# TD2

## Exercice 1 : Gestion de Compte Bancaire

Objectif : Travailler sur l'instanciation, les constructeurs et l'encapsulation.

Créer une classe **CompteBancaire** avec les attributs privés suivants :

- titulaire (chaîne de caractères)
- solde (flottant)

Implémenter les méthodes suivantes :

- Un **constructeur** qui initialise un compte bancaire avec le nom du titulaire et un solde initial de 0.
- Une méthode **deposer** qui permet d'ajouter un montant au solde du compte.
- Une méthode **retirer** qui permet de retirer un montant du solde du compte (si le solde est suffisant).
- Des méthodes **getSolde** et **getTitulaire** pour accéder aux attributs privés.

### Questions :

- Pourquoi l'encapsulation est-elle importante dans la gestion d'un compte bancaire ?
- Comment le constructeur est-il utilisé pour initialiser correctement les objets CompteBancaire ?

## Exercice 2 : Surcharge de Méthodes pour les Calculs

Objectif : Travailler sur la surcharge de méthodes.

Crée une classe **Calculatrice** avec les méthodes suivantes :

- Une méthode **addition** qui accepte deux ou trois nombres et retourne leur somme. Si un seul nombre est fourni, elle retourne ce nombre.
- Une méthode **soustraction** qui accepte deux ou trois nombres. Si deux nombres sont fournis, elle retourne la différence entre le premier et le deuxième. Si trois nombres sont fournis, elle soustrait les deux derniers au premier.

### Exemple :

```
calc = Calculatrice();
System.out.println(calc.addition(10, 5));  // Retourne 15
System.out.println(calc.addition(10, 5, 2));  // Retourne 17
System.out.println(calc.soustraction(10, 5));  // Retourne 5
System.out.println(calc.soustraction(10, 5, 2));  // Retourne 3
```

## Exercice 3 : Gestion des Produits

Objectif : Travailler sur l'instanciation, les constructeurs et l'encapsulation.

Créer une classe **Produit** avec les attributs privés suivants :

- nom (chaîne de caractères)
- prix (flottant)
- quantite (entier)

Implémenter les méthodes suivantes :

- Un constructeur qui initialise un produit avec un nom, un prix et une quantité.
- Une méthode **ajusterPrix** qui permet de changer le prix du produit.
- Une méthode **ajusterQuantite** qui permet de changer la quantité disponible du produit.
- Des méthodes **getNom**, **getPrix**, et **getQuantite** pour accéder aux attributs.

### Questions :

- En quoi l'encapsulation aide-t-elle à contrôler les modifications sur les attributs du produit ?
- Comment le constructeur facilite-t-il l'instanciation de nouveaux produits ?

## Exercice 4 : Classe pour les Vecteurs

Objectif : Travailler sur la surcharge des méthodes et l'instanciation.

Créer une classe **Vecteur** avec les attributs :

- x (flottant)
- y (flottant)

Implémenter les méthodes suivantes :

- Un constructeur qui initialise un vecteur avec deux coordonnées, x et y.
- Une méthode **ajouter** qui accepte un autre vecteur ou deux coordonnées séparées et retourne la somme des vecteurs ou de leurs coordonnées.
- Une méthode **soustraire** qui accepte un autre vecteur ou deux coordonnées séparées et retourne la différence des vecteurs ou de leurs coordonnées.

### Exemple :

```
Vecteur v1 = Vecteur(2, 3);
Vecteur v2 = Vecteur(5, 7);
System.out.println(v1.ajouter(v2));  // Retourne un nouveau vecteur (7, 10)
System.out.println(v1.ajouter(1, 1));  // Retourne un nouveau vecteur (3, 4)
```

### Questions :

- Comment la surcharge de la méthode ajouter permet-elle de gérer différents types d'arguments ?
- En quoi le constructeur facilite-t-il l'instanciation de nouveaux vecteurs ?

# Exercice 5 : Héritage et Polymorphisme avec des Animaux

Objectif : Travailler sur l'instanciation, les constructeurs et la surcharge.

- Créer une classe **Animal** avec un attribut nom et une méthode parler qui retourne une chaîne de caractères indiquant que l'animal ne parle pas.
- Créer des classes dérivées **Chien** et **Chat** qui héritent de Animal :
  - Chien doit surdéfinir la méthode parler pour retourner “Ouaf”.
  - Chat doit surdéfinir la méthode parler pour retourner “Miaou”.
- Créer un programme utilisant une liste d'animaux et faisant appel à leur méthode parler.

### Questions :

- Comment la surdéfinition (overriding) est-elle utilisée pour implémenter le polymorphisme ?
- Que se passe-t-il si on ajoute un nouvel animal, par exemple un Oiseau, avec sa propre méthode parler ?

# Exercice 6 : Bibliothèque de Livres

Objectif : Travailler sur l'instanciation, les constructeurs, l'encapsulation et les associations entre classes.

Crée une classe **Livre** avec les attributs privés suivants :

- titre (chaîne de caractères)
- auteur (chaîne de caractères)
- isbn (chaîne de caractères)
- nombrePages (entier)

Implementer les méthodes suivantes :

- Un constructeur qui initialise un livre avec un titre, un auteur, un ISBN et un nombre de pages.
- Des méthodes getTitre, getAuteur, getIsbn et getNombrePages pour accéder aux attributs.
- Des méthodes setTitre, setAuteur, setIsbn, et setNombrePages pour modifier ces attributs.

Créer ensuite une classe **Bibliotheque** avec les attributs suivants :

- nom (chaîne de caractères)
- adresse (chaîne de caractères)
- livres (liste d'objets Livre)

Implementer les méthodes suivantes :

- Un constructeur pour initialiser une bibliothèque avec un nom, une adresse, et une liste vide de livres.
- Une méthode ajouterLivre qui permet d'ajouter un livre à la bibliothèque.
- Une méthode afficherLivres qui affiche la liste des livres avec leur titre, auteur et ISBN.
- Une méthode rechercherLivre qui permet de rechercher un livre par son ISBN et le retourne.

### Question

- Représenter le diagramme de classes

# Escape game

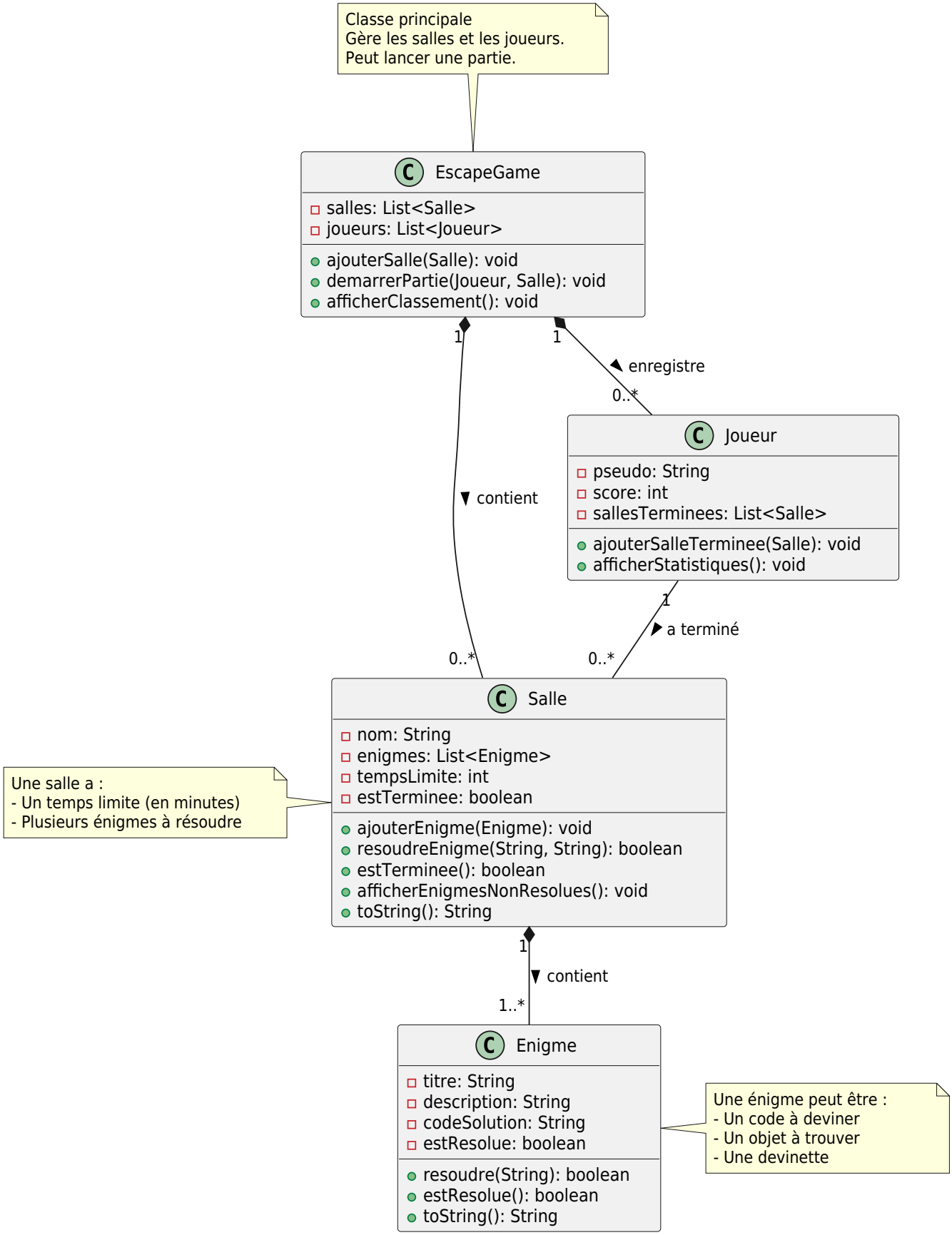
Objectif : Concevoir et implémenter un système simplifié pour un escape game en mode console. Les joueurs devront résoudre des énigmes dans des salles thématiques pour s'échapper avant la fin du temps imparti.

## Consignes Générales

- Travail en binôme.
- Langage : Java.

## Cahier des Charges

### 1. Classes à Implémenter




## 2. Fonctionnalités

### Gestion des Énigmes

**Une énigme a (attributs) :**

- Un titre et une description (affichés en console).
- Un code solution (ex : “CLEF”, “1984”, “MIROIR”).
- Un état (resolue ou non).

**Méthodes :**

- resoudre(String tentative) : Compare la tentative du joueur au code solution.
- toString() : Affiche l’énigme de manière lisible (sans spoiler la solution).

#### Gestion des Salles

**Une salle a :**

- Un nom (ex : “La Bibliothèque Maudite”).
- Une liste d’énigmes.
- Un temps limite (en minutes, simulé en console).

**Méthodes :**

- ajouterEnigme(Enigme) : Ajoute une énigme à la salle.
- resoudreEnigme(int index, String reponse) : Tente de résoudre une énigme par son index.
- estTerminee() : Retourne true si toutes les énigmes sont résolues.

#### Interaction en Console

**Le programme doit permettre de :**

- Afficher les énigmes disponibles dans une salle.
- Sélectionner une énigme (par numéro) et entrer une réponse.
- Recevoir un feedback (ex : “ Énigme résolue !” ou “ Mauvaise réponse”).
- Passer à la salle suivante si toutes les énigmes sont résolues.

#### Règles de gestion supplémentaires

- Une salle doit connaître la salle suivante.
- Il existe plusieurs types de salles :
  - Celles par défaut où il faut résoudre toutes les énigmes pour sortir
  - Celles où il faut juste en résoudre une seule (à condition que l'énigme soit identifiée comme “unique”)
  - Celles où l'une des énigmes ne doit pas être résolue...
  - La salle de sortie, correspondant à la fin du jeu