

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

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 інтервал

Об'єкт дозволяє кожні вказані кількість секунд отримувати передбачення

Властивості :

- масив можливоих 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 можливоих передбачень,
- інтервал між передбаченнями

Методи:

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

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

Як парметри 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

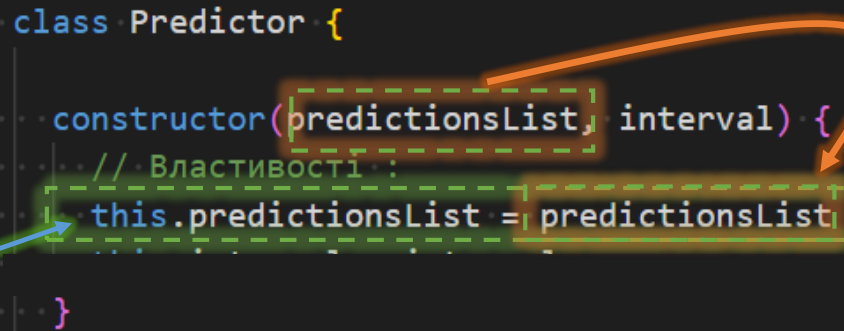
Властивості :

- масив можливоих 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 інтервал

Об'єкт дозволяє кожні вказані кількість секунд отримувати передбачення

Властивості :

- масив можливоих 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 інтервал.
Об'єкт дозволяє кожні вказані кількість секунд отримувати передбачення

Властивості :

- масив можливоих 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 кількість секунд отримувати передбачення

Властивості :

- масив можливоих передбачень,
- інтервал між передбаченнями

Методи:

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

```
назва_метода (що_доатково_треба_знати) {  
    .....  
}
```

```
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)  
    }  
}
```


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 можливоих 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

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 приватні поля (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 ) {  
    .....  
}
```

Загальна форма	Приклад
<code>class Назва класу {</code>	
<code>//---1) опис <u>приватних полів</u> 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
<pre>class Назва класу {</pre>	
<pre>//---1) опис приватних полів об'єкта з використанням "<u>#</u>" --- #ім'я закритого поля</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	Приклад
<pre>class Назва класу {</pre>	
<pre>//---1) опис <i>приватних полів</i> об'єкта з використанням "<u>#</u>" --- #<i>ім'я закритого поля</i></pre>	
<pre>//--- 2) метод зчитування значення закритого поля(геттер) //(дозволяє ззовні отримати значення закритого поля) get <i>ім'я властивості</i> () { return this. #<i>ім'я закритого поля</i> }</pre>	
<pre>//--- 3) метод запису значення закритого поля (сеттер) //(дозволяє перевірити коректність значення і зберегти) set <i>ім'я властивості</i> (<i>нове_значення</i>) { //---- якщо нове значення некоректне, то генеруємо //---- виключну ситуацію if(<i>нове_значення</i>-не_коректне) throw new Error('Значення некоректне') this. #<i>ім'я закритого поля</i> = <i>нове_значення</i> }</pre>	
<pre>}</pre>	<pre>}</pre>

Загальна форма	Приклад
<pre>class Назва класу { //---1) опис приватних полів об'єкта з використанням <code>"#"</code> --- #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 . . . }</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>	
<pre>//--- 3) метод запису значення закритого поля (setter) //(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> . . . }</pre>	<pre>}</pre>

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

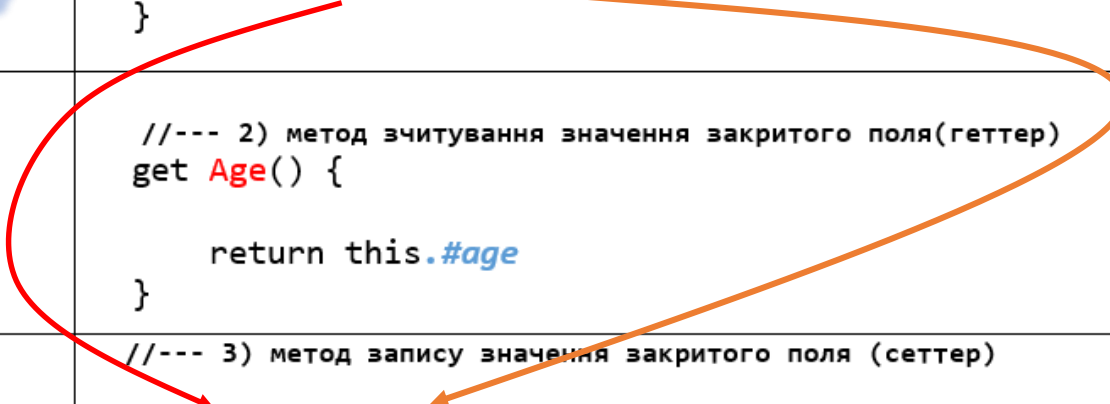
Загальна форма	Приклад
<pre>class Назва класу {</pre>	<pre>class Pupil {</pre>
<pre>//---1) опис приватних полів об'єкта з використанням " # " --- #ім'я закритого поля</pre>	
<pre>constructor(початкове_значення_властивості){ //--- 4) У конструкторі початкове значення присвоюємо // не напряму у закрите поле, а властивості (буде перевірка) this. ім'я властивості = початкове_значення_властивості }</pre>	<div data-bbox="1605 285 2497 371">Описуємо клас «<u>Pupil</u>»</div>
<pre>//--- 2) метод зчитування значення закритого поля(геттер) //(дозволяє ззовні отримати значення закритого поля) get ім'я властивості () { return this. #ім'я закритого поля }</pre>	
<pre>//--- 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>. }</pre>	<pre>}</pre>

Загальна форма	Приклад
<pre>class Назва класу {</pre>	<pre>class Pupil {</pre>
<pre>//---1) опис приватних полів об'єкта з використанням " # " --- #ім'я закритого поля </pre>	<pre>//---1) опис приватних полів #age</pre>
<pre>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 }</pre>	<div>1) описуємо приватне поле # age</div>
<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>	
<pre>//--- 3) метод запису значення закритого поля (setter) //(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>. }</pre>	<pre>}</pre>

Загальна форма	Приклад
<pre>class Назва класу {</pre>	<pre>class Pupil {</pre>
<pre>//---1) опис <i>приватних полів</i> об'єкта з використанням " # " --- #<i>ім'я закритого поля</i></pre>	<pre>//---1) опис <i>приватних полів</i> <i>#age</i></pre>
<pre>constructor(<i>початкове_значення_властивості</i>){ //--- 4) У конструкторі початкове значення присвоюємо // не напряму у закрите поле, а властивості (буде перевірка) this. <i>ім'я властивості</i> = <i>початкове_значення_властивості</i> }</pre>	<pre>.....</pre>
<pre>//--- 2) метод зчитування значення закритого поля(геттер) //(дозволяє ззовні отримати значення закритого поля) get <i>ім'я властивості</i> () { return this. <i>#ім'я закритого поля</i> }</pre>	<pre>//--- 2) метод зчитування значення закритого поля(геттер) get <i>Age</i>() { return this.<i>#age</i> }</pre>
<pre>//--- 3) метод запису значення закритого поля (сеттер) //(дозволяє перевірити коректність значення і зберегти) set <i>ім'я властивості</i> (<i>нове_значення</i>) { //---- якщо нове значення некоректне, то генеруємо //---- виключну ситуацію if(<i>нове_значення</i>-не_коректне) throw new Error('Значення некоректне') this. <i>#ім'я закритого поля</i> = <i>нове_значення</i> }</pre>	<pre>.....</pre> <div data-bbox="1600 829 2270 1029">2) описуємо метод – геттер для зчитування приватного поля</div>
<pre>..... }</pre>	<pre>..... }</pre>

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

3) описуємо метод – сеттер
для зміни приватного поля

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 } </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 } </pre> <div data-bbox="2091 51 2499 258" style="border: 1px dashed blue; padding: 5px; position: absolute; top: 36px; right: 36px;"> 4) у конструкторі при ініціалізації поля використовуємо властивість </div> 
<pre> //--- 2) метод зчитування значення закритого поля(геттер) //(дозволяє ззовні отримати значення закритого поля) get ім'я властивості () { return this. #ім'я закритого поля } </pre>	<pre> //--- 2) метод зчитування значення закритого поля(геттер) get Age() { return this.#age } </pre>
<pre> //--- 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> //--- 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> } </pre>	<pre> } </pre>

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(getter) //(дозволяє ззовні отримати значення закритого поля) get ім'я властивості () { return this. #ім'я закритого поля } //--- 3) метод запису значення закритого поля (setter) //(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(getter) get Age() { return this.#age } //--- 3) метод запису значення закритого поля (setter) set Age(newAgeVaLue) { if (newAgeVaLue < this.minAge) throw new Error('Значення віку учня некоректне') this.#age = newAgeVaLue } let p1 = new Pupil(10) p1.Age = 20 //Буде викликано set</pre>
<div>об'єкт . назва_властивості = нове_значення</div>	<div>При встановленні значення автоматично викликається метод setter</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>
<div>змінна = об'єкт . назва_властивості</div>	<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 властивість () { . . . }, set властивість (нове_значення_властивості) { . . . } }</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>