3 – UML

1MODE - Modélisation d'applications

Sommaire

1. Introduction
2. Diagrammes de classe

1. Introduction

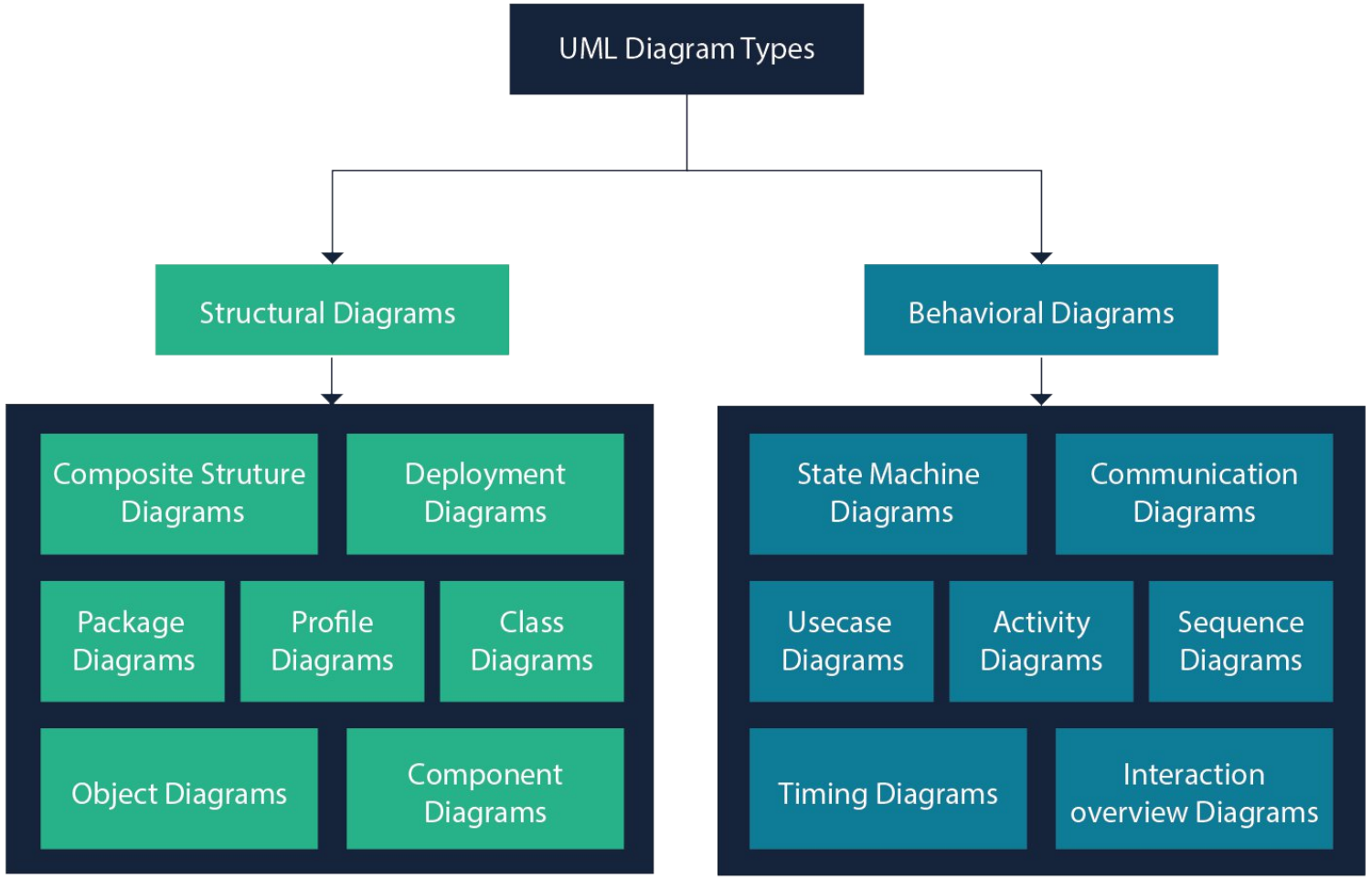
1. Introduction

Qu’est-ce que l’UML ?

- *Unified Modeling Language*
- Le langage de modélisation visuel utilisé pour la conception orientée objet
- Première version apparue en 1997
- Version actuelle : 2.5

1. Introduction

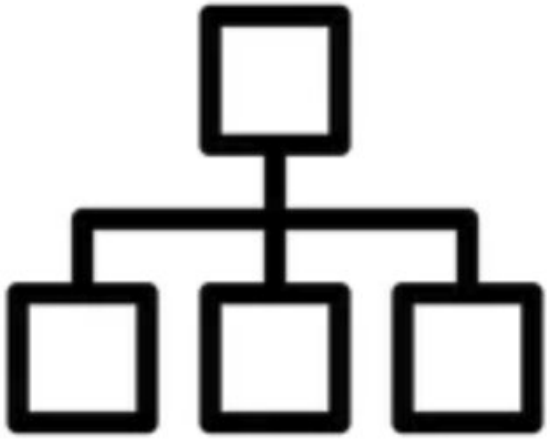
Types de diagrammes UML



1. Introduction

Diagrammes de classe

Dans ce cours, nous nous concentrerons uniquement sur les diagrammes de classe



1. Introduction

2. Diagrammes de classes

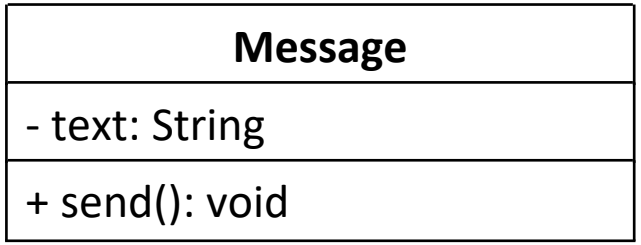
2. Diagrammes de classes

Diagrammes de classes

Comportent 3 sections :

- Nom de la classe
- Liste des attributs
- Liste des opérations

Exemple:



2. Diagrammes de classes

Conventions

- Le nom de la classe est écrit en **gras**.
- Si la classe est abstraite, son nom est écrit en *italique*.
- Les attributs et méthodes de classe sont écrits en souligné.

2. Diagrammes de classes

Grammaire - Attributs

- Les attributs désignent les propriétés, ou les champs, ou une classe
- Ils sont spécifiés de la manière suivante :

<visibilité><nom>: <type> = <valeur par défaut> <{modificateur}>

2. Diagrammes de classes

Grammaire - Attributs

<visibilité><nom>: <type> = <valeur par défaut> <{modificateur}>

Visibilité

- Désigne les modificateurs d'accès des attributs
- Diffèrent selon le langage de programmation
- Les modificateurs d'accès suivants sont supportés :
  - + : public
  - - : private
  - # : protected

2. Diagrammes de classes

Grammaire - Attributs

<visibilité><nom>: <type> = <valeur par défaut> <{modificateur}>

Nom
- Nom de l'attribut

Type
- Désigne le type de donnée
- Peut être soit un type primitif (*e.g.*, int, double), soit une classe

2. Diagrammes de classes

Grammaire - Attributs

<visibilité><nom>: <type> = <valeur par défaut> <{modificateur}>

Valeur par défaut
- Optionnelle (inclut le signe ‘=‘)
- Correspond au type précisé

Modificateur
- Optionnel
- Contient des informations supplémentaires (*e.g.*, {readOnly})

2. Diagrammes de classes

Grammaire - Opérations

- Les opérations désignent les comportements ou méthodes d’une classe
- Ils sont spécifiés de la manière suivante :

<visibilité><nom>: (<paramètres>): <type de retour>

2. Diagrammes de classes

Grammaire - Opérations

Visibilité et Nom
Identiques à précédemment

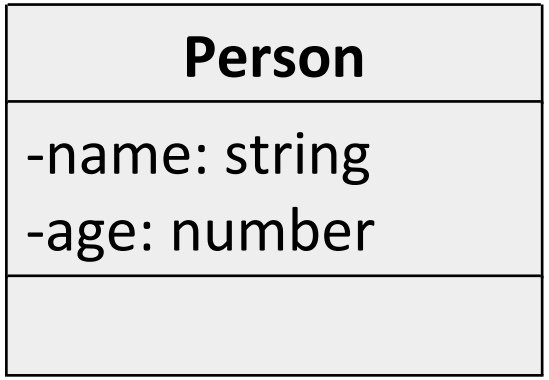
Paramètres
Représente les paramètres reçus en arguments pour cette opération
<parameter name>: <parameter type>

Type de retour
Le type du résultat de l'opération

2. Diagrammes de classes

Attributs

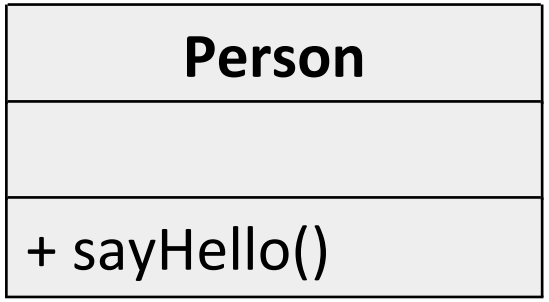
```
class Person {
  private name: string
  private age: number
}
```



2. Diagrammes de classes

Méthode

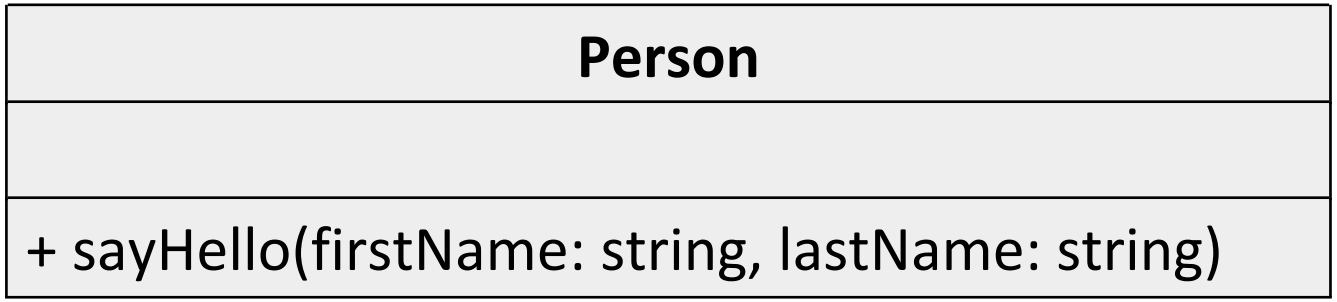
```
class Person {
  public sayHello() {
    console.log("Hello!")
  }
}
```



2. Diagrammes de classes

Méthode avec paramètres

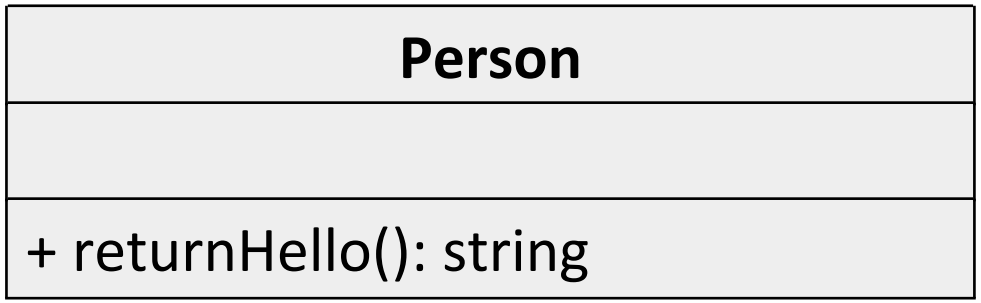
```
class Person {
  public sayHello(firstName: string, lastName: string) {
    console.log(`Hello, ${firstName} ${lastName}!`)
  }
}
```



2. Diagrammes de classes

Méthode avec un type de retour

```
class Person {
  public returnHello(): string {
    return "Hello"
  }
}
```



2. Diagrammes de classes

Relations

Les classes orientées objet ont des relations entre elles, qui peuvent être de différents types :

- Héritage : les classes s'étendent l'une sur l'autre
- Dépendance : les classes dépendent les unes des autres
- Association : les classes interagissent les unes avec les autres
- Agrégation et composition : les classes forment partie les unes des autres

2. Diagrammes de classes

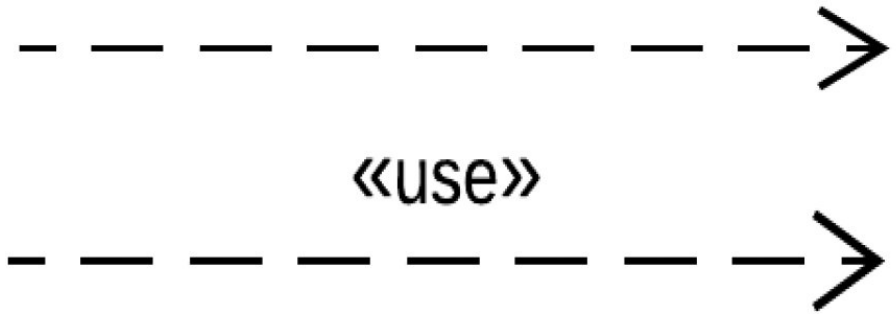
Relations

- Les relations ont également une multiplicité, qui montre combien d'instances d'une classe peuvent exister de chaque côté d'une relation
- Les relations de dépendance existent lorsque les classes dépendent les unes des autres de telle manière qu'un changement dans l'une peut affecter l'autre, comme lorsque l'une accepte une instance d'une autre classe comme paramètre d'une méthode

2. Diagrammes de classes

Relations : Dépendance

- Les relations de dépendance sont représentées par des flèches sur des lignes en pointillé
- Des stéréotypes peuvent être indiqués entre guillemets pour fournir des détails supplémentaires sur la nature de la relation



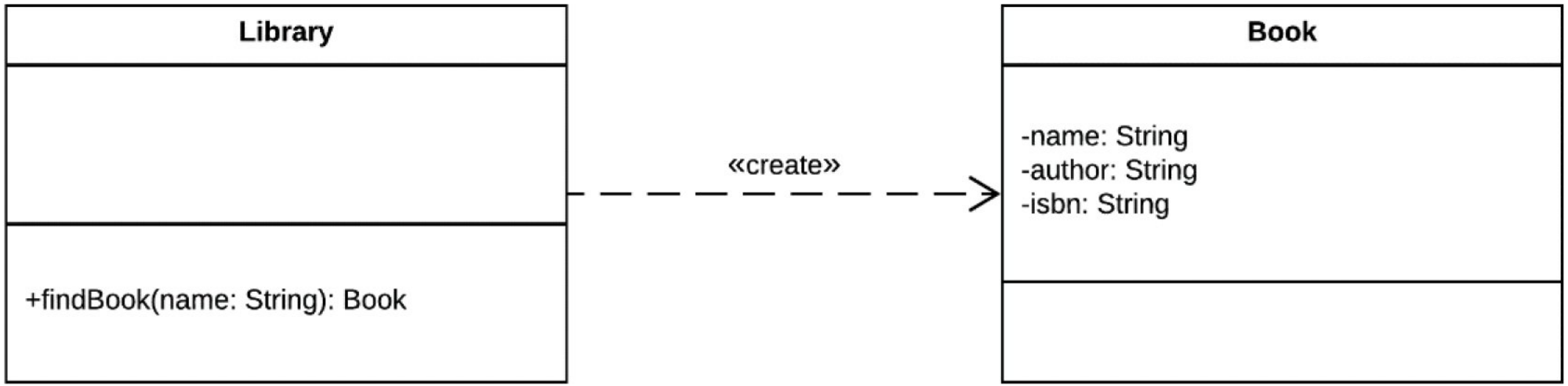
2. Diagrammes de classes

Relations : Dépendance – Exemple

- Une relation de dépendance peut exister lorsque nous avons une classe Bibliothèque qui gère des objets Livre
- Comme la classe Bibliothèque a une méthode qui renvoie un Livre, des changements apportés à la classe Livre pourraient entraîner des changements dans la classe Bibliothèque (en fonction de la façon dont les objets Livre sont créés)

2. Diagrammes de classes

Relations : Dépendance – Exemple



2. Diagrammes de classes

Relations : Dépendance – Exemple

```
class Book {
  private name: string
  private author: string
  private isbn: string

  constructor(name: string, author: string, isbn: string) {
    this.name = name
    this.author = author
    this.isbn = isbn
  }
}
```

2. Diagrammes de classes

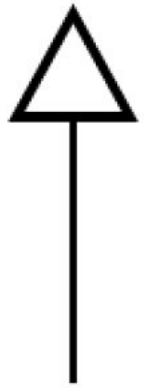
Relations : Dépendance – Exemple

```
class Library {
  private books: Book[] = []

  public create(name: string, author: string, isbn: string): Book {
    const book = new Book(name, author, isbn)
    this.books.push(book)
    return book
  }

  public findBook(name: string): Book | undefined {
    return this.books.find(book => book.name === name)
  }
}
```

2. Diagrammes de classes

Relations : Généralisation

- Les relations de généralisation existent lorsqu'une classe étend une autre classe (en la spécialisant, comme une voiture est une spécialisation d'un véhicule).
- Les relations de généralisation sont représentées par une flèche triangulaire sur une ligne pleine



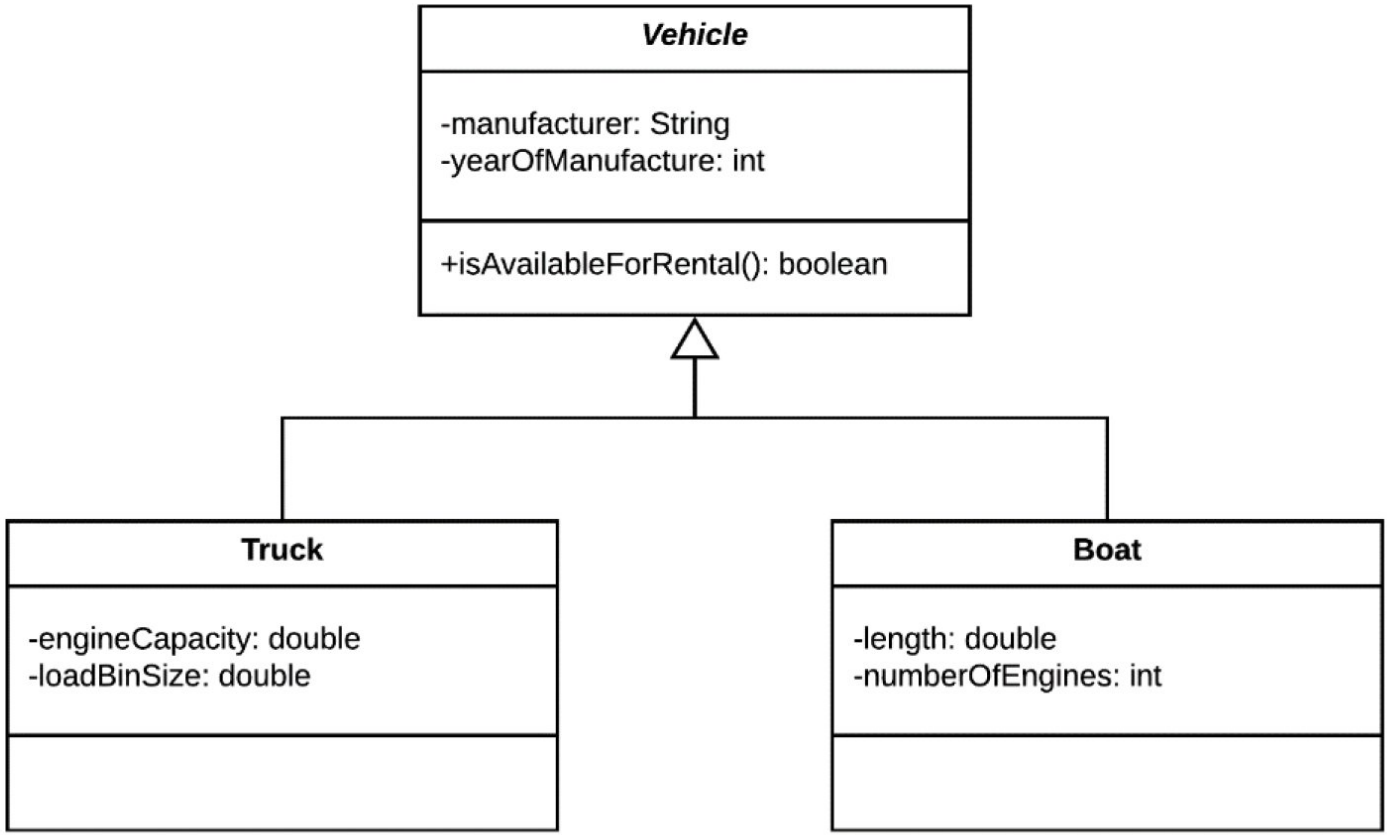
2. Diagrammes de classes

Relations : Généralisation - Exemple

- Les opérations et les attributs de la classe parent existent également dans les classes enfants, sans être spécifiés explicitement
- Une relation de généralisation peut exister lorsque nous avons un système qui suit la location de véhicules, où nous avons différents types de véhicules spécialisés

2. Diagrammes de classes

Relations : Généralisation - Exemple



2. Diagrammes de classes

Relations : Généralisation - Exemple

```
class Vehicle {
  protected manufacturer: string
  protected yearOfManufacture: number

  constructor(manufacturer: string, yearOfManufacture: number) {
    this.manufacturer = manufacturer
    this.yearOfManufacture = yearOfManufacture
  }

  public isAvailableForRental(): boolean {
    return true
  }
}
```

2. Diagrammes de classes

Relations : Généralisation - Exemple

```
class Truck extends Vehicle {
  private engineCapacity: number
  private loadBinSize: number

  constructor(manufacturer: string, yearOfManufacture: number,
engineCapacity: number, loadBinSize: number) {
    super(manufacturer, yearOfManufacture)
    this.engineCapacity = engineCapacity
    this.loadBinSize = loadBinSize
  }
}
```

2. Diagrammes de classes

Relations : Généralisation - Exemple

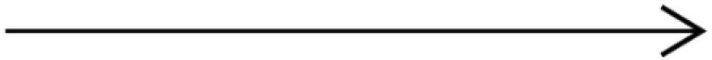
```
class Boat extends Vehicle {
  private length: number
  private numberOfEngines: number

  constructor(manufacturer: string, yearOfManufacture: number, length:
number, numberOfEngines: number) {
    super(manufacturer, yearOfManufacture)
    this.length = length
    this.numberOfEngines = numberOfEngines
  }
}
```

2. Diagrammes de classes

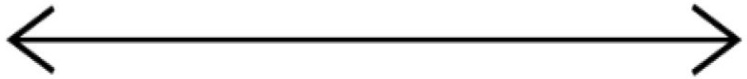
Relations : Association

Les relations d'association existent souvent lorsque les classes ont des attributs d'autres types sur lesquelles elles peuvent invoquer des opérations

- Les relations d'association sont représentées par une flèche sur une ligne pleine



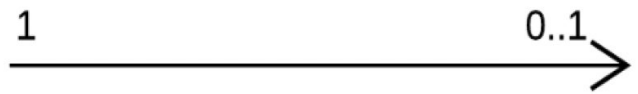
- Les relations d'association peuvent être bidirectionnelles, auquel cas les deux classes peuvent se référencer mutuellement.

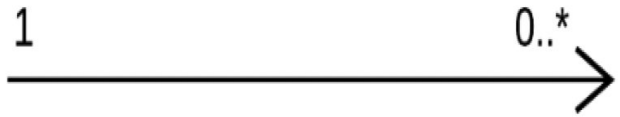


2. Diagrammes de classes

Relations : Association

- Les relations d'association peuvent également inclure une multiplicité, où nous pouvons avoir une instance d'un côté et exactement zéro ou une instance de l'autre côté, ou une instance d'un côté et zéro ou plusieurs instances de l'autre côté (* fait référence à n'importe quel nombre d'instances), ou toute autre combinaison possible

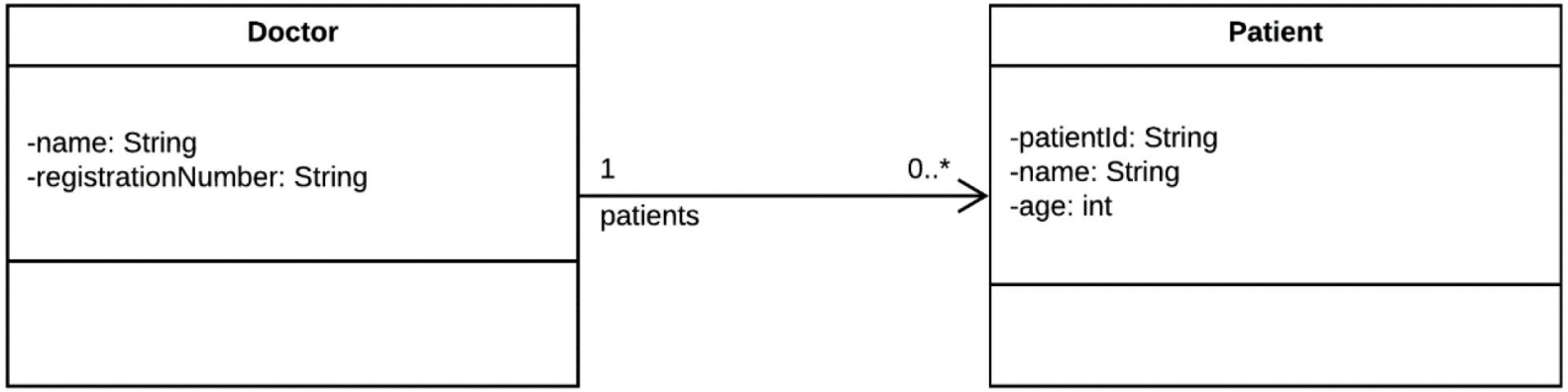




2. Diagrammes de classes

Relations : Association – Exemple

- Une relation d'association peut exister lorsque nous modélisons la relation entre les médecins et les patients, où un médecin peut avoir n'importe quel nombre de patients et un patient ne peut être traité que par un médecin à la fois



2. Diagrammes de classes

Relations : Association – Exemple

```
class Doctor {
  private name: string
  private registrationNumber: string
  private patients: Patient[] = []

  constructor(name: string, registrationNumber: string) {
    this.name = name
    this.registrationNumber = registrationNumber
  }

  public addPatient(patient: Patient): void {
    this.patients.push(patient)
  }
}
```

2. Diagrammes de classes

Relations : Association – Exemple

```
class Patient {
  private patientId: string
  private name: string
  private age: number
  private doctor: Doctor

  constructor(patientId: string, name: string, age: number, doctor:
Doctor) {
    this.patientId = patientId
    this.name = name
    this.age = age
    this.doctor = doctor
    doctor.addPatient(this)
  }
}
```

2. Diagrammes de classes

Relations : Agrégation

Les relations d'agrégation existent lorsque nous agrégeons (ou rassemblons) des objets d'une classe dans une autre classe

- Les relations d'agrégation sont représentées par un diagramme non rempli du côté "propriétaire" de la relation

2. Diagrammes de classes

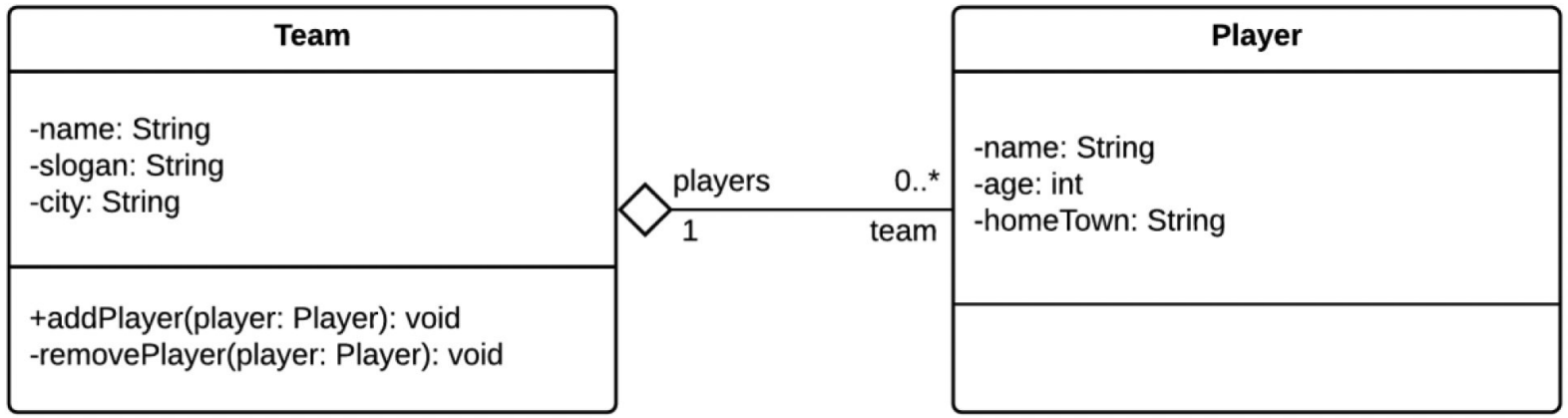
Relations : Agrégation

- Les objets des deux côtés d'une relation d'agrégation peuvent exister séparément
- Les relations d'agrégation peuvent avoir une multiplicité

2. Diagrammes de classes

Relations : Agrégation – Exemple

- Une relation d'association peut exister lorsque nous modélisons les équipes et les joueurs
- Un joueur peut exister sans appartenir à une équipe, et une équipe peut exister sans aucun joueur



2. Diagrammes de classes

Relations : Agrégation – Exemple

```
class Team {
  private name: string
  private slogan: string
  private city: string
  private players: Player[] = []

  constructor(name: string, slogan: string, city: string) {
    this.name = name
    this.slogan = slogan
    this.city = city
  }
  // ...
```

2. Diagrammes de classes

Relations : Agrégation – Exemple

```
 public addPlayer(player: Player): void {
    this.players.push(player)
    player.team = this
  }

  public removePlayer(player: Player): void {
    const index = this.players.indexOf(player)
    if (index !== -1) {
      this.players.splice(index, 1)
      player.team = null
    }
  }
}
```

2. Diagrammes de classes

Relations : Agrégation – Exemple

```
class Player {
  private name: string
  private age: number
  private homeTown: string
  private team?: Team = null

  constructor(name: string, age: number, homeTown: string) {
    this.name = name
    this.age = age
    this.homeTown = homeTown
  }
}
```

2. Diagrammes de classes

Relations : Composition

Les relations de composition existent lorsque des objets sont composés (ou constitués) d'autres objets

- Les relations de composition sont représentées par un losange rempli du côté "propriétaire"

2. Diagrammes de classes

Relations : Composition

- Les objets dans une relation de composition ne peuvent pas, conceptuellement, exister isolément
- Ceci n'est pas toujours facile à appliquer lors des développements
- Si l'objet parent dans une relation de composition est détruit, les objets enfants le sont aussi

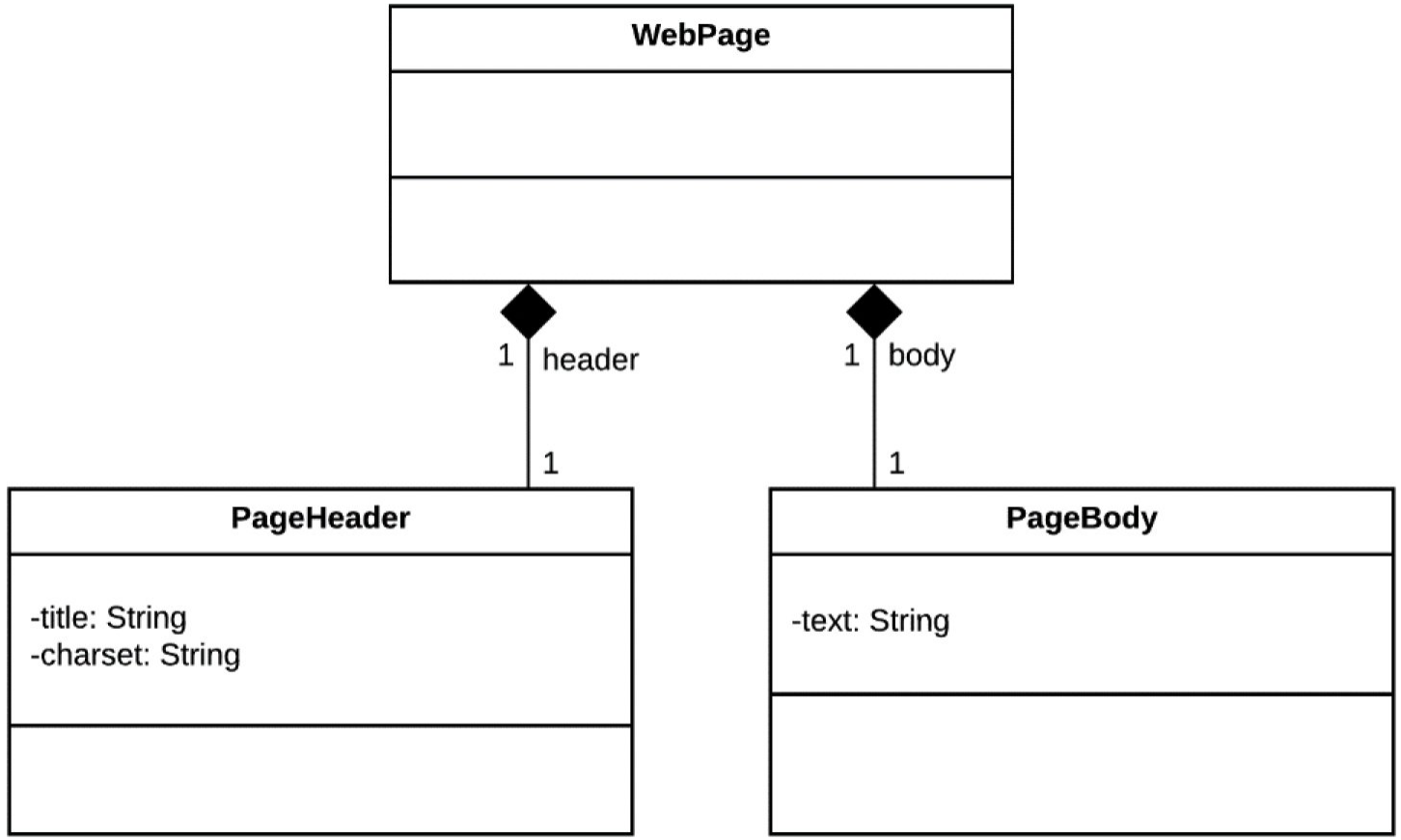
2. Diagrammes de classes

Relations : Composition – Exemple

- Une relation de composition peut exister lorsque nous modélisons un système pour créer des pages Web
- Les pages ne peuvent pas exister sans un en-tête de page et un corps de page, et chaque objet PageHeader et PageBody doit appartenir à un objet WebPage

2. Diagrammes de classes

Relations : Composition – Exemple



2. Diagrammes de classes

Relations : Composition – Exemple

```
class PageHeader {
  private title: string
  private charset: string

  constructor(title: string, charset: string) {
    this.title = title
    this.charset = charset
  }
}
```

2. Diagrammes de classes

Relations : Composition – Exemple

```
class PageBody {
  private text: string

  constructor(text: string) {
    this.text = text
  }
}
```

2. Diagrammes de classes

Relations : Composition – Exemple

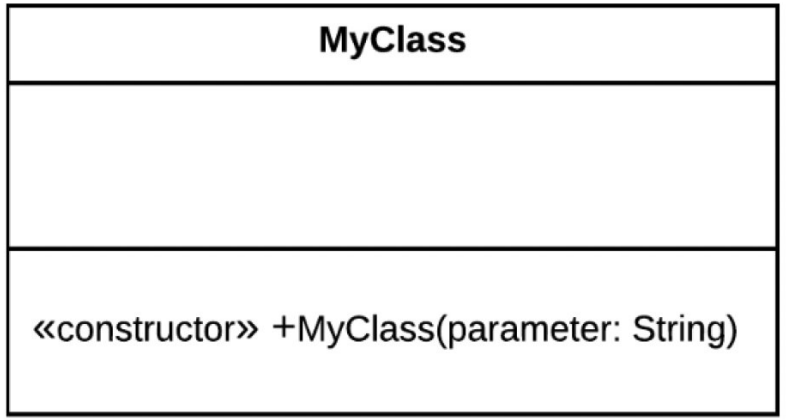
```
class WebPage {
  private header: PageHeader
  private body: PageBody

  constructor(headerTitle: string, headerCharset: string, bodyText:
string) {
    this.header = new PageHeader(headerTitle, headerCharset)
    this.body = new PageBody(bodyText)
  }
}
```

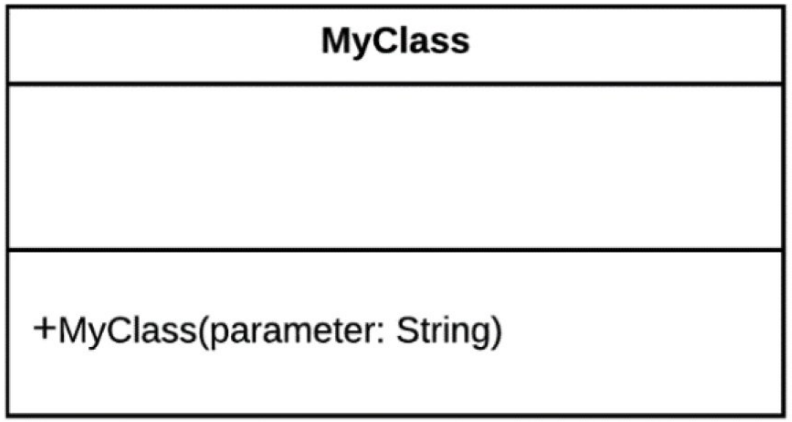
2. Diagrammes de classes

Constructeurs

Les constructeurs sont des opérations spéciales utilisées pour créer des instances d'une classe

- Les constructeurs portent le même nom que la classe et ne spécifient pas de type de retour
- Optionnellement, les constructeurs peuvent être indiqués avec le mot-clé "constructor"





2. Diagrammes de classes

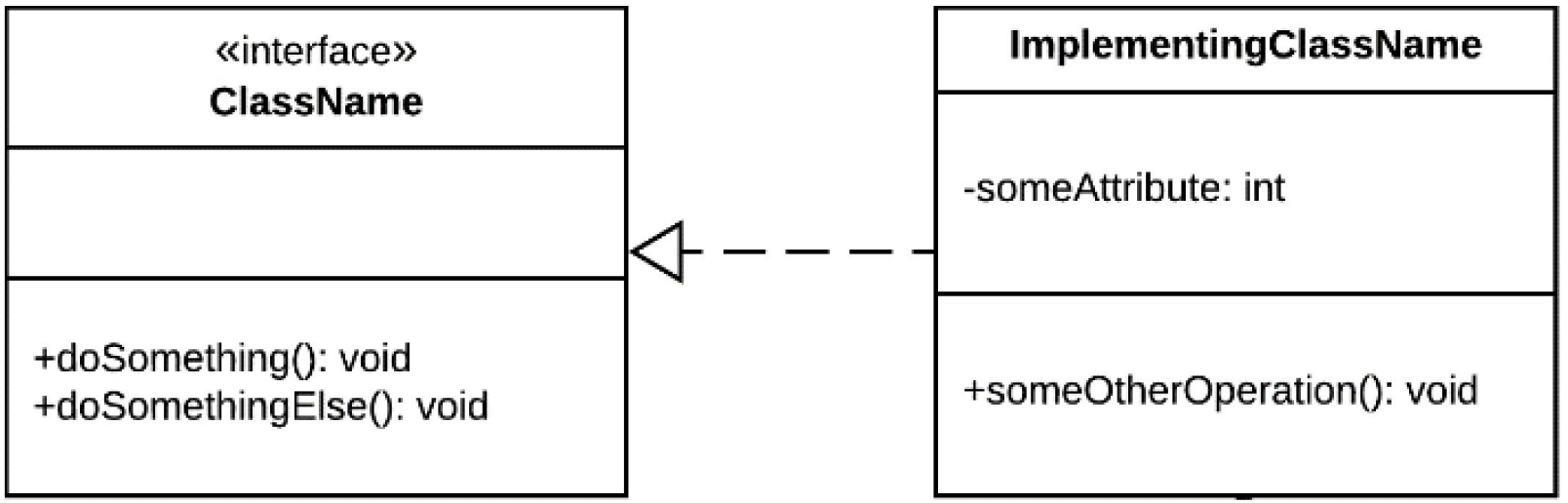
Interfaces

Les interfaces définissent des contrats pour le comportement, mais sans implémenter ce comportement directement

- Les interfaces sont réalisées (ou implémentées) par des classes concrètes
- Les interfaces sont indiquées par le mot-clé "interface"
- La réalisation est représentée par un triangle sur une ligne en pointillé
- Les méthodes d'interface sont implicitement incluses dans les classes qui réalisent une interface, même si elles ne sont pas affichées

2. Diagrammes de classes

Interfaces



2. Diagrammes de classes

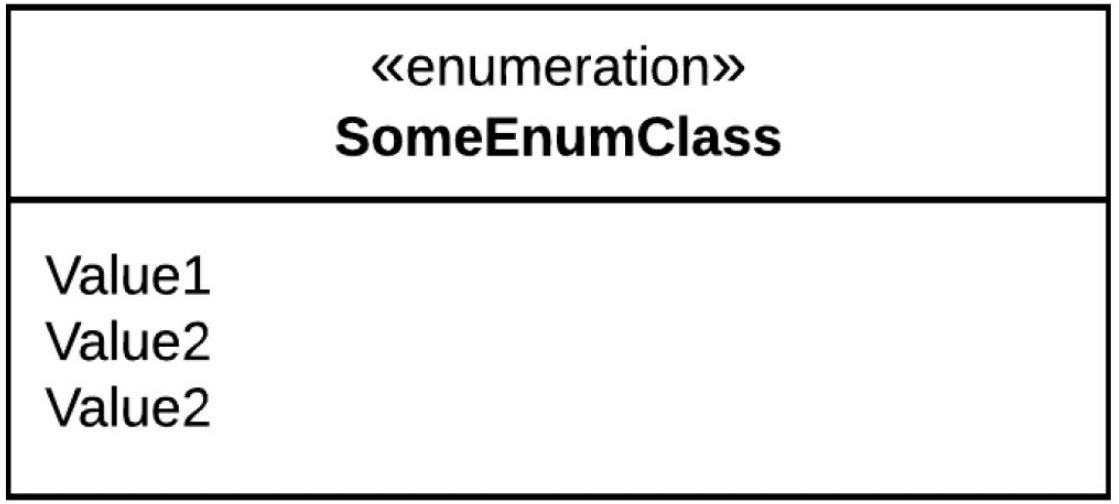
Énumérations

Les énumérations, sont des classes qui fournissent un ensemble fixe de valeurs littérales

- Les énumérations sont définies par le mot-clé ”enumeration"
  Les énumérations ont des attributs (valeurs), mais pas de comportement, donc ils peuvent être dessinés sans la section des opérations
- Les énumérations n'ont pas besoin de définir les types d'attributs, car tous les attributs sont du type de l’énumération elle-même
- Les énumérations n'ont pas non plus besoin de montrer les modificateurs d'accès, car tous les attributs sont implicites s'ils sont accessibles depuis l'enum elle-même

2. Diagrammes de classes

Énumérations



2. Diagrammes de classes

Énumérations

```
enum Direction {
  Up,
  Down,
  Left,
  Right,
}
```

2. Diagrammes de classes