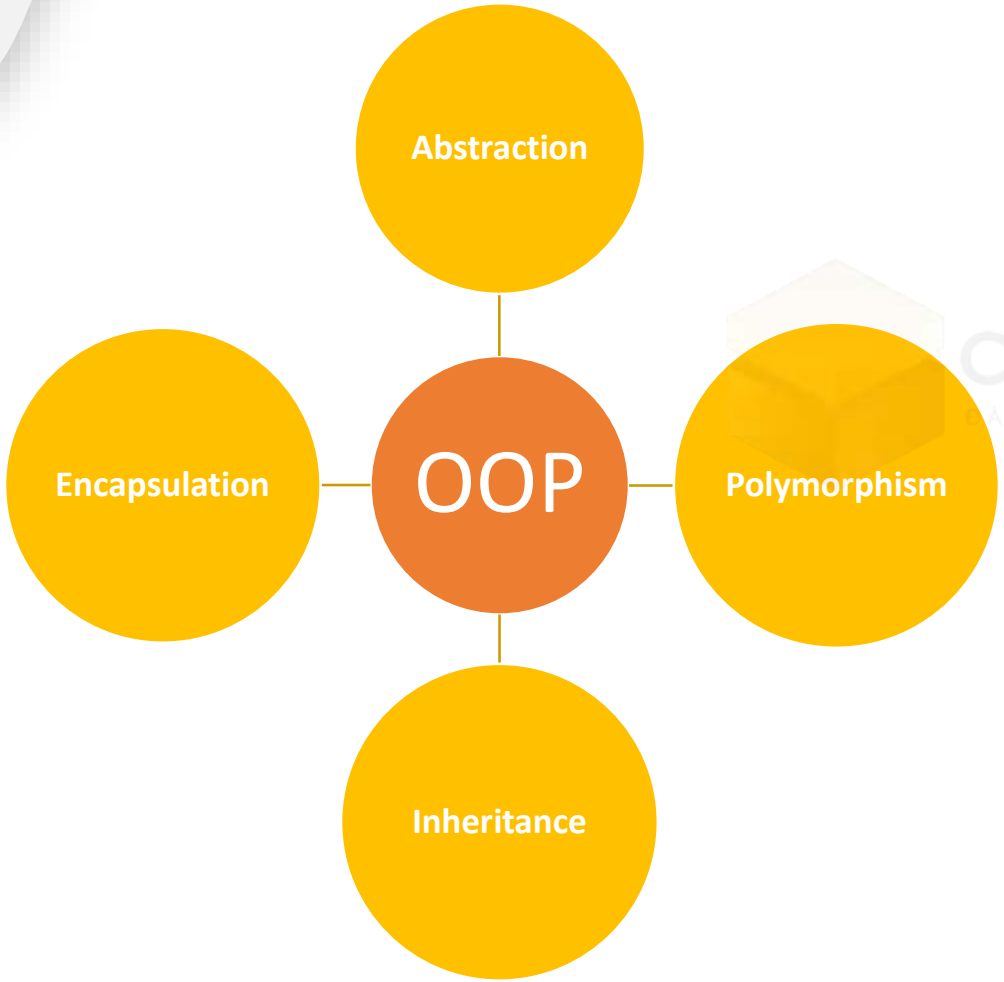
1

# OOP (Object Oriented Programming)

# OOP – 4 tính chất của lập trình hướng đối tượng

- Lập trình hướng đối tượng là một kỹ thuật lập trình cho phép lập trình viên tạo ra các đối tượng trong code để trừu tượng hóa các đối tượng thực tế trong cuộc sống.
- Các ưu điểm:
  - Dễ dàng quản lý code khi có sự thay đổi chương trình.
  - Dễ mở rộng dự án.
  - Có tính bảo mật cao.
  - Có thể sử dụng mã nguồn, tiết kiệm tài nguyên.

# OOP – 4 tính chất của lập trình hướng đối tượng




- Hầu hết các ngôn ngữ lập trình hiện nay như Java, PHP, .NET, Ruby, Python… đều hỗ trợ OOP
- ES6 cũng hỗ trợ OOP nhưng chỉ có 2 tính chất là Kế thừa và Đa hình, còn 2 tính còn lại thì hỗ trợ kém.

# OOP - Inheritance (kế thừa)

➢ **Tại sao cần Kế Thừa ?**

Công ty du lịch PT đang quản lý thông tin 2 loại chuyến xe như sau:

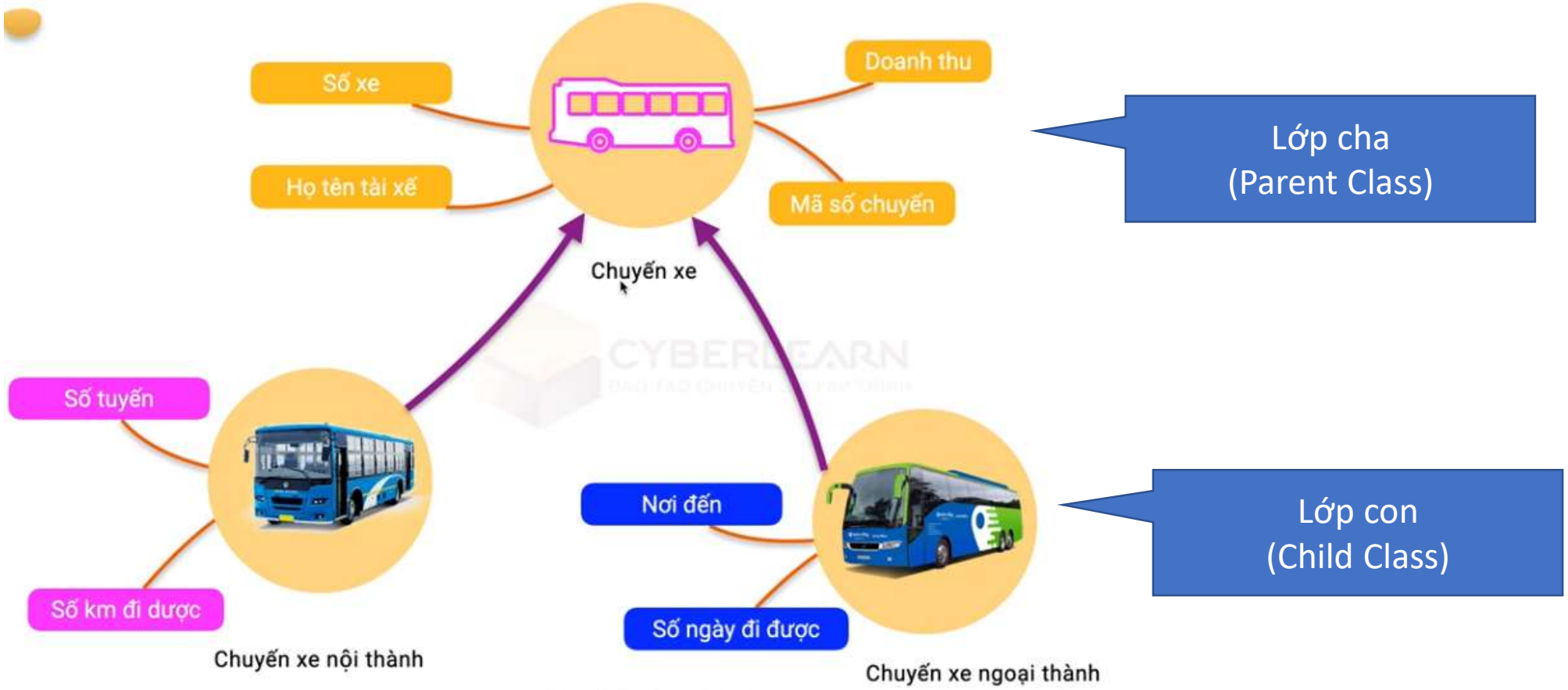
- Chuyến xe nội thành: Mã số chuyến, Họ tên tài xế, số xe, số km đi được, doanh thu.
- Chuyến xe ngoại thành: Mã số chuyến, Họ tên tài xế, số xe, nơi đến, số ngày đi được, doanh thu.
- Công ty cần tính tổng doanh thu cả công ty và doanh thu của từng chuyến

# OOP - Inheritance (kế thừa)

- Quan sát 2 lớp đối tương bên dưới, 2 lớp này có thuộc tính giống nhau không?


# OOP - Inheritance (kế thừa)

- Tạo thêm 1 lớp cha riêng, bao quát các đặc điểm chung của 2 lớp con bên dưới
- Những thuộc tính riêng sẽ để ở các lớp con
- Những thuộc tính chung sẽ được lớp con kế thừa từ lớp cha



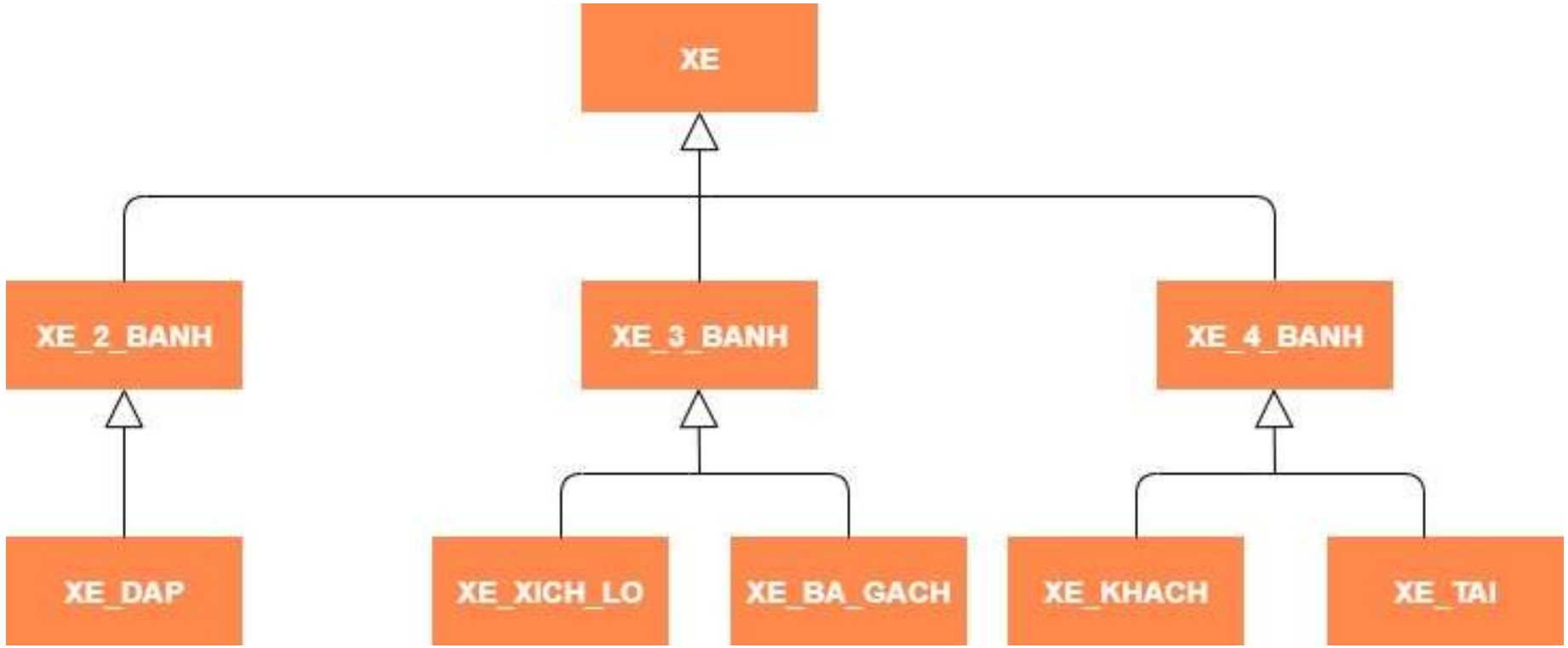
# OOP - Inheritance (kế thừa)

- **Inheritance** là khái niệm rất then chốt trong OOP
- **Inheritance** cho phép tạo 1 class mới dựa trên 1 class có sẵn
- **Inheritance** có thể làm đơn giản hóa thiết kế và cài đặt của ứng dụng
- **Class mới** kế thừa từ class có sẵn được gọi là **derived class, child class hoặc subclass**.
- **Class có sẵn** được gọi là một **base class, parent class hoặc super class**
- **Subclass** được thừa hưởng từ **super class** các field và các method của **super class**
- **Subclass** có thể extend (mở rộng) **super class** bằng cách bổ sung thêm các field, các constructor và các method
- **Subclass** có thể override (ghi đè) lại các method kế thừa của **super class** để tạo thành phiên bản method phù hợp cho **Subclass**

# OOP - Inheritance (kế thừa)

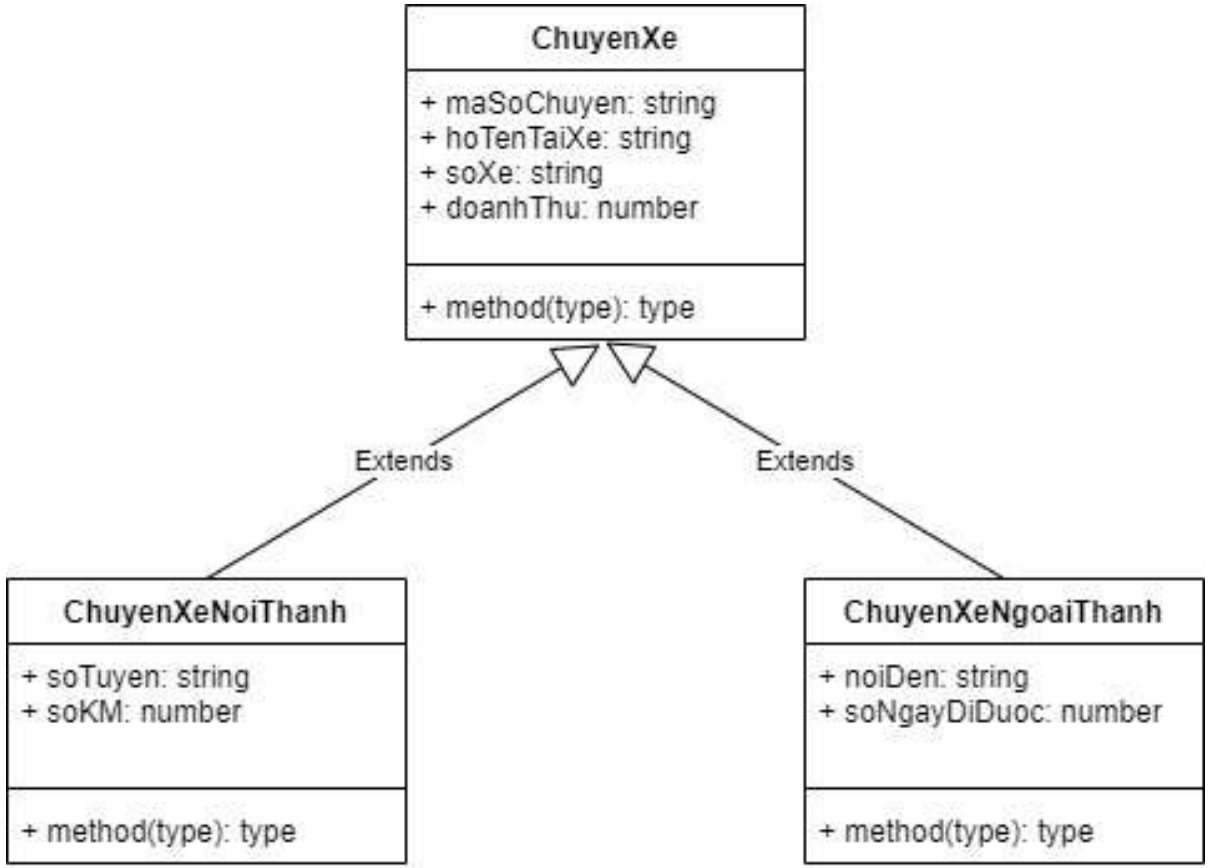
➢ Cây kế thừa

Là một cây đa nhánh thể hiện mối quan hệ giữa các lớp trong hệ thống, chương trình.



# OOP - Inheritance (kế thừa)

- Sơ đồ lớp tổng quát (UML)



# OOP - Inheritance (kế thừa)

- Class Inheritance
- ES6 giúp code chúng ta dễ đọc, dễ hiểu và tường minh hơn, như những ngôn ngữ thuần OOP
- Subclass sử dụng từ khóa **extends** để kế thừa thuộc tính và phương thức từ superclass

```
//ES6
class Mother {
    constructor(name) {
        this.name = name;
    }
    colorEyes() {
        console.log('Black');
    }
}
class Me extends Mother {
    colorSkin() {
        console.log('White');
    }
}
let me = new Me('Mị');
console.log(me.name);//Mị
me.colorEyes();//Black
me.colorSkin();//White
```

# OOP - Method Overriding
# (Phương thức ghi đè)

➢ Tại sao cần Method Overriding?

- Xét bài toán quản lý nhân sự bao gồm Nhân viên thường, Trưởng phòng và giám đốc. Mỗi nhân sự sẽ có lương theo ngày và phụ cấp khác nhau. Xây dựng các lớp đối tượng và tính lương cho nhân viên.

# OOP - Method Overriding
# (Phương thức ghi đè)

```
class NhanSu{
    constructor(hoTen,maSo,soNgayLam
){
        this.hoTen = hoTen;
        this.maSo = maSo;
        this.soNgayLam = soNgayLam;
        this.luong = 0;
    }
    tinhLuong(){
            this.luong = 0;
    }
}
```

Override

Override

```
class NhanVienThuong extends NhanSu{
    constructor(hoTen,maSo,soNgayLam){
        super(hoTen,maSo,soNgayLam)
         this.phuCap = 200;
         this.luongNgay = 100;
    }
    tinhLuong(){
            this.luong = this.soNgayLam * this.luongNgay
 + this.phuCap;
    }
}
```

```
class TruongPhong extends NhanSu{
    constructor(hoTen,maSo,soNgayLam){
        super(hoTen,maSo,soNgayLam)
         this.phuCap = 300;
         this.luongNgay = 200;
    }
    tinhLuong(){
            this.luong = this.soNgayLam * this.luongNgay
 + this.phuCap;
    }
}
```

# OOP - Method Overriding
# (Phương thức ghi đè)

- Class con sẽ kế thừa các thuộc tính và phương thức của class cha và class con cũng có thể ghi đè phương thức của class cha
- Để ghi đè phương thức của class cha, class con chỉ cần tạo phương thức trùng tên trong class con hoặc tạo thông qua prototype

```
class NhanVienThuong extends NhanSu{
    constructor(hoTen,maSo,soNgayLam){
        super(hoTen,maSo,soNgayLam)
         this.phuCap = 200;
         this.luongNgay = 100;
    }
}
NhanVienThuong.prototype.tinhLuong = function(){
    this.luong = this.soNgayLam * this.luongNgay
         + this.phuCap;
}
```

```
class NhanVienThuong extends NhanSu{
    constructor(hoTen,maSo,soNgayLam){
        super(hoTen,maSo,soNgayLam)
         this.phuCap = 200;
         this.luongNgay = 100;
    }
    tinhLuong(){
      this.luong = this.soNgayLam *
          this.luongNgay + this.phuCap;
    }
}
```

# OOP - Method Overriding
# (Phương thức ghi đè)

- Ở phiên bản trước ES5, cũng có cách để ghi đè phương thức nhưng cú pháp sẽ khó hiểu và không tường minh

```
// ES5 - method overriding
function Person(name) {
    this.name = name;
}
Person.prototype.getName = function () {
    return this.name;
}
var reader = new Person('Truc');
reader.getName = function () {
    let baseName = Person.prototype.getName.call(this);
    return "Hello" + baseName;
}
console.log(reader.getName());
```

Gọi hàm và cho phép bạn truyền vào một object

# OOP - Super()

➢ Super()

Super đại diện cho class cha giúp gọi lại constructor hoặc phương phức của class cha, mà phương thức đó đã bị override (ghi đè) trong lớp con.

```
// ES6 - Gọi lại thuộc tính và phương thức
class People {
    constructor(name, email) {
        this.name = name;
        this.email = email;
    }
    getInfo() {
        console.log(`${this.name} ${this.email}`)
    }
}
class Reader extends People {
    constructor(name, email, book) {
        //Gọi lại thuộc tính của class cha
        super(name, email);
        this.book = book;
    }
    getInfo() {
        // Gọi lại phương thức của class cha
        super.getInfo();
        console.log(this.book);
    }
}
let reader = new Reader('Mị', 'mi@gmail.com', 'Ngữ Văn 12');
reader.getInfo();
```

# OOP - Polymorphism (đa hình)

- Tính đa hình là gì?
- Ví dụ tính đa hình trong bài toán tính tiền nhân viên:

Khi ta yêu cầu cho cả 2 loại nhân viên thực hiện việc tính lương thì cả 2 Nhân viên đều hiểu thông điệp, nhưng mỗi loại nhân viên sẽ tính tiền lương khác nhau:

```
class NhanVienThuong extends NhanSu{
    constructor(hoTen,maSo,soNgayLam){
        super(hoTen,maSo,soNgayLam)
         this.phuCap = 200;
         this.luongNgay = 100;
    }
    tinhLuong(){
       this.luong = this.soNgayLam *
         this.luongNgay + this.phuCap;
    }
}
```

```
class TruongPhong extends NhanSu{
    constructor(hoTen,maSo,soNgayLam){
        super(hoTen,maSo,soNgayLam)
         this.phuCap = 300;
         this.luongNgay = 200;
    }
    tinhLuong(){
       this.luong = this.soNgayLam *
         this.luongNgay + this.phuCap;
    }
}
```

# OOP - Polymorphism (đa hình)

- Tính đa hình là hiện tượng các đối tượng thuộc các lớp khác nhau có thể hiểu cùng 1 thông điệp theo các cách khác nhau.
- Để thể hiện được tính đa hình:
  - Các lớp phải có quan hệ kế thừa với cùng 1 lớp cha nào đó.
  - Phương thức đa hình phải được ghi đè (override) ở các lớp con (như ví dụ tính lương ở phần Method Overriding ).

# OOP - Encapsulation (tính đóng gói)

- Encapsulation:
  - Tính che giấu thông tin.
  - Tính chất này không cho phép người sử dụng các đối tượng thay đổi thuộc tính của một đối tượng. Chỉ có các phương thức nội tại của đối tượng cho phép thay đổi trạng thái của nó.
  - Việc cho phép môi trường bên ngoài tác động lên các dữ liệu nội tại của một đối tượng theo cách nào là hoàn toàn tùy thuộc vào người viết mã.
  - Đây là tính chất đảm bảo sự toàn vẹn của đối tượng.

# OOP - Abstraction (tính trừu tượng)

- Abstraction :
  - Tính trừu tượng là chỉ nêu ra vấn đề mà không hiển thị cụ thể, chỉ hiện thị tính năng thiết yếu đối với đối tượng người dùng mà không nói quy trình hoạt động.
  - Ví dụ: như tạo ra tính năng gửi tin nhắn, ta chỉ cần hiểu là người dùng viết tin rồi nhấn gửi đi. Còn quy trình xử lý tin nhắn gửi như thế nào thì ta chưa đề cập đến.
  - Như vậy, tính trừu tượng là che giấu thông tin thực hiện từ người dùng, họ chỉ biết tính năng được cung cấp. Chỉ biết thông tin đối tượng thay vì cách nó sử dụng như thế nào.

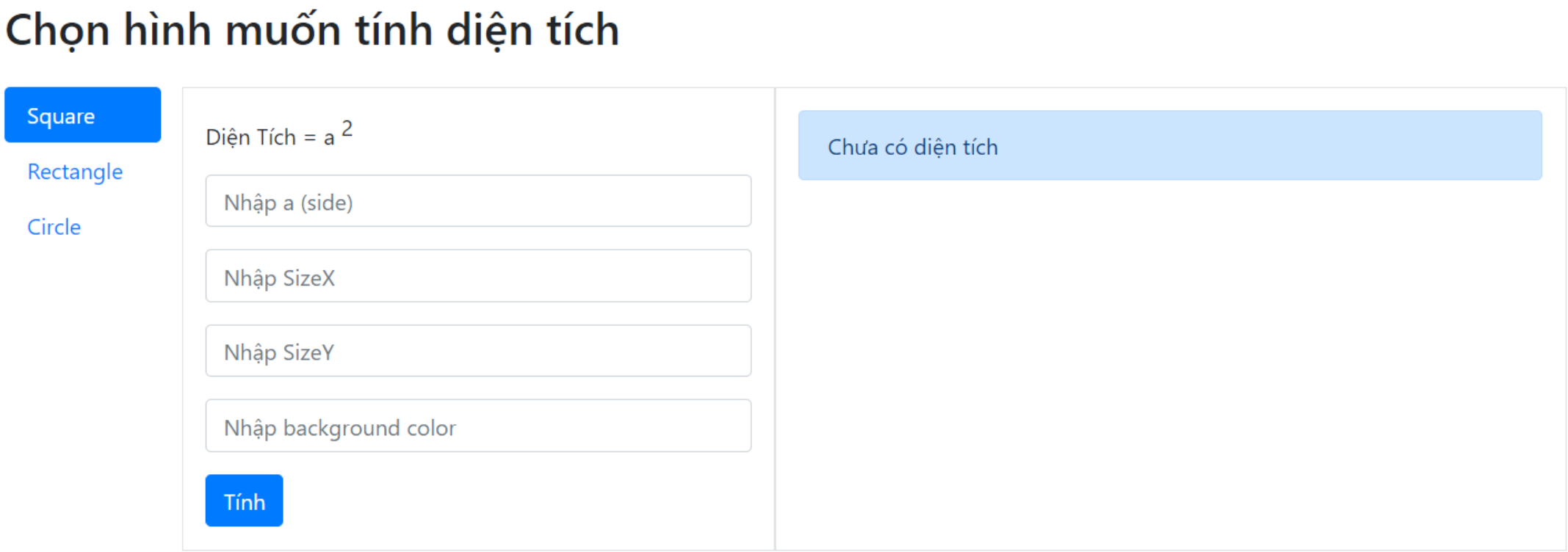
# OOP - Abstraction (tính trừu tượng)

- Abstraction :
- Nó có những ưu điểm sau:
  - Cho phép lập trình viên bỏ qua những phức tạp trong đối tượng mà chỉ đưa ra những khái niệm phương thức và thuộc tính cần thiết. Ta sẽ dựa những khái niệm đó để viết ra, nâng cấp và bảo trì.
  - Nó giúp ta tập trung cái cốt lõi đối tượng. Giúp người dùng không quên bản chất đối tượng đó làm gì.

# OOP - Bài tập: Tính diện tích hình học

# OOP – Bài tập: Tính diện tích hình học

[Nhấn vào đây để tải layout]



# OOP – Bài tập: Tính diện tích hình học

- Xây dựng chương trình tính diện tích các hình: hình vuông, hình chữ nhật, hình tròn. Các hình có thuộc tính như sau:

- **Square**: tên, màu nền, chiều dài (sizeX), chiều cao (sizeY),diện tích, cạnh hình vuông (side)

  **Diện tích (area) =** $side^2$

- **Rectangle**: tên, màu nền, chiều dài (sizeX), chiều cao (sizeY),diện tích, cạnh dài (length), cạnh rộng (width)

  **Diện tích (area) = length * width**

- **Circle**: tên, màu nền, chiều dài (sizeX), chiều cao (sizeY),diện tích, bán kính (radius - r)

  **Diện tích (area) =** $\pi r^2$

- Thực hiện các yêu cầu:

  1. Xây dựng các lớp với chức năng kế thừa
  2. Nhập các thông số tương ứng cho mỗi hình
  3. Tính diện tích
  4. Vẽ hình tương ứng dựa vào màu nền, chiều dài (sizeX), chiều cao (sizeY)
  5. Hiện diện tích và tên hình lên giao diện

# OOP – Bài tập: Tính diện tích hình học

- Kiến thức áp dụng
  1. Phân tích và xây dựng lớp đối tượng (Class)
  2. Tính kế thừa
  3. Phương thức Overriding
  4. Tính đa hình
  5. Hàm super

# OOP – Bài tập: Tính diện tích hình học

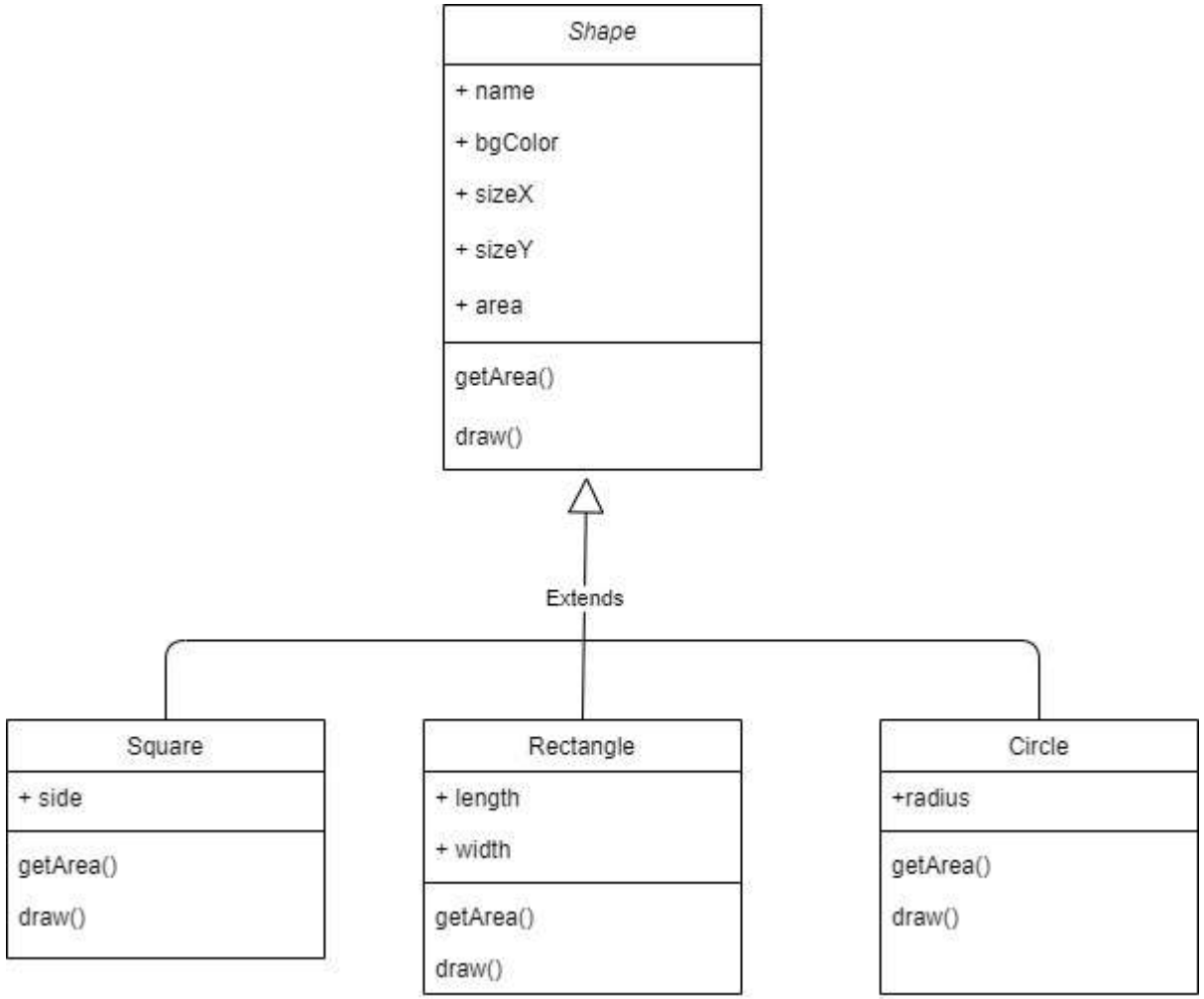
- Bước 1: Xác định đối tượng
- Bước 2: Vẽ sơ đồ lớp UML
- Bước 3: Liệt kê tất cả thuộc tính của đối tượng
- Bước 4: Liệt kê các phương thức
- Bước 5: Xây dựng lớp đối tượng
- Bước 6: Xây dựng phương thức khởi tạo
- Bước 7: Xây dựng phương thức tính diện tích
- Bước 8: Xây dựng phương thức vẽ hình
- Bước 9: Tạo đối tượng ở nơi cần xử lý
- Bước 10: Gọi các phương thức xử theo yêu cầu bài toán

# OOP – Bài tập: Tính diện tích hình học

## ➢ Thảo luận nhóm

1. Xác định đối tượng
2. Vẽ sơ đồ UML để liệt kê các thuộc tính và phương thức của đối tượng

# OOP – Bài tập: Tính diện tích hình học



# OOP - Bài tập: Testing Online

# OOP - Bài tập: Testing Online

[Nhấn vào đây để tải layout]

# OOP - Bài tập: Testing Online

- Xây dựng chương trình Testing Online. Hệ thống có 2 loại câu hỏi:

- **MultipleChoice, FillInBlank**
- Thực hiện các yêu cầu:
  1. Xây dựng các lớp với chức năng kế thừa
  2. Hiển thi danh sách câu hỏi
  3. Tính điểm, số câu đúng, số câu sai

# OOP - Bài tập: Testing Online

- Kiến thức áp dụng

1. Phân tích và xây dựng lớp đối tượng (Class)
2. Tính kế thừa
3. Phương thức Overriding
4. Tính đa hình
5. Axios và CRUD
6. Các phương pháp duyệt mảng

# OOP - Bài tập: Testing Online

- Bước 1: Xác định đối tượng
- Bước 2: Vẽ sơ đồ lớp UML
- Bước 3: Liệt kê tất cả thuộc tính của đối tượng
- Bước 4: Liệt kê các phương thức
- Bước 5: Xây dựng lớp đối tượng
- Bước 6: Xây dựng phương thức khởi tạo
- Bước 7: Xây dựng phương thức hiển thi câu hỏi
- Bước 8: Xây dựng phương thức tính điểm
- Bước 9: Tạo đối tượng ở nơi cần xử lý
- Bước 10: Gọi các phương thức xử theo yêu cầu bài toán

# OOP - Bài tập: Testing Online

## ➢ Thảo luận nhóm

1. Xác định đối tượng
2. Vẽ sơ đồ UML để liệt kê các thuộc tính và phương thức của đối tượng



# Các cách khác call API

# FETCH API

## ➢ Fetch API là gì?

- Fetch là hàm hỗ trợ tương tác với API của ES6
- Fetch hoạt động dựa trên Promises (sử dụng then, catch)
- Fetch là hàm có sẵn của ES6 nên không cần cài đặt thư viện như Axios.
- Fetch không hỗ trợ IE

# FETCH API

## ➢ Cách sử dụng Fetch:

- GET: fetch cần then 2 lần để lấy data từ API

```
fetch('http://svcy.myclass.vn/api/SinhVienApi/LayDanhSachSinhVien')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.log(error));
```

# FETCH API

- Cách sử dụng Fetch:
  - POST:

fetch yêu cầu chuyển kiểu dữ liệu object sang json và khai báo định dạng kiểu dữ liệu ở headers

```
const sv = {
        maSinhVien: 1553,
        tenSinhVien: "Nguyễn Thị Học Viên",
        loaiSinhVien: "Khó Khăn",
        diemToan: 5,
        diemLy: 5,
        diemHoa: 0,
        diemRenLuyen: 0,
        email: "hv.1997@gmail.com",
        soDienThoai: 051216665
};

fetch('http://svcy.myclass.vn/api/SinhVienApi/ThemSinhVien', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(sv),
})
.then(response => response.json())
.then(data => {
  console.log('Success:', data);
})
.catch((error) => {
  console.error('Error:', error);
});
```

# FETCH API

- Cách sử dụng Fetch:
  - PUT: cách sử dụng put tương tự như với post

```
const sv = {
        maSinhVien: 1553,
        tenSinhVien: "Nguyễn Thị Học Viên",
        loaiSinhVien: "Khó Khăn",
        diemToan: 5,
        diemLy: 5,
        diemHoa: 10,
        diemRenLuyen: 10,
        email: "hv.1997@gmail.com",
        soDienThoai: 051216665
};

fetch('http://svcy.myclass.vn/api/SinhVienApi/CapNhatThongTinSinhVien?maSinhVien=1553', {
  method: 'PUT',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(sv),
})
.then(response => response.json())
.then(data => {
  console.log('Success:', data);
})
.catch((error) => {
  console.error('Error:', error);
});
```

# FETCH API

## ➢ Cách sử dụng Fetch:

- DELETE:

```
fetch('http://svcy.myclass.vn/api/SinhVienApi/XoaSinhVien?maSinhVien=1553', {
  method: 'DELETE'
})
.then(response => response.json())
.then(data => {
  console.log('Success:', data);
})
.catch((error) => {
  console.error('Error:', error);
});
```

# Async và Await

- Async và Await giúp làm việc với các hàm bất đồng bộ
- Có thể sử dụng trong hầu hết các trình duyệt chính, ngoại trừ IE
- Nó được xây dựng trên Promises và tương thích với tất cả các Promise (Fetch, Axios)

# Async và Await

Async - khai báo một hàm bất đồng bộ

- Tự động biến đổi một hàm thông thường thành một Promise.
- Khi gọi tới hàm async nó sẽ xử lý mọi thứ và được trả về kết quả trong hàm của nó.

Async cho phép sử dụng Await.

- Khi được đặt trước một Promise, nó sẽ đợi cho đến khi Promise kết thúc và trả về kết quả.
- Await chỉ làm việc với Promises, nó không hoạt động với callbacks.
- Await chỉ có thể được sử dụng bên trong các function async.

Sử dụng then, catch

```
let getData = () =>{
    fetch('http://svcy.myclass.vn/api/SinhVienApi/LayDanhSachSinhVien')
   .then(response => response.json())
   .then(data => console.log(data))
   .catch(error => console.log(error));
}
getData();
```

Thay thế then, catch

bằng async và await

```
let getData = async () =>{
    try{
        //Xử lý thành công
        let response = await fetch('http://svcy.myclass.vn/api/SinhVienApi/LayDanhSachSinhVien');
        let data = await response.json();
        console.log(data);
    }
    catch (error){
        console.log(error);
    }

}
getData();
```

# Thank You