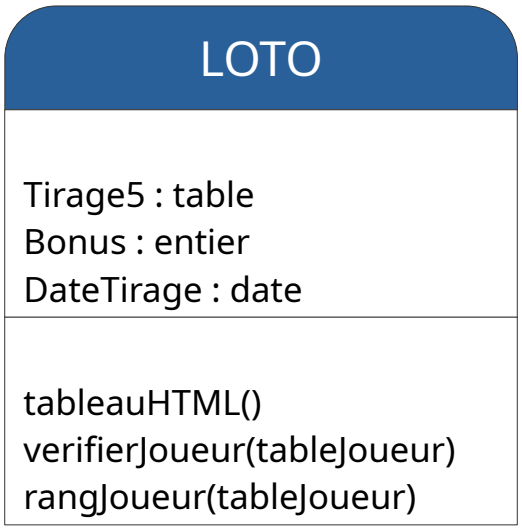
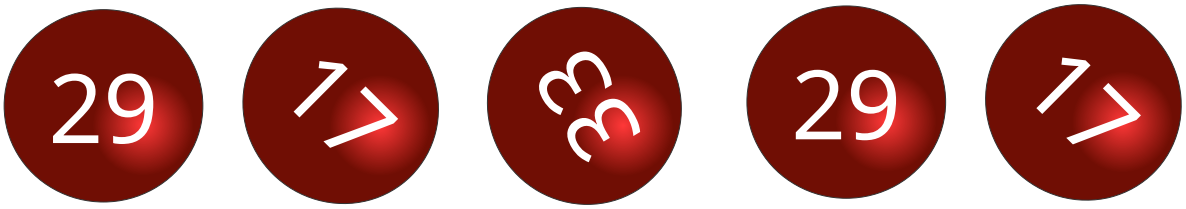
LA POO

Adaptation au projet Loto





date	commentaire
14/12/2019	Création
30/12/2019	Ajout des schémas UML

SOMMAIRE

1 Introduction....3
1.1 Objectifs....3
1.2 Prérequis....3
1.3 Notions abordées....3
2 Problématiques....4
2.1 Rappel des besoins....4
2.2 Évolutions complémentaires....5
2.2.1 Enregistrement de l'historique des tirages....5
2.2.2 Plusieurs tirages par semaine....5
2.2.3 Grilles de plusieurs joueurs....5
2.2.4 Complication ?....6
3 Solution orientée objet....7
3.1 Introduction à l'objet....7
3.2 Concept objet en informatique....8
3.2.1 Attributs ou propriétés....8
3.2.2 Méthodes....8
3.2.3 Classe....9
3.2.4 Instance et instanciation....10
3.2.5 Constructeur....10
4 En résumé....11
4.1 Classe Loto....12
4.1.1 Création de la classe Loto....12
4.1.2 Utilité de la classe Loto....13
4.1.3 Synoptique de la classe LOTO....14

1 INTRODUCTION

Dans le projet de simulation du loto national, nous avons vu que l'utilisation de plusieurs variables sont nécessaires, ainsi que plusieurs fonctions.

Nous allons étudier ce que la programmation orientée objet peut apporter à ce projet.

1.1 OBJECTIFS

Ce cours doit permettre de comprendre l'intérêt de la programmation orientée objet :

1. Par l'inventaire des inconvénients de la programmation actuelle
2. Par la définition de nouvelles notions
3. Par la mise en pratique de ces notions

1.2 PRÉREQUIS

Avoir lu et suivi le cours JavaScript SI4 et avoir fait le Projet 07 jeu du loto.

Pour partir d'un état commun, le code source sera fourni.

1.3 NOTIONS ABORDÉES

Les notions suivantes sont importantes :

- classe et objets
- attribut (ou propriété)
- méthode

2 PROBLÉMATIQUES

L'idée pour améliorer ce code, est de revoir les besoins de l'application et de déterminer ce qui rend le programme complexe.

Ensuite, il faut observer les évolutions et indiquer les complexités complémentaires.

Enfin, il faut découvrir les notions qui permettront de résoudre une partie des problèmes.

2.1 RAPPEL DES BESOINS

Le contexte est le tirage du loto .

Rappelez ici les demandes du projet :

Les complexités associées sont :

1. L'usage de fonctions multiples
2. L'usage de boucles
3. L'imbrication des boucles (il y a deux tableaux à comparer)
4. Le tirage multiple
5. L'utilisation de plusieurs variables pour valider un tirage de rang un

2.2 ÉVOLUTIONS COMPLÉMENTAIRES

Il faut maintenant envisager que le projet évolue.

Que va-t-il se passer, si nous souhaitons enregistrer l'historique de plusieurs tirages ?

Comment rendre possible d'avoir deux tirages par semaine (mercredi et samedi) ?

Comment enregistrer la grille de plusieurs joueurs ?

Comment changer les règles (diminuer ou augmenter le nombre de boules tirées) ?

Que faire si un autre script utilise un nom identique pour une de ses variables ou fonctions ?

2.2.1 Enregistrement de l'historique des tirages

Que modifieriez-vous dans le code pour permettre l'enregistrement de tous les tirages.

2.2.2 Plusieurs tirages par semaine

Que faut-il faire pour tester 2 tirages par semaine ?

2.2.3 Grilles de plusieurs joueurs

Comment mémoriser les grilles et les noms des joueurs. Comment vérifier pour un tirage, si chaque joueur a gagné ?

2.2.4 Complication ?

En réalité, améliorer ce programme devrait vous sembler assez lourd et complexe.

Pouvez-vous identifier les raisons ?

3 SOLUTION ORIENTÉE OBJET

La Programmation Orientée Objet (POO) permet de simplifier la programmation, en utilisant le concept d'objet.

3.1 INTRODUCTION À L'OBJET

Dans la vie réelle, la phrase "**j'ai vu un camion passer en grillant un feu rouge**" est perçu par tout le monde. Mais voyons ce que chacun perçoit avec 3 personnages :

Personnage	camion	grillant	Feu rouge
Einstein			
Vous			
Une IA ou Un robot			

Si le sens peut être différent pour l'IA ou le robot, notre cerveau humain est capable de manipuler le concept d'objet, sans le voir : votre esprit est capable de reconstituer le sens de la phrase avec vos propres concepts.

Ainsi, un objet (comme le camion) dispose d'attributs communs dans les trois cas : grande taille, massif, des roues, un espace de chargement, etc.

3.2 CONCEPT OBJET EN INFORMATIQUE

En informatique, un objet est un concept modélisable par des attributs et des méthodes.

3.2.1 Attributs ou propriétés

Les **attributs ou propriétés sont des états** et correspondent aux éléments permettant de décrire ce qu'objet à en commun avec un autre objet de même classe.

- Exemple d'attributs : taille, poids, couleur, nom, quantité

Pour un véhicule de classe "camion", circulant sur les routes Européennes, il y a quelques attributs importants, que l'on peut regrouper dans un cadre :

Camion
immatriculation NombreEssieux CapacitéChargement (tonne) Puissance (chevaux DIN) hauteur (mètre) largeur (mètre) PTAC (tonne)

3.2.2 Méthodes

Les **méthodes sont des actions** possibles sur ou avec l'objet. Il s'agit souvent d'un verbe.

- Exemple de méthodes : rouler, démarrer, arrêter, atteler, dételer...

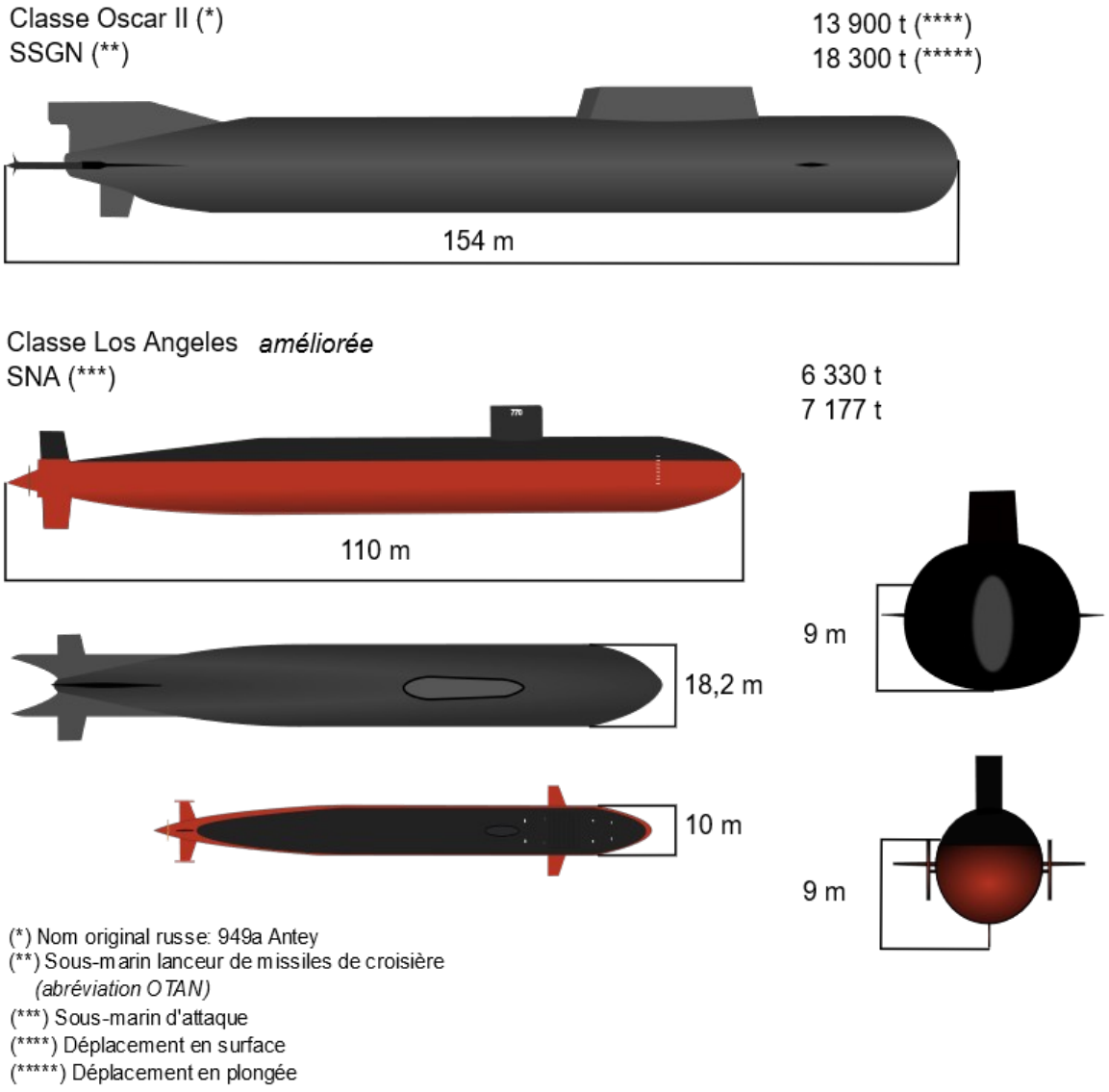
Pour un "camion", la plupart ont des fonctions communes (on ne parlera pas de suite des camions-grues). On regroupe ces actions dans un autre cadre.

Camion
immatriculation NombreEssieux CapacitéChargement (tonne) Puissance (chevaux DIN) hauteur (mètre) largeur (mètre) PTAC (tonne)
DémarrerMoteur() ArrêterMoteur() Freiner() Accélerer() Klaxonner() Charger() Décharger() ...

3.2.3 Classe

Une **classe est une famille**. Ce terme se retrouve dans les engins militaires ou les armes.

Exemple de classe : (voir film "à la poursuite d'octobre rouge")



Ainsi, une classe est une sorte de plan 'type' de différents objets ayant des attributs et des méthodes communes.

3.2.4 Instance et instanciation

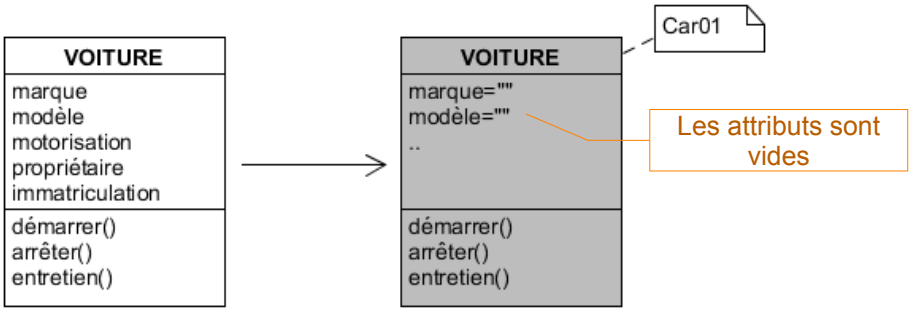
Une classe n'est qu'un modèle qui décrit les membres de celle-ci. Les membres sont :

- les attributs
- les méthodes

Pour créer un véritable objet dans le programme, il faut demander à utiliser ce modèle, le terme est instancier.

Instancier la classe VOITURE revient à **créer une nouvelle** voiture, avec des attributs réels. On utilise le mot-clé 'new' pour créer cette instance :

Car01 = new VOITURE()



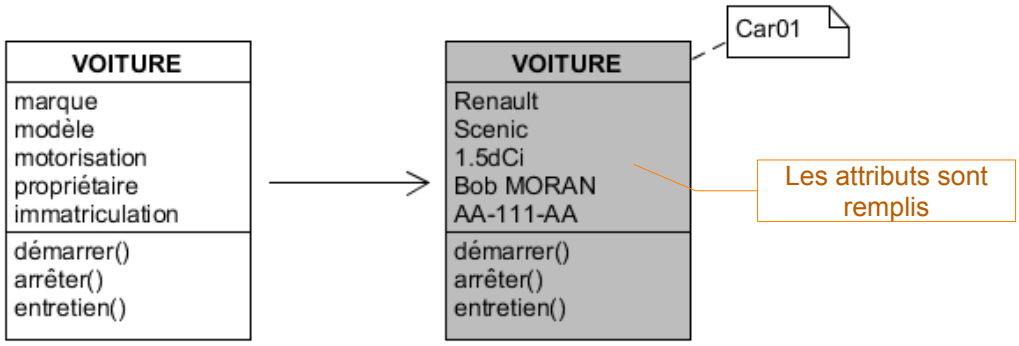
Il faut cependant remplir les attributs, ce qui se fait en utilisant la syntaxe objet.attribut = valeur ou bien en utilisant une méthode appelée mutateur (setMarque("Renault")).

Notez qu'un objet doit être stocké dans une variable (en C# et Java, on précisera une variable du type de la classe).

3.2.5 Constructeur

Le **constructeur est une méthode** qui porte un nom spécial (constructor en JavaScript, __constructor() en PHP ou le nom de la classe en C# et Java). On peut lui passer des paramètres pour remplir les attributs.

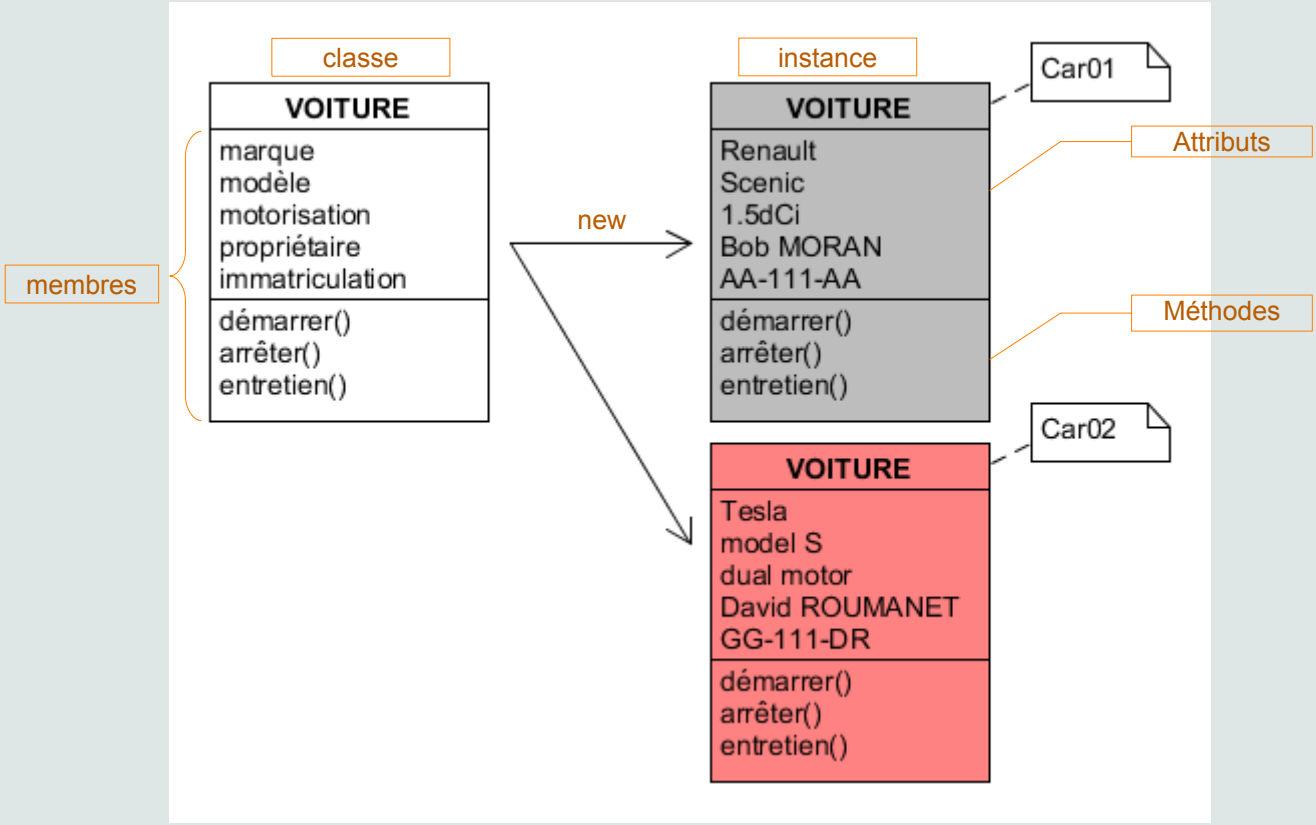
Car01 = new VOITURE("Renault", "Scenic", "1.5dCi", "Bob MORAN", "AA-111-AA")



4 En résumé

Voici les informations essentielles à retenir et assimiler.

Une **classe** est une **famille** d'objet ayant des propriétés et méthodes communes.



Un **objet** est une **instance** particulière de la classe, stocké dans une variable. L'accès aux **membres** de cet objet se fait en utilisant l'étiquette suivie d'un point et, soit d'une **propriété**, soit d'une **méthode**.

```
console.log("La voiture de "+ Car02.propriétaire + " est une "+Car02.marque)
```

affichera

```
La voiture de David ROUMANET est une Tesla
```

```
Car02.démarrer()
```

exécutera la méthode démarrer() sur l'objet Car02.

Pour créer un nouvel objet de la classe, on **l'instancie** avec le mot clé '**new**' qui appelle le **constructeur** de la classe.

```
Car03 = new VOITURE("Peugeot", "e-208", "100kW","Chuck Norris", "CH-000-CK")
```

4.1 Classe Loto

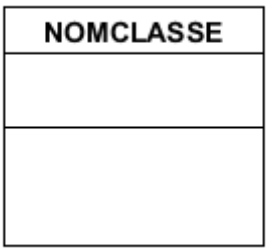
Il faut maintenant imaginer une famille Loto qui permettrait de contenir les attributs utiles pour différents tirages.

4.1.1 Création de la classe Loto

Dans le cadre ci-après, placez les attributs que vous jugez utile d'être contenu pour répondre aux différents besoins. Pour rappel :

- un tirage se fait actuellement sur 6 nombres
  - 5 nombres communs entre 1 et 49. Aucune redondance possible.
  - 1 nombre chance entre 1 et 10. Nombre indépendant des 5 autres.
- pour avoir un historique, un tirage est associé à une semaine
- chaque joueur ayant un nom peut avoir un tirage
- Si les règles changent, il doit être possible de changer le nombre de boules
- Sur les 5 nombres on peut les **insérer** manuellement (cas des grilles de joueurs) ou bien automatiquement (cas d'un tirage)
- Sur les 5 nombres on peut **vérifier** s'il n'y a pas de redondance
- Sur le nombre chance, on peut l'**insérer** manuellement ou automatiquement
- Sur les 5 nombres, on peut les **comparer** à 5 autres nombres (transmis)
- Sur un tirage on peut **modifier** ou **lire** le nom du joueur
- Sur un tirage on peut **modifier** ou **lire** le numéro de semaine
- Une méthode de choix de nombre au hasard dans un intervalle doit être proposée...

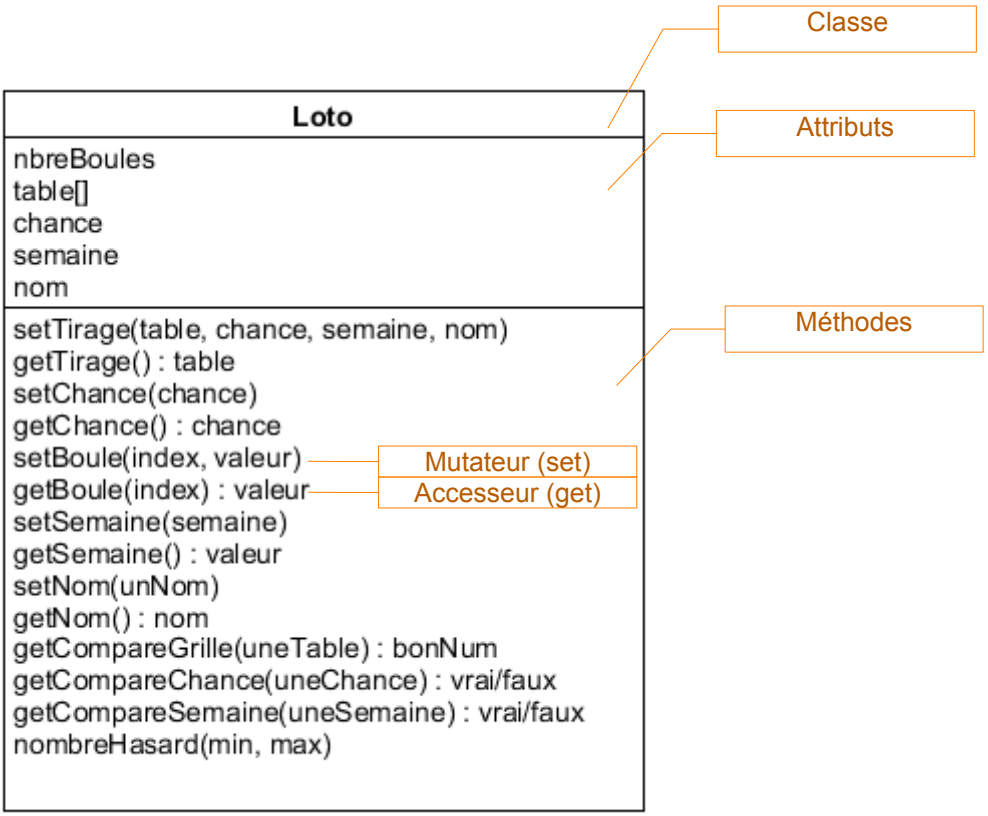
Entraînez-vous sur une feuille à remplir les cadres d'une classe (formalisme UML)



La correction se trouve à la page suivante.

4.1.2 Utilité de la classe Loto

Voici un exemple de réponse (mais elle n'est pas unique) de classe Loto.



Ce que peut apporter la classe Loto au développeur ? Plusieurs avantages :

- La classe peut-être enregistrée dans un fichier à part
  - Ce fichier ne contient que ce qui concerne la classe
  - C'est plus lisible
  - C'est plus facile à déboguer
  - N'importe qui peut utiliser ce code
- Les attributs et méthodes sont propres à la classe
  - Même s'il existe une autre méthode (ou fonction) dans le programme avec le même nom, seule celle de la classe s'applique sur un objet de cette classe (créé depuis cette classe).
  - Dans certaines conditions, les propriétés ne sont pas accessibles directement (il faut utiliser une méthode de la classe). Cela sécurise les manipulations de données. Dans ce cas, les propriétés doivent commencer par #.

4.1.3 Synoptique de la classe LOTO

Voici un exemple écrit de la classe, en JavaScript

```
// ----------  CLASSE  ----------------------------------------------------------------
class Loto {
    nbreBoules=0;                   // les attributs (chaque objet aura ses propres valeurs)
    table=[]
    chance=0
    semaine=0
    nom=""

    // constructeur
    constructor(nbreBoules) {
    nombreHasard(min, max) {
    setHasard(semaine=0) {
    setGrille(table, chance, semaine=0, unNom="") {
    setBoule(index, valeur) {
    getBoule(index) {
    getGrille() {
        return this.table
    }
    getChance() {
    getSemaine() {
    getCompareGrille(uneTable) {
    getCompareChance(uneChance) {
    getCompareSemaine(uneSemaine) {
}
```

Le travail pratique sera de modifier l'application originale pour utiliser les objets.

Note : pour créer un attribut privé, JavaScript propose de le faire précéder d'un symbole #.

Ainsi, pour rendre les attributs privés dans la classe ci-dessus, on écrira :

```
#nbreBoules=0 ;
#table=[] ;
#chance=0 ;
#semaine=0 ;
#nom="" ;
```

L'accès pour un getter est alors le suivant :

```
getChance() {
   return this.#chance ;
}
```