面向对象三大特性

> Author：zhangzhang
>
> Version：1.0.0

一、引言

> 面向对象三大特性：封装、继承、多态。

二、封装【 重点 】

2.1 封装的必要性

```java
public class TestEncapsulation {
    public static void main(String[] args) {
        Student s1 = new Student();
        s1.name = "tom";
        s1.age = 20000;
        s1.sex = "male";
        s1.score = 100D;
    }
}

class Student{
    String name;
    int age;
    String sex;
    double score;
}
```

在对象的外部，为对象的属性赋值，可能存在非法数据的录入。

就目前的技术而言，并没有办法对属性的赋值加以控制。

- 在对象的外部，为对象的属性赋值，可能存在非法数据的录入。
- 就目前的技术而言，并没有办法对属性的赋值加以控制。

2.2 什么是封装

概念：尽可能隐藏对象的内部实现细节，控制对象的修改及访问的权限。

访问修饰符：private （可将属性修饰为私有，仅本类可见）

```java
public class TestEncapsulation {
    public static void main(String[] args) {
        Student s1 = new Student();
        s1.age = 20000;
    }
}

class Student{
    String name;
    private int age;
    String sex;
    double score;
}
```

The field Student.age is not visible

编译错误：私有属性在类的外部不可访问

如何在提供正常的对外访问渠道的同时，又能控制录入的数据有效？

2.3 公共访问方法

```java
public class TestEncapsulation {
    public static void main(String[] args) {
        Student s1 = new Student();
        s1.setAge(20000);
        System.out.println(s1.getAge());
    }
}

class Student{
    String name;
    private int age;
    String sex;
    double score;

    public void setAge(int age){
        this.age = age;
    }

    public int getAge(){
        return this.age;
    }
}
```

以访问方法的形式，进而完成赋值与取值操作。
问题：依旧没有解决到非法数据录入！

提供公共访问方法，以保证数据的正常录入。

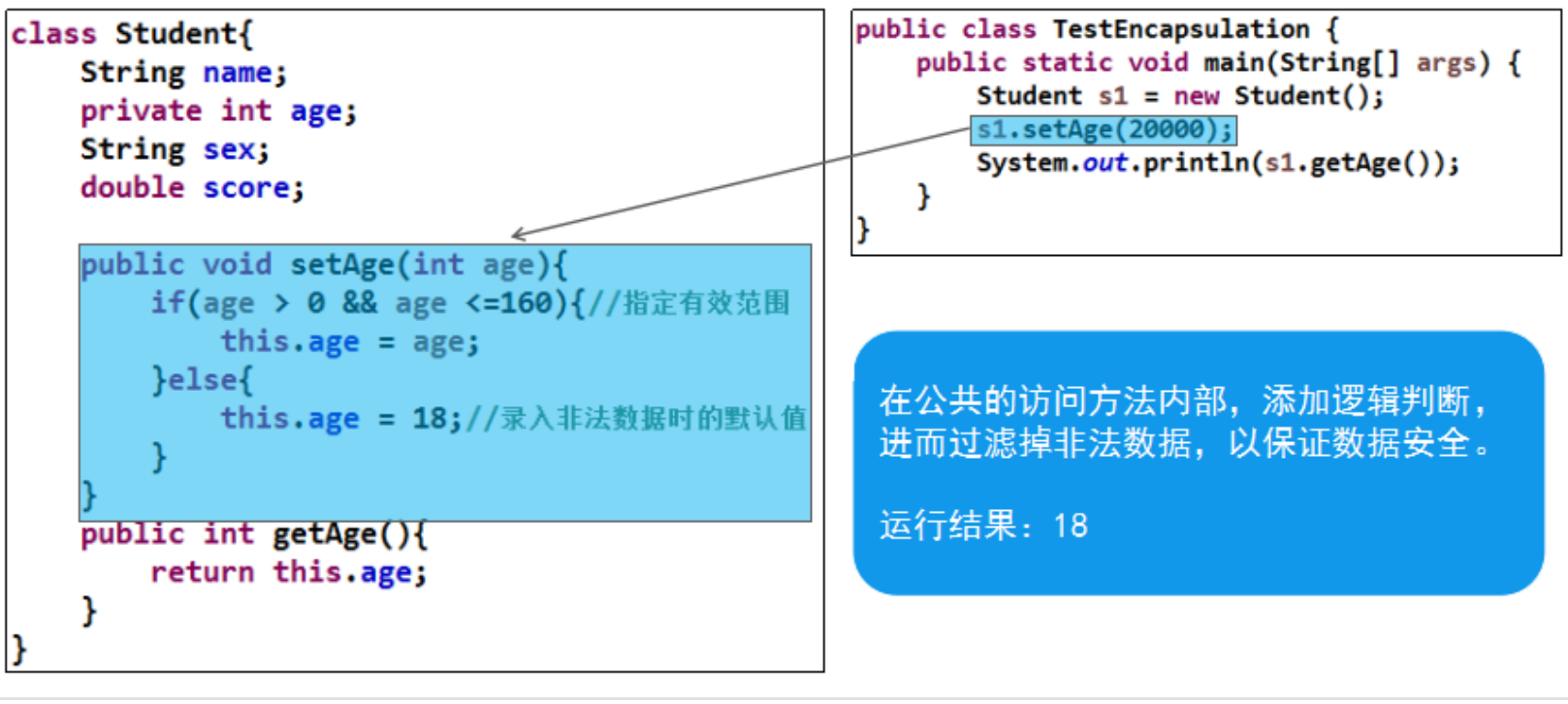
命名规范：
赋值：setXXX() //使用方法参数实现赋值
取值：getXXX() //使用方法返回值实现取值

以访问方法的形式，进而完成赋值与取值操作。
问题：依旧没有解决到非法数据录入！

- 提供公共访问方法，以保证数据的正常录入。
- 命名规范：

- 赋值：setXXX() //使用方法参数实现赋值
- 取值：getXXX() //使用方法返回值实现取值

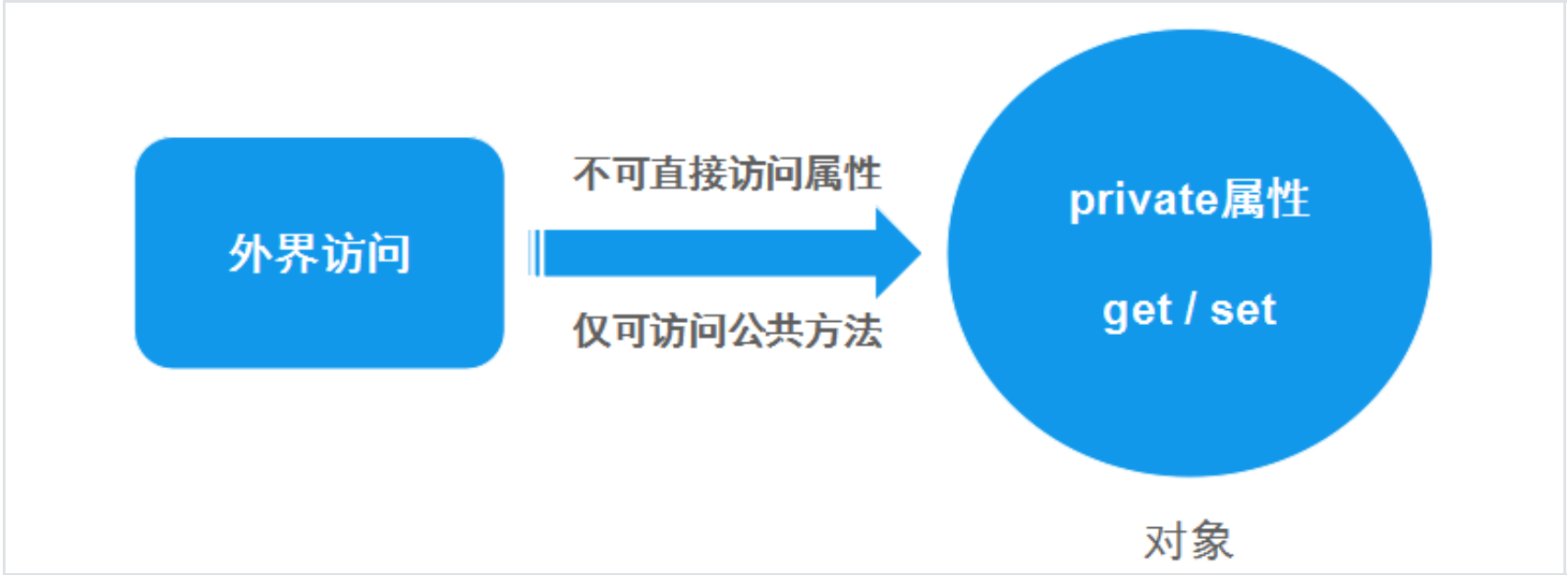
2.4 过滤有效数据



> 在公共的访问方法内部，添加逻辑判断，进而过滤掉非法数据，以保证数据安全。

```
public class TestEncapsulation {
  public static void main(String[] args) {

    Student s1 = new Student();

    s1.name = "tom";
    //s1.age = 20;//安全，无法直接访问到私有属性
    s1.setAge(20);//调用方法完成赋值操作

    s1.sex = "male";
    s1.score = 99.0;

    System.out.println( s1.getAge() );//调用方法完成取值操作

  }
}

class Student{
  String name; //属性也称为字段
  private int age;//私有属性，本类可见，其他位置不可见
  String sex;
  double score;

  public Student() {}

  /**
   * 为age属性赋值的方法
   */
  public void setAge(int age) {//20000
    if(age >= 0 && age <= 153) {//合法区间
      this.age = age;
    }else {
      this.age = 18;//如果录入的数据不合法，默认使用18代表用户的年龄
    }
  }

  /**
   * 为age属性取值的方法
   */
  public int getAge() {
    return this.age;
  }
}
```

2.5 总结



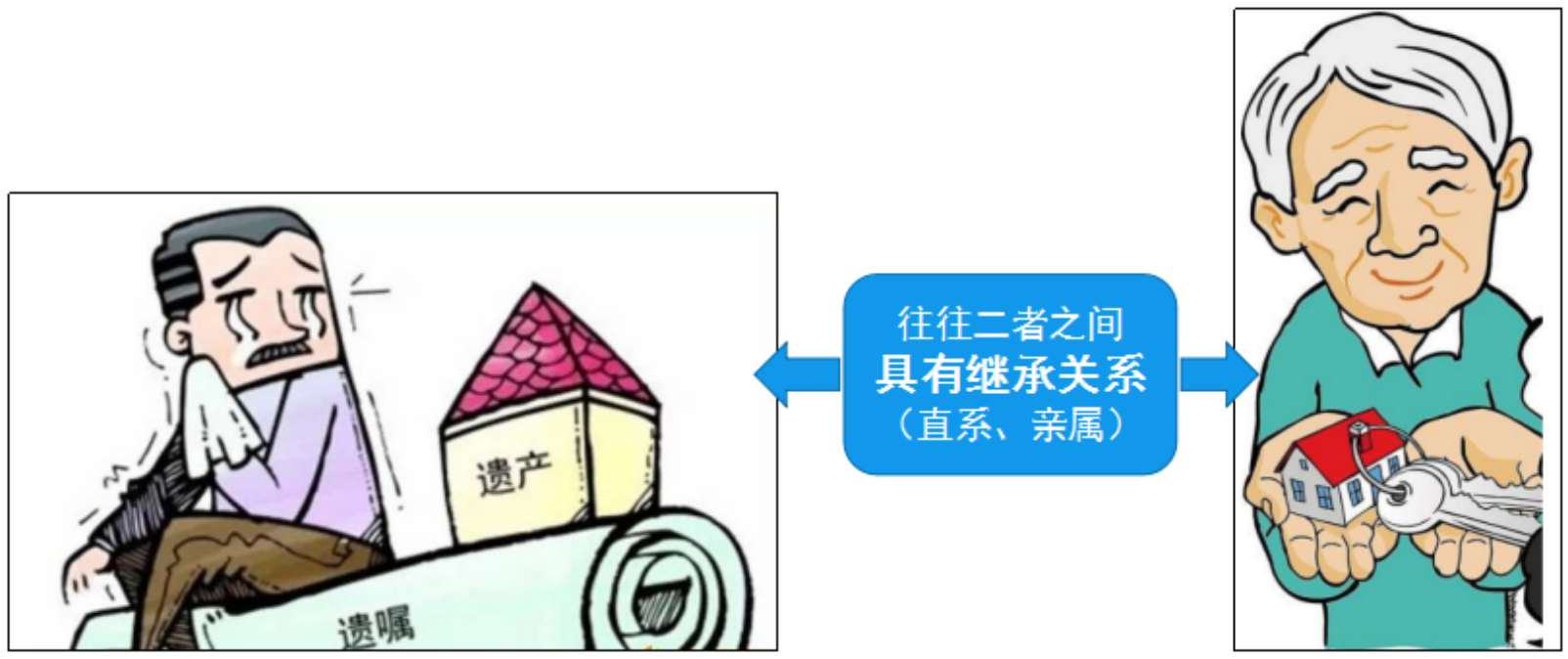
> get/set方法是外界访问对象私有属性的唯一通道，方法内部可对数据进行检测和过滤。

```
public class TestEncapsulation2 {

  public static void main(String[] args) {

    Teacher t1 = new Teacher();

    t1.setName("jack");
    t1.setAge(20);
    t1.setSex("male");
    t1.setSalary(2000.0);

    System.out.println(t1.getName() +"\t"+ t1.getAge() +"\t"+ t1.getSex() +"\t"+ t1.getSalary());

  }

}

class Teacher{
  private String name;
  private int age;
  private String sex;
  private double salary;

  public Teacher() {}

  public void setName(String name) {
    this.name = name;
  }

  public String getName() {
    return this.name;
  }

  public void setAge(int age) {
    this.age = age;
  }

  public int getAge() {
    return this.age;
  }

  public void setSex(String sex) {
    this.sex = sex;
  }

  public String getSex() {
    return this.sex;
  }

  public void setSalary(double salary) {
    this.salary = salary;
  }

  public double getSalary() {
    return this.salary;
  }


}
```

三、继承【重点】

3.1 生活中的继承

- 生活中的“继承”是施方的一种赠与，受方的一种获得。
- 将一方所拥有的东西给予另一方。



3.2 程序中的继承

- 程序中的继承，是类与类之间特征和行为的一种赠与或获得。

- 两个类之间的继承关系，必须满足“is a”的关系。

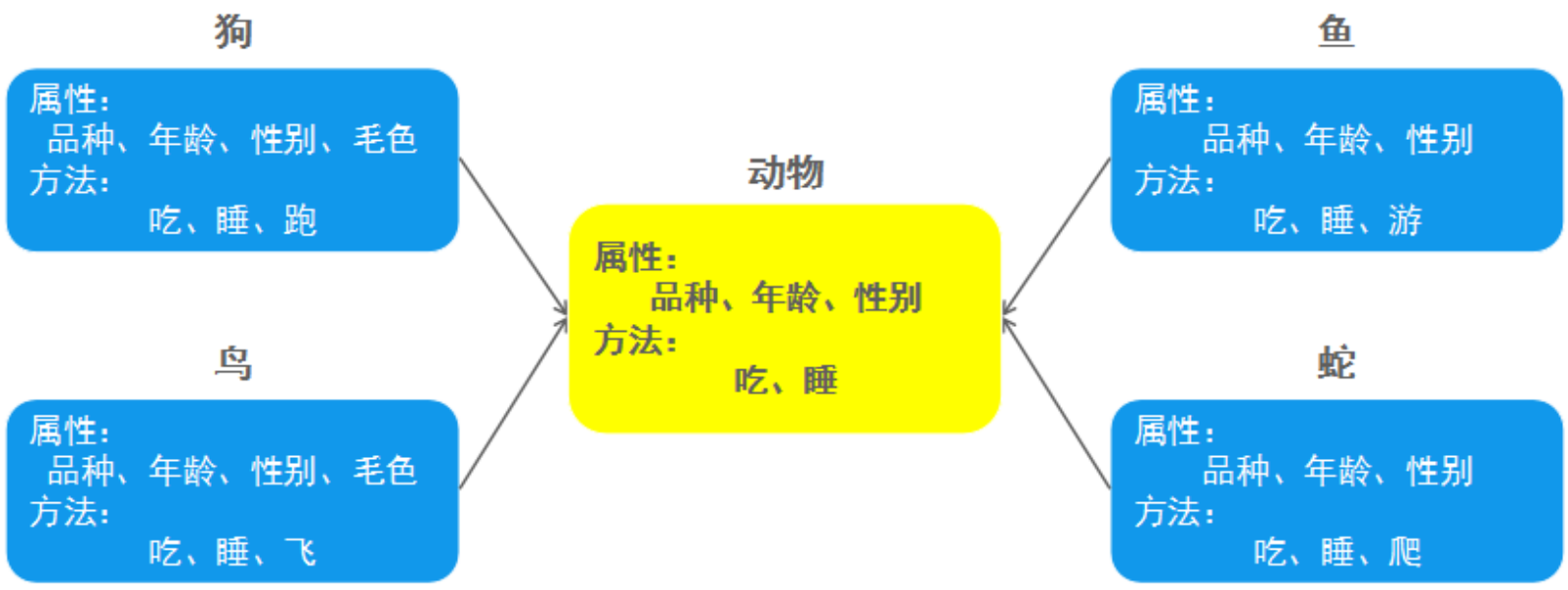
3.3 父类的选择

- 现实生活中，很多类别之间都存在着继承关系，都满足“is a”的关系。
- 狗是一种动物、狗是一种生物、狗是一种物质。
- 多个类别都可作为“狗”的父类，需要从中选择出最适合的父类。

- 功能越精细，重合点越多，越接近直接父类。
- 功能越粗略，重合点越少，越接近Object类。（万物皆对象的概念）

3.4 父类的抽象

实战：可根据程序需要使用到的多个具体类，进行共性抽取，进而定义父类。



> 在一组相同或类似的类中，抽取出共性的特征和行为，定义在父类中，实现重用。

```
public class TestExtends {
  public static void main(String[] args) {

    Dog dog1 = new Dog();
    dog1.breed = "哈士奇"; //继承自父类
    dog1.age = 2;  //继承自父类
    dog1.sex = "公";  //继承自父类
    dog1.furColor = "灰白"; //子类独有的

    dog1.eat();  //继承自父类
    dog1.sleep();  //继承自父类
    dog1.run(); //子类独有的

    //----------------------------------------

    Bird bird1 = new Bird();
    bird1.breed = "麻雀";
    bird1.age = 1;
    bird1.sex = "雄";
    bird1.furColor = "棕";

    bird1.eat();
    bird1.sleep();
    bird1.fly();

  }
}

//父类
class Animal{
  String breed;
  int age;
  String sex;

  public void eat() {}
  public void sleep() {}
}


class Dog extends Animal{ //完成继承关系
  String furColor;

  public void run() {}
}


class Bird extends Animal{
  String furColor;

  public void fly() {}
}


class Fish extends Animal{
  public void swim() {}
}


class Snake extends Animal{
  public void climb() {}
}
```

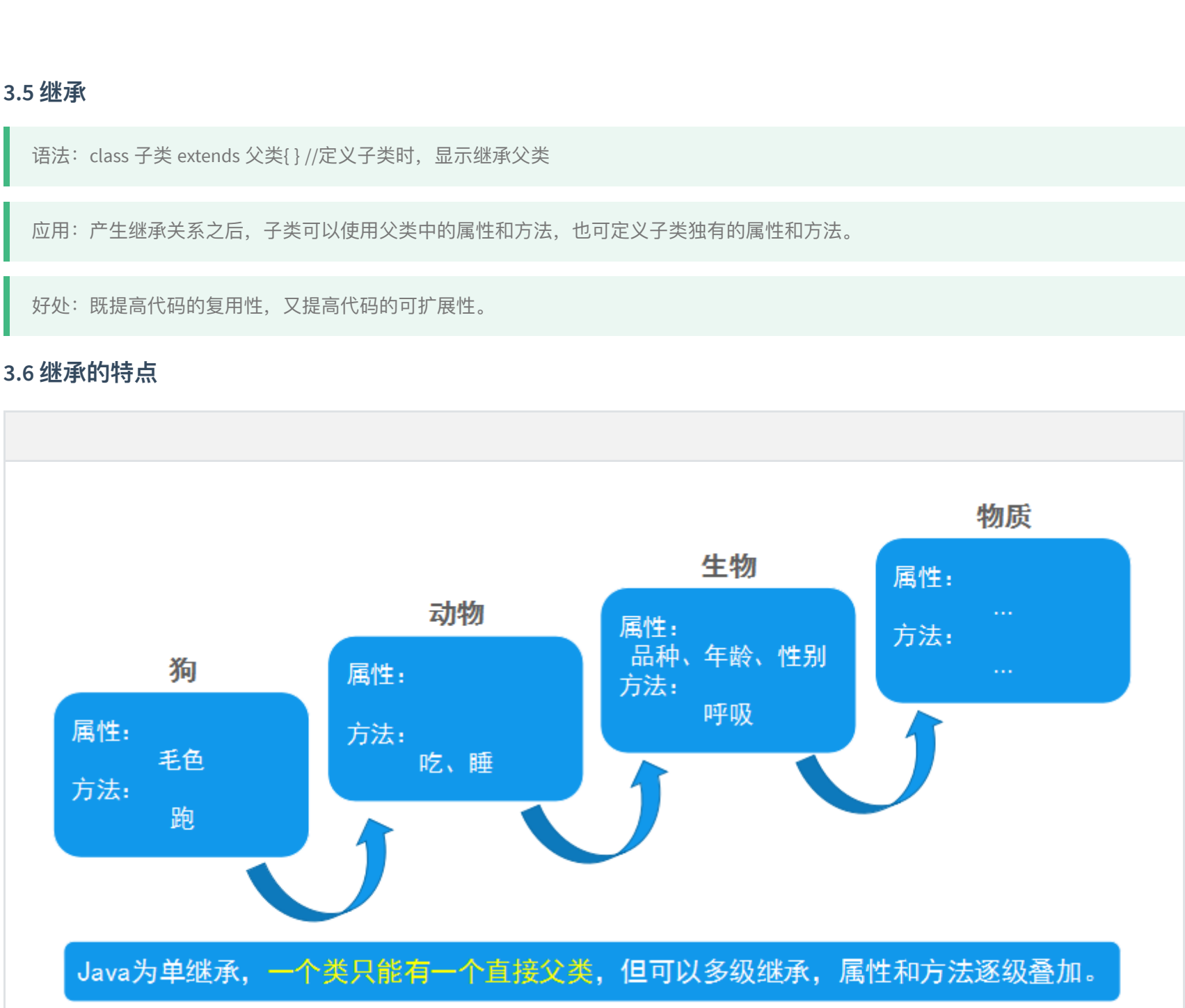
3.5 继承

> 语法：class 子类 extends 父类{ } //定义子类时，显示继承父类

> 应用：产生继承关系之后，子类可以使用父类中的属性和方法，也可定义子类独有的属性和方法。

> 好处：既提高代码的复用性，又提高代码的可扩展性。

3.6 继承的特点



> Java为单继承，一个类只能有一个直接父类，但可以多级继承，属性和方法逐级叠加。

```
public class TestExtends2 {

  public static void main(String[] args) {

    Car car = new Car();
    car.price = 1000000.0;
    car.speed = 120;
    car.brand = "保时捷";
    System.out.println("一辆" + car.brand +"品牌的小汽车正在以" + car.speed +"/H的速度前进");
    car.run();


    Bus bus = new Bus();
    bus.price = 1500000.0;
    bus.speed = 60;
    bus.seatNum = 16;
    System.out.println(bus.price +"\t"+ bus.speed +"\t"+ bus.seatNum);
    bus.run();


    Bicycle bic = new Bicycle();
    bic.price = 2000.0;
    bic.speed = 20;
    bic.color = "红";
    System.out.println(bic.price +"\t"+ bic.speed +"\t"+ bic.color);
    bic.run();

}
}

//交通工具类
class Vehicle{
  double price;//价格
  int speed;//速度

  public void run() {
    System.out.println("行驶中...");
  }
}

class Car extends Vehicle{
  String brand;//品牌

}

class Bus extends Vehicle{
  int seatNum;//座位数
}
```

```
class Bicycle extends Vehicle{
  String color;//颜色
}
```

3.7 不可继承

> 构造方法：类中的构造方法，只负责创建本类对象，不可继承。

> private修饰的属性和方法：访问修饰符的一种，仅本类可见。（详见下图）

> 父子类不在同一个package中时，default修饰的属性和方法：访问修饰符的一种，仅同包可见。（详见下图）

四、访问修饰符

4.1 访问修饰符

	本类	同包	非同包子类	其他
private	√	×	×	×
default	√	√	×	×
protected	√	√	√	×
public	√	√	√	√


```
package com.qf.t2.range.p1;
/**
 * 访问本类
 * 4个访问修饰符，均有效
 */
public class TestSelfClass {

  private String a = "A";//私有
  String b = "B";//默认 default
  protected String c = "C";//受保护
  public String d = "D";//公开

  public void m1() {
    System.out.println(  this.a  );
    System.out.println(  this.b  );
    System.out.println(  this.c  );
    System.out.println(  this.d  );
  }

}
```

```
package com.qf.t2.range.p2;
/**
 * 访问同包下的另一个类
 * 4个访问修饰符，除private以外，其他三个有效
 */
public class TestSamePackage {

  public static void main(String[] args) {

    Target target = new Target();

    System.out.println(target.f);
    System.out.println(target.g);
    System.out.println(target.h);

  }

}
```

```java
package com.qf.t2.range.p2;
public class Target {

  private String e = "E";
  String f = "F";
  protected String g = "G";
  public String h = "H";

}
```

```java
package com.qf.t2.range.p3;

import com.qf.day13.t2.range.p2.Target;

public class ExtendsTarget extends Target {

  public void m2() {

    // 普通访问方式，创建对象，调用属性，跟两者是否具有继承关系无关
    //  Target target = new Target();
    //  System.out.println( target.g );//无法体现继承的关系，protected所修饰的内容，不可见
    //  System.out.println( target.h );

    // 基于继承关系产生后，子类访问父类定义的属性的方式
    System.out.println(this.g); //子类访问父类由protected所修饰的内容，可见
    System.out.println(this.h);

  }
}
```

```java
package com.qf.day13.t2.range.p3;

import com.qf.day13.t2.range.p2.Target;

/**
 * 访问非同包下的另一个类
 * 4个访问修饰符，除public以外，其他三个无效
 */
public class TestNotSamePackage {

  public static void main(String[] args) {
    //Scanner
    //java.util.Arrays.copyOf()
    //java.util.Arrays.sort()

    //写全限定名（包名+类名）
    Target target = new Target();

    System.out.println(target.h);
  }
}
```

五、方法重写

5.1 方法的重写/覆盖

> 思考：子类中是否可以定义和父类相同的方法?

> 思考：为什么需要在子类中定义和父类相同的方法?

> 分析：当父类提供的方法无法满足子类需求时，可在子类中定义和父类相同的方法进行重写（Override）。

> 方法重写原则：
>
> - 方法名称、参数列表、返回值类型必须与父类相同。
> - 访问修饰符可与父类相同或是比父类更宽泛。

> 方法重写的执行：
>
> - 子类重写父类方法后，调用时优先执行子类重写后的方法。

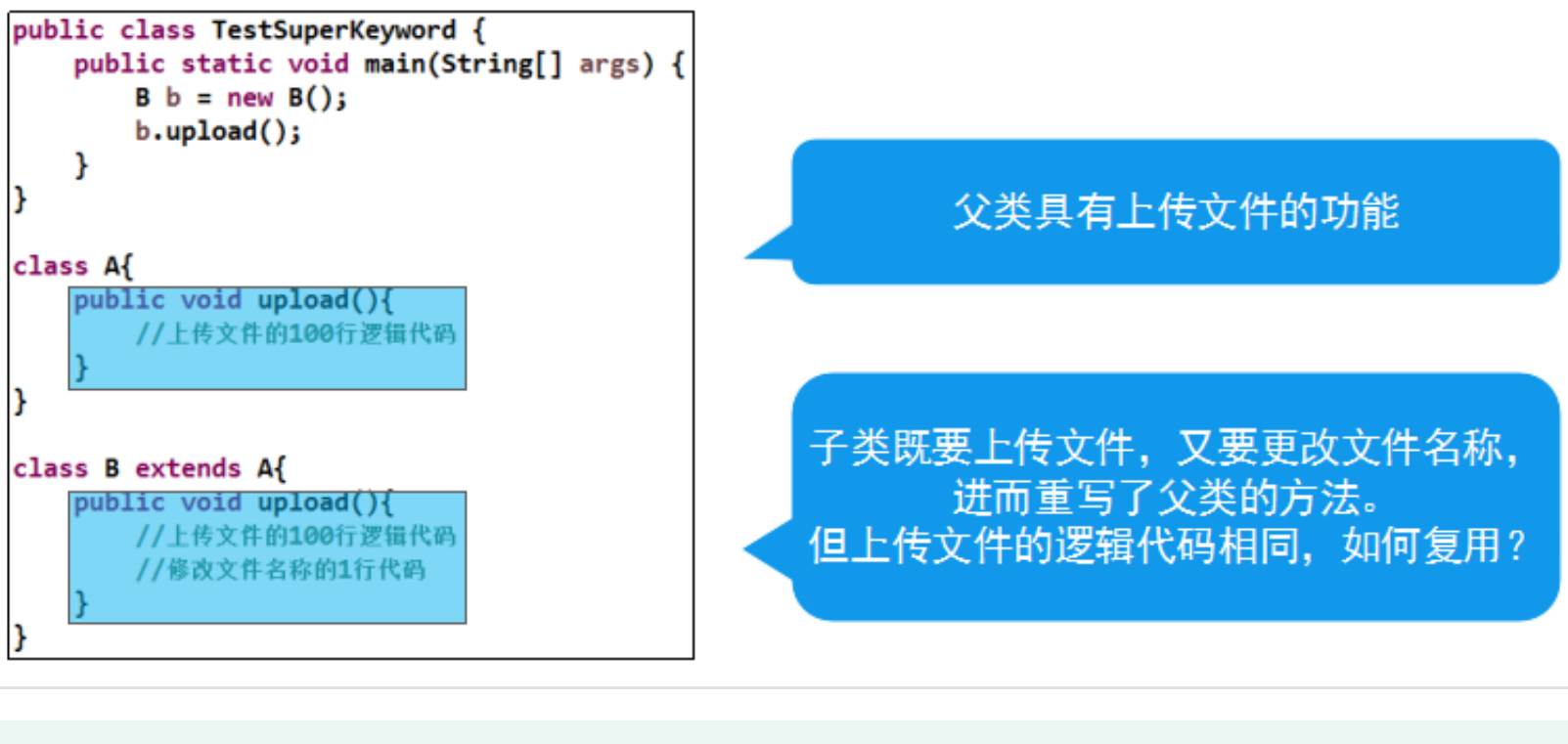
```java
public class TestOverride {
  public static void main(String[] args) {
    Dog dog = new Dog();

    dog.eat();//狗在吃骨头（覆盖后，优先执行子类覆盖之后的版本）

    Cat cat = new Cat();
```

```
    cat.eat();//猫在吃鱼
  }
}

class Animal{
  String breed;
  int age;
  String sex;

  public void eat() {
    System.out.println("动物在吃...");
  }

  public void sleep() {
    System.out.println("动物在睡...");
  }
}


class Dog extends Animal{
  String furColor;

  //子类中定义和父类相同的方法进行覆盖
  public void eat() {
    System.out.println("狗在吃骨头...");
  }

  public void sleep() {
    System.out.println("狗在趴着睡...");
  }

  public void run() {

  }
}

class Cat extends Animal{

}

class Fish extends Animal{

}
```

六、super关键字

6.1 super关键字

> 在子类中，可直接访问从父类继承到的属性和方法，但如果父子类的属性或方法存在重名（属性遮蔽、方法重写）时，需要加以区分，才可专项访问。



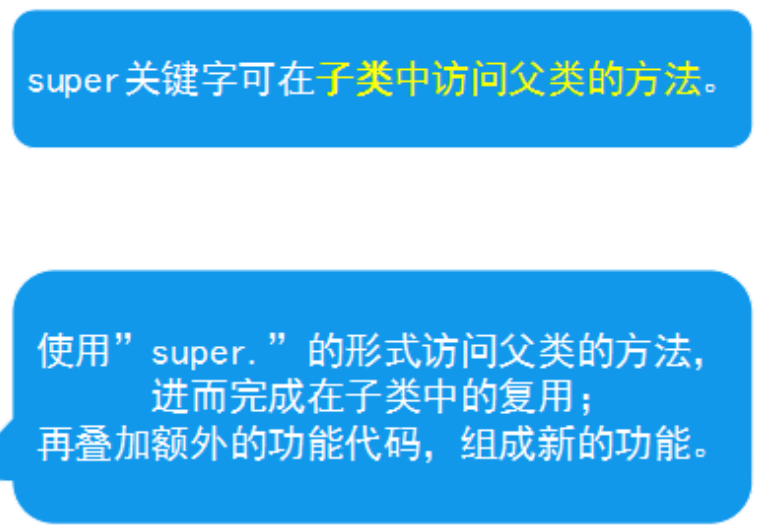
> 父类具有上传文件的功能

> - 子类既要上传文件，又要更改文件名称，进而重写了父类的方法。
> - 但上传文件的逻辑代码相同，如何复用?

6.2 super访问方法

```java
public class TestSuperKeyword {
    public static void main(String[] args) {
        B b = new B();
        b.upload();
    }
}

class A{
    public void upload(){
        //上传文件的100行逻辑代码
    }
}

class B extends A{
    public void upload(){
        super.upload();//上传文件的100行逻辑代码
        //修改文件名称的1行代码
    }
}
```

super关键字可在子类中访问父类的方法。

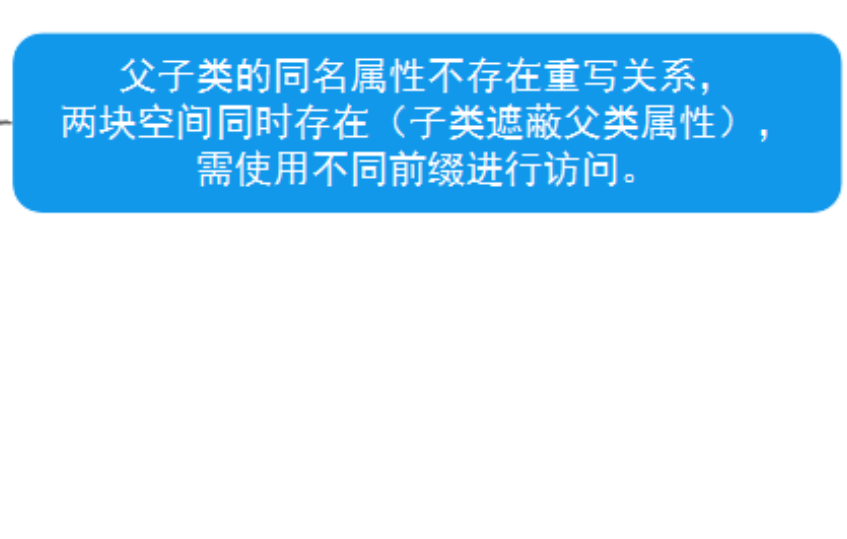


> super关键字可在子类中访问父类的方法。

- 使用”super.”的形式访问父类的方法，进而完成在子类中的复用；
- 再叠加额外的功能代码，组成新的功能。

6.3 super访问属性

```java
public class TestSuperKeyword {
    public static void main(String[] args) {
        B b = new B();
        b.print();
    }
}

class A{
    int value = 10;
}

class B extends A{
    int value = 20;

    public void print(){
        int value = 30;
        System.out.println(value);
        System.out.println(this.value);
        System.out.println(super.value);
    }
}
```



> 运行结果：
> 30
> 20
> 10

> 父子类的同名属性不存在重写关系，两块空间同时存在（子类遮蔽父类属性），需使用不同前缀进行访问。

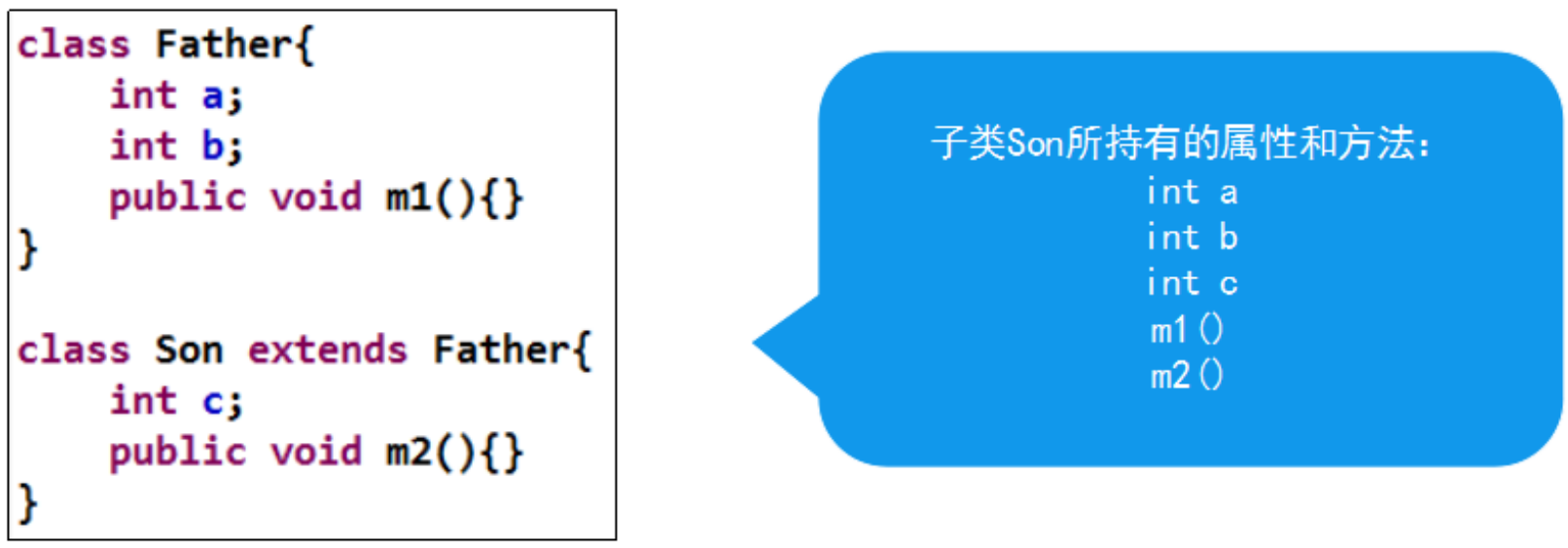
```java
public class TestBasicSuperKeyword {
    public static void main(String[] args) {

        Son son = new Son();

        son.method();

    }
}

class Father{
    int field = 10;
    String flag = "Hello";
}

class Son extends Father{
    int field = 20;

    public void method() {
        System.out.println(super.field);//在子类中访问父类的成员
        System.out.println(this.field);
```

```
    System.out.println(flag);
  }
}
```

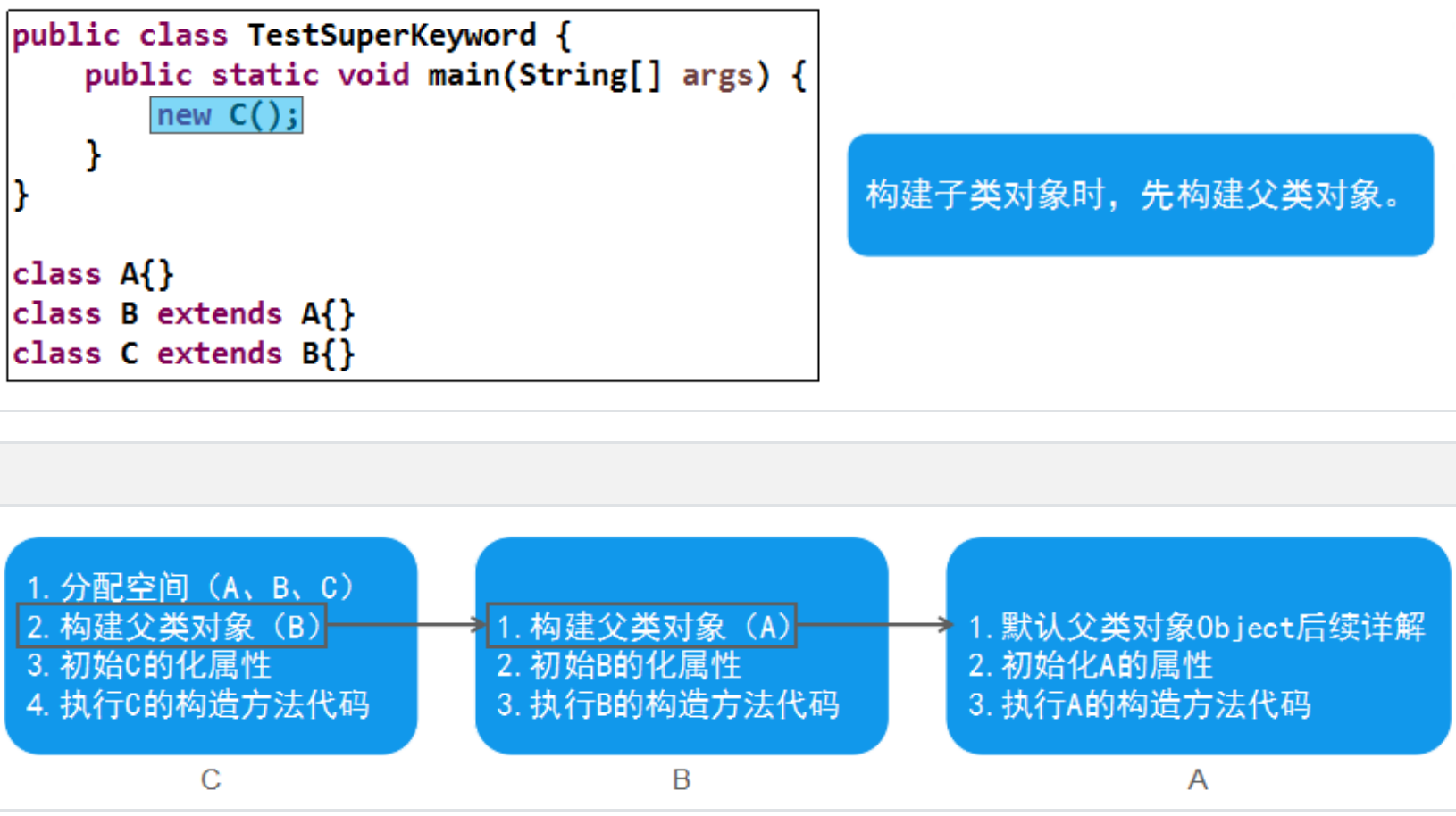
6.4 继承中的对象创建

> 在具有继承关系的对象创建中，构建子类对象会先构建父类对象。

> 由父类的共性内容，叠加子类的独有内容，组合成完整的子类对象。



6.5 继承后的对象构建过程

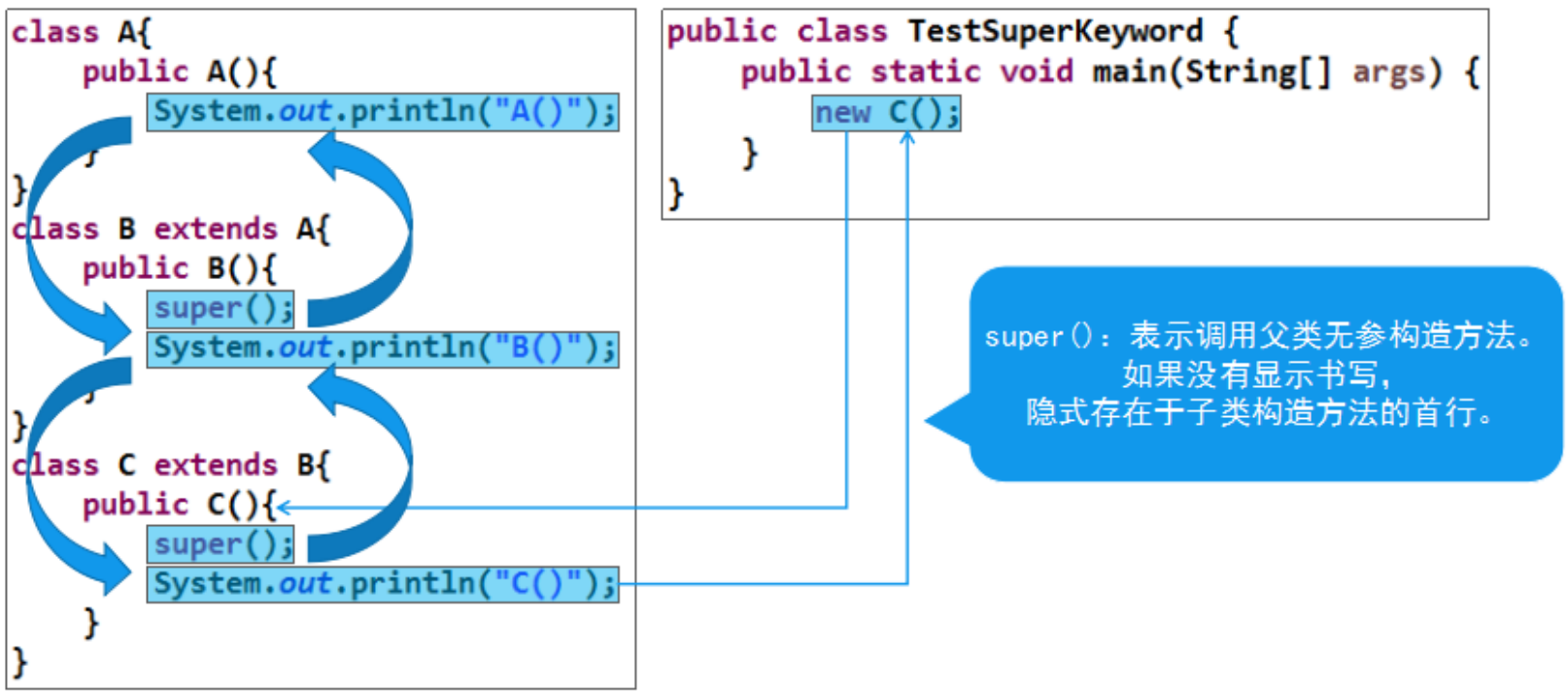


> 构建子类对象时，先构建父类对象。

```
public class TestCreateSort {//extends Object

  public static void main(String[] args) {

    C c  =new C();//field1  field2  field3

  }

}

class A{//extends Object
  String field1 = "A的属性";//1.初始化属性

  public A() {
    super();//调用Object类的无参构造方法
    System.out.println("A - 构造方法被执行  " + field1);//2.执行构造方法中的逻辑代码
  }
}

class B extends A{
  String field2 = "B的属性";//2.初始化属性

  public B() {
    super();//1.调用父类的构造方法（默认调用父类无参的构造方法）隐式存在
```

```
    System.out.println("B - 构造方法被执行  " + field2);//3.执行构造方法中的逻辑代码
  }
}

class C extends B{
  String field3 = "C的属性";//2.初始化属性

  public C() {
    super();//1.调用父类的构造方法（默认调用父类无参的构造方法）隐式存在
    System.out.println("C - 构造方法被执行  " + field3);//3.执行构造方法中的逻辑代码
  }
}

//1.构建父类对象
//2.初始化自身属性
//3.执行自身构造方法中的逻辑代码
```

6.6 super调用父类无参构造

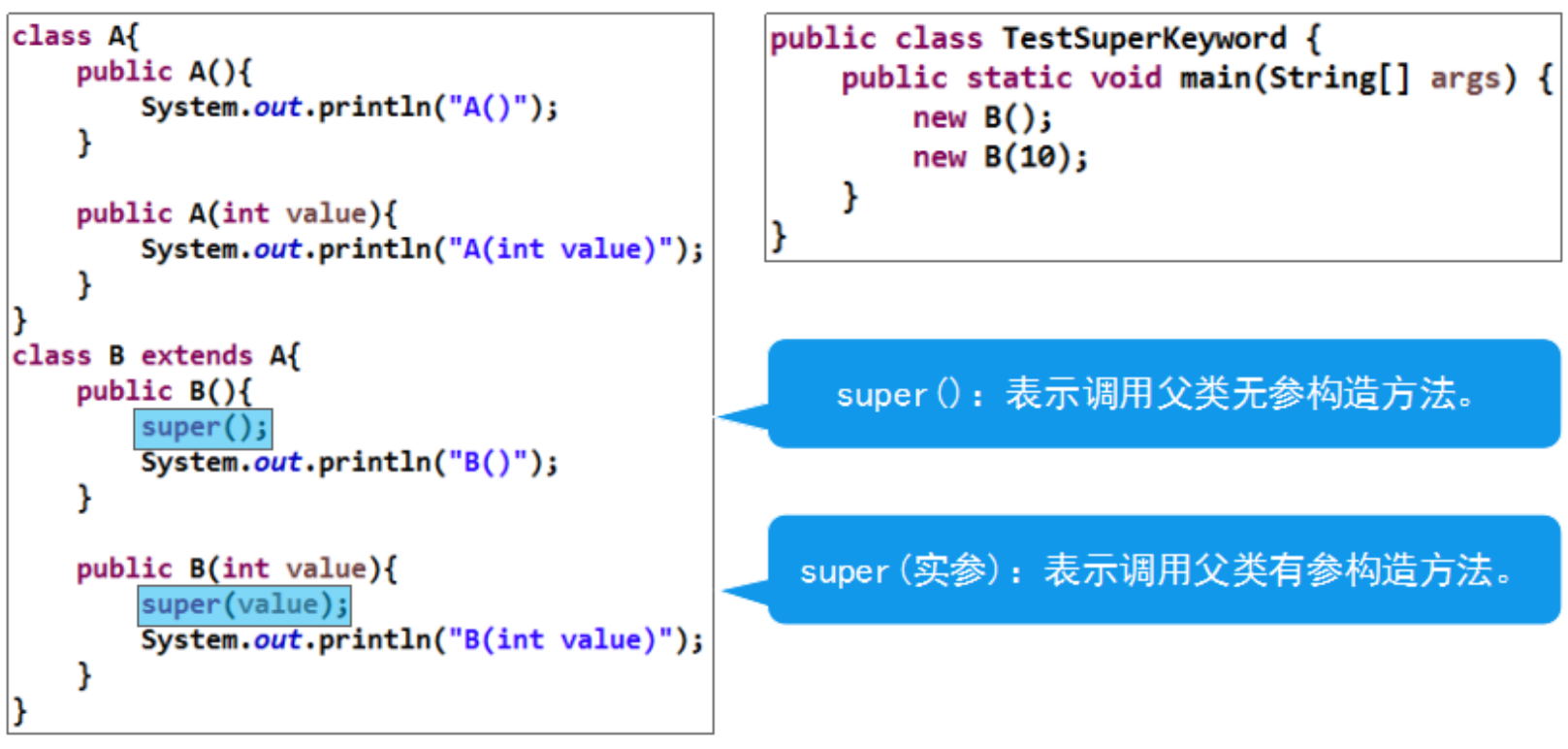


运行结果：

A()
B()
C()

super()：表示调用父类无参构造方法。如果没有显示书写，隐式存在于子类构造方法的首行。

6.7 super调用父类有参构造



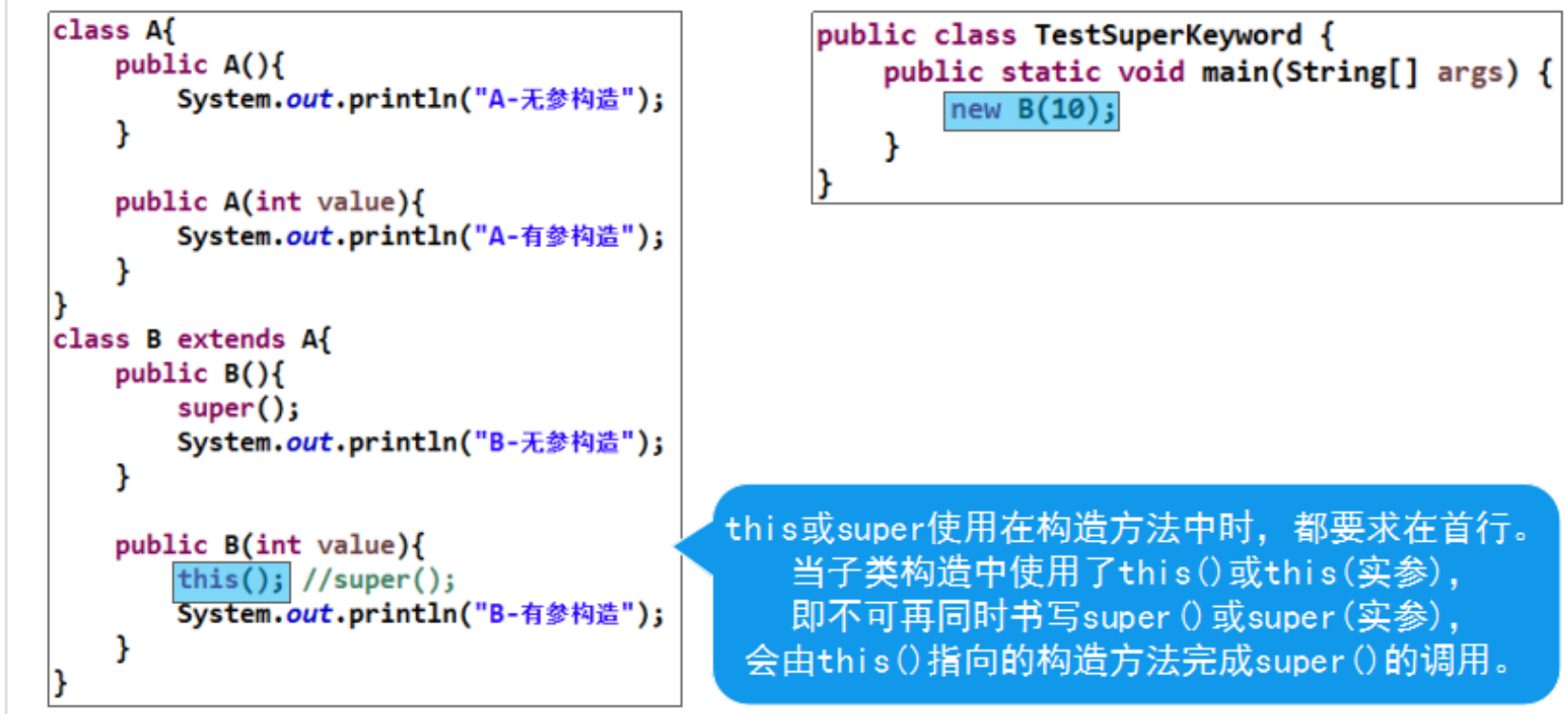
super()：表示调用父类无参构造方法。

super(实参)：表示调用父类有参构造方法。

```
public class TestArgsConstructor {
  public static void main(String[] args) {
```

```
    new Son();

    System.out.println("-------------");

    Son son = new Son(10);

    System.out.println("-------------");

    new Son(3.5);

    System.out.println(son.field);
  }
}


class Father{

  int field;//父类提供的属性

  public Father() {
    System.out.println("Father() - 无参构造被执行");
  }

  public Father(int a) {
    this.field = a;
    System.out.println("Father(int a) - 一参构造被执行");
  }

}

class Son extends Father{

  public Son() {
    //super();
    System.out.println("Son() - 无参构造被执行");
  }

  public Son(int b) {
    super(b);
    System.out.println("Son(int b) - 一参构造被执行");
  }

  public Son(double c) {
    super(1);
    System.out.println("Son(double c) - 一参构造被执行");
  }
}
```

6.8 this与super



运行结果：
A-无参构造
B-无参构造
B-有参构造

this或super使用在构造方法中时，都要求在首行。
当子类构造中使用了this()或this(实参)，即不可再同时书写super()或super(实参)，会由this()指向z构造方法完成super()调用。

```
public class TestThisAndSuper {
```

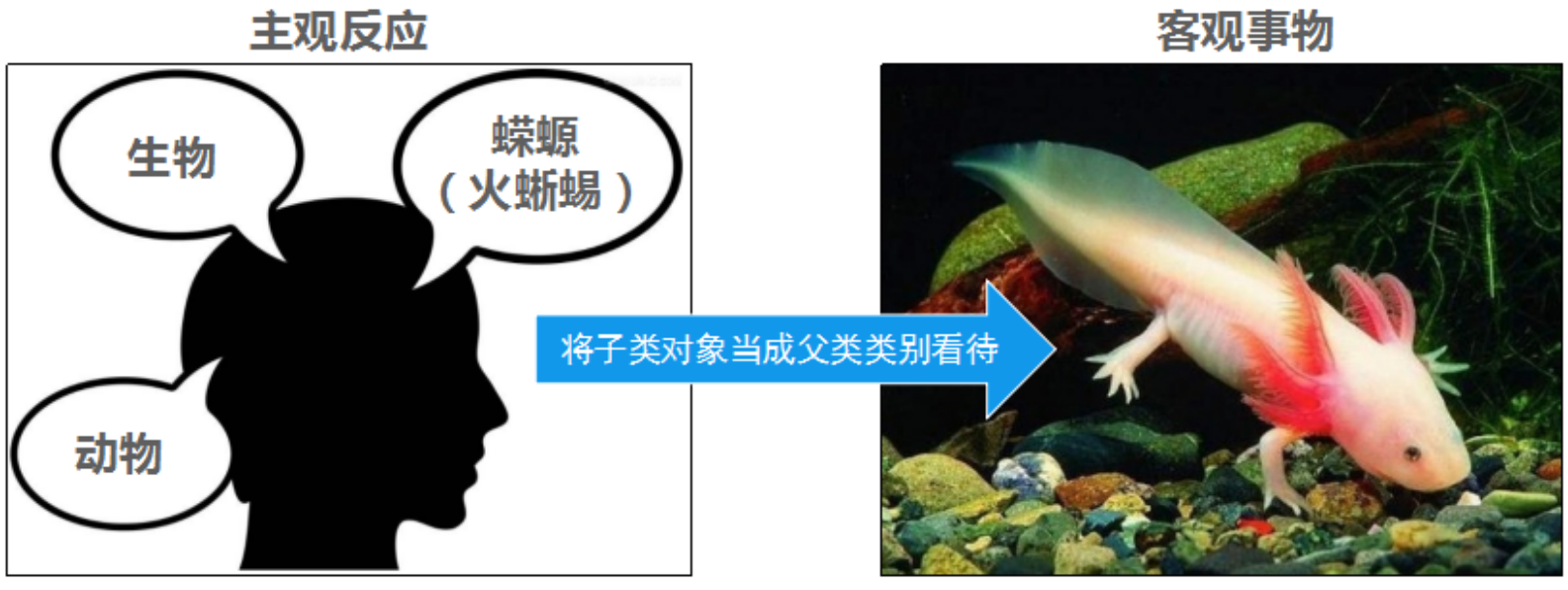
```
  public static void main(String[] args) {
    // TODO Auto-generated method stub
    Dog dog = new Dog("哈士奇",2,"公","灰白");

    //Dog dog2 = new Dog("萨摩",3,"母");

    System.out.println(dog.breed +"\t"+ dog.age +"\t"+ dog.sex +"\t"+ dog.furColor);

    //System.out.println(dog2.breed +"\t"+ dog2.age +"\t"+ dog2.sex +"\t"+ dog2.furColor);
  }

}


class Animal{
  String breed;
  int age;
  String sex;

  public Animal() {
    System.out.println("---Animal() Executed---");
  }
}

class Dog extends Animal{
  String furColor;

  public Dog() {
    super();
  }

  public Dog(String breed , int age , String sex) {
    super();//this(xx,xx)
    super.breed = breed;
    super.age = age;
    super.sex = sex;
    System.out.println("Dog(String breed , int age , String sex) Executed");
  }

  public Dog(String breed , int age , String sex , String furColor) {
    //super();
    //this(breed , age , sex);//调用本类中的其他构造方法（调用三参构造完成赋值）
    this.furColor = furColor;
    System.out.println("Dog(String breed , int age , String sex , String furColor) Executed");
  }

  public void method() {
    System.out.println(this.furColor);
    System.out.println(breed);//直接访问
    System.out.println(this.breed);
    System.out.println(super.breed);
  }
}
```

七、多态【重点】

7.1 生活中的人物视角

> 生活中，不同人物角色看待同一个对象的视角不同，关注点也不相同。

7.2 生活中的多态

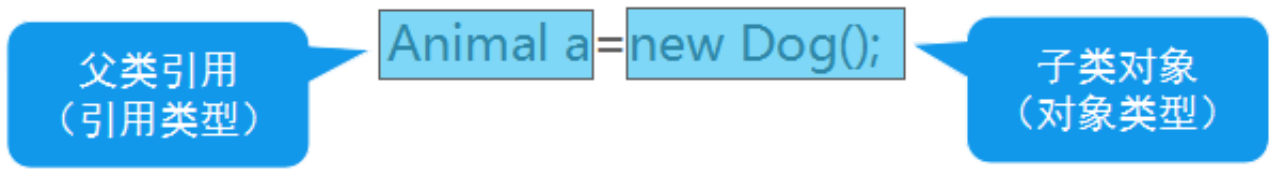


> 生活中的多态是指“客观事物在人脑中的主观反应”。

> 主观意识上的类别与客观存在的对象具有“is a”关系时，即形成多态。

7.3 程序中的多态

> 概念：父类引用指向子类对象，从而产生多种形态。



> 二者具有直接或间接的继承关系时，父类引用可指向子类对象，即形成多态。

> 父类引用仅可调用父类所声明的属性和方法，不可调用子类独有的属性和方法。

```
public class TestBasicPolymorphic {
  public static void main(String[] args) {

    Animal a = new Dog();//将狗对象，当成动物来看待
    System.out.println(a.breed);
    System.out.println(a.age);
    System.out.println(a.sex);
    a.eat();
    a.sleep();

    Dog d = new Dog();//将狗对象，当成狗来看待
    System.out.println(d.breed);
    System.out.println(d.age);
    System.out.println(d.sex);
    System.out.println(d.furColor);
    d.eat();
    d.sleep();
    d.run();

  }
}

class Animal{
  String breed;
  int age;
  String sex;

  public void eat() {}
  public void sleep() {}
}

class Dog extends Animal{
  String furColor;

  public void run() {}
}
```

7.4 多态中的方法重写

> 思考：如果子类中重写了父类中的方法，以父类类型引用调用此方法时，优先执行父类中的方法还是子类中的方法?

> 实际运行过程中，依旧遵循重写原则，如果子类重写了父类中的方法，执行子类中重写后的方法，否则执行父类中的方法。

7.5 多态的应用

```
class Master{
    public void feed(Dog dog){
        dog.eat();
    }

    public void feed(Cat cat){
        cat.eat();
    }

    public void feed(Fish fish){
        fish.eat();
    }

    public void feed(Bird bird){
        bird.eat();
    }

    //......
}
```

> 方法重载可以解决接收不同对象参数的问题，但其缺点也比较明显。

> - 首先，随着子类的增加，Master类需要继续提供大量的方法重载，多次修改并重新编译源文件。
> - 其次，每一个feed方法与某一种具体类型形成了密不可分的关系，耦合太高。

> 场景一：使用父类作为方法形参实现多态，使方法参数的类型更为宽泛。

> 场景二：使用父类作为方法返回值实现多态，使方法可以返回不同子类对象。

八、装箱、拆箱

8.1 向上转型（装箱）

```
public class TestConvert {
    public static void main(String[] args) {
        Animal a = new Dog();
    }
}

class Animal{
    public void eat(){
        System.out.println("动物在吃...");
    }
}

class Dog extends Animal{
    public void eat(){
        System.out.println("狗在吃骨头...");
    }
}
```

父类引用中保存真实子类对象，
称为向上转型（即多态核心概念）。

注意：
仅可调用Animal中所声明的属性和方法。

> 父类引用中保存真实子类对象，称为向上转型（即多态核心概念）。

> 注意： 仅可调用Animal中所声明的属性和方法。

8.2 向下转型（拆箱）

```java
public class TestConvert {
    public static void main(String[] args) {
        Animal a = new Dog();
        Dog dog = (Dog)a;
    }
}

class Animal{
    public void eat(){
        System.out.println("动物在吃...");
    }
}

class Dog extends Animal{
    public void eat(){
        System.out.println("狗在吃骨头...");
    }
}
```

将父类引用中的真实子类对象，
强转回子类本身类型，称为向下转型。

注意：
只有转换回子类真实类型，才可调用子类独有的属性和方法。

> 将父类引用中的真实子类对象，强转回子类本身类型，称为向下转型。

> 注意：只有转换回子类真实类型，才可调用子类独有的属性和方法。

8.3 类型转换异常

```java
class Animal{
    public void eat(){
        System.out.println("动物在吃...");
    }
}

class Dog extends Animal{
    public void eat(){
        System.out.println("狗在吃骨头...");
    }
}

class Cat extends Animal{
    public void eat(){
        System.out.println("猫在吃鱼...");
    }
}
```

```java
public class TestConvert {
    public static void main(String[] args) {
        Animal a = new Dog();
        Cat cat = (Cat)a;
    }
}
```

```
Exception in thread "main" java.lang.ClassCastException
```

向下转型时，如果父类引用中的子类对象类型和目标类型不匹配，则会发生类型转换异常。

> 向下转型时，如果父类引用中的子类对象类型和目标类型不匹配，则会发生类型转换异常。

8.4 instance of关键字

> 向下转型前，应判断引用中的对象真实类型，保证类型转换的正确性。

> 语法：父类引用 instance of 类型 //返回boolean类型结果

```java
public class TestConvert {
    public static void main(String[] args) {
        Animal a = new Dog();

        if(a instanceof Dog){
            Dog dog = (Dog)a;
            dog.eat();
        }else if(a instanceof Cat){
            Cat cat = (Cat)a;
            cat.eat();
        }
    }
}
```

当“a”引用中存储的对象类型确实为Dog时，
再进行类型转换，进而调用Dog中的独有方法。

运行结果：
狗在吃骨头...