Классы в typescript

Основны:

Класс в TypeScript — это шаблон для создания объектов. Он определяет свойства и методы, которые будут доступны для экземпляров этого класса.

Пример создания класса:

```
class Person {
    name: string;
    age: number;

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

    greet() {
        console.log(`Привет, меня зовут ${this.name} и мне ${this.age}
лет.`);
    }
}
```

- Свойства: `name` и `age` — это свойства класса, которые определяют характеристики объекта.

- Конструктор: `constructor` — это специальный метод, который вызывается при создании экземпляра класса. Он инициализирует свойства объекта.

- Методы: `greet` — это метод класса, который выполняет определенное действие.

Создание экземпляров класса:

```
const person1 = new Person("Иван", 30);
person1.greet();
```

Модификаторы доступа

- public: Доступен из любого места. Это значение по умолчанию.

- private: Доступен только внутри класса.

- protected: Доступен внутри класса и его подклассов.

```
class Animal {
    protected species: string;

    constructor(species: string) {
        this.species = species;
    }
}

class Dog extends Animal {
    bark() {
        console.log(`Гав! Я ${this.species}.`);
    }
}

const dog = new Dog("собака");
dog.bark();
```

Использование emit в vue

emit — это метод, который позволяет дочернему компоненту отправлять события родительскому компоненту. Это особенно полезно, когда дочерний компонент должен уведомить родительский о каком-либо действии или изменении состояния.

Синтаксис использования emit:

Метод emit вызывается на экземпляре компонента и принимает два основных параметра:

1. Имя события: строка, представляющая имя события.

2. Данные (опционально): любые данные, которые вы хотите передать с событием.

```
this.$emit('eventName', eventData);
```

Пример использования emit:

Создание дочернего компонента:

Создадим дочерний компонент `ChildComponent.vue`, который будет содержать кнопку. При нажатии на кнопку он будет отправлять событие `buttonClicked`.

```
<template>
  <div>
    <button @click="handleClick">Нажми меня</button>
  </div>
</template>

<script lang="ts">
import { defineComponent } from 'vue';

export default defineComponent({
  methods: {
    handleClick() {
      this.$emit('buttonClicked', 'Дочерний компонент нажал кнопку!');
    },
  },
});
```

Создание родительского компонента:

Теперь создадим родительский компонент `ParentComponent.vue`, который будет слушать событие `buttonClicked` от дочернего компонента.

```
<template>
  <div>
    <h1>Родительский компонент</h1>
    <ChildComponent @buttonClicked="handleButtonClick" />
  </div>
</template>

<script lang="ts">
import { defineComponent } from 'vue';
import ChildComponent from './ChildComponent.vue';

export default defineComponent({
  components: { ChildComponent },
```

```
  methods: {
    handleButtonClick(message: string) {
      console.log(message);
    },
  },
});
</script>
```

Пояснение:

1. При нажатии на кнопку в **ChildComponent** вызывается метод **handleClick**.

2. Этот метод использует **this.$emit** для отправки события **buttonClicked** вместе с сообщением.

3. Родительский компонент **ParentComponent** слушает событие **buttonClicked** с помощью директивы **@buttonClicked**.

4. Когда событие срабатывает, вызывается метод **handleButtonClick**, который получает сообщение и выводит его в консоль.

Создание приложения TaskTracker

1. **Создание иерархии файлов:**

```
src/
├── store/
│   ├── TaskStore.ts
├── components/
│   ├── TaskList.vue
│   ├── TaskItem.vue
├── models/
│   ├── Task.ts
├── views/
│   ├── Home.vue
├── App.vue
├── main.ts
```

2. **Создание файла src/models/Task.ts**

```
export class Task {
     public id: number,
     public title: string,
     public completed: boolean = false
    constructor(id:number,title:string)
    {
        this.id = id;
        this.title = title;
    }
}
```

3. **Создание файла src/store/TaskStore.ts**

```
import { Task } from '../models/Task';

export class TaskStore {
    private tasks: Task[] = [];
    private nextId: number = 1;

    constructor() {
```

```
        // Инициализация с некоторыми задачами
        this.tasks.push(new Task(this.nextId++, 'Изучить
Vue.js'));
        this.tasks.push(new Task(this.nextId++, 'Создать
проект на Vue'));
    }

    // Получить все задачи
    public getTasks(): Task[] {
        return this.tasks;
    }

    // Добавить новую задачу
    public addTask(title: string): void {
        this.tasks.push(new Task(this.nextId++, title));
    }

    // Удалить задачу по ID
    public removeTask(id: number): void {
        this.tasks = this.tasks.filter(task => task.id !==
id);
    }

    // Переключить состояние задачи
    public toggleTask(id: number): void {
        const task = this.tasks.find(task => task.id === id);
        if (task) {
            task.completed = !task.completed;
        }
    }
}
```

4. **Создание представления src/views/Home.vue**

```
<template>
  <div>
    <h1>Планер задач</h1>
    <TaskList :tasks="tasks" @toggle="toggleTask" />
  </div>
</template>

<script lang="ts">
import { defineComponent } from 'vue';
import { Task } from '../models/Task';
import TaskList from '../components/TaskList.vue';
import { TaskStore } from '../services/TaskStore';

export default defineComponent({
  components: { TaskList },
  data() {
    return {
      taskStore: new TaskStore(),
      newTaskTitle: '',
    };
  },
  computed: {
    tasks(): Task[] {
      return this.taskStore.getTasks();
```

```
    },
  },
  methods: {
    toggleTask(task: Task) {
      this.taskStore.toggleTask(task.id);
    },
  },
});
</script>
```

5. **Создание компонента src/components/TaskItem.vue**

```
<template>
  <li>
    <span :class="{ completed: task.completed }"
@click="toggle">{{ task.title }}</span>
  </li>
</template>

<script lang="ts">
import { defineComponent } from 'vue';
import { Task } from '../models/Task';

export default defineComponent({
  props: {
    task: {
      type: Object as () => Task,
      required: true,
    },
  },
  methods: {
    toggle() {
      this.$emit('toggle', this.task);
    },
  },
});
</script>

<style>
.completed {
  text-decoration: line-through;
}
</style>
```

6. **Создание компонента src/components/TaskList.vue**

```
<template>
  <div>
    <h2>Список задач</h2>
    <ul>
      <TaskItem
        v-for="task in tasks"
        :key="task.id"
        :task="task"
        @toggle="toggleTask"
      />
    </ul>
  </div>
</template>
```

```
      <script lang="ts">
      import { defineComponent } from 'vue';
      import { Task } from '../models/Task';
      import TaskItem from './TaskItem.vue';

      export default defineComponent({
        components: { TaskItem },
        props: {
          tasks: {
            type: Array as () => Task[],
            required: true,
          },
        },
        methods: {
          toggleTask(task: Task) {
            task.completed = !task.completed;
          },
        },
      });
</script>
```

Задание

1. Добавьте возможность добавления новых задач: Реализуйте форму, которая позволит пользователю вводить название новой задачи и добавлять её в список.
2. Добавьте возможность удаления задач: Реализуйте функциональность, которая позволит пользователю удалять задачи из списка.