

Класи

Інкапсуляція

Класи

У нових версіях додано можливість описувати класи, що спрощують опис і створення об'єктів

Об'єкт реальної дійсності

----- Властивості -----

(параметри, характеристики)

- властивість-характеристика 1
- властивість-характеристика 2
-
- властивість-характеристика N

----- Функціональні можливості -----

(дії, які може виконувати сам об'єкт або можна виконувати з об'єктами)

- функціональна можливість (дія) 1
- функціональна можливість (дія) 2
-
- функціональна можливість (дія) M

Класи

У нових версіях додано можливість описувати класи, що спрощують опис і створення об'єктів

Об'єкт реальної дійсності

----- Властивості -----

(параметри, характеристики)

- властивість-характеристика 1
- властивість-характеристика 2
-
- властивість-характеристика N

*властивості описуємо
у конструкторі*

----- Функціональні можливості -----

(дії, які може виконувати сам об'єкт або можна виконувати з об'єктами)

- функціональна можливість (дія) 1
- функціональна можливість (дія) 2
-
- функціональна можливість (дія) M

*функціональні можливості описуємо як
методи*

Приклад опису класу

```
class Назва класу {
```

```
    constructor(форм.параметри){  
        //--- Опис полів (індивідуальні дані об'єктів)---  
        this.властивість1 = значення1;  
        this.властивість2 = значення2;  
        . . .  
    }
```

```
//-- Опис методів (спільні дані для усіх об'єктів) --
```

```
    функція-метод_1 (форм.парам.)  
    {  
        . . .  
    }  
    функція-метод_2 (форм.парам.)  
    {  
        . . .  
    }  
    . . .  
}
```

```
//---- створення об'єкта ----
```

```
об'єкт = new Назва класу (... параметри ...)
```

1. Створюємо клас (називаємо з великої літери)

Приклад. Розробити клас «Передбачувач».

Користувач задає масив можливих передбачень і інтерв

Об'єкт дозволяє кожні вказані кількість секунд отримувати передбачення

Властивості :

- масив можливих передбачень,
- інтервал між передбаченнями

Методи:

- вибір випадкового передбачення
- метод *run*, що ініціює запуск таймера і генерування передбачень

2.Описуємо конструктор

Приклад. Розробити клас «Передбачувач».

Користувач задає масив можливих передбачень і інтервал. Об'єкт дозволяє кожні вказані кількість секунд отримувати передбачення.

Властивості :

- масив можливих передбачень,
- інтервал між передбаченнями

Методи:

- вибір випадкового передбачення
- метод *run*, що ініціює запуск таймера і генерування передбачень

```
class Predictor {  
    constructor(predictionsList, interval) {  
  
    }  
}
```

Як параметри передаємо дані, які необхідно знати при створенні об'єкта.

3. Поступово описуємо властивості (всередині конструктора)

Приклад. Розробити клас «Передбачувач».

Користувач задає масив можливих передбачень і інтервал. Об'єкт дозволяє кожні вказані кількість секунд отримувати передбачення.

Властивості:

- масив можливих передбачень,
- інтервал між передбаченнями

Методи:

- вибір випадкового передбачення
- метод *run*, що ініціює запуск таймера і генерування передбачень

```
class Predictor {  
    constructor(predictionsList, interval) {  
        // Властивості:  
        this.predictionsList = predictionsList;  
    }  
}
```

this. назва_властивості = початкове значення

3. Поступово описуємо властивості (всередині конструктора)

Приклад. Розробити клас «Передбачувач».

Користувач задає масив можливих передбачень і інтерв.
Об'єкт дозволяє кожні вказані кількість секунд отримувати передбачення

Властивості :

- масив можливих передбачень,
- інтервал між передбаченнями

Методи:

- вибір випадкового передбачення
- метод *run*, що ініціює запуск таймера і генерування передбачень

```
class Predictor {  
    constructor(predictionsList, interval) {  
        // Властивості :  
        this.predictionsList = predictionsList  
        this.interval = interval  
    }  
}
```

this. назва_властивості = початкове значення

4. Поступово описуємо методи (всередині класу) (після конструктора)

Приклад. Розробити клас «Передбачувач».

Користувач задає масив можливих передбачень і інтерв.
Об'єкт дозволяє кожні вказані кількість секунд отримувати передбачення

Властивості:

- масив можливих передбачень,
- інтервал між передбаченнями

Методи:

- вибір випадкового передбачення
- метод *run*, що ініціює запуск таймера і генерування передбачень

```
class Predictor {  
  constructor(predictionsList, interval) {  
    // Властивості:  
    this.predictionsList = predictionsList  
    this.interval = interval  
  }  
  
  // -----Методи: -----  
  // ----- вибір випадкового передбачення -----  
  getRandomPrediction() {  
    const randomIndex = Math.floor(  
      Math.random() * this.predictionsList.length  
    )  
    return this.predictionsList[randomIndex]  
  }  
  
  // ----- назва_метода (що_доатково_треба_знати) -----  
  .....  
}
```


4. Поступово описуємо методи (всередині класу) (після конструктора)

Приклад. Розробити клас «Передбачувач».

Користувач задає масив можливих передбачень і інтервал.
Об'єкт дозволяє кожні вказані кількості секунд отримувати передбачення

Властивості:

- масив можливих передбачень,
- інтервал між передбаченнями

Методи:

- вибір випадкового передбачення
- метод *run*, що ініціює запуск таймера і генерування передбачень

```
class Predictor {  
  constructor(predictionsList, interval) {  
    // Властивості:  
    this.predictionsList = predictionsList  
    this.interval = interval  
  }  
  
  // -----Методи: -----  
  // ----- вибір випадкового передбачення -----  
  getRandomPrediction() {  
    const randomIndex = Math.floor(  
      Math.random() * this.predictionsList.length  
    )  
    return this.predictionsList[randomIndex]  
  }  
  // ----- метод run, що ініціює запуск таймера і  
  // ----- генерування передбачень -----  
  run() {  
    setInterval(() => {  
      alert(this.getRandomPrediction())  
    }, this.interval * 1000)  
  }  
}
```

назва_метода (що_доатково_треба_знати) {

}

5. Створюємо об'єкт з використанням класу

Приклад. Розробити клас «Передбачувач».

Користувач задає масив можливих передбачень і інтервал. Об'єкт дозволяє кожні вказані кількість секунд отримувати передбачення.

Властивості:

- масив можливих передбачень,
- інтервал між передбаченнями

Методи:

- вибір випадкового передбачення
- метод *run*, що ініціює запуск таймера і генерування передбачень

let об'єкт = new назва_класу (що_треба_передати)

```
class Predictor {  
  constructor(predictionsList, interval) {  
    // Властивості:  
    this.predictionsList = predictionsList  
    this.interval = interval  
  }  
  
  // ----- Методи: -----  
  // ----- вибір випадкового передбачення -----  
  getRandomPrediction() {  
    const randomIndex = Math.floor(  
      Math.random() * this.predictionsList.length  
    )  
    return this.predictionsList[randomIndex]  
  }  
  // ----- метод run, що ініціює запуск таймера і  
  // ----- генерування передбачень -----  
  run() {  
    setInterval(() => {  
      alert(this.getRandomPrediction())  
    }, this.interval * 1000)  
  }  
}
```

let p1 = new Predictor(['love', 'money', 'friends', 'PEACE'], 2)
p1.run()

Задача. Створити клас TTime для роботи із часом у форматі “години:хвилини”. Час представляється структурою із двома полями. Реалізувати методи збільшення/зменшення часу на певну кількість годин чи хвилин.

Приклад. Створити об'єкт учень.

----- Властивості - характеристики -----

- ПІБ (прізвище, ім'я, по-батькові)
- клас, у якому навчається
- вік
- середній бал

----- Методи (функціональні можливості) -----

- визначення того, ким він є (відмінник, хорошист, ...)
- визначити кількості років до закінчення школи

Ці поля не можуть мати довільні значення !!!!

- клас (від 1 до 11)
- вік (наприклад від 7 до 17 років)
- середній бал (від 0 до 12)

Для захисту використати приватні поля!!!

Інкапсуляція – базовий принцип об'єктно-орієнтованого програмування, згідно з яким поля об'єкта є внутрішніми даними і прямий доступ до них ззовні повинен бути заборонений.

- Інкапсуляція реалізується з використанням **приватних (закритих) полів**
- Звертання до закритих полів повинно здійснюватися з використанням спеціальних методів (геттерів (*get*) та сеттерів (*set*))

Приватні поля

- приватні поля можуть бути використані тільки всередині інших методів цього ж класу!!
- імена приватних полів починаються з символу «#»
#приватне_поле
- для доступу до приватних полів (зчитування та зміни значень) як правило описують **відкриту властивість** (набір спеціальних функцій: геттерів і сеттерів з однаковою назвою)
- для отримання значення приватного поля можна використати (не обов'язково) спеціальну функцію геттер

```
get назва_відкритої_властивості () {  
    .....  
}
```

- для контрольованої зміни значень приватних полів (не обов'язково) використовуємо спеціальну функцію сеттер

```
set назва_відкритої_властивості ( нове_значення_яке_хочемо_зберегти ) {  
    .....  
}
```

Приклад. Створити об'єкт учень.

----- Властивості - характеристики -----

- ПІБ (прізвище, ім'я, по-батькові)
- клас, у якому навчається
- вік
- середній бал

Розробимо захищену властивість
«вік» - Age
Ця властивість не може бути меншою
за мінімальний вік учня minAge

----- Методи (функціональні можливості) -----

- визначення того, ким він є (відмінник, хорошист, ...)
- визначити кількості років до закінчення школи

Загальна форма	Приклад
<code>class Назва класу {</code>	<code>class Pupil {</code>
<code>//---1) опис приватних полів об'єкта з використанням "<u>#</u>" ---</code> <code>#ім'я закритого поля</code>	
	<div>1) описуємо приватне поле (його ім'я повинна починатися з символу «#»)</div>
<code>}</code>	<code>}</code>

Загальна форма

Приклад

`class Назва класу {``class Pupil {``//---1) опис приватних полів об'єкта з використанням " # " ---``//---1) опис приватних полів``#ім'я закритого поля``#age`

1) описуємо приватне поле (його ім'я повинна починатися з символу «#»)

`}``}`

Загальна форма	Приклад
<pre>class Назва класу { //---1) опис приватних полів об'єкта з використанням "<u>#</u>" --- #ім'я закритого поля </pre>	<pre>class Pupil { //---1) опис приватних полів #age</pre>
<pre> //--- 2) метод зчитування значення закритого поля(getter) //(дозволяє ззовні отримати значення закритого поля) get ім'я властивості () { return this. #ім'я закритого поля }</pre>	
}	}

Загальна форма

```
class Назва класу {  
    //---1) опис приватних полів об'єкта з використанням "#" ---  
    #ім'я закритого поля  
    . . . . .  
}
```

```
//--- 2) метод зчитування значення закритого поля(геттер)  
//(дозволяє ззовні отримати значення закритого поля)  
get ім'я властивості ( )  
{  
    return this. #ім'я закритого поля  
}
```

Приклад

```
class Pupil {  
    //---1) опис приватних полів  
    #age
```

```
//--- 2) метод зчитування значення закритого поля(геттер)  
get Age() {  
    return this.#age  
}
```

2) описуємо метод – геттер
для **зчитування**
приватного поля

Загальна форма

```
class Назва класу {
```

```
//---1) опис приватних полів об'єкта з використанням " # " ---
```

```
#ім'я закритого поля
```

```
.....
```

```
//--- 2) метод зчитування значення закритого поля(геттер)
```

```
//(дозволяє ззовні отримати значення закритого поля)
```

```
get ім'я властивості ( )
```

```
{
```

```
    return this. #ім'я закритого поля
```

```
}
```

```
//--- 3) метод запису значення закритого поля (сеттер)
```

```
//(дозволяє перевірити коректність значення і зберегти)
```

```
set ім'я властивості (нове_значення)
```

```
{
```

```
    //---- якщо нове значення некоректне, то генеруємо
```

```
    //---- виключну ситуацію
```

```
    if(нове_значення-не_коректне)
```

```
        throw new Error('Значення некоректне')
```

```
    this. #ім'я закритого поля = нове_значення
```

```
}
```

```
}
```

Приклад

```
class Pupil {
```

```
//---1) опис приватних полів
```

```
#age
```

```
//--- 2) метод зчитування значення закритого поля(геттер)
```

```
get Age() {
```

```
    return this.#age
```

```
}
```

```
}
```


Загальна форма

```
class Назва класу {  
  //---1) опис приватних полів об'єкта з використанням " # " ---  
  #ім'я закритого поля  
  . . . . .  
}
```

```
//--- 2) метод зчитування значення закритого поля(геттер)  
//(дозволяє ззовні отримати значення закритого поля)  
get ім'я властивості ( )  
{  
  return this. #ім'я закритого поля  
}
```

```
//--- 3) метод запису значення закритого поля (сеттер)  
//(дозволяє перевірити коректність значення і зберегти)  
set ім'я властивості (нове_значення)  
{  
  //---- якщо нове значення некоректне, то генеруємо  
  //---- виключну ситуацію  
  
  if(нове_значення-не_коректне)  
    throw new Error('Значення некоректне')  
  
  this. #ім'я закритого поля = нове_значення  
}
```

}

Приклад

```
class Pupil {  
  //---1) опис приватних полів  
  #age
```

```
//--- 2) метод зчитування значення закритого поля(геттер)  
get Age() {  
  
  return this.#age  
}
```

```
//--- 3) метод запису значення закритого поля (сеттер)  
set Age(newAgeValue) {  
  //3) описуємо метод-сеттер  
  //для зміни приватного поля  
  
  if (newAgeValue < this.minAge)  
    throw new Error('Значення віку учня некоректне')  
  
  this.#age = newAgeValue  
}
```

}

Загальна форма

```

class Назва класу {
  //---1) опис приватних полів об'єкта з використанням " # " ---
  #ім'я закритого поля
  . . . . .
  constructor(початкове_значення_властивості){
    //--- 4) У конструкторі початкове значення присвоюємо
    // не напряму у закрите поле, а властивості (буде перевірка)
    this. ім'я властивості = початкове_значення_властивості
    . . . . .
  }
  //--- 2) метод зчитування значення закритого поля(геттер)
  //(дозволяє ззовні отримати значення закритого поля)
  get ім'я властивості ( )
  {
    return this. #ім'я закритого поля
  }
  //--- 3) метод запису значення закритого поля (сеттер)
  //(дозволяє перевірити коректність значення і зберегти)
  set ім'я властивості (нове_значення)
  {
    //---- якщо нове значення некоректне, то генеруємо
    //---- виключну ситуацію

    if(нове_значення-не_коректне)
      throw new Error('Значення некоректне')

    this. # ім'я закритого поля = нове_значення
  }
  . . . . .
}

```

Приклад

```

class Pupil {
  //---1) опис приватних полів
  #age

  //--- 2) метод зчитування значення закритого поля(геттер)
  get Age() {
    return this.#age
  }
  //--- 3) метод запису значення закритого поля (сеттер)
  set Age(newAgeValue) {
    if (newAgeValue < this.minAge)
      throw new Error('Значення віку учня некоректне')

    this.#age = newAgeValue
  }
}

```

Загальна форма

```

class Назва класу {
  //---1) опис приватних полів об'єкта з використанням " # " ---
  #ім'я закритого поля
  . . . . .
  constructor(початкове_значення_властивості){
    //--- 4) У конструкторі початкове значення присвоюємо
    // не напряму у закрите поле, а властивості (буде перевірка)
    this. ім'я властивості = початкове_значення_властивості
    . . . . .
  }

  //--- 2) метод зчитування значення закритого поля(геттер)
  //(дозволяє ззовні отримати значення закритого поля)
  get ім'я властивості ( )
  {
    return this. #ім'я закритого поля
  }

  //--- 3) метод запису значення закритого поля (сеттер)
  //(дозволяє перевірити коректність значення і зберегти)
  set ім'я властивості (нове_значення)
  {
    //---- якщо нове значення некоректне, то генеруємо
    //---- виключну ситуацію

    if(нове_значення-не_коректне)
      throw new Error('Значення некоректне')

    this. # ім'я закритого поля = нове_значення
  }
  . . . . .
}

```

Приклад

```

class Pupil {
  //---1) опис приватних полів
  #age
  . . . . .
  constructor(initialAge, minAge = 7) {
    this.minAge = minAge
    //--- 4) У конструкторі початкове значення присвоюємо
    this.Age = initialAge
  }

  //--- 2) метод зчитування значення закритого поля(геттер)
  get Age() {
    return this.#age
  }

  //--- 3) метод запису значення закритого поля (сеттер)
  set Age(newAgeValue) {
    if (newAgeValue < this.minAge)
      throw new Error('Значення віку учня некоректне')

    this.#age = newAgeValue
  }
}

```

4) у конструкторі при ініціалізації поля використовуємо властивість

Загальна форма	Приклад
<pre>class Назва класу { //---1) опис приватних полів об'єкта з використанням " # " --- #ім'я закритого поля constructor(початкове_значення_властивості){ //--- 4) У конструкторі початкове значення присвоюємо // не напряму у закрите поле, а властивості (буде перевірка) this. ім'я властивості = початкове_значення_властивості } //--- 2) метод зчитування значення закритого поля(геттер) //(дозволяє ззовні отримати значення закритого поля) get ім'я властивості () { return this. #ім'я закритого поля } //--- 3) метод запису значення закритого поля (сеттер) //(дозволяє перевірити коректність значення і зберегти) set ім'я властивості (нове_значення) { //---- якщо нове значення некоректне, то генеруємо //---- виключну ситуацію if(нове_значення-не_коректне) throw new Error('Значення некоректне') this. # ім'я закритого поля = нове_значення } }</pre>	<pre>class Pupil { //---1) опис приватних полів #age constructor(initialAge, minAge = 7) { this.minAge = minAge //--- 4) У конструкторі початкове значення присвоюємо this.Age = initialAge } //--- 2) метод зчитування значення закритого поля(геттер) get Age() { return this.#age } //--- 3) метод запису значення закритого поля (сеттер) set Age(newAgeValue) { if (newAgeValue < this.minAge) throw new Error('Значення віку учня некоректне') this.#age = newAgeValue } }</pre>
<pre>. назва_властивості = нове_значення</pre>	<pre>let p1 = new Pupil(10) p1.Age = 20 //Буде викликано set</pre> <div data-bbox="1854 1096 2265 1318"> <p>При встановленні значення автоматично викликається метод сеттер</p> </div>

Загальна форма	Приклад
<pre>class Назва класу { //---1) опис приватних полів об'єкта з використанням "#" --- #ім'я закритого поля constructor(початкове_значення_властивості){ //--- 4) У конструкторі початкове значення присвоюємо // не напряму у закрите поле, а властивості (буде перевірка) this. ім'я властивості = початкове_значення_властивості } //--- 2) метод зчитування значення закритого поля(геттер) //(дозволяє ззовні отримати значення закритого поля) get ім'я властивості () { return this. #ім'я закритого поля } //--- 3) метод запису значення закритого поля (сеттер) //(дозволяє перевірити коректність значення і зберегти) set ім'я властивості (нове_значення) { //---- якщо нове значення некоректне, то генеруємо //---- виключну ситуацію if(нове_значення-не_коректне) throw new Error('Значення некоректне') this. # ім'я закритого поля = нове_значення } }</pre>	<pre>class Pupil { //---1) опис приватних полів #age constructor(initialAge, minAge = 7) { this.minAge = minAge //--- 4) У конструкторі початкове значення присвоюємо this.Age = initialAge } //--- 2) метод зчитування значення закритого поля(геттер) get Age() { return this.#age } //--- 3) метод запису значення закритого поля (сеттер) set Age(newAgeValue) { if (newAgeValue < this.minAge) throw new Error('Значення віку учня некоректне') this.#age = newAgeValue } }</pre>
<pre> назва_властивості </pre>	<pre> let p1 = new Pupil(10) p1.Age = 20 //Буде викликано set let s = p1.Age //Буде викликано get </pre>

При зчитуванні значення автоматично викликається метод геттер

Приклад. Створити об'єкт учень.

----- Властивості - характеристики -----

- ПІБ (прізвище, ім'я, по-батькові)
- клас, у якому навчається
- вік
- середній бал

----- Методи (функціональні можливості) -----

- визначення того, ким він є (відмінник, хорошист, ...)
- визначити кількості років до закінчення школи

Ці поля не можуть мати довільні значення !!!!

- клас (від 1 до 11)
- вік (наприклад від 7 до 17 років)
- середній бал (від 0 до 12)

Для захисту використати приватні поля!!!

Приклад. Створити клас «Клієнт»

(ім'я – довільний доступ (відкрите поле),

номер рахунку – тільки для читання,

кількість грошей – контрольований доступ (і читання і запис))

Задача 2. Створити клас **Product**, що представляє товар на складі

поля:

Назва товару

Кількість одиниць

Ціна одиниці

методи:

зменшення кількості товару

збільшення кількості товару

визначення вартості вказаної кількості товару

нарахування вказаної знижки (у відсотках)

визначення загальної вартості товару

1. Створити клас TMoney для роботи з грошовими сумами. Сума повинна зберігатися у вигляді доларового еквіваленту. Реалізувати методи додавання/вилучення грошової маси, вказуючи необхідну суму у гривнях, та визначення курсу долара, при якому сума у гривнях збільшиться на 100. Курс долара зберігати в окремому полі.

Геттери і сеттери у літералах

Загальна форма	Приклад
<pre>{ <u>властивість1</u>: <u>значення1</u>, <u>властивість2</u>: <u>значення2</u>, ... get <u>властивість</u> () { . . . }, set <u>властивість</u> (<u>нове_значення_властивості</u>) { . . . } }</pre>	<pre>let user = { firstName: "Іван", surname: "Сірко", get fullName() { return this.firstName + ' ' + this.surname; }, set fullName(value) { var split = value.split(' ') this.firstName = split[0] this.surname = split[1] } }</pre>
	<pre>alert(user.fullName); // Іван Сірко user.fullName = "Петро Галушка" alert(user.firstName) // Петро alert(user.surname) // Галушка</pre>

Класи в TypeScript

Клас - це шаблон для створення об'єктів з певними властивостями та методами. В TypeScript класи підтримують типізацію, модифікатори доступу, наслідування та конструктори.

Загальна форма

```
class ClassName {  
    властивість1: Type1;  
    властивість2: Type2;  
  
    constructor(властивість1: Type1, властивість2: Type2) {  
        this.властивість1 = властивість1;  
        this.властивість2 = властивість2;  
    }  
  
    метод1(): ReturnType {  
        // код  
    }  
}
```

Приклад

```
class Person {  
    name: string;  
    age: number;  
  
    constructor(name: string, age: number) {  
        this.name = name;  
        this.age = age;  
    }  
  
    greet(): void {  
        console.log(`Привіт, мене звати ${this.name}`);  
    }  
}  
  
const p1 = new Person("Іван", 30);  
p1.greet(); // Привіт, мене звати Іван
```

Модифікатори доступу

- `public` - доступна з будь-якого місця (за замовчуванням)
- `private` - доступна лише всередині класу
- `protected` - доступна всередині класу та у підкласах

```
class Employee {  
    public name: string;  
    private salary: number;  
    protected department: string;  
  
    constructor(name: string, salary: number, department: string) {  
        this.name = name;  
        this.salary = salary;  
        this.department = department;  
    }  
  
    getSalary(): number {  
        return this.salary;  
    }  
}
```


Геттери і сеттери Геттер (getter) - це метод, який дозволяє отримати значення приватної або захищеної властивості об'єкта.
Сеттер (setter) - це метод, який дозволяє встановити або змінити значення приватної або захищеної властивості, з можливістю перевірки або обробки значень.
Використання геттерів і сеттерів підвищує інкапсуляцію, дозволяючи контролювати доступ до внутрішніх властивостей об'єкта.

Загальна форма

```
class ClassName {  
    private _property: Type;  
  
    constructor(value: Type) {  
        this._property = value;  
    }  
  
    // Геттер  
    get property(): Type {  
        return this._property;  
    }  
  
    // Сеттер  
    set property(value: Type) {  
        // можна додати перевірку  
        this._property = value;  
    }  
}
```

Приклад

```
class Rectangle {  
    private _width: number;  
    private _height: number;  
  
    constructor(width: number, height: number) {  
        this._width = width;  
        this._height = height;  
    }  
  
    get area(): number {  
        return this._width * this._height;  
    }  
  
    set width(value: number) {  
        if (value > 0) this._width = value;  
    }  
}  
  
const rect = new Rectangle(10, 5);  
console.log(rect.area); // 50  
rect.width = 20;  
console.log(rect.area); // 100
```

Приватні поля JavaScript (#)

- Приватні поля - це спеціальний синтаксис для створення прихованих властивостей у класах.
- Вони позначаються #ім'я і доступні лише всередині класу.
- На відміну від private у TypeScript, який існує лише на етапі типізації, # - це реальна обмежена видимість у runtime (JavaScript).

Загальна форма

```
class ClassName {  
  #privateField: Type;    // приватне поле  
  publicField: Type;      // звичайне публічне поле  
  
  constructor(value: Type) {  
    this.#privateField = value;  
  }  
  
  getPrivateField() {  
    return this.#privateField;  
  }  
}
```

Приклад

```
class BankAccount {  
  #balance: number; // приватне поле  
  
  constructor(initialBalance: number) {  
    this.#balance = initialBalance;  
  }  
  
  deposit(amount: number) {  
    this.#balance += amount;  
  }  
  
  getBalance() {  
    return this.#balance;  
  }  
}
```

```
const account = new BankAccount(1000);  
  
account.deposit(500);  
console.log(account.getBalance()); // 1500 ✅  
  
console.log(account.balance); // ❌ undefined
```

Використання readonly

```
class Point {  
    constructor(  
        public x: number,  
        public y: number,  
        readonly label: string  
    ) {}  
}  
  
const p = new Point(10, 20, "A");  
console.log(p.label); // "A"  
// p.label = "B"; // ❌ Помилка: readonly властивість
```

Параметричні властивості в TypeScript

- Параметричні властивості дозволяють оголошувати і ініціалізувати поля класу напрямку в конструкторі.
- Для цього перед параметром конструктора додається модифікатор доступу (public, private, protected, readonly).
- TypeScript автоматично:
 - Створює властивість у класі.
 - Ініціалізує її значенням із конструктора.

Загальна форма

```
class ClassName {  
  constructor(  
    public field1: Type,  
    private field2: Type,  
    protected field3: Type,  
    readonly field4: Type  
  ) {}  
}
```

Це еквівалентно

```
class ClassName {  
  public field1: Type;  
  private field2: Type;  
  protected field3: Type;  
  readonly field4: Type;  
  
  constructor(field1: Type, field2: Type, field3: Type, field4: Type) {  
    this.field1 = field1;  
    this.field2 = field2;  
    this.field3 = field3;  
    this.field4 = field4;  
  }  
}
```

Приклад

```
class Person {  
  constructor(  
    public name: string,  
    private age: number  
  ) {}  
  
  introduce() {  
    console.log(`Мене звати ${this.name}, мені ${this.age} років`);  
  }  
}  
  
const p = new Person("Оля", 25);  
console.log(p.name);    // ✅ "Оля"  
// console.log(p.age); // ❌ Помилка: private  
p.introduce(); // Мене звати Оля, мені 25 років
```