const md = require('markdown-it')() .use(require('markdown-it-plantuml'));

Développement orienté objet

Abstraction

Définition

L'**abstraction** est un mécanisme qui permet de définir une classe en ne considérant que les caractéristiques essentielles de l'objet. Une classe abstraite peut donc contenir :

- des attributs
- des méthodes
- des méthodes abstraites

Une **méthode abstraite** est une méthode dont on fournit la signature (le nom, les paramètres et le type de retour) mais pas le corps.

Une classe abstraite ne peut pas être instanciée. Elle sert de modèle pour d'autres classes qui en héritent. Ces classes filles doivent implémenter les méthodes abstraites de la classe mère. Si une classe fille ne le fait pas, elle devient elle-même une classe abstraite.

Déclaration d'une classe abstraite en TypeScript

```
abstract class Animal {
  protected nom: string;
  protected age: number;

  constructor(nom: string, age: number) {
    this.nom = nom;
    this.age = age;
  }

  abstract parler(): void;

  toString(): string {
    return `${this.nom} a ${this.age} ans`;
  }
}
```

Syntaxe : le mot-clef `abstract` permet de déclarer une classe abstraite. Les méthodes abstraites sont déclarées sans corps et précéder du mot-clef `abstract`. Une méthode abstraite n'a pas de corps.

Remarque : il peut bien sûr y avoir plusieurs méthodes abstraites.

Si on essaie d'instancier la classe `Animal`, on obtient une erreur :

```
const animal = new Animal("Médor", 5); // Erreur : Impossible de créer une instance
d'une classe abstraite
```

```
...
src/animal.ts:17:16 - error TS2511: Cannot create an instance of an abstract class.
...
```

Héritage d'une classe abstraite

Une classe abstraite peut être héritée par une autre classe. La classe fille doit implémenter les méthodes abstraites de la classe mère.

```
class Chien extends Animal {
  constructor(nom: string, age: number) {
    super(nom, age);
  }

  parler(): void {
    console.log("Wouf !");
  }
}
```

Syntaxe : la classe `Chien` hérite de la classe `Animal` avec le mot-clef `extends`. La classe `Chien` doit implémenter la méthode abstraite `parler()` de la classe `Animal`.

Si la classe fille ne le fait pas, elle devient elle-même une classe abstraite. Le code ci-dessous lève une erreur :

```
class Mamifère extends Animal {
  constructor(nom: string, age: number) {
    super(nom, age);
  }

  toString(): string {
    return super.toString() + "\n c'est un mamifère";
  }
}
```

```
...
src/mamifere.ts:1:7 - error TS2515: Non-abstract class 'Mamifere' does not implement
inherited abstract member 'parler' from class 'Animal'.
...
```

La classe `Mamifère` peut hériter de la classe `Animal` mais doit être déclarée comme abstraite car elle ne définit pas la méthode abstraite `parler()`.

```
abstract class Mamifère extends Animal {
  private _duréeGestation: number;

  constructor(nom: string, age: number, duréeGestation: number) {
    super(nom, age);
    this._duréeGestation = duréeGestation;
  }

  toString(): string {
    return super.toString() + "\n c'est un mamifère";
  }

  get duréeGestation(): number {
    return this._duréeGestation;
  }
}
```

Instanciation d'une classe fille

On peut instancier une classe fille.

```
const chien = new Chien("Médor", 5);
console.log(chien.toString());
chien.parler();
```

```
Médor a 5 ans
Wouf !
```

Classe abstraite et polymorphisme

Une classe abstraite peut être utilisée comme type de variable. On peut donc déclarer une variable de type `Animal` et lui affecter une instance de la classe `Chien`.

```
const animal: Animal = new Chien("Médor", 5);
console.log(animal.toString());
animal.parler();
```

```
Médor a 5 ans
Wouf !
```

Cela est notamment utile si on a besoin de manipuler des objets de différentes classes filles de la classe mère.

```
const animaux: Animal[] = [
  new Chien("Médor", 5),
  new Chien("Rex", 3),
  new Chat("Félix", 2),
  new Chat("Minou", 4),
];

for (const animal of animaux) {
  console.log(animal.toString());
  animal.parler();
}
```

```
Médor a 5 ans
Wouf !
Rex a 3 ans
Wouf !
Félix a 2 ans
Miaou !
Minou a 4 ans
Miaou !
```

Ou si on a besoin de passer en paramètre d'une fonction / méthode une instance de la classe `Chien` ou de la classe `Chat`.

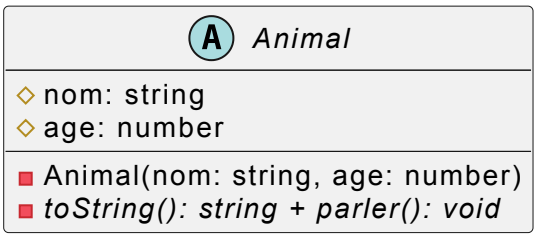
```
function faireParler(animal: Animal): void {
  animal.parler();
}

faireParler(new Chien("Médor", 5));
faireParler(new Chat("Félix", 2));
```

```
Wouf !
Miaou !
```

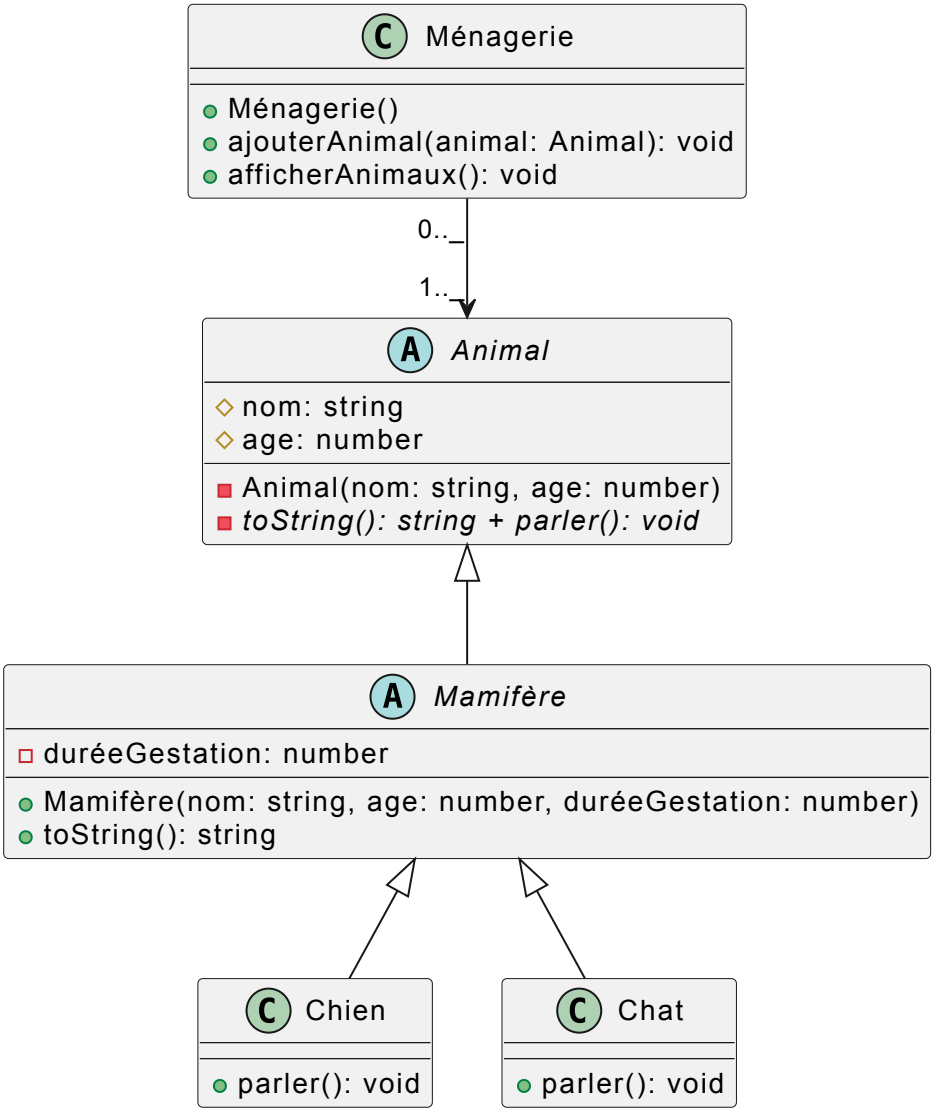
Classes abstraites en UML

Une classe abstraite est représentée par un nom en italique. Les méthodes abstraites sont représentées par un nom en italique.

Remarque : il n'est pas aisée d'écrire à la main en italique, vous pouvez dans un diagramme UML fait à la main indiqué explicitement le mot-clef `abstract`



On peut ajouter des associations entre une classe abstraite et d'autres classes :



Remarque : on indique les méthodes abstraites qui ont été implémentées par la classe fille.

Le code ci-dessous est une implémentation possible de ce diagramme UML.

```
abstract class Animal {
  protected nom: string;
```

```
  protected age: number;

  constructor(nom: string, age: number) {
    this.nom = nom;
    this.age = age;
  }

  abstract parler(): void;

  toString(): string {
    return `${this.nom} a ${this.age} ans`;
  }
}

abstract class Mamifère extends Animal {
  private _duréeGestation: number;

  constructor(nom: string, age: number, duréeGestation: number) {
    super(nom, age);
    this._duréeGestation = duréeGestation;
  }

  toString(): string {
    return super.toString() + "\n c'est un mamifère";
  }

  get duréeGestation(): number {
    return this._duréeGestation;
  }
}

class Chien extends Mamifère {
  constructor(nom: string, age: number) {
    super(nom, age, 63);
  }

  parler(): void {
    console.log("Wouf !");
  }
}

class Chat extends Mamifère {
  constructor(nom: string, age: number) {
    super(nom, age, 63);
  }

  parler(): void {
    console.log("Miaou !");
  }
}

class Ménagerie {
  private _animaux: Animal[];
```

```
  constructor() {
    this._animaux = [];
  }

  ajouterAnimal(animal: Animal): void {
    this._animaux.push(animal);
  }

  afficherAnimaux(): void {
    for (const animal of this._animaux) {
      console.log(animal.toString());
      animal.parler();
    }
  }
}

const ménagerie = new Ménagerie();
ménagerie.ajouterAnimal(new Chien("Médor", 5));
ménagerie.ajouterAnimal(new Chat("Félix", 2));
ménagerie.afficherAnimaux();
```

```
Médor a 5 ans
 c'est un mamifère
Wouf !
Félix a 2 ans
 c'est un mamifère
Miaou !
```