Abstract Class & Interface

Module: Advanced Programming with JavaScript

Mục tiêu

- Trình bày được Interface
- Trình bày được Abstract class
- Trình bày được Abstract method
- Khai báo được Interface
- Khai báo được Abstract class
- Khai báo được lớp kế thừa từ abstract class
- Khai báo được lớp triển khai từ Interface
- Thiết kế được các giải pháp có sử dụng Interface và Abstract class

Abstract class và Abstract method

Từ khoá abstract

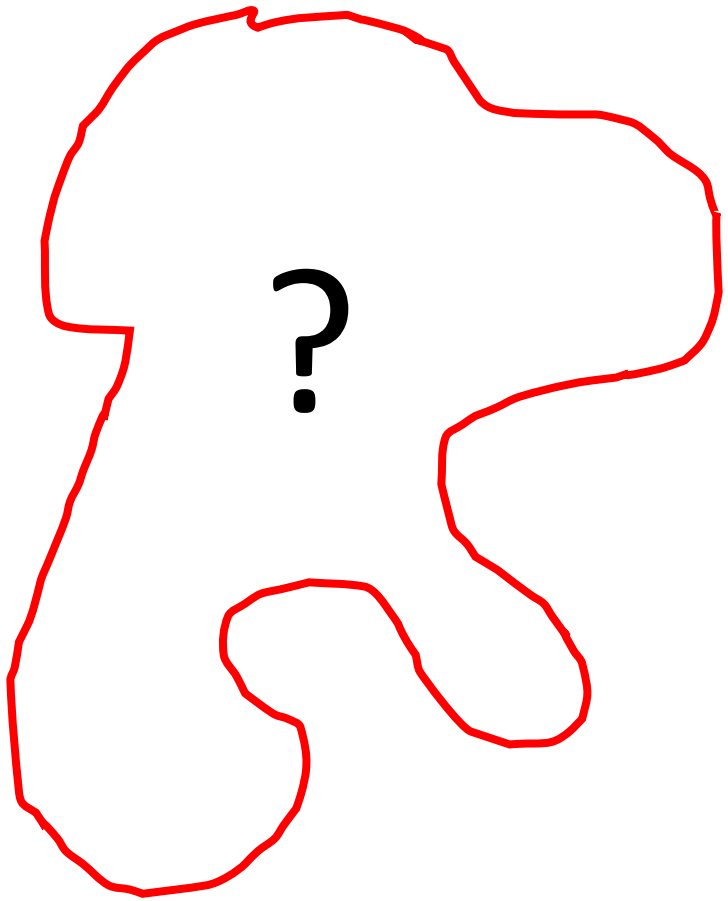
Abstract class

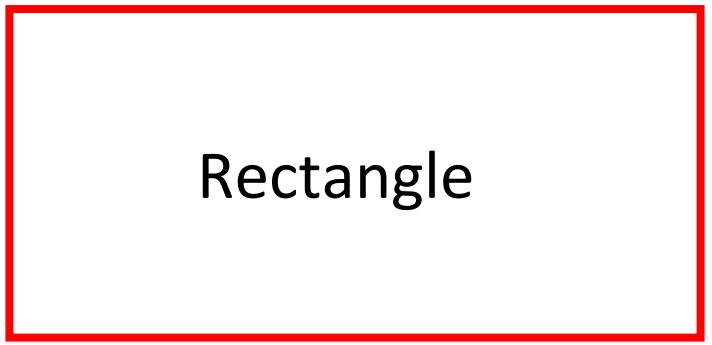
Abstract method

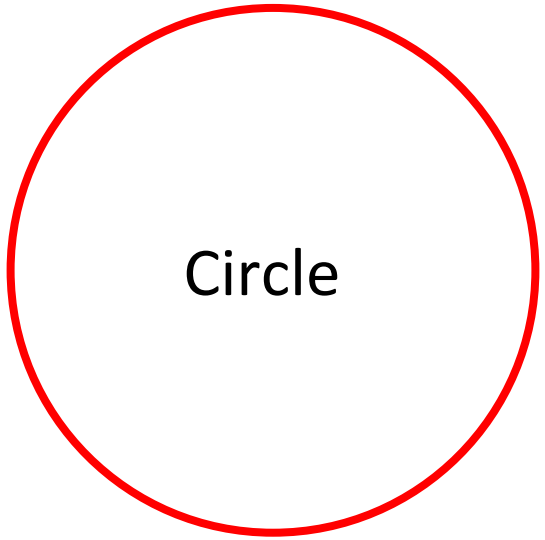
Abstract class

- Trong kế thừa, lớp cha định nghĩa các phương thức chung cho các lớp con
- Lớp con cụ thể hơn lớp cha, lớp cha “chung chung” hơn lớp con
- Trong hệ kế thừa, càng lên cao thì tính cụ thể càng ít đi, tính trừu tượng càng tăng lên
- Những lớp có tính trừu tượng rất cao, đến mức không thể tạo được các đối tượng của lớp đó thì được gọi là lớp trừu tượng (abstract class)
- Ví dụ: Lớp *Geometric* là một lớp rất trừu tượng, do đó nó phù hợp để trở thành một lớp abstract

Abstract class







Abstract method

- Abstract method (phương thức trừu tượng) là những phương thức được khai báo (declare) nhưng không có phần thân (không được implement)
- Ví dụ:
  - Lớp Geometric có thể khai báo phương thức getArea() và getPerimeter() nhưng không có phần thân của hai phương thức này
  - Tất cả các “Hình” đều có thể tính được diện tích và chu vi
  - Không thể tính được diện tích và chu vi ở bên trong lớp Geometric bởi vì chưa xác định rõ “Hình” này là hình gì
- Phương thức trừu tượng được bổ sung phần thân (tức là implement) ở các lớp con

Từ khoá abstract

- Từ khoá abstract được sử dụng để khai báo lớp trừu tượng và phương thức trừu tượng
- Ví dụ:

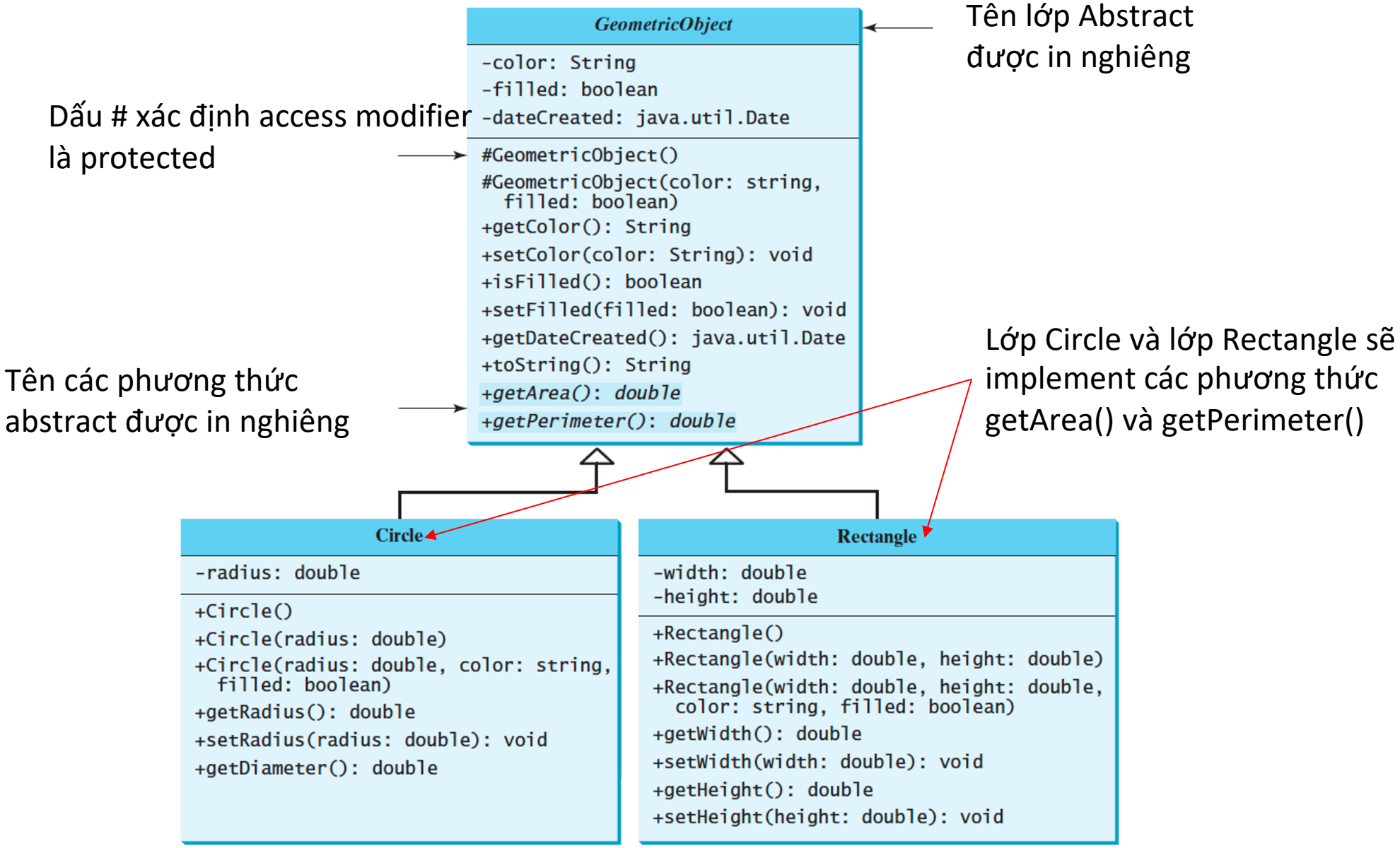
```
abstract class Geometric {
    private $name;

    getName() {
        return $this->name;
    }

    abstract getArea();

    abstract getPerimeter();
}
```

Các ký hiệu UML



Các tính chất của lớp abstract

- Không thể tạo đối tượng của lớp abstract
- Lớp abstract có thể có các thuộc tính và phương thức bình thường
- Một lớp chứa phương thức abstract thì lớp đó phải là abstract
- Một lớp không phải là abstract kế thừa từ một lớp cha abstract thì phải implement tất cả các phương thức abstract của lớp cha
- Một lớp abstract kế thừa từ một lớp cha abstract thì có thể không implement các phương thức abstract của lớp cha

Lớp trừu tượng - ví dụ

Định nghĩa lớp trừu tượng AbstractClass

```
abstract class AbstractClass {

    // Force extending class to define this method
    abstract getValue(): string;
    abstract prefixValue(value: string): string;

    // Common method
    printOut(): void {
        console.log(this.getValue());
    }
}
```

Lớp trừu tượng - ví dụ

Tạo ra class con ConcreteClass1 thừa kế AbstractClass

```
class ConcreteClass1 extends AbstractClass {
    getValue(): string {
        return "ConcreteClass1";
    }

    prefixValue(prefix: string) {
        return `${prefix}ConcreteClass1`;
    }
}
```

Lớp trừu tượng - ví dụ

Tạo ra class con ConcreteClass2 thừa kế AbstractClass

```
class ConcreteClass2 extends AbstractClass {
    getValue(): string {
        return "ConcreteClass2";
    }

    prefixValue(prefix: string) {
        return `${prefix}ConcreteClass2`;
    }
}
```

Interface

Interface

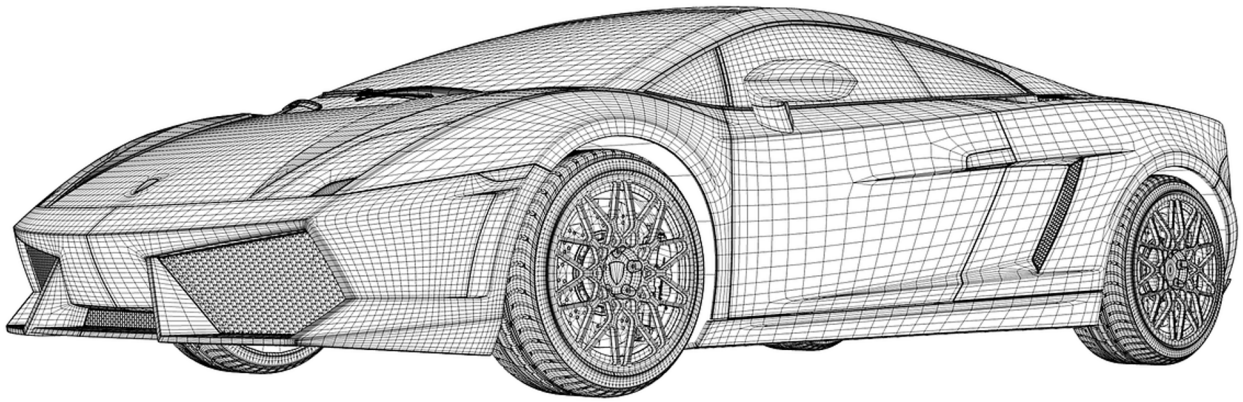
- Interface là một cấu trúc tương tự như lớp, nhưng chỉ chứa các hằng số và abstract method
- Interface quy định các hành vi chung cho các lớp triển khai nó
- Sử dụng từ khoá interface để định nghĩa interface
- Cú pháp:

```
interface InterfaceName {
    /** Constant declarations */
    /** Abstract method signatures */
}
```

- Ví dụ:

```
interface Flyable{
}
```

Interface: Ví dụ



Các tính chất

- Định nghĩa một interface tạo ra một kiểu dữ liệu mới
- Không thể tạo đối tượng của interface
- Interface không thể chứa các phương thức không abstract
- Khi một lớp triển khai interface thì cần triển khai tất cả các phương thức được khai báo trong interface đó
- Interface có thể được thiết kế để khai báo các phương thức chung cho các lớp không liên quan với nhau (khác với abstract class, được kế thừa bởi các lớp có liên quan với nhau)
- Interface bổ sung cho việc Java không hỗ trợ “đa kế thừa”

Triển khai interface

- Một lớp triển khai interface bằng cách sử dụng từ khoá *implements*
- Cú pháp: `class ClassName implements IntefaceName{}`
- Ví dụ:

```
interface Flyable{
    fly();
}

class Bird implements Flyable{
    public fly() {
        return "Flying with wings";
    }
}
```

Triển khai nhiều interface

- Tạo interface
- Một class implements nhiều interface
- Trong lớp định nghĩa rõ nội dung của phương thức

```
interface Animal {
    move();
}

interface Flyable {
    fly();
}

class Bird implements Animal, Flyable {
    move() {
        return "Running...";
    }

    fly() {
        return "Flying with to wings";
    }
}
```

Kế thừa interface

- Một interface có thể kế thừa interface khác
- Interface con thừa hưởng các phương thức và hằng số được khai báo trong interface cha
- Interface con có thể khai báo thêm các thành phần mới
- Từ khoá extends được sử dụng để kế thừa interface
- Ví dụ:

```
interface Flyable{
    fly();
}

interface AnimalFlyable extends Flyable{}

interface EngineFlyable extends Flyable{}
```

Access modifier của interface

- Mặc định, các phương thức abstract của interface đều có access modifier là public
- Không thể sử dụng private hoặc protected cho các phương thức của interface
- Không cần thiết phải chỉ rõ access modifier là public cho các phương thức của interface
- Ví dụ:

```
interface Flyable {
        public fly();
}
```

Không cần thiết

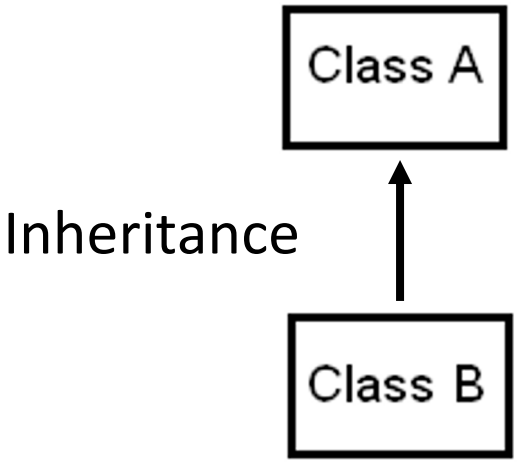
Khai báo hằng số trong Interface

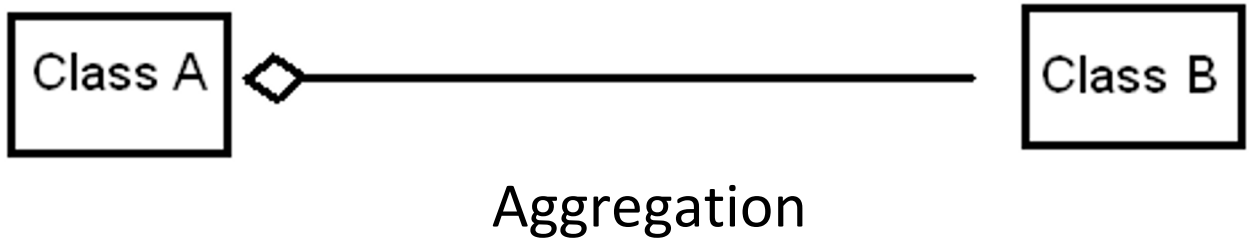
- Có thể khai báo các hằng số trong interface
- Không cần thiết phải chỉ rõ access modifier cho hằng số, mặc định là public
- Cần phải ghi rõ từ khóa const cho hằng số
- Ví dụ:

```
interface Flyable{
    const ORIENTATION_LEFT = 1;
    const ORIENTATION_RIGHT = 2;
    const ORIENTATION_UP = 3;
    const ORIENTATION_DOWN = 4;
}
```

Lựa chọn Inheritance hay Aggregation

- Trong nhiều trường hợp, có thể chuyển đổi qua lại giữa việc sử dụng inheritance (kế thừa) và aggregation (tập hợp)
- Inheritance thể hiện mối quan hệ *is-a*
- Aggregation thể hiện mối quan hệ *has-a*
- Ví dụ:
  - Lớp Apple và lớp Fruit: Mối quan hệ *is-a*
  - Lớp Customer và lớp Address: Mối quan hệ *has-a*





Lựa chọn Interface hay Abstract class

- Trong nhiều trường hợp, có thể chuyển đổi giữa việc sử dụng Interface và Abstract class
- Nếu có sự gần gũi, rõ ràng giữa các lớp về mối quan hệ *cha-con* thì nên sử dụng lớp (mối quan hệ *is-a*)
  - Ví dụ: *Apple is a Fruit (Táo là một Quả)*
- Nếu không có mối quan hệ gần gũi thì nên chọn interface (mối quan hệ *can-do*)
  - Ví dụ: *Bird can fly (Chim có thể bay)*

Tóm tắt bài học

- Abstract method là phương thức chỉ có phần khai báo mà không có phần thân
- Abstract class là lớp có tính chất trừu tượng, không tạo được đối tượng
- Lớp có phương thức abstract thì bắt buộc phải abstract
- Lớp kế thừa lớp abstract thì phải triển khai toàn bộ các phương thức abstract
- Interface chỉ có thể chứa hằng số và các phương thức abstract
- Từ khóa implement được sử dụng để triển khai interface
- Interface có thể kế thừa interface khác

Hướng dẫn

- Hướng dẫn làm bài thực hành và bài tập
- Chuẩn bị bài tiếp: ***Cấu trúc dữ liệu và giải thuật cơ bản***