



Bài 6

Inheritance

Kiểm tra bài trước

Hỏi và trao đổi về các khó khăn gặp phải trong bài “Access Modifier”

Tóm tắt lại các phần đã học từ bài “Access Modifier”

- Trình bày được cơ chế kế thừa
- Triển khai được cơ chế kế thừa giữa các lớp
- Trình bày được cơ chế ghi đè phương thức (method overriding)
- Biểu diễn được mối quan hệ kế thừa bằng các ký hiệu
- Trình bày được ý nghĩa của từ khoá final
- Trình bày được khái niệm Polymorphism
- Trình bày được phương thức toString() của lớp Object
- Trình bày được cơ chế ép kiểu (casting)

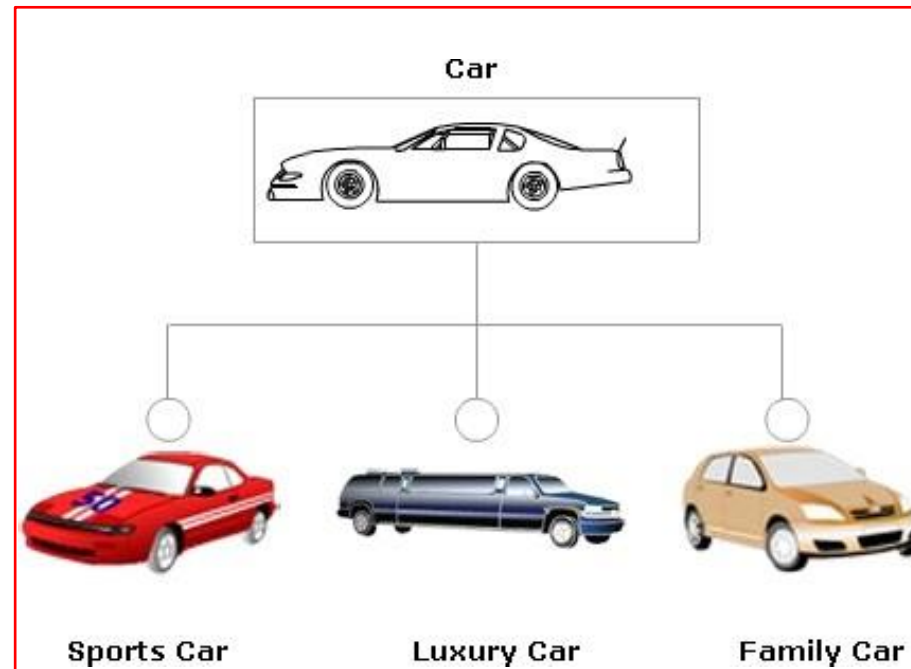


Inheritance

Kế thừa



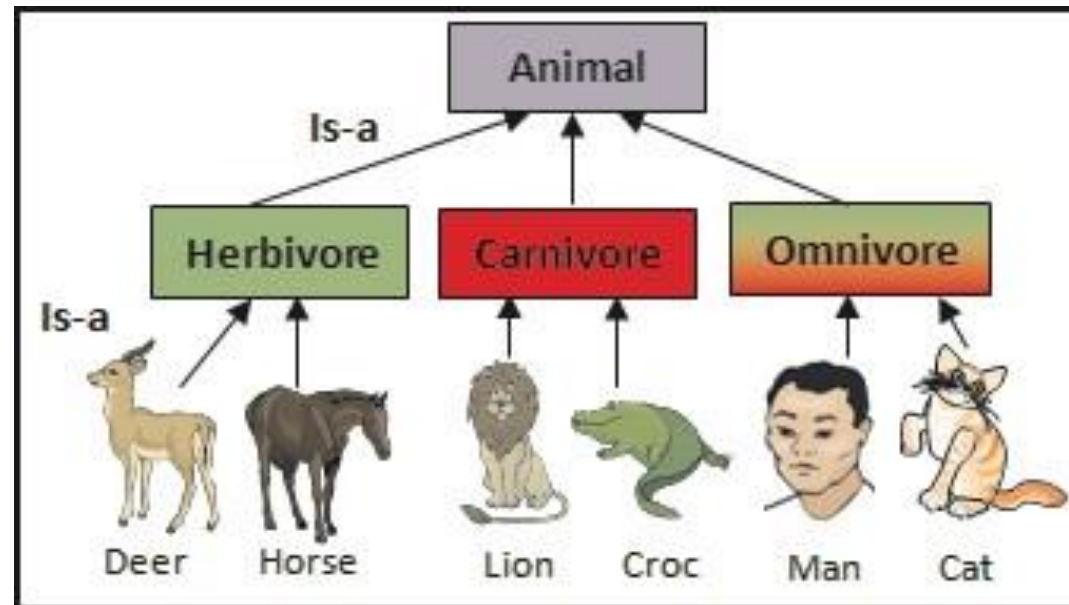
- Kế thừa là cơ chế cho phép một lớp Con sử dụng lại các đặc điểm và hành vi đã được định nghĩa trong lớp Cha
- Ví dụ
 - Lớp Cha: Car
 - Lớp Con: Sports Car, Luxury Car, Family Car



Quan hệ is-a



- Quan hệ giữa lớp con và lớp cha là quan hệ *is-a* (là-một)
- Ví dụ: Ngựa là một động vật ăn cỏ, sư tử là một động vật ăn thịt, động vật ăn cỏ là một động vật...



Các khái niệm



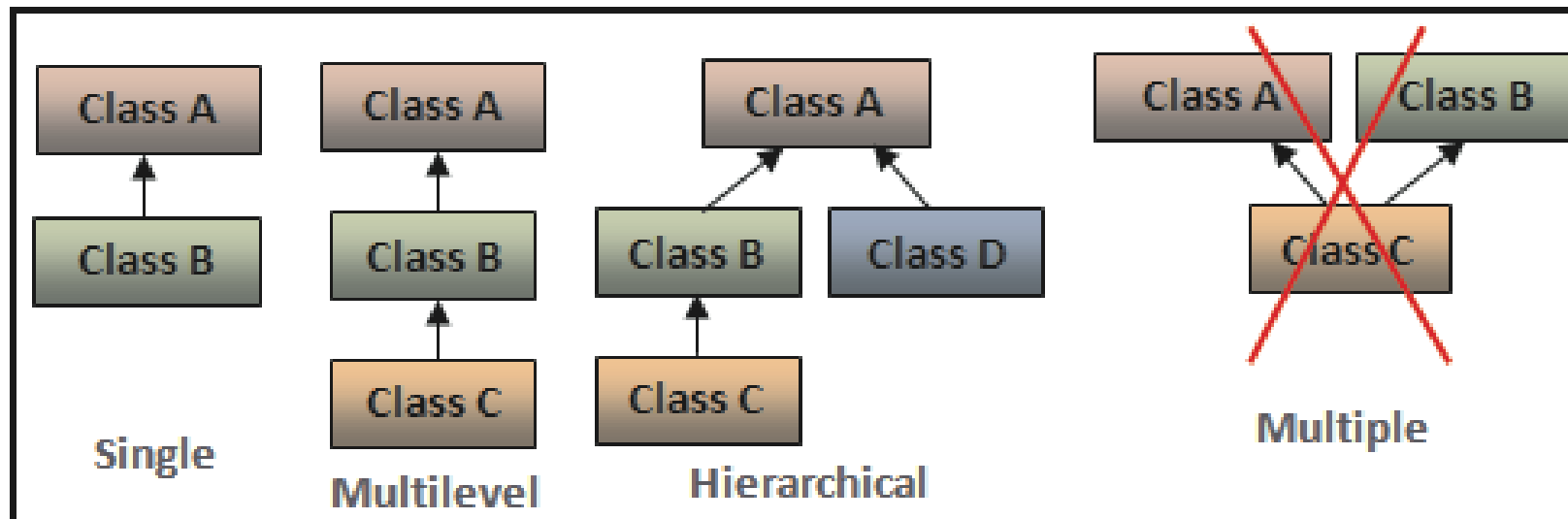
- Lớp được kế thừa gọi là lớp cha (parent class) hoặc lớp cơ sở (base class) hoặc superclass
- Lớp kế thừa gọi là lớp con (child class) hoặc lớp dẫn xuất (derived class) hoặc subclass
- Lớp con kế thừa tất cả các thành phần của lớp cha, ngoại trừ các thành phần được khai báo là *private*
- Constructor không được kế thừa
- Lớp con có thể gọi constructor của lớp cha
- Lớp con có thể định nghĩa thêm các thuộc tính và phương thức mới
- Java không cho phép đa kế thừa (một lớp kế thừa nhiều lớp cha)



Một số dạng kế thừa



- **Single:** Một lớp kế thừa từ chỉ một lớp cha
- **Multilevel:** Một lớp kế thừa từ một lớp cha, lớp cha lại kế thừa từ lớp khác ở trên nó
- **Hierarchical:** Một lớp cha có nhiều lớp con với nhiều level khác nhau
- **Multiple:** Một lớp con kế thừa từ nhiều lớp cha



Cú pháp kế thừa



- Từ khoá `extends` được sử dụng để kế thừa một lớp
- Cú pháp:

```
class SubClass extends SupperClass
{
    //Class body
}
```



Trong đó:

- SubClass là tên của lớp con
- SupperClass là tên của lớp cha

Kế thừa: Ví dụ



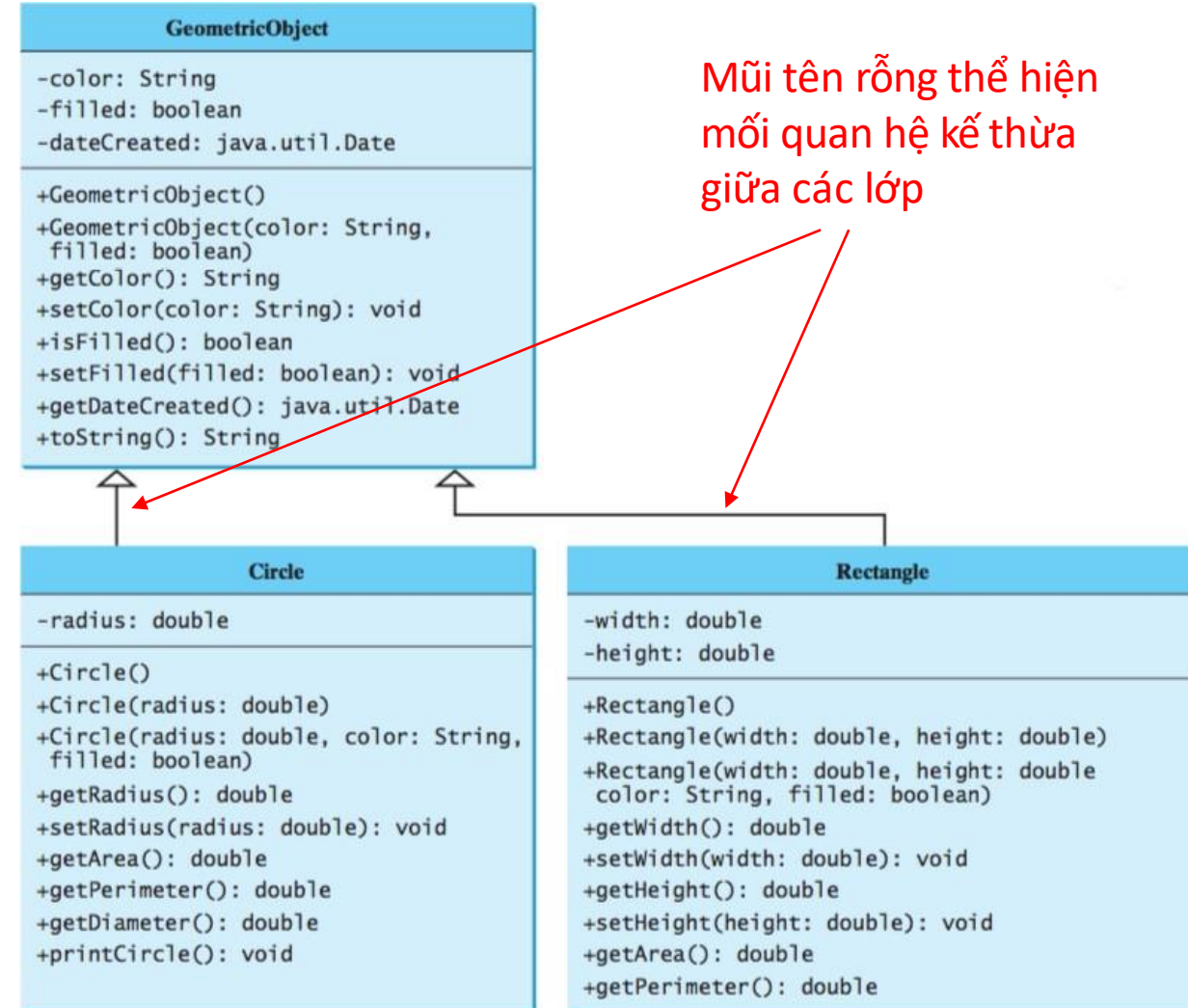
```
class GeometricObject{  
}
```

```
class Circle extends GeometricObject{  
}
```

```
class Rectangle extends GeometricObject{  
}
```

```
Circle circle = new Circle(1);
```

```
System.out.println("A circle " + circle.toString());  
System.out.println("The color is " + circle.getColor());  
System.out.println("The radius is " + circle.getRadius());  
System.out.println("The area is " + circle.getArea());  
System.out.println("The diameter is " + circle.getDiameter());
```





Method overriding

Method overriding



- Method Overriding (ghi đè phương thức) là cơ chế cho phép lớp con định nghĩa lại các phương thức đã được định nghĩa trước đó ở lớp cha
- Phương thức override ở lớp con có cùng tên, cùng danh sách tham số và kiểu dữ liệu trả về so với phương thức ở lớp cha
- Phương thức ở lớp con phải có access modifier có level bằng hoặc cao hơn so với phương thức ở lớp cha
- Từ khoá override được sử dụng để ghi đè phương thức
- `@Override` có thể được sử dụng để đánh dấu phương thức ghi đè

Method overriding: Ví dụ



```
public class Geometric {  
    public String getName() {  
        return "I am a Geometric object";  
    }  
}
```

```
public class Rectangle extends Geometric {
```

```
    @Override
```

```
    public String getName() {  
        return "I am a Rectangle object";  
    }  
}
```

```
public static void main(String[] args) {  
    Geometric geoObj = new Geometric();  
    System.out.println(geoObj.getName());
```

```
    Rectangle rectObj = new Rectangle();  
    System.out.println(rectObj.getName());  
}
```

I am a Geometric object
I am a Rectangle object

Từ khoá super



- Từ khoá `super` được sử dụng ở lớp con để gọi đến constructor hoặc phương thức của lớp cha
- Ví dụ, gọi constructor của lớp cha:

```
public class Geometric{  
    private String name;  
  
    public Geometric(String name) {  
        this.name = name;  
    }  
}
```

```
public class Rectangle extends Geometric {  
    private int width;  
    private int height;  
  
    public Rectangle(String name, int width, int height) {  
        super(name);  
        this.width = width;  
        this.height = height;  
    }  
}
```

Từ khoá super: Gọi phương thức của lớp cha



- Ví dụ:

```
public class Circle {  
    private int radius;  
  
    public Circle(int radius) {  
        this.radius = radius;  
    }  
  
    public double getArea(){  
        return Math.PI * this.radius * this.radius;  
    }  
  
    public double getPerimeter(){  
        return Math.PI * 2 * this.radius;  
    }  
}
```

```
public class Cylinder extends Circle {  
    private int height;  
  
    public Cylinder(int radius, int height) {  
        super(radius);  
        this.height = height;  
    }  
  
    @Override  
    public double getArea() {  
        return super.getArea() * 2 + super.getPerimeter() * height;  
    }  
  
    public double getVolume(){  
        return super.getArea() * this.height;  
    }  
}
```



Phương thức toString()

Lớp Object và phương thức toString()



- Lớp Object là lớp gốc của tất cả các lớp trong Java
- Tất cả các lớp trong Java đều kế thừa từ lớp Object
- Lớp Object có một phương thức được sử dụng thông dụng đó là toString(): Trả về một chuỗi mô tả đối tượng
- Mô tả của phương thức toString() là:

public String toString()

- Ví dụ:



```
System.out.println(circle.toString());
```

Override phương thức toString()



- Các lớp có thể ghi đè phương thức toString() để mô tả đối tượng tốt hơn

• Ví dụ:

```
public class Circle{  
    private int radius;  
  
    public Circle(int radius){  
        this.radius = radius;  
    }  
  
    @Override  
    public String toString(){  
        return "I am a Circle, my radius is " + this.radius;  
    }  
}
```



Polymorphism

Polymorphism

Dynamic Binding

Polymorphism



- Polymorphism (Đa hình) là cơ chế cho phép một biến thuộc kiểu dữ liệu cha có thể trở đến một đối tượng thuộc lớp con
- Khi khai báo một lớp, đồng nghĩa với tạo ra một kiểu dữ liệu mới
- Khi một lớp con kế thừa lớp cha thì kiểu dữ liệu của lớp cha được gọi là supertype, kiểu dữ liệu của lớp con được gọi là subtype
- Ví dụ: *Geometric* là supertype của *Circle*, và *Circle* là subtype của *Geometric*. Tất cả các đối tượng của lớp *Circle* đều là *Geometric*, nhưng không phải ngược lại.
- Tính đa hình, cho phép khai báo sau:

Geometric geometricObj = **new** Circle(1);



Polymorphism: Ví dụ



```
public static void main(String[] args) {  
    displayGeometricObject(new Circle(1, "red", false));  
    displayGeometricObject(new Rectangle(1, 1, "black", true));  
}
```

```
public static void displayGeometricObject(Geometric geometricObj){  
    System.out.println("Created on " + geometricObj.getDateCreated()+  
        ". Color is " + geometricObj.getColor());  
}
```

Phương thức displayGeometricObject() sẽ cho kết quả khác nhau, tùy thuộc vào đối tượng cụ thể được truyền vào

Dynamic Binding



- Khi triển khai kế thừa, một phương thức có thể *được viết* (và viết lại - override) ở những lớp khác nhau
- Dynamic Binding là cơ chế của JVM để xác định sẽ gọi phương thức nào tại thời điểm thực thi
- Ví dụ, phương thức toString() nào sẽ được gọi?

```
Object o = new Circle();  
System.out.println(o.toString());
```

Kiểu khai báo và Kiểu thực tế



- Kiểu khai báo của một biến (declared type) là kiểu sử dụng trong bước khai báo
- Kiểu thực tế (actual type) của một biến là kiểu của đối tượng mà biến trỏ đến
- Ví dụ:

Object o = **new** Circle();

Trong đó:

- Kiểu khai báo của biến o là Object
 - Kiểu thực tế của biến o là Circle
- Cơ chế Dynamic Binding sẽ dựa vào kiểu thực tế để gọi phương thức



Ép kiểu (Casting)

Ép kiểu

Toán tử instanceof

Ép kiểu (Casting)



- Ép kiểu là cơ chế chuyển đổi một tham chiếu đến đối tượng thuộc loại này thành tham chiếu đến đối tượng thuộc loại khác
- Có 2 loại ép kiểu:
 - Implicit casting (ép kiểu ngầm định): Ép từ subtype lên supertype
 - Explicit casting (ép kiểu tường minh): Ép từ supertype xuống subtype

- Ví dụ:

Object o = **new** Circle(); ← Implicit casting

Geometric g = **new** Circle(); ←

Circle c = (Circle)g; ← Explicit casting

Explicit casting



- Sẽ xảy ra lỗi Compile nếu 2 kiểu dữ liệu không nằm trong chuỗi kế thừa
 - Ví dụ, 2 lớp Animal và Circle độc lập với nhau trong chuỗi kế thừa:

```
Animal a = new Animal();
```

```
Circle c = (Circle)a; _____ Lỗi compile
```

- Sẽ xảy ra lỗi Runtime nếu kiểu dữ liệu thực tế không phù hợp với kiểu dữ liệu được ép sang
 - Ví dụ, kiểu dữ liệu Rectangle không thể ép sang kiểu Circle:

```
Geometric g = new Rectangle();
```

```
Circle c = (Circle)g; _____ Lỗi runtime: ClassCastException
```

Toán tử instanceof



- Toán tử instanceof giúp kiểm tra kiểu của một đối tượng
- Giá trị trả về có kiểu boolean
- Ví dụ:

```
if (myObject instanceof Circle){  
    System.out.println("The circle diameter is " + ((Circle)myObject).getDiameter());  
}
```


Lưu ý: Nếu không ép sang kiểu Circle thì không thể gọi phương thức getDiameter() được.



Từ khoá final

Từ khoá final



- Từ khoá final áp dụng cho lớp và phương thức để ngăn ngừa việc kế thừa và ghi đè phương thức
- Ví dụ: 

```
public final class Animal{  
  
}
```

```
public class Circle{  
    public final void display(){  
  
    }  
}
```

- Từ khoá static *được sử dụng để* khai báo các thành phần thuộc lớp
- package *được sử dụng để* nhóm các lớp có liên quan đến nhau trong cùng một đơn vị
- Getter/setter là cơ chế để kiểm soát truy cập đến các trường dữ liệu của đối tượng

- Kế thừa là cơ chế cho phép một lớp thừa hưởng các đặc điểm và hành vi của một lớp khác
- Lớp được kế thừa gọi là lớp cha, lớp kế thừa gọi là lớp con
- Ghi đè phương thức là hình thức lớp con viết lại các phương thức đã có của lớp cha
- Sử dụng mũi tên rỗng để biểu diễn mối quan hệ kế thừa giữa các lớp
- Java không hỗ trợ đa kế thừa
- Từ khoá final được sử dụng để ngăn chặn việc kế thừa từ một lớp và việc ghi đè phương thức
- Đa hình là cơ chế cho phép một biến kiểu cha có thể trỏ đến các đối tượng kiểu con
- Lớp Object là lớp cha của tất cả các lớp trong Java
- Phương thức toString() được sử dụng để trả về một chuỗi mô tả đối tượng
- Ép kiểu là hình thức chuyển đổi tham chiếu đối tượng từ một kiểu này sang tham chiếu đối tượng thuộc kiểu khác

Hướng dẫn

Hướng dẫn làm bài thực hành và bài tập

Chuẩn bị bài tiếp theo: *Interface và Abstract class*