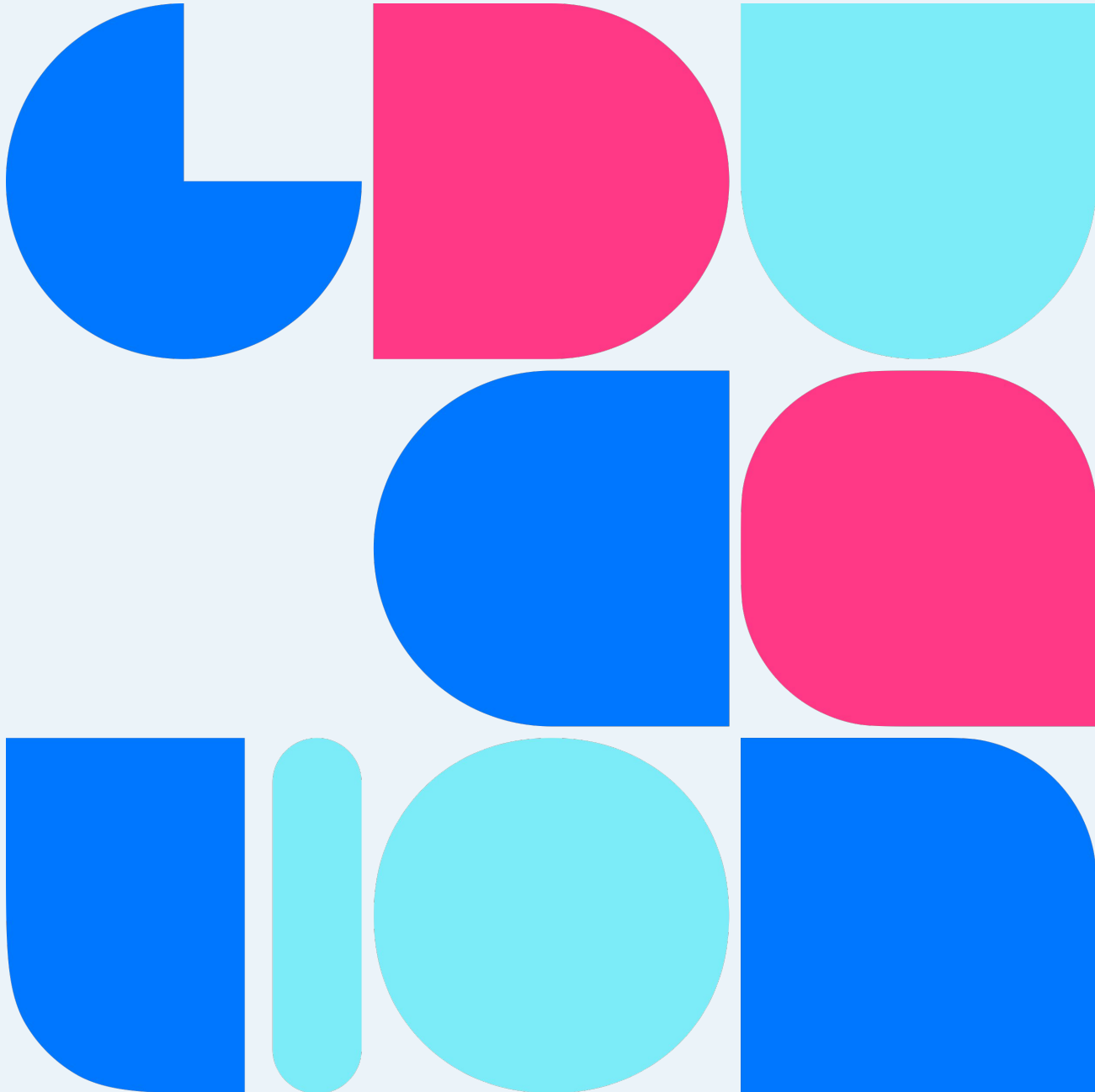
education

Лекция 14
TypeScript 2

Дмитрий Зайцев
Мартин Комитски



План на сегодня

- Ещё больше TS

План на сегодня

1. Возможности TypeScript
    - Модули
    - Utility Types
    - Decorators
    - Mixins
2. React TSX

Минутка бюрократии

- Внимание
- Отметки о посещении занятий
- Обратная связь о лекциях

Модули

Модули

- export
- import
- modules/first.ts
- modules/second.ts
- modules/third.ts
- modules/types.ts

Export

1. `export default () => …`
2. `export {}`
3. `export { … as … }`
4. `export { … as … } from …`
5. `export * as … from …`

Import

1. `import … from …`
2. `import * as … from`
3. `import {...} from …`

Typescript

Внешние модули. Ambient

.D.TS

```
declare module "Backend" {
    export interface ITeacher {
        name: string;
        level: number;
        exp?: number;
    }

    export interface ILesson {
        title: string;
        day: number;
        hours: number;
        teacher: ITeacher;
    }

    export function makeLesson(title: string, day: number, hours: number, teacher: ITeacher): ILesson;
}
```

declare **module “CumbersomeBackend”**

Namespaces

~~Запутывание~~ упорядочивание пространства имен объектов

- namespaces/validators.ts
- namespaces/mergeQuirqs.ts

```
1.  namespace Shapes {
2.      export namespace Polygons {
3.          export class Triangle { }
4.          export class Square { }
5.      }
6.  }
```

1. `const square = new Shapes.Polygons.Square()`

```
1. import polygons = Shapes.Polygons
1. const square = new polygons.Square()
```

- Модули и пространства имен нужны для упорядочивания кода
- Использовать и модули и пространства имен одновременно - избыточно
- Проще всего работать с модулями

Declaration merging

Typescript. Declaration merging

```
1.  interface Car {
2.      manufacturer: string;
3.      name: string;
4.      vehicleInfo: string;
5.  }
6.
7.  interface Car {
8.      horsePower: number;
9.      torque: number;
10.     stickers: string[];
11. }
12.
13. const realCar: Car {
14.     manufacturer: 'VAZ',
15.     name: 'Granta',
16.     vehicleInfo: 'Realnii avtomobil dlya realnoi jizni',
17.     horsePower: 98,
18.     torque: 145,
19.     stickers: ['my life - my rules', 'dolbit normalno'],
20. }
```

Typescript. Declaration merging

НЕФУНКЦИОНАЛЬНЫЕ СВОЙСТВА

- атрибуты интерфейсов должны быть уникальными или одного типа

ФУНКЦИОНАЛЬНЫЕ СВОЙСТВА

- каждое новое свойство считается перегрузкой
- последний интерфейс имеет превосходство над предыдущими
- специализированное свойство будет иметь превосходство над обычными

Typescript. Declaration merging

```
interface Document {
    createElement(tagName: any): Element;  // any
}

interface Document {
    createElement(tagName: "div"): HTMLDivElement;  // specialized
    createElement(tagName: "span"): HTMLSpanElement;  // specialized
}

interface Document {
    createElement(tagName: string): HTMLElement;  // general
    createElement(tagName: "canvas"): HTMLCanvasElement;  // specialized
}
```

Typescript. Declaration merging

```
interface Document {
    createElement(tagName: "canvas"): HTMLCanvasElement;
    createElement(tagName: "div"): HTMLDivElement;
    createElement(tagName: "span"): HTMLSpanElement;
    createElement(tagName: string): HTMLElement;
    createElement(tagName: any): Element;
}
```

Typescript. Declaration merging

```
interface Cloner {
    clone(animal: Animal): Animal;
}

interface Cloner {
    clone(animal: Sheep): Sheep;
}

interface Cloner {
    clone(animal: Dog): Dog;
    clone(animal: Cat): Cat;
}
```

Typescript. Declaration merging

```
interface Cloner {
    clone(animal: Dog): Dog;
    clone(animal: Cat): Cat;
    clone(animal: Sheep): Sheep;
    clone(animal: Animal): Animal;
}
```

Typescript. Namespace merging

```
1.  namespace Prison {
2.      export class Inspector { }
3.      export class Prisoner { }
4.  }
5.
6.  namespace Prison {
7.      export interface Aparts { facilities: string[]; }
8.      export class Yard { facilities: string[] }
9.  }
```

Typescript. Namespace merging

```
namespace Prison {
    export class Inspector { }
    export class Prisoner { }

    export interface Aparts { facilities: string[]; }
    export class Yard { facilities: string[] }
}
```

UTILITY TYPES

\U1F4AA

Typescript

- **PARTIAL**

  Все свойства необязательные

- **READONLY**

  Все свойства – readonly

- **REQUIRED**

  Все свойства обязательны

- **NONNULLABLE**

  Все свойства, кроме null и undefined

Typescript

utilityTypes/commonProps.ts

Typescript

- **PICK**

  Использовать выбранные свойства

- **OMIT**

  Использовать все свойства, кроме выбранных

Typescript

utilityTypes/pickNOmit.ts

Typescript. Record

Присваивание свойств одного типа к другому типу.

```
interface PageInfo {
    title: string;
}

type Page = 'home' | 'about' | 'contact';

const x: Record<Page, PageInfo> = {
    about: { title: 'about' },
    contact: { title: 'contact' },
    home: { title: 'home' },
};
```

Typescript. Exclude. Extract.

EXCLUDE

Взять только те свойства, которых нет во втором объекте

```
1. type T0 = Exclude<"a" | "b" | "c", "a">;  // "b" | "c"
2. type T1 = Exclude<"a" | "b" | "c", "a" | "b">;  // "c"
3. type T2 = Exclude<string | number | (() => void), Function>;  // string | number
```

EXTRACT

Взять только те свойства, которые есть во втором объекте

```
1. type T0 = Extract<"a" | "b" | "c", "a" | "f">;  // "a"
2. type T1 = Extract<string | number | (() => void), Function>;  // () => void
```

Typescript

- **PARAMETERS**
- **CONSTRUCTORPARAMETERS**
- **RETURNTYPE**
- **INSTANCETYPE**
- **THISPARAMETERTYPE**

Conditional types

Infer

Conditional types. Infer

Conditional types

```
type TypeA = { id: string }
type TypeB = { id: number }

type ConditionalType<T> = T extends TypeA ? TypeA : never

type ResultType1 = ConditionalType<TypeA> // TypeA
type ResultType2 = ConditionalType<TypeB> // never
```

Infer in conditional types

```
type TypeA = { id: string }
type TypeB = { id: number }

type InferType<T> = T extends { id: infer P } ? P extends string ? string : number : any

type ResultType1 = InferType<TypeA> // string
type ResultType2 = InferType<TypeB> // number
type ResultType3 = InferType<object> // any
```

Decorators

Typescript. Decorators

OOP Design pattern

Расширение функционала вызываемого объекта без его модификации

O from solid

/oop/commonDecorators.ts

```
const noisy = (func) => {
        const wrapper = (...args) => {
            console.log(-----)
            const res = func(...args)
            console.log(-----)
            return res
        }
    return wrapper
}
```

Typescript. Decorators

>> A Decorator is a special kind of declaration that can be attached to a class declaration, method, accessor, property, or parameter.

```
1. const sum = (a, b) => (a + b)
2. const noisySum = noisy(sum)
3. noisySum(1, 15)
```

Typescript. Decorators

tsc --target ES5 --experimentalDecorators

"experimentalDecorators": true

oop/typescriptDecorators.ts

Typescript. Decorators

```
// first
@first @second myFunction

// second
@first
@second
myFunction
```

Typescript. Сигнатура декоратора

1. target - метод, к которому применяется декоратор
2. key - имя метода
3. value - дескриптор свойства

Typescript. Decorators

Дескриптор свойства

Object.getOwnPropertyDescriptor

oop/typescriptDecorators.ts

СОЗДАНИЕ ДЕКОРАТОРА. Decorator factories

```
1. function enumerable(value: boolean) {
2.     return function (target: any, propertyKey: string, descriptor: PropertyDescriptor) {
3.         descriptor.enumerable = value;
4.     };
5. }
```

Mixins

Примеси для классов

Typescript. Mixins

```
1. function applyMixins(derivedCtor: any, baseCtors: any[]) {
2.     baseCtors.forEach(baseCtor => {
3.         Object.getOwnPropertyNames(baseCtor.prototype).forEach(name => {
4.             Object.defineProperty(derivedCtor.prototype, name,
   Object.getOwnPropertyDescriptor(baseCtor.prototype, name));
5.         });
6.     });
7. }
```

Typescript. Mixins

```
// Activatable Mixin
class Activatable {
    isActive: boolean;
    activate() {
        this.isActive = true;
    }
    deactivate() {
        this.isActive = false;
    }
}
```

Typescript. Mixins

```
// Disposable Mixin
class Disposable {
    isDisposed: boolean;
    dispose() {
        this.isDisposed = true;
    }
}
```

Typescript. Mixins

```
class SmartObject {
    constructor() {
        setInterval(() => console.log(this.isActive + " : " + this.isDisposed), 500);
    }

    interact() {
        this.activate();
    }
}
```

Typescript. Mixins

oop/mixins.ts

```
interface SmartObject extends Disposable, Activatable {}
applyMixins(SmartObject, [Disposable, Activatable]);

const smartObj = new SmartObject();
setTimeout(() => smartObj.interact(), 1000);
```

Typescript.TSX

tsx/button.tsx

Typescript.Типы

https://github.com/microsoft/TypeScript/blob/master/src/lib/dom.generated.d.ts

Typescript. Еще типы

https://github.com/DefinitelyTyped/DefinitelyTyped/tree/master/types

Typescript2?

Вопросы?

Полезные ссылки

- https://www.typescriptlang.org/
- Серия коротких роликов про TS
- ?

Домашнее задание №14

1. Перевести проект на TypeScript
2. ?

Расширенное описание задания, подсказки, а также презентации с лекций всегда есть в репозитории.

Срок сдачи

26 декабря

Мем дня


TYPESCRIPT
// @TS-IGNORE
imgflip.com


How do I feel when I write TypeScript code
TypeScript
Me
DEVS.LOL

Спасибо за внимание!

education

Пока!

Присоединяйтесь к сообществу про образование в VK

- VK Образование

education