



EXPERT
EXPOSÉ

2^e édition

Design Patterns en PHP

Les 23 modèles de conception :
descriptions et solutions illustrées
en **UML 2** et **PHP**

En téléchargement



exemples

Version en ligne
OFFERTE !
pendant 1 an

Laurent DEBRAUWER
Yannick EVAIN
Sébastien FERRANDEZ

eni

Les éléments à télécharger sont disponibles à l'adresse suivante :
<http://www.editions-eni.fr>
Saisissez la référence ENI de l'ouvrage **EI2PHDES** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

Avant-propos

Partie 1 : Introduction

Chapitre 1

Introduction aux design patterns

| | |
|---|----|
| 1. Principes de base du concepteur objet | 15 |
| 1.1 Les principes SOLID | 15 |
| 1.1.1 Le principe de responsabilité unique | 16 |
| 1.1.2 Ouvert à l'extension mais fermé à la modification | 18 |
| 1.1.3 Le principe de substitution de Liskov | 19 |
| 1.1.4 Séparation des interfaces | 20 |
| 1.1.5 L'inversion des dépendances | 21 |
| 2. Design patterns ou modèles de conception | 22 |
| 3. Description des design patterns | 23 |
| 4. Catalogue des design patterns | 24 |
| 5. Organisation du catalogue des design patterns | 26 |
| 6. Aspects spécifiques des exemples de code PHP | 27 |

2 _____ Design Patterns en PHP

Les 23 modèles de conception

Chapitre 2

Une étude de cas : la vente en ligne de véhicules

| | |
|--|----|
| 1. Description du système | 29 |
| 2. Cahier des charges | 29 |
| 3. Prise en compte des design patterns | 31 |

Partie 2 : Design patterns de construction

Chapitre 3

Introduction aux design patterns de construction

| | |
|--|----|
| 1. Présentation | 33 |
| 2. Problèmes liés à la création d'objets | 34 |
| 2.1 Problématique | 34 |
| 2.2 Solutions proposées par les design patterns de construction | 35 |

Chapitre 4

Le design pattern Abstract Factory

| | |
|---------------------------------|----|
| 1. Description | 37 |
| 2. Exemple | 37 |
| 3. Structure | 40 |
| 3.1 Diagramme de classes | 40 |
| 3.2 Participants | 41 |
| 3.3 Collaborations | 41 |
| 4. Domaines d'utilisation | 42 |
| 5. Exemple en PHP | 42 |

Chapitre 5
Le design pattern Builder

- 1. Description 49
- 2. Exemple..... 49
- 3. Structure 51
 - 3.1 Diagramme de classes..... 51
 - 3.2 Participants 52
 - 3.3 Collaborations..... 52
- 4. Domaines d'utilisation..... 53
- 5. Exemple en PHP 54

Chapitre 6
Le design pattern Factory Method

- 1. Description 59
- 2. Exemple..... 59
- 3. Structure 61
 - 3.1 Diagramme de classes..... 61
 - 3.2 Participants 62
 - 3.3 Collaborations..... 62
- 4. Domaines d'utilisation..... 62
- 5. Exemple en PHP 63

4 _____ Design Patterns en PHP

Les 23 modèles de conception

Chapitre 7

Le design pattern Prototype

| | |
|-------------------------------------|----|
| 1. Description | 67 |
| 2. Exemple | 67 |
| 3. Structure | 70 |
| 3.1 Diagramme de classes | 70 |
| 3.2 Participants | 71 |
| 3.3 Collaboration | 71 |
| 4. Domaines d'utilisation | 71 |
| 5. Exemple en PHP | 72 |

Chapitre 8

Le design pattern Singleton

| | |
|------------------------------------|----|
| 1. Description | 79 |
| 2. Exemple | 79 |
| 3. Structure | 80 |
| 3.1 Diagramme de classes | 80 |
| 3.2 Participants | 80 |
| 3.3 Collaboration | 81 |
| 4. Domaine d'utilisation | 81 |
| 5. Exemples en PHP | 81 |
| 5.1 La liasse vierge | 81 |
| 5.2 La classe Vendeur | 83 |

Partie 3 : Design patterns de structuration

Chapitre 9

Introduction aux design patterns de structuration

- 1. Présentation 85
- 2. Composition statique et dynamique..... 86

Chapitre 10

Le design pattern Adapter

- 1. Description 89
- 2. Exemple..... 89
- 3. Structure 91
 - 3.1 Diagramme de classes..... 91
 - 3.2 Participants..... 91
 - 3.3 Collaborations..... 92
- 4. Domaines d'application 92
- 5. Exemple en PHP 93

Chapitre 11

Le design pattern Bridge

- 1. Description 97
- 2. Exemple..... 97
- 3. Structure 100
 - 3.1 Diagramme de classes..... 100
 - 3.2 Participants..... 101
 - 3.3 Collaborations..... 101
- 4. Domaines d'application 102
- 5. Exemple en PHP 102

6 _____ Design Patterns en PHP

Les 23 modèles de conception

Chapitre 12

Le design pattern Composite

| | |
|-------------------------------------|-----|
| 1. Description | 109 |
| 2. Exemple | 109 |
| 3. Structure | 112 |
| 3.1 Diagramme de classes | 112 |
| 3.2 Participants | 112 |
| 3.3 Collaborations | 113 |
| 4. Domaines d'application | 114 |
| 5. Exemple en PHP | 115 |

Chapitre 13

Le design pattern Decorator

| | |
|-------------------------------------|-----|
| 1. Description | 119 |
| 2. Exemple | 119 |
| 3. Structure | 124 |
| 3.1 Diagramme de classes | 124 |
| 3.2 Participants | 125 |
| 3.3 Collaborations | 125 |
| 4. Domaines d'application | 125 |
| 5. Exemple en PHP | 126 |

Chapitre 14

Le design pattern Facade

| | |
|--------------------------|-----|
| 1. Description | 129 |
| 2. Exemple | 129 |

- 3. Structure 132
 - 3.1 Diagramme de classes..... 132
 - 3.2 Participants..... 133
 - 3.3 Collaborations..... 133
- 4. Domaines d'application 134
- 5. Exemple en PHP 135

Chapitre 15
Le design pattern Flyweight

- 1. Description 141
- 2. Exemple..... 141
- 3. Structure 144
 - 3.1 Diagramme de classes..... 144
 - 3.2 Participants..... 145
 - 3.3 Collaborations..... 145
- 4. Domaine d'application..... 145
- 5. Exemple en PHP 146

Chapitre 16
Le design pattern Proxy

- 1. Description 151
- 2. Exemple..... 151
- 3. Structure 155
 - 3.1 Diagramme de classes..... 155
 - 3.2 Participants..... 156
 - 3.3 Collaborations..... 156
- 4. Domaines d'application..... 156
- 5. Exemple en PHP 157

8 _____ Design Patterns en PHP

Les 23 modèles de conception

Partie 4 : Design Patterns de comportement

Chapitre 17

Introduction aux design patterns de comportement

- 1. Présentation 161
- 2. Distribution par héritage ou par délégation 162

Chapitre 18

Le design pattern Chain of Responsibility

- 1. Description 165
- 2. Exemple 165
- 3. Structure 169
 - 3.1 Diagramme de classes 169
 - 3.2 Participants 170
 - 3.3 Collaborations 170
- 4. Domaines d'application 170
- 5. Exemple en PHP 171

Chapitre 19

Le design pattern Command

- 1. Description 177
- 2. Exemple 177
- 3. Structure 181
 - 3.1 Diagramme de classes 181
 - 3.2 Participants 182
 - 3.3 Collaborations 182
- 4. Domaines d'application 184
- 5. Exemple en PHP 184

Chapitre 20
Le design pattern Interpreter

- 1. Description 193
- 2. Exemple..... 193
- 3. Structure 196
 - 3.1 Diagramme de classes..... 196
 - 3.2 Participants 197
 - 3.3 Collaborations..... 197
- 4. Domaines d'application 198
- 5. Exemple en PHP 198

Chapitre 21
Le design pattern Iterator

- 1. Description 205
- 2. Exemple..... 205
- 3. Structure 207
 - 3.1 Diagramme de classes..... 207
 - 3.2 Participants 207
 - 3.3 Collaborations..... 208
- 4. Domaines d'application 208
- 5. Exemple en PHP 208

Chapitre 22
Le design pattern Mediator

- 1. Description 213
- 2. Exemple..... 213

10 _____ Design Patterns en PHP

Les 23 modèles de conception

| | |
|---------------------------------|-----|
| 3. Structure | 217 |
| 3.1 Diagramme de classes | 217 |
| 3.2 Participants | 217 |
| 3.3 Collaborations | 218 |
| 4. Domaines d'application | 218 |
| 5. Exemple en PHP | 218 |

Chapitre 23

Le design pattern Memento

| | |
|---------------------------------|-----|
| 1. Description | 225 |
| 2. Exemple | 225 |
| 3. Structure | 228 |
| 3.1 Diagramme de classes | 228 |
| 3.2 Participants | 228 |
| 3.3 Collaborations | 229 |
| 4. Domaines d'application | 229 |
| 5. Exemple en PHP | 229 |

Chapitre 24

Le design pattern Observer

| | |
|---------------------------------|-----|
| 1. Description | 237 |
| 2. Exemple | 237 |
| 3. Structure | 240 |
| 3.1 Diagramme de classes | 240 |
| 3.2 Participants | 241 |
| 3.3 Collaborations | 241 |
| 4. Domaines d'application | 241 |
| 5. Exemple en PHP | 242 |

Chapitre 25
Le design pattern State

- 1. Description 247
- 2. Exemple..... 247
- 3. Structure 251
 - 3.1 Diagramme de classes..... 251
 - 3.2 Participants 251
 - 3.3 Collaborations..... 252
- 4. Domaines d'application 252
- 5. Exemple en PHP 252

Chapitre 26
Le design pattern Strategy

- 1. Description 261
- 2. Exemple..... 262
- 3. Structure 264
 - 3.1 Diagramme de classes..... 264
 - 3.2 Participants 264
 - 3.3 Collaborations..... 265
- 4. Domaines d'application 265
- 5. Exemple en PHP 266

Chapitre 27
Le design pattern Template Method

- 1. Description 271
- 2. Exemple..... 271

12 _____ Design Patterns en PHP

Les 23 modèles de conception

| | |
|---------------------------------|-----|
| 3. Structure | 276 |
| 3.1 Diagramme de classes | 276 |
| 3.2 Participants | 276 |
| 3.3 Collaborations | 277 |
| 4. Domaines d'application | 277 |
| 5. Exemple en PHP | 277 |

Chapitre 28

Le design pattern Visitor

| | |
|---------------------------------|-----|
| 1. Description | 281 |
| 2. Exemple | 281 |
| 3. Structure | 285 |
| 3.1 Diagramme de classes | 285 |
| 3.2 Participants | 286 |
| 3.3 Collaborations | 286 |
| 4. Domaines d'application | 287 |
| 5. Exemple en PHP | 287 |

Partie 5 : Application des design patterns

Chapitre 29

Compositions et variations de design patterns

| | |
|--|-----|
| 1. Préliminaire | 293 |
| 2. Le design pattern Pluggable Factory | 294 |
| 2.1 Introduction | 294 |
| 2.2 Structure | 299 |
| 2.3 Exemple en PHP | 300 |

- 3. Le design pattern Reflective Visitor 308
 - 3.1 Discussion 308
 - 3.2 Structure 312
 - 3.3 Exemple en PHP 314
- 4. Le design pattern Multicast 321
 - 4.1 Description et exemple 321
 - 4.2 Structure 324
 - 4.3 Exemple en PHP 325
 - 4.4 Discussion : comparaison avec le design pattern Observer ... 332

Chapitre 30
Le design pattern composite MVC

- 1. Introduction au problème 333
- 2. Le design pattern composite MVC 334
- 3. Exemple en PHP 341
 - 3.1 Introduction 341
 - 3.2 Architecture 343
 - 3.3 Étude du code 344

Chapitre 31
Les design patterns dans la conception de logiciels

- 1. Modélisation et conception avec les design patterns 363
- 2. Autres apports des design patterns 366
 - 2.1 Un référentiel commun 366
 - 2.2 Un ensemble récurrent de techniques de conception 366
 - 2.3 Un outil pédagogique de l'approche orientée objet 366
- 3. Des pratiques bien répandues 367

Annexe Exercices

| | |
|--|---------|
| 1. Énoncés des exercices | 369 |
| 1.1 Création de cartes de paiement | 369 |
| 1.1.1 Création en fonction du client | 369 |
| 1.1.2 Création à l'aide d'une fabrique. | 370 |
| 1.2 Autorisation des cartes de paiement | 370 |
| 1.3 Système de fichiers | 370 |
| 1.4 Browser graphique d'objets | 371 |
| 1.5 États de la vie professionnelle d'une personne | 372 |
| 1.6 Cache d'un dictionnaire persistant d'objets | 372 |
| 2. Correction des exercices | 375 |
| 2.1 Création de cartes de paiement | 375 |
| 2.1.1 Création en fonction du client | 375 |
| 2.1.2 Création à l'aide d'une fabrique. | 376 |
| 2.2 Autorisation des cartes de paiement | 377 |
| 2.3 Système de fichiers | 378 |
| 2.4 Browser graphique d'objets | 385 |
| 2.5 États de la vie professionnelle d'une personne | 386 |
| 2.6 Cache d'un dictionnaire persistant d'objets | 387 |
| Index | 389 |

Chapitre 4

Le design pattern Abstract Factory

1. Description

Le but du design pattern `Abstract Factory` est la création d'objets regroupés en familles sans avoir à connaître leurs classes concrètes.

2. Exemple

Le système de vente de véhicules gère des véhicules fonctionnant à l'essence et des véhicules fonctionnant à l'électricité. Cette gestion est confiée à l'objet `Catalogue`, à qui incombe la responsabilité de créer de tels objets.

Pour chaque produit, nous disposons d'une classe abstraite, d'une sous-classe concrète décrivant la version du produit fonctionnant à l'essence et d'une sous-classe concrète décrivant la version du produit fonctionnant à l'électricité. Par exemple, à la figure 4.1, pour l'objet `scooter`, il existe une classe abstraite `Scooter` et deux sous-classes concrètes : `ScooterElectricite` et `ScooterEssence`.

L'objet `Catalogue` peut utiliser ces sous-classes concrètes pour instancier les produits. Cependant, si par la suite de nouvelles familles de véhicules doivent être prises en compte (diesel ou hybride essence-électricité), les modifications à apporter à l'objet `Catalogue` peuvent s'avérer assez fastidieuses.

Le design pattern `Abstract Factory` résout ce problème en introduisant une interface `FabriqueVehiculeInterface` qui contient la signature des méthodes à utiliser pour créer chaque produit. Le type de retour de ces méthodes est constitué par l'une des classes abstraites de produit. Ainsi, l'objet `Catalogue` n'a pas besoin de connaître les sous-classes concrètes et reste parfaitement indépendant des familles de produits. En révélant l'interface de nos fabriques et non leur implémentation, nous découplons le code client des produits concrets : notre objet client `Catalogue` demande des produits à la fabrique qu'on lui passe en paramètre lors de sa construction sans avoir la moindre idée de qui elle est ni de ce qui se passe en coulisses pour qu'il obtienne le bon produit.

Une classe implémentant `FabriqueVehiculeInterface` est créée pour chaque famille de produits, à savoir les classes `FabriqueVehiculeElectricite` et `FabriqueVehiculeEssence`. Une telle classe a la responsabilité d'implémenter les opérations de création du véhicule appropriée pour la famille à laquelle elle est associée.

L'objet client `Catalogue` prend alors pour paramètre un objet se conformant à l'interface `FabriqueVehiculeInterface`, c'est-à-dire soit une instance de `FabriqueVehiculeElectricite`, soit une instance de `FabriqueVehiculeEssence`. Avec une telle instance, le catalogue peut créer et manipuler des véhicules sans devoir connaître les familles de véhicules et les classes concrètes correspondantes.

L'ensemble des classes du design pattern Abstract Factory pour cet exemple est détaillé à la figure 4.1.

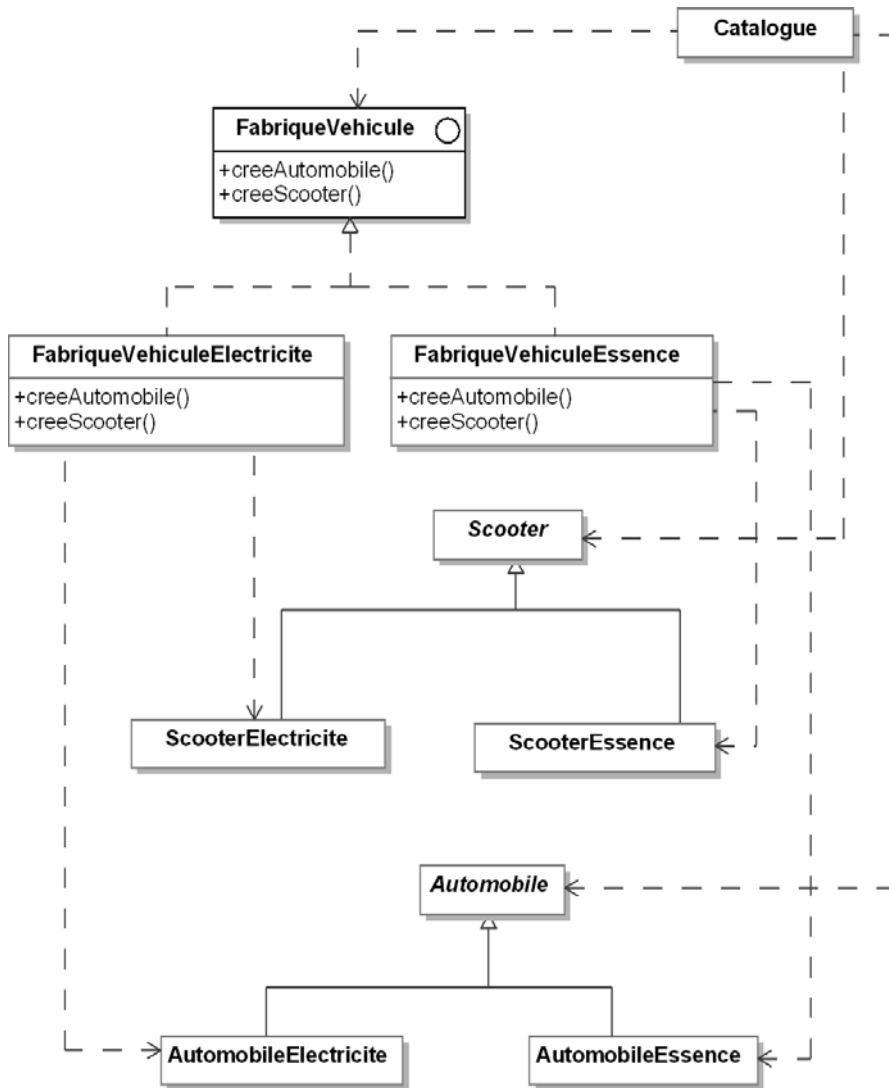


Figure 4.1 - Le design pattern Abstract Factory appliqué à des familles de véhicules

3. Structure

3.1 Diagramme de classes

La figure 4.2 détaille la structure générique du design pattern Abstract Factory.

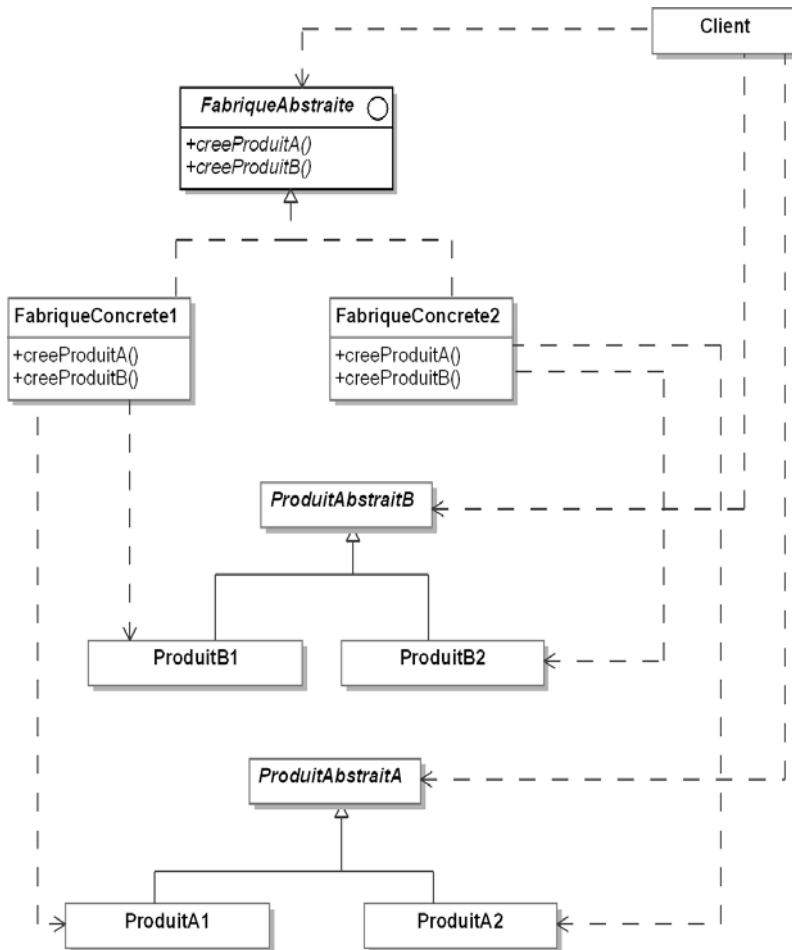


Figure 4.2 - Structure du design pattern Abstract Factory

3.2 Participants

Les participants au design pattern Abstract Factory sont les suivants :

- FabriqueAbstraite (FabriqueVehiculeInterface) est une interface spécifiant les signatures des méthodes créant les différents produits.
- FabriqueConcretel, FabriqueConcrete2 (FabriqueVehicule-
Electricite, FabriqueVehiculeEssence) sont les classes concrètes implémentant les méthodes créant les produits pour chaque famille de produits. Connaissant la famille et le produit, elles sont capables de créer une instance du produit pour cette famille.
- ProduitAbstraitA et ProduitAbstraitB (AbstractScooter et AbstractAutomobile) sont les classes abstraites des produits indépendamment de leur famille. Les familles sont introduites dans leurs sous-classes concrètes.
- Client (Catalogue) est la classe qui utilise l'interface FabriqueAbstraite.

3.3 Collaborations

La classe Catalogue utilise une instance de l'une des fabriques concrètes pour créer ses produits au travers de l'interface exposée par FabriqueAbstraiteInterface.

■ Remarque

Il est recommandé de ne créer qu'une seule instance des fabriques concrètes, celle-ci pouvant être partagée par plusieurs clients. Nous verrons plus tard un design pattern capable de garantir qu'une seule instance d'une classe est disponible à l'exécution : Singleton.

4. Domaines d'utilisation

Le design pattern Abstract Factory est utilisé dans les domaines suivants :

- Un système utilisant des produits a besoin d'être indépendant de la façon dont ces produits sont créés et regroupés.
- Un système est paramétré par plusieurs familles de produits qui peuvent évoluer.

5. Exemple en PHP

Voici maintenant un exemple d'utilisation du design pattern écrit en PHP. Le code PHP correspondant à la classe abstraite AbstractAutomobile et ses sous-classes est donné à la suite. Il est très simple, il décrit les quatre propriétés des automobiles ainsi que la méthode afficheCaracteristiques qui permet de les afficher.

```
<?php
declare(strict_types=1);

namespace ENI\DesignPatterns\AbstractFactory;

abstract class AbstractAutomobile
{
    protected string $marque;

    protected string $couleur;

    protected int $puissance;

    protected float $espace;

    public function __construct(string $marque, string $couleur,
int $puissance, float $espace)
    {
        $this->marque = $marque;
        $this->couleur = $couleur;
        $this->puissance = $puissance;
    }
}
```

```
        $this->espace = $espace;
    }

    abstract public function afficheCaracteristiques(): void;
}

<?php
declare(strict_types=1);

namespace ENI\DesignPatterns\AbstractFactory;

class AutomobileElectricite extends AbstractAutomobile
{
    public function afficheCaracteristiques(): void
    {
        echo "Automobile électrique - marque: $this->marque"
            . ", couleur: $this->couleur"
            . ", puissance: $this->puissance"
            . ", espace: $this->espace" . PHP_EOL;
    }
}

<?php
declare(strict_types=1);

namespace ENI\DesignPatterns\AbstractFactory;

class AutomobileEssence extends AbstractAutomobile
{
    public function afficheCaracteristiques(): void
    {
        echo "Automobile à essence - marque: $this->marque"
            . ", couleur: $this->couleur"
            . ", puissance: $this->puissance"
            . ", espace: $this->espace" . PHP_EOL;
    }
}
}
```

Le code PHP correspondant à la classe abstraite `AbstractScooter` et ses sous-classes est donné à la suite. Il est similaire à celui des automobiles, à