehicle($vC);
$driverPaul->addVehicle($vD);

//affectation des véhicules à Juliette
$driverJuliette->addVehicle($vA);
$driverJuliette->addVehicle($vB);

//déplacement des véhicules
$vA->moveForward(120);
$vC->moveForward(84);
$vC->moveBack(25);

//liste des véhicules conduits par Paul
echo "\n *** Liste des véhicules de {$driverPaul->getName()} ***";
foreach ($driverPaul->getVehicles() as $vehicle ){
    echo "\n-{$vehicle->getRegistrationNumber()}";
}

//liste des véhicules conduits par Juliette
echo "\n *** Liste des véhicules de {$driverJuliette->getName()} ***";
foreach ($driverJuliette->getVehicles() as $vehicle ){
    echo "\n-{$vehicle->getRegistrationNumber()}";
}

//suppression des véhicules A et E pour Paul
$driverPaul->removeVehicle($vA);
```

```php
$driverPaul->removeVehicle($vE);

//liste des véhicules restant à Paul
echo "\n *** Liste des véhicules restant à {$driverPaul->getName()} ***";
foreach ($driverPaul->getVehicles() as $vehicle ){
    echo "\n-{$vehicle->getRegistrationNumber()}";
}

echo "\n==== FIN DU TEST EXERCICE 3 ====";
```

① Ne pas oublier d'initialiser la collection

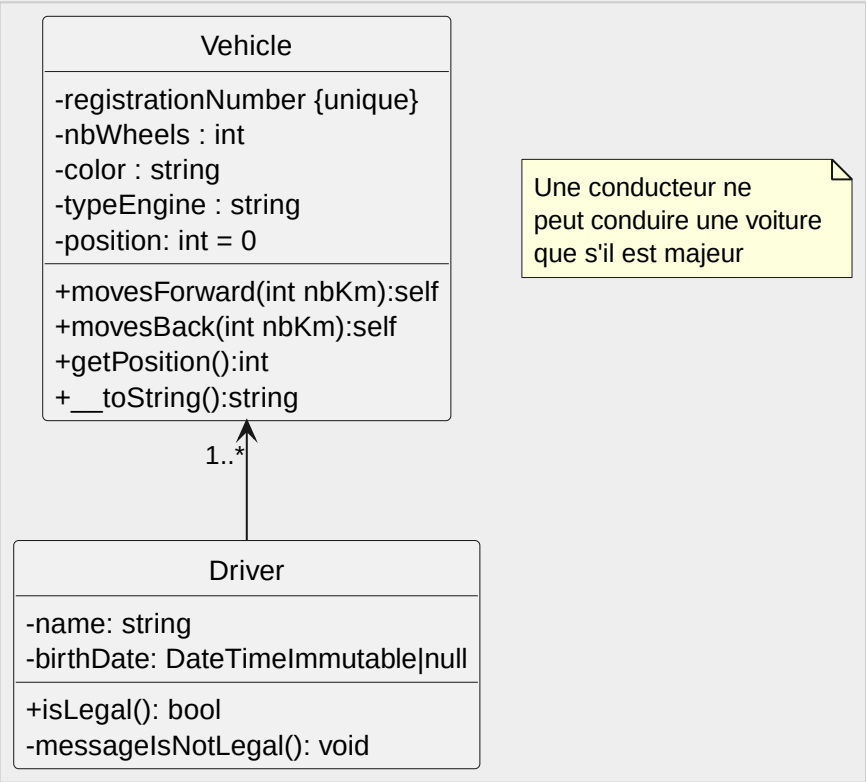
② La méthode `diff` sur un objet de type DateTimeInterface permet de calculer un écart entre deux dates.

③ Formate l'intervalle de façon à retourner le nombre d'années sous forme de chaîne (qui est ensuite casté pour être un entier)

2.4. Implémenter une association unidirectionnelle multiple avec cardinalité minimum à 1

Q4) Travail à faire

- Implémentez le diagramme suivant :



- Créez le conducteur Paul dont la date de naissance le conduit à être majeur. Il conduira le véhicule A.
- Affectez un véhicule supplémentaire B à Paul

- Créez un véhicule C (sans conducteur)
- Faire avancer les véhicules A et C respectivement de 123km et 257km
- Faire reculer le véhicule A de 70km
- Afficher depuis la variable référençant Paul la liste de ses véhicules et leur position.
- Retirez les véhicules A et **C** à Paul
- Afficher depuis la variable référençant Paul la liste de ses véhicules et leur position.

Correction de Q4

```php
<?php

//inclusion de la classe Vehicle
include_once 'exo-1-correction.php';

/**
 * Navigabilité unidirectionnelle de Driver vers Vehicle
 */
class Driver
{
    /**
     * @var array|Vehicle[] $vehicles collection d'objets de type Vehicle
     */
    private array $vehicles = []; //ne pas oublier d'initialiser la collection !

    /**
     * @param string            $name      nom du conducteur
     * @param DateTimeImmutable $birthDate date de naissance du conducteur
     * @param Vehicle      $vehicle   un objet de type Vehicle (ce n'est pas une variable d'objet !)
     */
    public function __construct(
        private string $name,
        private DateTimeImmutable $birthDate,
        Vehicle $vehicle ①
    ) {
        //conformément au diagramme, on associe un véhicule dès la création du conducteur
        $this->addVehicle($vehicle); ②
    }

    //region ****** mutateurs et accesseurs ******

    /**
     * @return string
     */
```

```php
    public function getName(): string
    {
        return $this->name;
    }

    /**
     * @param string $name
     *
     * @return Driver
     */
    public function setName(string $name): self
    {
        $this->name = $name;

        return $this;
    }

    /**
     * @return DateTimeImmutable
     */
    public function getBirthDate(): DateTimeImmutable
    {
        return $this->birthDate;
    }

    /**
     * @param DateTimeImmutable $birthDate
     *
     * @return Driver
     */
    public function setBirthDate(DateTimeImmutable $birthDate): self
    {
        $this->birthDate = $birthDate;

        //si jamais la personne change sa date de naissance pour une date qui la conduit à être mineure, alors il faut désassocier les véhicules
        if (!$this->isLegal()) {
            $this->messageIsNotLegal();
            $this->vehicle = [];
        }

        return $this;
    }

    /**
     * @param Vehicle $vehicle ajoute un item de type Vehicle à la collection
     */
    public function addVehicle(Vehicle $vehicle): bool
    {
        //tester si le conducteur est mineur
        if(!$this->isLegal()){
```

```
85              $this->messageIsNotLegal();
86              return false;
87          }
88          if (!in_array($vehicle, $this->vehicles, true)) {
89              $this->vehicles[] = $vehicle;
90
91              return true;
92          }
93
94          return false;
95      }
96
97      /**
98       * @param Vehicle $vehicle retire l'item de la collection
99       */
100     public function removeVehicle(Vehicle $vehicle): bool
101     {
102         //s'il ne reste plus qu'un véhicule, il ne faut pas le retirer
    conformément au diagramme
103         if(count($this->vehicles) === 1){ ③
104             return false;
105         }
106
107         $key = array_search($vehicle, $this->vehicles, true);
108
109         if ($key !== false) {
110             unset($this->vehicles[$key]);
111
112             return true;
113         }
114
115         return false;
116     }
117
118     /**
119      * @return Vehicle[]
120      */
121     public function getVehicles(): array
122     {
123         return $this->vehicles;
124     }
125
126     private function messageIsNotLegal(): void
127     {
128         echo "\n La personne est mineure, il n'est pas possible de lui
    associer un véhicule.\n";
129
130     }
131
132     //endregion ****** mutateurs et accesseurs ******
133
```

```php
    //region ****** Autres opérations ******

    public function isLegal(): bool
    {
        //date courante
        $now = new DateTimeImmutable(
        ); //j'aurais pu prendre une date de type DateTime
        //calcul de la différente entre la date du jour et la date de naissance
        //documentation : https://www.php.net/manual/fr/datetime.diff.php
        $interval = $this->birthDate->diff($now);
        //cet interval est un objet, il faut retourner l'écart en nombre d'années
        $years = (int)$interval->format('%r%y');

        return $years > 18;

    }

    //endregion ****** Autres opérations ******
}

echo "\n\n==== DEBUT DU TEST EXERCICE 4 ==== \n";

//Création du véhicule A
$vA = (new Vehicle())->setRegistrationNumber('AAAA');

//création de Paul
$driverPaul = new Driver('Paul', new DateTimeImmutable('2000-04-12'),$vA);

//Création du véhicule B
$vB = (new Vehicle())->setRegistrationNumber('BBBB');
//affectation du véhicule B à Paul
$driverPaul->addVehicle($vB);
//Création du véhicule C
$vC = (new Vehicle())->setRegistrationNumber('CCCC');
//circulation des véhicules A et C
$vA->moveForward(123);
$vC->moveForward(257);
$vA->moveBack(70);

//liste des véhicules conduits par Paul
echo "\n *** Liste des véhicules de {$driverPaul->getName()} ***";
foreach ($driverPaul->getVehicles() as $vehicle ){
    echo "\n-{$vehicle->getRegistrationNumber()}";
}

//suppression des véhicules A et C pour Paul
$driverPaul->removeVehicle($vA);
$driverPaul->removeVehicle($vC);

```

```php
//liste des véhicules conduits par Paul
echo "\n *** Liste des véhicules restant à {$driverPaul->getName()} ***";
foreach ($driverPaul->getVehicles() as $vehicle ){
    echo "\n-{$vehicle->getRegistrationNumber()}";
}

echo "\n==== FIN DU TEST EXERCICE 4 ====";
```

① à l'instanciation d'un `Driver`, il est attendu un véhicule afin de l'associer au conducteur dès sa création

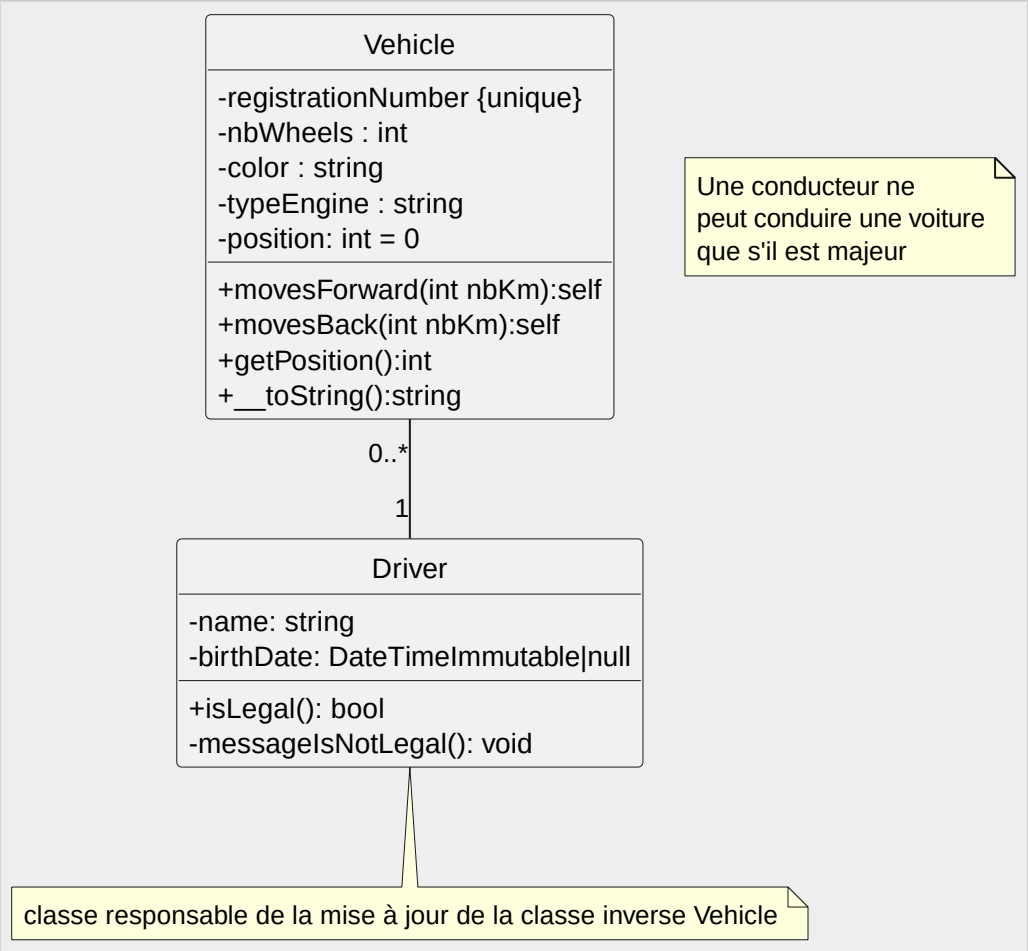
② on ajoute le premier véhicule au conducteur

③ lorsque l'on retire un véhicule à un conducteur, il faut s'assurer qu'il lui en reste forcément un avant de procéder à sa suppression

2.5. Implémenter une association bidirectionnelle one-to-many

Q5) Travail à faire

- Implémentez le diagramme suivant :



- Créez le conducteur Paul dont la date de naissance le conduit à être majeur.

- Créez le conducteur Juliette dont la date de naissance la conduit à être mineure.
- Créez un véhicule A et tenter de lui affecter Juliette.
- Créez un véhicule B et tenter de lui affecter Paul.
- Afficher la liste des véhicules associés à Paul.
- Afficher la liste des véhicules associés à Juliette.

Correction de Q5

- Voici les modifications à apporter aux classes `Vehicle` et `Driver` utilisée précédemment :

```
①
class Vehicle implements Stringable
{


    //ici déclaration des autres attributs d'objets
    /**
     * @param Driver $driver conducteur associé au véhicule
     */
    public function __construct(
        private Driver $driver ②
    )
    {
        // ATTENTION ATTENTION ATTENTION ATTENTION
        // ATTENTION ATTENTION ATTENTION ATTENTION

        //Même si vous utilisez la promotion des arguments, il faut passer par la méthode setDriver de façon à mettre à jour l'objet inverse (ici le conducteur)
        $this->setDriver($driver); ⑤
    }


    /**
     * @return Driver
     */
    public function getDriver(): Driver
    {
        return $this->driver;
    }

    /**
     * @param Driver $driver
     *
     * @return Vehicle
     */
    public function setDriver(Driver $driver): Vehicle
```

```
36      {
37          $this->driver = $driver;
38          //mise à jour de l'objet inverse (le conducteur)
39          $driver->addVehicle($this); ③
40
41          return $this;
42      }
43
44      // ici les autres opérations (mutateurs, setters, ...)
45
46  }
47
48  class Driver
49  {
50      /**
51       * @param string            $name      nom du conducteur
52       * @param DateTimeImmutable $birthDate date de naissance du conducteur
53       * @param array|Vehicle[]   $vehicles  collection d'objets de type
    Vehicle
54       */
55      public function __construct(
56          private string $name,
57          private DateTimeImmutable $birthDate,
58          private array $vehicles = []
59      ) {
60      }
61
62      //autres mutateurs, accesseurs et méthodes
63
64
65      /**
66       * @param Vehicle $vehicle ajoute un item de type Vehicle à la collection
67       */
68      public function addVehicle(Vehicle $vehicle): bool
69      {
70          //tester si le conducteur est mineur
71          if (!$this->isLegal()) {
72              $this->messageIsNotLegal();
73
74              return false;
75          }
76          if (!in_array($vehicle, $this->vehicles, true)) {
77              $this->vehicles[] = $vehicle;
78              ④
79              //pas de mise à jour de l'objet lié car c'est l'objet Vehicle qui
    est responsable de la navigabilité bidirectionnelle
80              return true;
81          }
82
83          return false;
84      }
```

```php

    /**
     * @param Vehicle $vehicle retire l'item de la collection
     */
    public function removeVehicle(Vehicle $vehicle): bool
    {

        $key = array_search($vehicle, $this->vehicles, true);

        if ($key !== false) {
            unset($this->vehicles[$key]);

            return true;
        }

        return false;
    }

    /**
     * @return Vehicle[]
     */
    public function getVehicles(): array
    {
        return $this->vehicles;
    }

}


echo "\n\n==== DEBUT DU TEST EXERCICE 5 ==== \n";

//création de Paul (majeur)
$driverPaul = new Driver('Paul', new DateTimeImmutable('2000-04-12'));

//création de Juliette (mineure)
$driverJuliette = new Driver('Juliette', new DateTimeImmutable('2019-02-24'));

//Tentative d'affectation d'un véhicule A à Juliette
$vA = (new Vehicle($driverJuliette))->setRegistrationNumber('AAAA');

//Tentative d'affectation d'un véhicule B à Paul
$vB = (new Vehicle($driverPaul))->setRegistrationNumber('BBBB');

//liste des véhicules conduits par Paul
echo "\n *** Liste des véhicules de {$driverPaul->getName()} ***";
foreach ($driverPaul->getVehicles() as $vehicle) {
    echo "\n-{$vehicle->getRegistrationNumber()}";
}

//liste des véhicules conduits par Juliette
```

```php
echo "\n *** Liste des véhicules de {$driverJuliette->getName()} ***";
foreach ($driverJuliette->getVehicles() as $vehicle) {
    echo "\n-{$vehicle->getRegistrationNumber()}";
}

echo "\n==== FIN DU TEST EXERCICE 5 ====";
```

① La classe véhicule est obligatoirement la classe responsable de la mise à jour de la classe inverse puisqu'elle doit être associée à un Driver dès son instanciation

② Attribut qui va contenir l'objet lié

③ Mise à jour de l'objet lié (navigabilité de `Driver` vers Vehicle)

④ pas de mise à jour de l'objet lié car c'est l'objet Vehicle qui est responsable de la navigabilité bidirectionnelle

⑤ Même si vous utilisez la promotion des arguments pour initialiser l'attribut `Vehicle::driver`, il faut passer par la méthode `setDriver` de façon à mettre à jour l'objet inverse (ici le conducteur)

Index