Les classes et l'instanciation d'objet

Table des matières

I. Contexte

Durée : 1 h

Prérequis : avoir étudié le cours jusqu'ici

Environnement de travail : Replit

Contexte

Comme vous le savez, les objets sont en POO des entités caractérisées par des propriétés. Les propriétés d'un objet peuvent être définies sur une valeur, sur une référence vers un autre objet, ou bien sur des fonctions, nous parlons alors de méthodes. La Programmation Orientée Objet permet d'aborder la programmation sous un angle différent. En effet, elle permet d'organiser les codes d'une manière plus accessible et plus proche des concepts de la réalité. Elle permet par ailleurs de centraliser les modifications de codes qui impacteront toutes les instances effectuées.

Pour créer une instance d'un objet, il nous faut appeler un constructeur, une méthode qui construit un objet d'un certain type. Il est d'ailleurs assez simple d'appeler les constructeurs de types natifs comme **Number**, **Array**, etc. Mais là où la Programmation Orientée Objet va prendre tout son sens, c'est avec le concept de **classes**. En effet, les classes vont nous permettre de créer nos propres types d'objets, et de les instancier.

Ainsi, en apprenant à coder des classes, nous serons à même de construire nos propres objets personnalisés, et de centraliser les instructions définissant toutes les instances dans les classes. La compréhension et la maîtrise des classes en JS va donc nous permettre de gravir des échelons en POO.

Ce cours sera constitué de nombreux exemples de codes, d'exercices pratiques et de vidéos, tous réalisés via l'interface Replit. Vous pourrez donc les tester librement, et apprendre à définir vos propres classes en POO. Dans la première partie, nous parlerons du concept de classe, et de la définition d'une classe avec un constructeur. Puis dans la seconde partie, nous parlerons de la définition de méthodes.

II. Définition et instanciation d'une classe

A. Définition et instanciation d'une classe

Classe

Une classe, c'est un peu comme un plan qui permet de construire autant d'objets que nécessaire. Par exemple, une classe peut être comme un plan d'une maison. Si on applique ce plan plusieurs fois, on peut construire autant de maisons que nécessaire grâce à celui-ci. C'est exactement le même principe que pour une classe. Une classe est un espace de code qui définit toutes les propriétés d'un type, et donc des objets créés à partir de celle-ci.

Méthode **Définir une classe**

Maintenant, comment définir une classe ? La convention veut qu'une classe commence toujours par une majuscule. Pour définir une classe, on utilise donc cette syntaxe :

```
class NomDeLaClasse {
  //définition des propriétés
}
```

Pour bien comprendre, tout au long de ce cours, nous allons définir une classe permettant de construire des objets de type **Ordinateur**. Notre classe va donc définir les propriétés des ordinateurs. Donc, voici notre classe :

```
class Ordinateur {
  //définition des propriétés
}
```

Cette classe va nous permettre de définir le type **Ordinateur**.

Différents types de propriétés

Comme vous le savez, il existe plusieurs types de propriétés.

Les **propriétés statiques** sont des propriétés que l'on appelle directement via le nom de la classe et non via une instance.

Les **propriétés de classe** sont quant à elles des propriétés définies dans la classe hors du constructeur, et partagées par toutes les instances de la classe. Elles sont donc accessibles via les instances de la classe, et par défaut, sont définies sur la même valeur, quelle que soit l'instance.

Les **propriétés d'instance** sont en revanche des propriétés qui sont définies dans le constructeur de la classe, et qui auront une valeur propre pour chaque instance.

Méthode **Définir des propriétés statiques**

Pour créer des propriétés statiques, nous pouvons simplement les définir avec le mot clé **static** dans notre classe. Voyons-le directement en exemple :

```
class Ordinateur {

  //définition de propriétés statiques

  static OBJETS_PRIS_EN_CHARGE = "ordinateurs";
}
```

Nous définissons une propriété statique, associée à la classe **Ordinateur**. Nous l'écrivons en majuscule pour être plus conventionnel, car c'est une propriété constante. Pour accéder à sa valeur, nous pouvons procéder ainsi :

```
class Ordinateur {

  //définition de propriétés statiques

  static OBJETS_PRIS_EN_CHARGE = "ordinateurs";
}

console.log(Ordinateur. OBJETS_PRIS_EN_CHARGE); //ordinateurs
```

On peut voir qu'on accède à la propriété statique via le nom de la classe, il n'y a pas besoin de l'instancier. Voyons maintenant comment créer des propriétés de classe.

Méthode **Définir des propriétés de classe**

Pour définir une propriété de classe, il faut simplement définir une propriété sur une valeur dans la classe, hors de la définition du constructeur. Pour l'instant, nous n'avons pas défini de constructeur, donc pas de problème, nous pouvons les définir ou nous voulons dans la classe.

Créons par exemple deux propriétés de classe :

```
class Ordinateur {

  //définition de propriétés statiques

  static OBJETS_PRIS_EN_CHARGE = "ordinateurs";

  //définition de propriétés de classe

  type = "laptop";
  clavier = "azerty";
}
```

Dans cet exemple, toutes les instances d'**Ordinateur** hériteront des propriétés **type** et **clavier**. Ce qui veut dire que par défaut, tous les ordinateurs auront une propriété **type** définie sur « *laptop* » et une propriété **clavier** définie sur à « *azerty* ». Ce sont des propriétés de classe. Elles seront accessibles via les instances de la classe **Ordinateur** et non via la classe elle-même.

Constructeur

Le constructeur va nous permettre d'instancier notre classe, donc de construire de vrais objets à partir de notre classe. Le constructeur est une méthode, qui va par ailleurs permettre de définir des propriétés d'instances, c'est-à-dire des propriétés dont la valeur sera spécifique à chaque instance. La valeur de ses propriétés pourra être passée comme argument du constructeur lors de son appel. Si nous ne définissons pas de constructeur dans la classe, un constructeur par défaut sera créé et appelé par JavaScript, qui ne passera aucun paramètre.

Méthode **Définir un constructeur et des propriétés d'instance**

Pour définir un constructeur, il faut utiliser le mot clé **constructor** et initialiser les paramètres qui stockeront les valeurs que prendront les propriétés d'instances.

```
class Ordinateur {

  //déclaration des propriétés qui seront définies comme propriétés d'instance dans le constructeur (facultatif mais bonne pratique)

  marque;
  ram;
  stockage;

  //définition de propriétés statiques

  static OBJETS_PRIS_EN_CHARGE = "ordinateurs";

  //définition de propriétés de classe

  type = "laptop";
  clavier = "azerty";

  //définition du constructeur

  constructor(marque, ram, stockage) {
    this.marque = marque;
    this.ram = ram;
    this.stockage = stockage;
  }
}
```

Nous définissons donc notre constructeur et nous initialisons 3 paramètres : **marque**, **ram**, et **stockage**. Puis nous définissons 3 propriétés d'instance en utilisant le mot clé **this**, qui permet de faire référence à l'instance et non à la classe. Nous définissons donc la propriété d'instance **this.marque** sur la valeur du paramètre **marque**, la propriété d'instance **this.ram** sur la valeur du paramètre **ram** et enfin la propriété d'instance **this.stockage** sur la valeur du paramètre **stockage**.

Méthode **Instancier la classe**

Nous pouvons maintenant instancier notre classe en faisant appel au constructeur. Pour cela, nous allons lors de l'appel passer comme argument les valeurs correspondant à la marque, la ram et le stockage. Pour appeler le constructeur, nous devons utiliser le mot clé **new** suivi de l'appel du constructeur.

```
class Ordinateur {

  //déclaration des propriétés qui seront définies comme propriétés d'instance dans le constructeur (facultatif mais bonne pratique)

  marque;
  ram;
  stockage;

  //définition de propriétés statiques

  static OBJETS_PRIS_EN_CHARGE = "ordinateurs";

  //définition de propriétés de classe

  type = "laptop";
  clavier = "azerty";

  //définition du constructeur

  constructor(marque, ram, stockage) {
    this.marque = marque;
    this.ram = ram;
    this.stockage = stockage;
  }
}

let monOrdi = new Ordinateur("Apple", 8, 512);
let monOrdi2 = new Ordinateur("HP", 16, 256);
```

Dans cet exemple, nous instancions deux fois **Ordinateur**, et nous définissons les arguments qui seront les valeurs de chaque attribut d'instance. On peut voir que chaque ordinateur accèdera aux propriétés de classes, et aura des propriétés d'instance définies sur des valeurs propres à chacune. D'ailleurs, nous pouvons afficher des propriétés de classe et d'instance de chaque objet :

```
class Ordinateur {

  //déclaration des propriétés qui seront définies comme propriétés d'instance dans le constructeur (facultatif mais bonne pratique)

  marque;
  ram;
  stockage;

  //définition de propriétés statiques

  static OBJETS_PRIS_EN_CHARGE = "ordinateurs";

  //définition de propriétés de classe

  type = "laptop";
  clavier = "azerty";

  //définition du constructeur

  constructor(marque, ram, stockage) {
    this.marque = marque;
    this.ram = ram;
    this.stockage = stockage;
```

```
  }
}

let monOrdi = new Ordinateur("Apple", 8, 512);
let monOrdi2 = new Ordinateur("HP", 16, 256);

console.log(monOrdi.marque); //Apple

console.log(monOrdi2.marque); //HP

console.log(monOrdi.type) ; //"laptop"

console.log(monOrdi2.type) ; //"laptop"
```

Nous avons donc défini des propriétés de nos objets grâce à la classe **Ordinateur**. Les classes permettent donc de définir un type, dans ce cas, c'est le type **Ordinateur** qui est défini. Pour rappel, chaque type d'objets que nous définissons est un sous-type du type « *racine* » **Object**. **Ordinateur** est donc un sous-type de **Object**.

B. Exercice

Question 1

[solution n°1 p.15]

Créez une classe **Personne** avec les propriétés d'instance **nom**, **prenom** et **age**, afin que l'instanciation fonctionne.

```
//définition de la classe

const personne1 = new Personne("Peter", "Parker", 20);
console.log(personne1);
```

Question 2

[solution n°2 p.15]

Créez une classe **Voiture** avec les propriétés **marque**, **modele** et **annee** et la propriété statique **roues = 4**, afin que l'instanciation fonctionne.

```
//définition de la classe

const voiture1 = new Voiture('Renault', 'Clio', 2020);
console.log(voiture1.marque + " à " + Voiture.ROUES + " roues");
```

Question 3

[solution n°3 p.15]

Créez une classe **Rectangle** avec les propriétés **longueur** et **largeur** afin que l'instanciation fonctionne.

```
//définition de la classe

const rectangle1 = new Rectangle(10, 5);
console.log(rectangle1.largeur);
```

Question 4

[solution n°4 p.15]

Créez un objet de type **Telephone** ayant pour **marque** « *Apple* », pour **modele** « *Iphone 11* », et pour **ram** 4.

```
class Telephone {
  constructor(marque, modele, ram) {
    this.marque = marque;
    this.modele = modele;
    this.ram = ram;
  }
}

const t1 = //instance

console.log(t1.ram);
```

Question 5 [solution n°5 p.16]

Corrigez l'erreur.

```
class Car {
  constructor(marque) {
    this.marque = marque;
  }
}

const car = Car("Ford");
console.log(car.marque);
```

III. Définir des méthodes dans une classe

A. Définir des méthodes dans une classe

Maintenant que nous savons définir une classe, nous allons pouvoir définir des méthodes personnalisées. Pour cela, nous allons voir en premier lieu comment définir des méthodes statiques, c'est-à-dire des méthodes accessibles via le nom de la classe directement.

Méthode **Définir une méthode statique**

En JavaScript, définir une méthode statique revient à définir une propriété statique, mais sur une fonction. Pour cela, nous allons utiliser le mot clé **static**. Plutôt que d'utiliser la syntaxe **maMethode = function() {}**, nous pouvons utiliser le raccourci **maMethode() {}**. Nous n'avons donc pas besoin d'utiliser le mot **function**.

Par exemple, définissons une méthode qui permet d'afficher simplement une chaîne contenant une explication du concept d'ordinateur :

```
class Ordinateur {

  //définition de propriétés statiques

  static OBJETS_PRIS_EN_CHARGE = "ordinateurs";

  //définition de propriétés de classe

  type = "laptop";
  clavier = "azerty";

  constructor(marque, ram, stockage) {
    this.marque = marque;
    this.ram = ram;
    this.stockage = stockage;
  }

  //définition de méthodes statiques

  static definition() {
    console.log("Les ordinateurs sont des machines automatisées permettant le traitement d'informations.")
  }

}

let monOrdi = new Ordinateur("Apple", 8, 512);
let monOrdi2 = new Ordinateur("HP", 16, 256);

Ordinateur.definition(); //appel de la méthode statique
```

Maintenant que nous avons vu un exemple de définition de méthode statique, voyons comment définir des méthodes d'instance.

Méthode **Définir des méthodes d'instance**

Pour créer une méthode d'instance, il nous faut simplement définir cette méthode sans utiliser le mot clé **static**. Nous aurons besoin d'utiliser le mot clé **this** pour faire référence à l'instance.

Pour voir un exemple, définissons une méthode permettant d'afficher dans une chaîne les caractéristiques d'un ordinateur (donc d'une instance d'**Ordinateur**).

```
class Ordinateur {

  //définition de propriétés statiques

  static OBJETS_PRIS_EN_CHARGE = "ordinateurs";

  //définition de propriétés de classe

  type = "laptop";
  clavier = "azerty";
  //définition du constructeur

  constructor(marque, ram, stockage) {
    this.marque = marque;
    this.ram = ram;
    this.stockage = stockage;
  }

  //définition de méthodes statiques

  static definition () {
    console.log("Les ordinateurs sont des machines automatisées permettant le traitement d'informations.");
  }

  //définition de méthodes d'instance

  ficheProduit() {
    console.log("Marque : " + this.marque + "\n" + "Mémoire vive : " + this.ram + "\n" + "Stockage : " + this.stockage);
  }

}

let monOrdi = new Ordinateur("Apple", 8, 512);
let monOrdi2 = new Ordinateur("HP", 16, 256);

monOrdi.ficheProduit();

monOrdi2.ficheProduit();
```

Nous pouvons voir que ça fonctionne, nous avons défini notre méthode d'instance **ficheProduit**. Maintenant, définissons une méthode permettant de calculer un prix de notre ordinateur. Cette méthode réalisera le calcul suivant : . C'est un exemple vraiment fictif.

```
class Ordinateur {

//définition de propriétés statiques

  static OBJETS_PRIS_EN_CHARGE = "ordinateurs";

  //définition de propriétés de classe

  type = "laptop";
  clavier = "azerty";

  //définition du constructeur

  constructor(marque, ram, stockage) {
    this.marque = marque;
    this.ram = ram;
    this.stockage = stockage;
  }

  //définition de méthodes statiques

  static definition() {
    console.log("Les ordinateurs sont des machines automatisées permettant le traitement d'informations.");
  }

  //définition de méthodes d'instance

  ficheProduit() {
    console.log("Marque : " + this.marque + "\n" + "Mémoire vive : " + this.ram + "\n" + "Stockage : " + this.stockage);
  }

  calcPrice() {
    return this.ram * 100 + this.stockage * 2;
  }
}

let monOrdi = new Ordinateur("Apple", 8, 512);
let monOrdi2 = new Ordinateur("HP", 16, 256);

Ordinateur.definition();

console.log(monOrdi.calcPrice());

console.log(monOrdi2.calcPrice());
```

Puis définissons une méthode qui par exemple permettra d'augmenter la ram d'un ordinateur :

```
class Ordinateur {

//définition de propriétés statiques

  static OBJETS_PRIS_EN_CHARGE = "ordinateurs";

  //définition de propriétés de classe

  type = "laptop";
  clavier = "azerty";
```

```
11
12  //définition du constructeur
13
14  constructor(marque, ram, stockage) {
15    this.marque = marque;
16    this.ram = ram;
17    this.stockage = stockage;
18  }
19
20  //définition de méthodes statiques
21
22  static definition() {
23    console.log("Les ordinateurs sont des machines automatisées permettant le traitement
    d'informations.");
24  }
25
26  //définition de méthodes d'instance
27
28  ficheProduit() {
29    console.log("Marque : " + this.marque + "\n" + "Mémoire vive : " + this.ram + "\n" +
    "Stockage : " + this.stockage);
30  }
31
32  calcPrice() {
33    return this.ram * 100 + this.stockage * 2;
34  }
35
36  addRam(value) {
37    return this.ram += value;
38  }
39 }
40
41 let monOrdi = new Ordinateur("Apple", 8, 512);
42 let monOrdi2 = new Ordinateur("HP", 16, 256);
43
44 Ordinateur.definition();
45
46 console.log(monOrdi.addRam(8));
```

Voilà, nous pouvons tester le code et appeler les méthodes pour voir qu'elles fonctionnent bien. Nous venons de voir pratiquement comment définir des méthodes personnalisées dans une classe, pour que toutes les instances en héritent.

Getters et Setters

Maintenant que nous savons définir des méthodes dans une classe, parlons de méthodes fondamentales, les getters et les setters, et voyons comment les définir via une vidéo.

B. Exercice

Question 1

[solution n°6 p.16]

Définissez une classe **Rectangle** avec 2 propriétés d'instance (**longueur** et **largeur**) et une méthode pour calculer son périmètre.

```
//definition de la classe

const monRectangle = new Rectangle(10, 8);
console.log(monRectangle.calculerPerimetre());
```

Question 2

[solution n°7 p.16]

Dans la classe **Cercle**, définissez une méthode permettant de renvoyer l'aire du cercle :

```
class Cercle {
  constructor(rayon) {
    this.rayon = rayon;
  }

  //définition de la méthode
  }
}

cercle = new Cercle(4);

console.log(cercle.aire());
```

Question 3

[solution n°8 p.17]

Dans la même classe **Cercle**, définissez une méthode permettant de renvoyer le périmètre du cercle :

```
 class Cercle {
  constructor(rayon) {
    this.rayon = rayon;
  }

  aire() {
    return Math.PI * Math.pow(this.rayon, 2)
  }

  //définition de la méthode
}

cercle = new Cercle(4);

console.log(cercle.perimetre());
```

Question 4

[solution n°9 p.17]

Définissez un getter **marque** pour la propriété **_marque**.

```
class Produit {
  constructor(marque) {
    this._marque = marque;
  }

  //définition du getter
  }
}

produit = new Produit("HP");

console.log(produit.marque);
```

Question 5

[solution n°10 p.17]

Définissez un setter pour la propriété **_marque**.

```
class Produit {
  constructor(marque) {
    this._marque = marque;
  }

  get marque() {
    return this._marque;
  }

  //définition du setter
}

produit = new Produit("HP");

produit.marque = "Acer" ;

console.log(produit.marque);
```

IV. Essentiel

Dans ce cours, nous avons parlé de la définition de classes et de l'instanciation d'objets en JavaScript. Une classe est comme un plan qui permet de définir les propriétés d'un type. Elle définit donc des propriétés statiques, des propriétés de classe et des propriétés d'instance. Cela prend en compte des propriétés définies sur des valeurs, sur des références d'objets, et des méthodes. Nous avons aussi vu comment définir le constructeur d'une classe, et comment créer des propriétés d'instances définies sur des arguments passés lors de l'appel du constructeur.

Dans la deuxième partie, nous avons compris comment définir des méthodes dans une classe, en distinguant les méthodes statiques et les méthodes d'instances. Nous avons également introduit les concepts de getters et de setters pour contrôler l'accès aux propriétés d'une classe.

Les classes sont un concept clé en programmation orientée objet et sont largement utilisées dans JavaScript et d'autres langages de programmation. Elles permettent d'organiser le code en blocs logiques et réutilisables et facilitent la création de nouveaux objets à partir d'une même structure. Les méthodes ajoutent une dimension fonctionnelle à la classe, permettant aux objets de réaliser des actions spécifiques. Les getters et setters permettent de contrôler l'accès aux propriétés de la classe, offrant un moyen plus sécurisé de manipuler les données.

En résumé, ce cours a posé les bases pour comprendre les classes et l'instanciation d'objet en JavaScript, une compétence fondamentale pour tout développeur JavaScript. Le concept de classe vous est maintenant bien moins étranger.

V. Auto-évaluation

A. Exercice

Vous êtes un vendeur d'écouteurs audio et vous cherchez à mettre au point un système de e-commerce pour vendre vos produits. Les écouteurs que vous vendez ont tous des propriétés communes :

- boite = true;
- nbEcouteurs = 2;

En revanche, chaque produit a des propriétés spécifiques :

- marque; (chaîne)
- sansFil; (booléen)
- price; (nombre)

Question 1

[solution n°11 p.18]

Créez une classe définissant le type **Ecouteur**. Définissez une méthode d'instance **fiche()** permettant grâce à une boucle d'afficher les différentes propriétés (clé et valeur) de l'instance d'**Ecouteur**. Instanciez 2 fois **Ecouteur** pour créer deux objets différents de type **Ecouteur**, et appelez pour chaque instance la méthode **fiche()**.

Question 2

[solution n°12 p.19]

Créez une classe **Panier** permettant de définir des objets de type **Panier**. Cette classe ne définira pas d'attributs d'instance, nous n'avons donc pas besoin de définir le constructeur, un constructeur par défaut sera créé et appelé automatiquement lors de l'instanciation. Définissez une propriété statique **contenu** qui sera définie sur un tableau. Définissez trois méthodes statiques :

- Une permettant d'ajouter un écouteur au tableau (la référence de l'objet)
- Une permettant de supprimer un écouteur au tableau (la référence de l'objet)
- Une permettant d'afficher la marque de chaque produit présent dans le panier

Appelez chaque méthode

B. Test

Exercice 1 : Quiz

[solution n°13 p.20]

Question 1

Qu'est-ce qu'une classe ?

- ○ Une entité qui définit les propriétés d'un type
- ○ Une méthode qui permet de manipuler des données
- ○ Une structure de contrôle de flux

Question 2

Comment définit-on une classe en JavaScript en respectant la convention ?

- ○ Avec le mot clé **class**, suivi du nom de la classe écrit tout en minuscule
- ○ Avec le mot clé **class**, suivi du nom de la classe commençant par une majuscule
- ○ Avec le mot clé **function**, suivi du nom de la classe commençant par une majuscule

Question 3

Quel mot clé permet de créer une instance d'une classe en JS ?

- ○ new
- ○ function
- ○ static

Question 4

Qu'est-ce qu'un getter dans une classe ?

- ○ Une méthode qui permet de récupérer la valeur d'une propriété d'instance
- ○ Une méthode qui permet de définir la valeur d'une propriété d'instance
- ○ Une méthode qui permet de définir une propriété statique

Solutions des exercices

p. 7 Solution n°1

Sortie attendue : 20

```
class Personne {
  constructor(nom, prenom, age) {
    this.nom = nom;
    this.prenom = prenom;
    this.age = age;
  }
}
const personne1 = new Personne("Peter", "Parker", 20);
console.log(personne1.age);
```

p. 7 Solution n°2

Sortie attendue : "*Renault à 4 roues*"

```
class Voiture {

  static ROUES = 4;

  constructor(marque, modele, annee) {
    this.marque = marque;
    this.modele = modele;
    this.annee = annee;
  }
}

const voiture1 = new Voiture('Renault', 'Clio', 2020);
console.log(voiture1.marque + " à " + Voiture.ROUES + " roues");
```

p. 7 Solution n°3

Sortie attendue : 5

```
class Rectangle {
  constructor(longueur, largeur) {
    this.longueur = longueur;
    this.largeur = largeur;
  }
}

const rectangle1 = new Rectangle(10, 5);

console.log(rectangle1.largeur);
```

p. 7 Solution n°4

Sortie attendue : 4

```
class Telephone {
  constructor(marque, modele, ram) {
    this.marque = marque;
    this.modele = modele;
    this.ram = ram;
  }
}

const t1 = new Telephone("Apple", "Iphone 11", 4);

console.log(t1.ram);
```

p. 8

Solution n°5

Sortie attendue : "*Ford*"

```
class Car {
  constructor(marque) {
    this.marque = marque;
  }
}

const car = new Car("Ford");
console.log(car.marque);
```

Explications : il ne faut pas oublier le mot new pour instancier la classe.

p. 12

Solution n°6

Sortie attendue : 36

```
class Rectangle {
  constructor(longueur, largeur) {
    this.longueur = longueur;
    this.largeur = largeur;
  }
  perimetre() {
    return 2 * (this.longueur + this.largeur);
  }
}
const monRectangle = new Rectangle(10, 8);
console.log(monRectangle.perimetre());
```

p. 12

Solution n°7

Sortie attendue : 50.26548245743669

```
class Cercle {
  constructor(rayon) {
    this.rayon = rayon;
  }

  aire() {
    return (Math.PI) * (this.rayon * this.rayon);
```

```
  }
}

cercle = new Cercle(4);

console.log(cercle.aire());
```

p. 12 **Solution n°8**

Sortie attendue : 25.132741228718345

```
class Cercle {
  constructor(rayon) {
    this.rayon = rayon;
  }

  aire() {
    return Math.PI * Math.pow(this.rayon, 2);
  }

  perimetre() {
    return (Math.PI * 2) * this.rayon;
  }
}

cercle = new Cercle(4);

console.log(cercle.perimetre());
```

p. 12 **Solution n°9**

Sortie attendue : « *HP* »

```
class Produit {
  constructor(marque) {
    this._marque = marque;
  }

  get marque() {
    return this._marque;
  }
}

produit = new Produit("HP");

console.log(produit.marque);
```

p. 13 **Solution n°10**

Sortie attendue : « *Acer* »

```
class Produit {
  constructor(marque) {
    this._marque = marque;
  }

  get marque() {
    return this._marque;
  }

  set marque(value) {
    this._marque = value;
  }
}

produit = new Produit("HP");

produit.marque = "Acer" ;

console.log(produit.marque);
```

p. 14

Solution n°11

Voici une classe qui correspond à ces caractéristiques :

```
class Ecouteur {

  //propriétés de classe

  boite = true;
  nbEcouteurs = 2;

  //constructeur et propriétés d'instance

  constructor(marque, sansFil, price) {
    this.marque = marque;
    this.sansFil = sansFil;
    this.prix = price;
  }

  fiche() {
    for(let property in this) {
      console.log(property + " : " + this[property]);
    }
  }
}

const ecouteurs1 = new Ecouteur("JBL", true, 50);

ecouteurs1.fiche();

console.log ("\n")

const ecouteurs2 = new Ecouteur("Bose", "false", 20);

ecouteurs2.fiche();
```

p. 14 **Solution n°12**

Voici un code qui fonctionne :

```
class Ecouteur {

  //propriétés de classe

  boite = true;
  nbEcouteurs = 2;

  //constructeur et propriétés d'instance

  constructor(marque, sansFil, price) {
    this.marque = marque;
    this.sansFil = sansFil;
    this.prix = price;
  }

  fiche() {
    for(let property in this) {
      console.log(property + " : " + this[property]);
    }
  }
}

const ecouteurs1 = new Ecouteur("JBL", true, 50);
const ecouteurs2 = new Ecouteur("Bose", "false", 20);

class Panier {

  //propriété statique

  static contenu = new Array();

  //méthodes statiques

  static addToPanier(obj) {
    this.contenu.push(obj);
  }
static rmToPanier(obj) {
    this.contenu.splice(this.contenu.indexOf(obj), 1);
  } //on utilise la méthode splice pour retire 1 item à partir de l'index de l'item ayant pour valeur obj

  static affPanier() {
    for(let produit of this.contenu) {
      console.log (produit.marque);
    }
  }
}

Panier.addToPanier(ecouteurs1); //On ajoute l'objet ecouteurs1 du panier

Panier.addToPanier(ecouteurs2); //On ajoute l'objet ecouteurs2 du panier

Panier.rmToPanier(ecouteurs1);  //On retire écouteur1 du panier

```

```
54 Panier.affPanier(); //On affiche la marque de tous les objets présents dans le panier
```

Exercice p. 14 **Solution n°13**

Question 1

Qu'est-ce qu'une classe ?

- ◉ Une entité qui définit les propriétés d'un type
- ○ Une méthode qui permet de manipuler des données
- ○ Une structure de contrôle de flux

Une classe est une entité qui définit un type d'objets, donc qui définit ses propriétés.

Question 2

Comment définit-on une classe en JavaScript en respectant la convention ?

- ○ Avec le mot clé **class**, suivi du nom de la classe écrit tout en minuscule
- ◉ Avec le mot clé **class**, suivi du nom de la classe commençant par une majuscule
- ○ Avec le mot clé **function**, suivi du nom de la classe commençant par une majuscule

On définit une classe en JavaScript avec le mot clé class, suivi du nom de la classe qui commence par une majuscule.

Question 3

Quel mot clé permet de créer une instance d'une classe en JS ?

- ◉ new
- ○ function
- ○ static

On crée une nouvelle instance d'une classe en JavaScript en utilisant le mot clé new suivi du nom de la classe.

Question 4

Qu'est-ce qu'un getter dans une classe ?

- ◉ Une méthode qui permet de récupérer la valeur d'une propriété d'instance
- ○ Une méthode qui permet de définir la valeur d'une propriété d'instance
- ○ Une méthode qui permet de définir une propriété statique

Un getter est une méthode qui permet de récupérer la valeur d'une propriété d'instance spécifique.