Bài 7
Interface và Abstract class

Module: ADVANCED PROGRAMMING WITH JAVA

Kiểm tra bài trước

Hỏi và trao đổi về các khó khăn gặp phải trong bài "Inheritance"

Tóm tắt lại các phần đã học từ bài "Inheritance"

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

Thảo luận

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 method

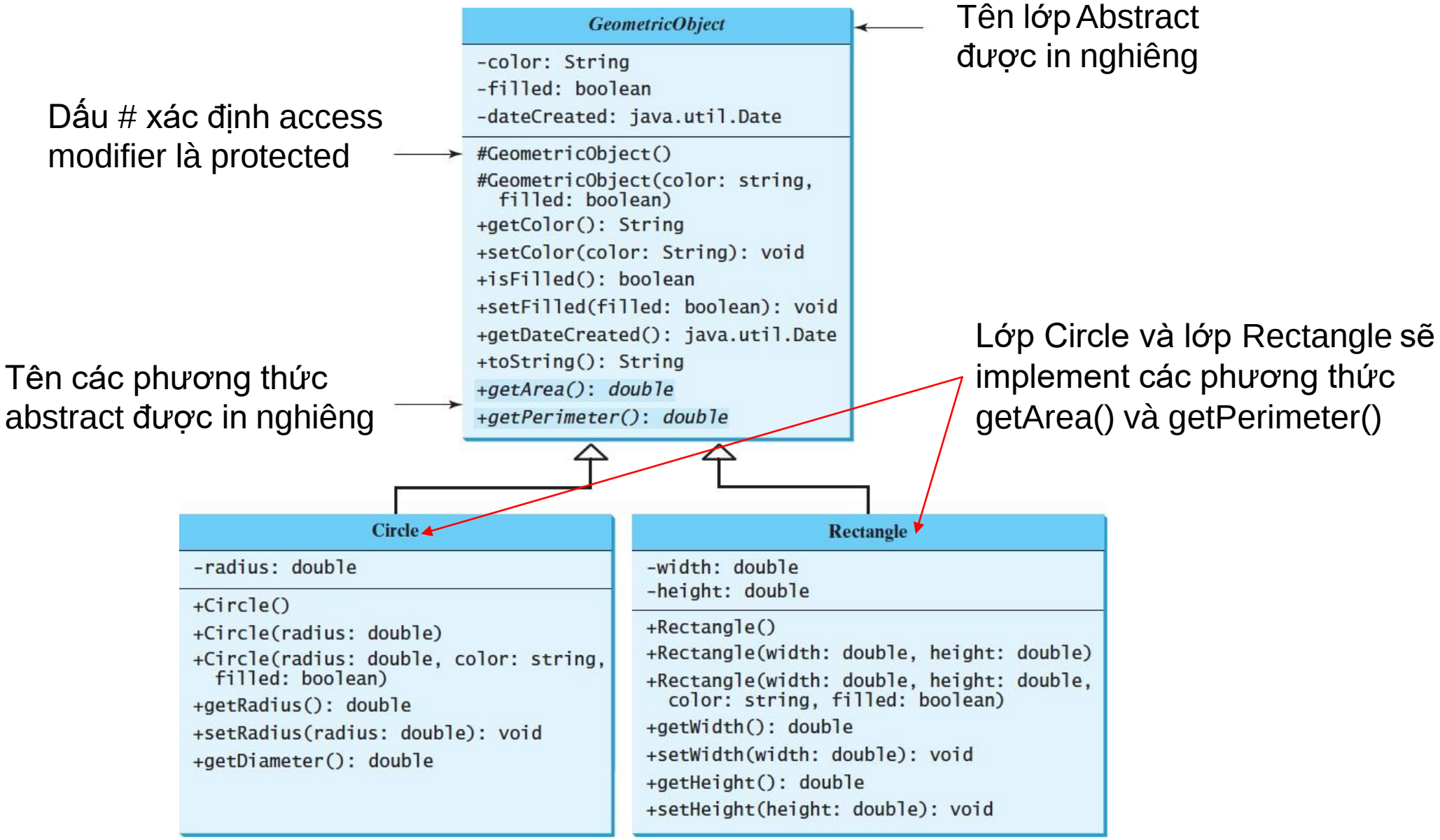
- Abstract method (phương thức trừu tượng) là những phương thức được khai báo (declare) nhưng không có phần thân (không được implement)
- Ví dụ:
  - Lớp Geometric có thể khai báo phương thức getArea() và getPerimeter() nhưng không có phần thân của hai phương thức này
  - Tất cả các "Hình" đều có thể 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:

```
public abstract class Geometric {
    private String name;

    protected Geometric(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public abstract double getArea();

    public abstract double 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 abstract thì không thể là final
- Phương thức abstract thì không thể là final

Demo

Từ khoá abstract

Abstract class

Abstract method

Thảo luận

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:

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

- Ví dụ:

```
public interface Flyable{
}
```

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:

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

public class Bird implements Flyable{
    @Override
    public String fly(){
        return "Flying with 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:

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

public interface AnimalFlyable extends Flyable{}

public 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:

```
public interface Flyable{
    public String 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 modifiler cho hằng số, mặc định là public
- Không cần thiết phải ghi rõ từ khoá final cho hằng số
- Ví dụ:

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

Demo

Interface

Thảo luận

Anonymous class

Anonymous class

- Anonymous class (lớp nặc danh) là một lớp đặc biệt, được khai báo và khởi tạo đối tượng tại cùng một thời điểm
- Anonymous class cần kế thừa một lớp hoặc triển khai một interface
- Ví dụ:

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

```
public static void main(String[] args) {
    Flyable flyableObj = new Flyable() {
        @Override
        public String fly(){
            return "Flying...";
        }
    };

    System.out.println(flyableObj.fly());
}
```

Thảo luận

Cohesion
Consistency
Encapsulation
Clarity

Cohesion

- Cohesion (tính gắn kết) có nghĩa là mỗi lớp chỉ nên đại diện cho một thực thể nhất định
- Tất cả các phương thức của lớp cần phối hợp cùng nhau hợp lý để hỗ trợ cho tính chất cohesion
- Ví dụ
  - Tính cohesion thấp:
    - Định nghĩa lớp Student (sinh viên) và thực hiện thêm các chức năng của Staff (nhân viên)
  - Tính cohsion cao:
    - Định nghĩa lớp Student và lớp Staff để thực hiện các nhiệm vụ riêng phù hợp
    - Có thể định nghĩa thêm lớp User để thực hiện các nhiệm vụ chung của hai thực thể

Cohesion: Ví dụ

- Các lớp String, StringBuilder và StringBuffer đều thực hiện các thao tác với chuỗi
- Tác thành 3 lớp khác nhau giúp nhiệm vụ của mỗi lớp rõ ràng hơn
  - String: Thao tác với chuỗi cố định (inmutable string)
  - StringBuilder: Thao tác với chuỗi thay đổi được (mutable string)
  - StringBuffer: Tương tự StringBuilder nhưng có thêm các thao tác làm việc đồng bộ (synchronized)

Consistency

- Consistency (tính đồng nhất) là tuân thủ các tiêu chuẩn của Java và các quy ước đặt tên
- Chọn các tên phù hợp cho lớp, thuộc tính và phương thức
- Cấu trúc của một lớp lần lượt là: Các trường dữ liệu, các constructor, các phương thức
- Nên định nghĩa một constructor không có tham số cho lớp

Encapsulation

- Nên sử dụng từ khoá private đối với các trường dữ liệu
- Định nghĩa phương thức getter nếu muốn lấy được giá trị của thuộc tính
- Định nghĩa phương thức setter nếu muốn thay đổi giá trị của thuộc tính

Clarity

- Clarity (tính rõ ràng) có nghĩa là nhiệm vụ của các lớp, của các phương thức cần phải dễ hiểu, dễ giải thích
- Các lớp, các phương thức có thể được sử dụng kết hợp với nhau theo nhiều cách khác nhau, do đó sự rõ ràng là cần thiết
- Các thuộc tính trong một lớp nên độc lập với nhau, tránh thừa dữ liệu
- Ví dụ, lớp Person:

```
class Person{
    private Date birthDay;
    private int age;
}
```

  - Thuộc tính *age* có thể tính được dựa vào thuộc tính *birthDay*

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)*

Tổng kết

- 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 khoá implement được sử dụng để triển khai interface
- Interface có thể kế thừa interface khác
- Anonymous class là lớp được khai báo và khởi tạo đối tượng tại một thời điểm
- Thiết kế các lớp cần tuân thủ: Tính gắn kết, tính bao gói, tính đồng nhất, tính rõ ràng

Hướng dẫn

Hướng dẫn làm bài thực hành và bài tập

Chuẩn bị bài tiếp theo: Clean Code