Mục tiêu

- 1.1 Classes
- 1.2 Công cụ sửa đổi truy cập (Access modifiers) với: Private, Public và Readonly
- 1.4 Kế thừa (inheritance)
- 1.5 Phương thức ghi đè (Overriding) & Protected modifier
- 1.6 Phương thức và thuộc tính với Static
- 1.7 Abstract classes

Mục tiêu

- 2.1 Interfaces
- 2.2 Interfaces như là function types
- 2.3 Sử dụng interfaces với classes
- 2.4 Readonly properties
- 2.5 Thuộc tính tuỳ chọn (Optional properties)

FPT Education

FPT POLYTECHNIC

cdio™

Conceive Design Implement Operate

THỰC HỌC – THỰC NGHIỆP

LẬP TRÌNH TYPESCRIPT

CLASSES & INTERFACES

http://www.poly.edu.vn

Phần 1
Classes

CLASSES

- 1 class trong typescript có thể gồm những thành phần sau:
  - Constructor
  - Properties
  - Methods

```
class Department {
  name: string;

  constructor(n: string) {
    this.name = n;
  }

  describe() {
    console.log('Department: ' + this.name);
  }
}
```

TỪ KHOÁ THIS

- Từ khoá **this** trong constructor truy xuất đến *tham số và các thuộc tính của class.*

- Từ khoá **new**: tạo *instance của class*

```
const accounting = new Department('Accounting');

accounting.describe();

const accountingCopy = { name: 'DUMMY', describe: accounting.describe };

accountingCopy.describe();
```

PRIVATE, PUBLIC, READONLY

- ***Access modifiers*** xác định khả năng hiển thị (chỉ định/sửa đổi truy cập) của các thành phần dữ liệu trong lớp.
- Các access modifiers được hỗ trợ bởi typescript gồm:
  - ***Public (mặc định)***: Thành phần dữ liệu được đánh dấu public có khả năng truy cập tất cả.
  - ***Private***: Thành phần dữ liệu được đánh dấu là private chỉ được truy xuất trong class của nó
  - ***Readonly***: Thành phần dữ liệu được đánh dấu là readonly có thể được truy cập bên ngoài lớp nhưng không thể thay đổi giá trị

PRIVATE, PUBLIC, READONLY

- Ví dụ

```
class Department {
  public name: string;
  private employees: string[] = [];

  constructor(n: string) {…
  }

  describe(this: Department) {…
  }

  addEmployee(employee: string) {…
  }

  printEmployeeInformation() {…
  }
}

const accounting = new Department('Accounting');

accounting.describe();
accounting.name = 'NEW NAME';
accounting.printEmployeeInformation();
```

KẾ THỪA (INHERITANCE)

- Một lớp (lớp con, lớp dẫn xuất) kế thừa từ một lớp khác (lớp cha, lớp cơ sở) sử dụng từ khoá ***extends***
- Các lớp con *kế thừa tất cả các thuộc tính và phương thức* từ lớp cha)ngoại trừ các thành phần dữ liệu được đánh dấu là private và constructors)
- Cú pháp:

```
class <child_class_name> extends <parent_class_name>
```

KẾ THỪA (INHERITANCE)

- Phân loại kế thừa
  - Single: mỗi class được kế thừa từ một lớp cha
  - Multiple: một lớp có thể kế thừa từ nhiều lớp (typescript không hỗ trợ đa kế thừa)
  - Multi-level

```
class ITDepartment extends Department{}
class AccountingDepartment extends ITDepartment{}
```

KẾ THỪA (INHERITANCE)

super()

- Được sử dụng để gọi constructor của lớp cha và để truy xuất đến các thuộc tính và phương thức của lớp cha.

```
class ITDepartment extends Department {
  admins: string[];
  constructor(id: string, admins: string[]) {
    super(id, 'IT');
    this.admins = admins;
  }
}
```

- Quy tắc quan trọng về super() trong typescript: mỗi lớp con có hàm constructor phải gọi super() để thực thi constructor của lớp cha trước khi truy cập vào phần thân hàm constructor

KẾ THỪA (INHERITANCE)

- Lớp con có thể gọi phương thức của lớp cha bằng từ khoá **super**

```
class ITDepartment extends Department {
  admins: string[];
  constructor(id: string, admins: string[]) {…
  }
  addEmployee(name:string) {
    console.log("IT");
    super.addEmployee(name);
  }
}
```

PHƯƠNG THỨC GHI ĐÈ (OVERRIDING)

- Phương thức ghi đè là quá trình một phương thức ở lớp cha được định nghĩa lại bằng một phương thức cùng tên, cùng tham số ở lớp con

```
class Department {
  private readonly id: string;
  private name: string;
  protected employees: string[] = [];

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

  describe(this: Department) {
    console.log(`Department (${this.id}): ${this.name}`);
  }

  addEmployee(employee: string) {
    this.employees.push(employee);
  }
}
```

```
class AccountingDepartment extends Department {
  constructor(id: string, private reports: string[]) {
    super(id, 'Accounting');
  }

  addEmployee(name: string) {
    if (name === 'Max') {
      return;
    }
    this.employees.push(name);
  }
}
```

PROTECTED

- Access modifiers được hỗ trợ bởi typescript:
  - ***Protected***:
    - Thành phần dữ liệu được đánh dấu là private chỉ được truy xuất trong class của nó
    - Lớp con có thể truy xuất được thành phần dữ liệu đánh dấu là protected ở lớp cha

```
class Department {

    protected employees: string[] = [];

    constructor(private readonly id: string, public name: string) {…
    }

    describe(this: Department) {…
    }

    addEmployee(employee: string) {…
    }

    printEmployeeInformation() {…
    }
}
```

PHƯƠNG THỨC VÀ THUỘC TÍNH STATIC

- Khi thành phần dữ liệu (properties hoặc method) được đánh dấu là ***Static*** => các thành phần dữ liệu có thể được truy xuất trực tiếp trong class mà không dùng từ khoá *this* và truy xuất trực tiếp ngoài lớp mà không từ từ khoá *new* để tạo instance

PHƯƠNG THỨC VÀ THUỘC TÍNH STATIC

```
class Department {
  static  id: string;
  private name: string;


  constructor(id: string, name: string) {
    Department.id = id;
    this.name = name;
  }

  static describe(name: string) {
    console.log(`Department (${Department.id}): ${this.name}`);
  }
}

Department.id = 'd1';
Department.describe('Max');
```

ABSTRACT CLASS

- Abstract class: là một class cha cho tất cả các class có cùng bản chất (kiểu, loại, nhiệm vụ của class)
- Các method ở abstract class không chứa phần triển khai và phải được thực hiện trong lớp dẫn xuất (lớp con)
- Cú pháp

ABSTRACT CLASS

```
abstract class Department {
    static fiscalYear = 2020;
    protected employees: string[] = [];

    constructor(protected readonly id: string, public name: string) {}

    static createEmployee(name: string) {
      return { name: name };
    }

    abstract describe(this: Department): void;
}

class ITDepartment extends Department {
    admins: string[];
    constructor(id: string, admins: string[]) {
      super(id, 'IT');
      this.admins = admins;
    }

    describe() {
      console.log('IT Department - ID: ' + this.id);
    }
}
```


demo

Phần 2
Interfaces

INTERFACES

- Interface mô tả/định nghĩa cấu trúc của đối tượng.
- Interface chỉ chứa phần khai báo thuộc tính, phương thức, sự kiện
- ***Chú ý***: *typescript compiler không chuyển đổi interface sang javascript*
- Cú pháp

```
interface <ten_interface>  {
  // code here
}
```

Ví dụ

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

  greet(phrase: string): void;
}

let user1: Person;

user1 = {
  name: 'Max',
  age: 30,
  greet(phrase: string) {
    console.log(phrase + ' ' + this.name);
  }
};

user1.greet('Hi there - I am');
```

INTERFACES NHƯ LÀ FUNCTION

- Interface có thể được mô tả như là 1 function.
- Cách thực hiện:
  - Cung cấp function signature (chữ ký hàm): khai báo hàm chỉ có danh sách tham số và kiểu trả về
- Ví dụ

```
interface AddFn {
  (a: number, b: number): number;
}
```

SỬ DỤNG INTERFACES VỚI CLASSES

- Dùng từ khoá ***implements*** để mô tả 1 class implements từ 1 interface
- Cú pháp

```
class <ten_class> implements <ten_interface> {
  //code here
}
```

- Dùng từ khoá ***extends*** để mô tả 1 interface kế thừa từ 1 interface khác
- Cú pháp

```
interface <ten_interface_1> extends <ten_interface_2 {
  //code here
}
```

SỬ DỤNG INTERFACES VỚI CLASSES

Ví dụ: implements và extends

```
interface Named {
  readonly name: string;
}

interface Greetable extends Named {
  greet(phrase: string): void;
}

class Person implements Greetable {
  name: string;
  age = 30;

  constructor(n: string) {
    this.name = n;
  }

  greet(phrase: string) {
    console.log(phrase + ' ' + this.name);
  }
}

let user1: Greetable;

user1 = new Person('Max');

user1.greet('Hi there - I am');
console.log(user1);
```

READONLY PROPERTIES

- Thuộc tính trong interface được đánh dấu là readonly: thuộc tính chỉ có thể sửa đổi khi một đối tượng được tạo lần đầu tiên
- Ví dụ

```
interface Named {
  readonly name: string;
  outputName: string;
}
```

OPTIONAL PROPERTIES

- Không phải tất cả các thuộc tính trong interface đều bắt buộc (Giống optional function)
- Ví dụ

```
interface Named {
  readonly name?: string;
  outputName?: string;
}
```


demo


thank
you!