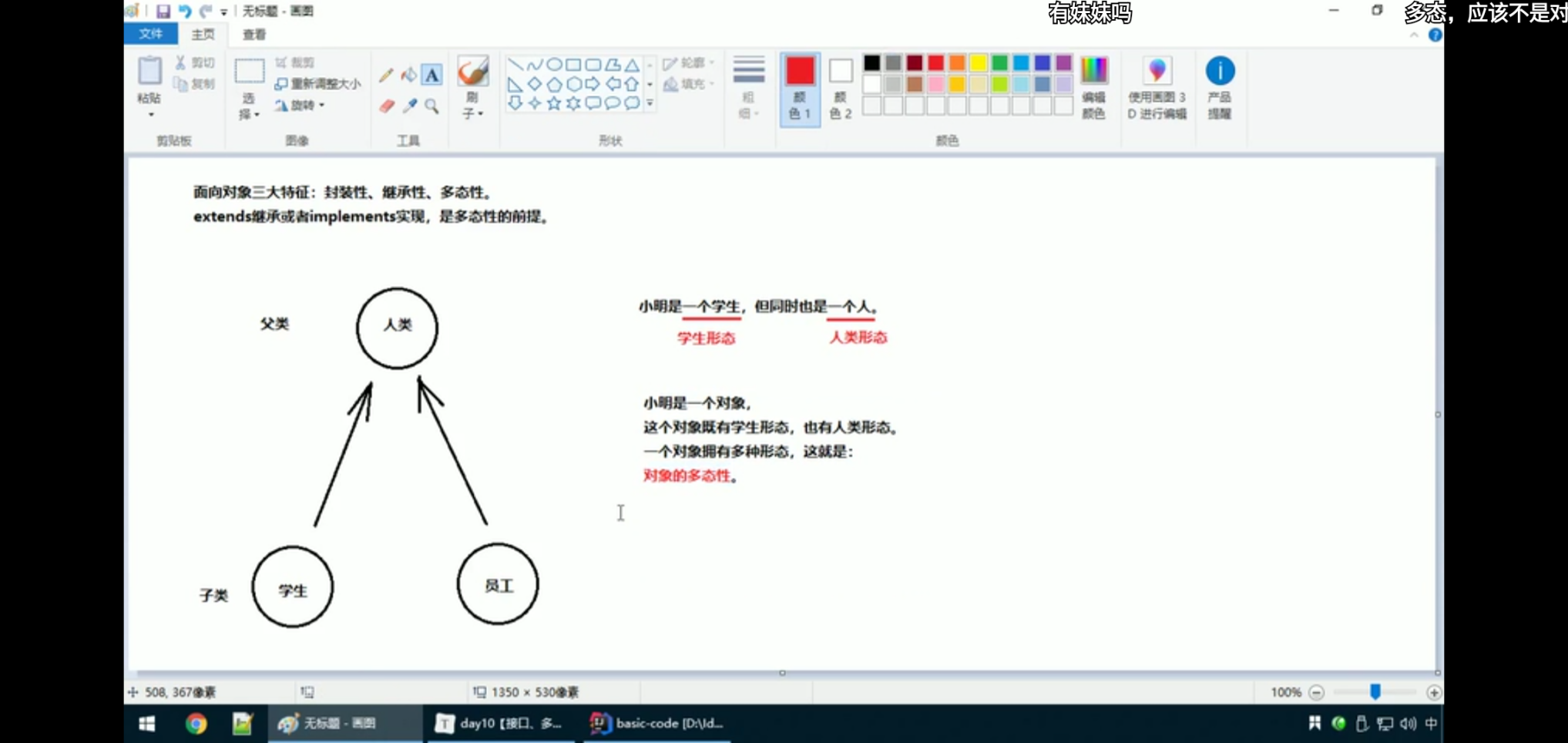
* + - * 1. 多态的概念



* + - * 1. 多态的格式与使用

/\*  
代码当中体现多态性，其实就是一句话：父类引用指向子类。  
  
格式：  
父类名称 对象名 = new 子类名称();  
或者：  
接口名称 对象名 = new 实现类名称();  
\*/

父类：

public class Fu {  
  
 public void method(){  
 System.out.println("父类方法");  
 }  
 public void methodFu(){  
 System.out.println("父类方法");  
 }  
}

子类：

public class Zi extends Fu{  
  
 @Override  
 public void method(){  
 System.out.println("子类方法");  
 }  
}

主函数：

public class zhu {  
 public static void main(String[] args) {  
 // 使用多态的写法  
 // 左侧父类的引用，指向了右侧子类的对象。  
 Fu obj = new Zi();  
 obj.method(); // 子类方法  
 obj.methodFu(); // 父类方法  
 }  
}

* + - * 1. 多态中成员变量的使用特点

/\*  
1. 直接通过对象名称访问成员变量，看等号左边是谁，优先用谁，没有则向上找。  
2. 间接通过成员方法访问成员变量，看该方法属于谁，优先用谁，没有则向上找。  
 \*/

父类：

public class Fu /\*extends Object\*/{  
  
 int num = 10;  
 public void showNum(){  
 System.out.println(num);  
 }  
}

子类：

public class Zi extends Fu{  
  
 int num = 20;  
 int age = 16;  
  
 @Override  
 public void showNum() {  
 System.out.println(num);  
 }  
}

主函数：

public class zgu {  
 public static void main(String[] args) {  
 // 使用多态的写法，父类引用指向子类对象  
 Fu obj = new Zi();  
 System.out.println(obj.num); // 父类：10  
// System.out.println(obj.num); // 错误写法  
 System.out.println("=================");  
  
 // 子类没有覆盖重写，就是分：20  
 // 子类如果没有覆盖重写，就是子：20  
 obj.showNum();  
 }  
}

* + - * 1. 多态中成员方法的使用特点
        2. /\*  
           在多态的代码当中，成员方法的访问规则是：  
            看new的是谁，就优先用谁，没有则向上找。  
             
           口诀：编译看左边，运行看右边。  
             
           对比一下：  
           成员变量：编译看左边，运行还看左边。  
           成员方法：编译看左边，运行看右边。  
            \*/

父类：

public class Fu /\*extends Object\*/ {  
  
 public void method(){  
 System.out.println("父类方法");  
 }  
 public void methodfu(){  
 System.out.println("父类的特有方法");  
 }  
}

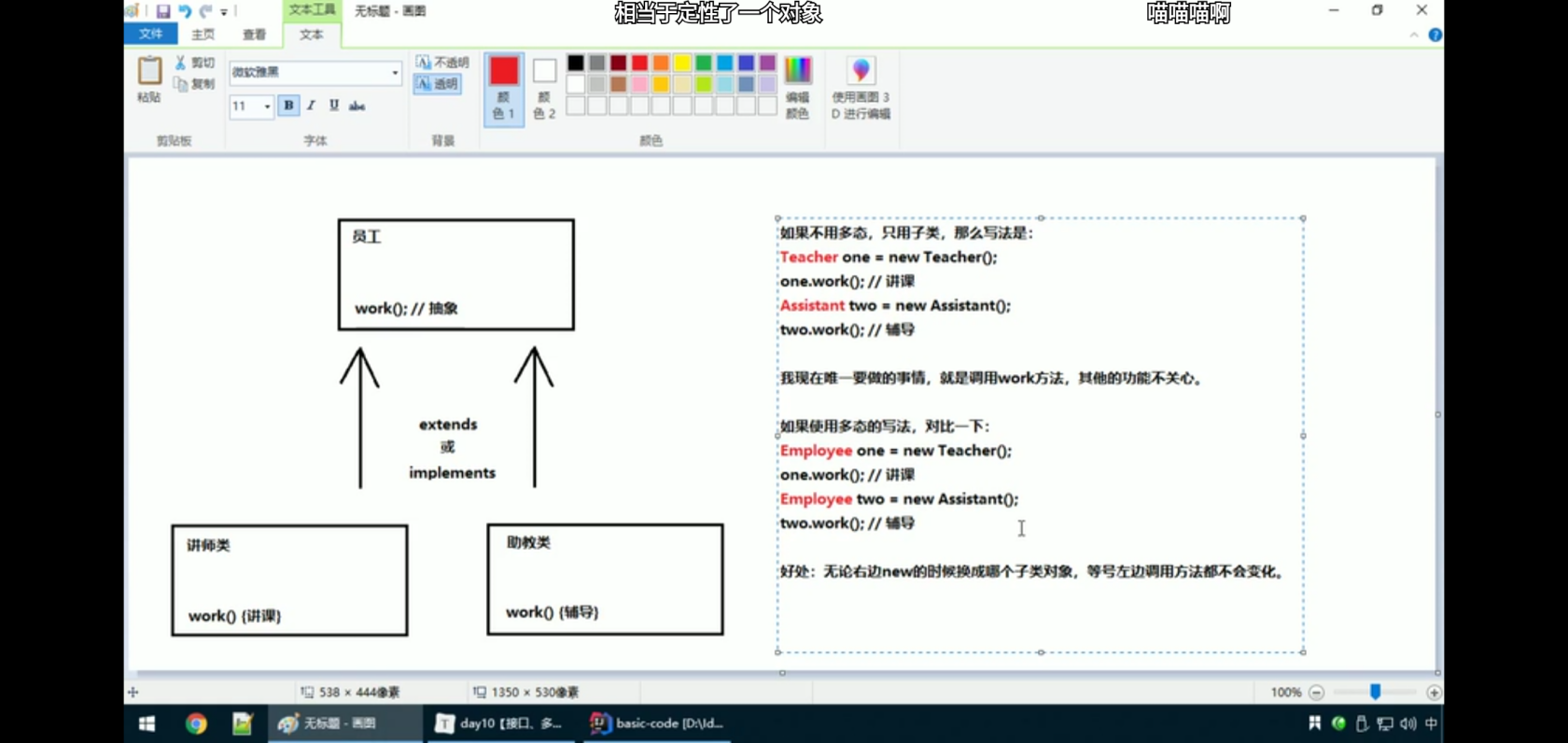
子类：

public class Zi extends Fu{  
  
 @Override  
 public void method() {  
 System.out.println("子类方法");  
 }  
}

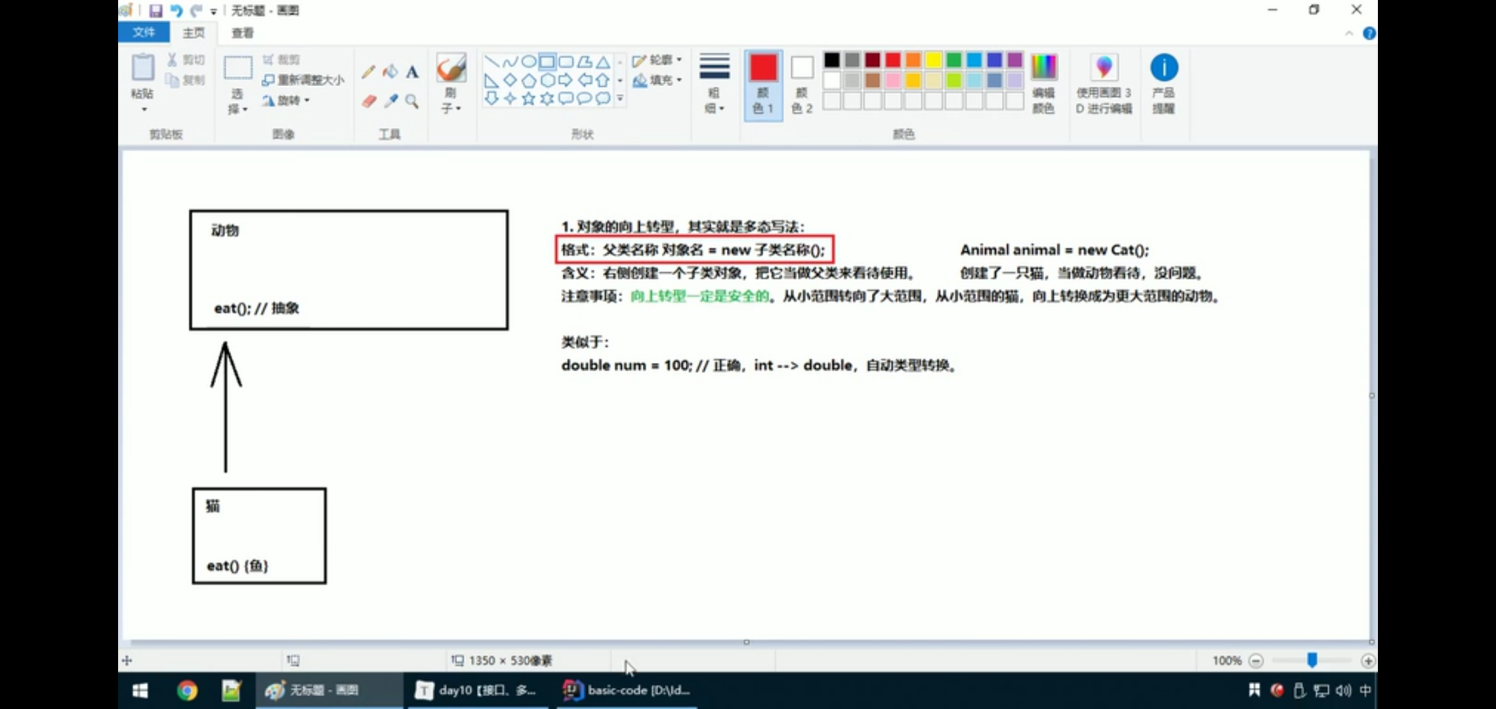
主函数：

public class zhu {  
 public static void main(String[] args) {  
 Fu obj = new Zi(); // 多态  
  
 obj.method(); // 父子都有，优先用子  
 obj.methodfu(); // 子类没有，父类有，向上找父类。  
 }  
}

* + - * 1. 使用多态的好处



* + - * 1. 对象的向上转型



父类（抽象类）：

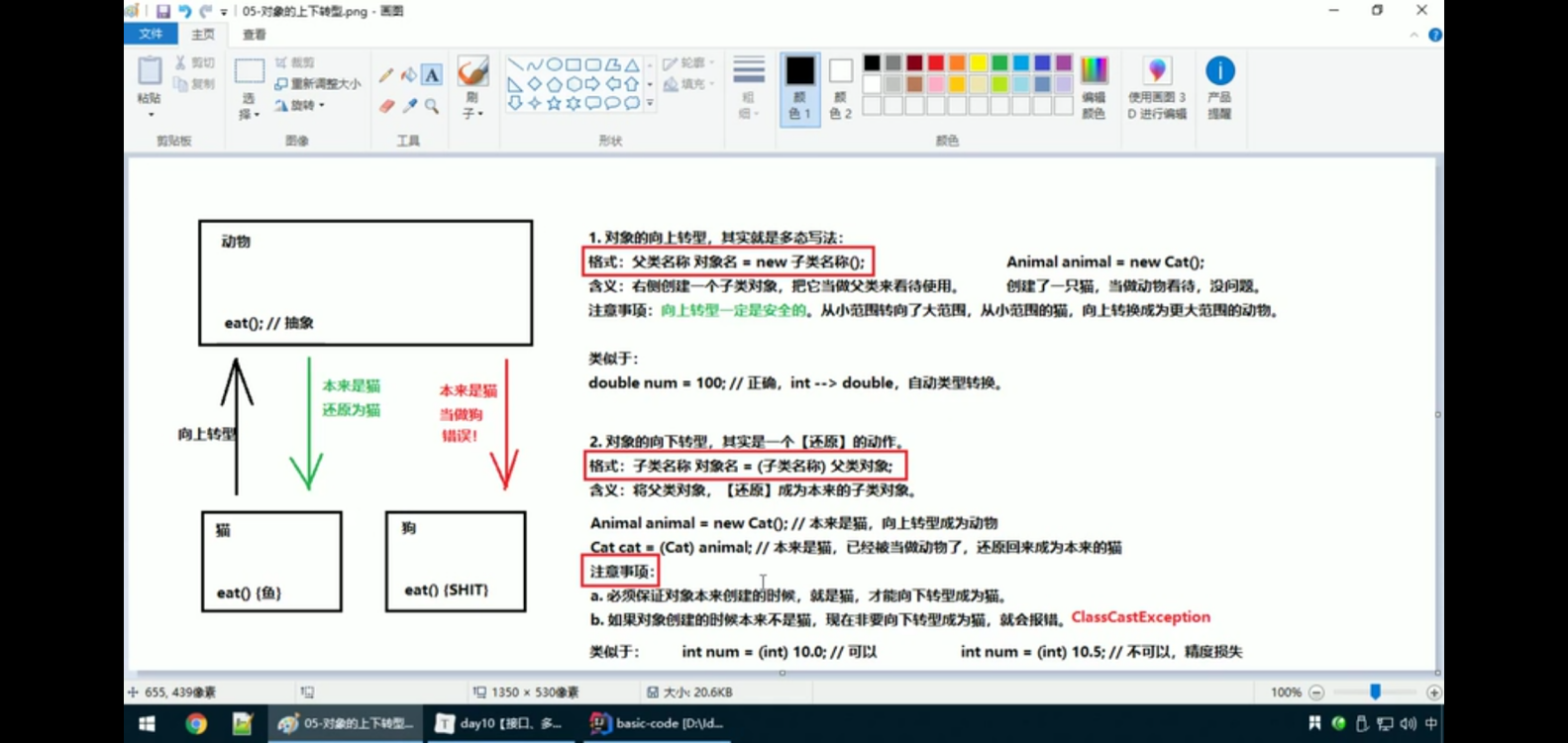
public abstract class Animal {  
 // 定义一个抽象类吃  
 public abstract void eat();  
}

子类（实现类）：

public class Cat extends Animal{  
 // 编写一个实现类（子类）  
 @Override  
 public void eat() {  
 System.out.println("猫吃鱼");  
 }}

主函数：

* + - * 1. public class DemoMain {  
             
            public static void main(String[] args) {  
            // 对象的向上转型，就是：父类引用指向子类对象  
            Animal animal = new Cat();  
            animal.eat(); // 猫吃鱼 }}
        2. 对象的向下转型



父类（抽象类）：

public abstract class Animal {  
  
 // 定义一个抽象类吃  
 public abstract void eat();  
}

子类猫（实现类）：

public class Cat extends Animal{  
 // 编写一个实现类（子类）  
 @Override  
 public void eat() {  
 System.out.println("猫吃鱼");  
 }  
  
 public void catchMouse(){  
 System.out.println("猫抓老鼠");  
 }  
}

子类狗（实现类）：

public class Dog extends Animal {  
 @Override  
 public void eat() {  
 System.out.println("狗吃SHIT");  
 }  
  
 public void watchHouse(){  
 System.out.println("狗看家");  
 }  
}

主函数：

public class DemoMain {  
  
 public static void main(String[] args) {  
 // 对象的向上转型，就是：父类引用指向子类对象  
 Animal animal = new Cat(); // 本来创建的时候是一只猫。  
 animal.eat(); // 猫吃鱼  
 animal.catchMouse(); // 错误写法！  
  
 // 向下转型，进行”还原“动作  
 Cat cat = (Cat) animal;  
 cat.catchMouse(); // 猫抓老鼠  
  
 // 下面是错误的向下转型  
 // 本来new的时候是一只猫，现在非要当作狗  
 // 错误写法！编译不会报错，但是运行时会出现异常  
 // java.lang.ClassException，类转换异常  
 Dog dog = (Dog) animal; // 错误上边定义的是猫，不是狗  
 }  
}

* + - * 1. 用instanof 关键字进行判断

主函数：（其余子类父类的代码和上边一样）

public class DemoMain {  
  
 public static void main(String[] args) {  
 // 对象的向上转型，就是：父类引用指向子类对象  
 Animal animal = new Dog(); // 本来创建的时候是一只狗。  
 animal.eat(); // 狗吃SHIT  
  
 // 如果希望调用子类的特有方法，需要向下转型  
 // 判断一下父类引用animal本来是不是Dog  
 if(animal instanceof Dog) {  
 Dog dog = (Dog) animal;  
 dog.watchHouse();  
 }  
 if (animal instanceof Cat) {  
 Cat cat = (Cat) animal;  
 cat.catchMouse();  
 }  
  
 giveMeApet(new Dog());  
 giveMeApet(new Cat());  
 }  
  
 public static void giveMeApet(Animal animal) {  
 if(animal instanceof Dog) {  
 Dog dog = (Dog) animal;  
 animal.eat();  
 dog.watchHouse();  
 }  
 if (animal instanceof Cat) {  
 Cat cat = (Cat) animal;  
 cat.catchMouse();  
 animal.eat();  
 }  
 }  
}

结果：

狗吃SHIT

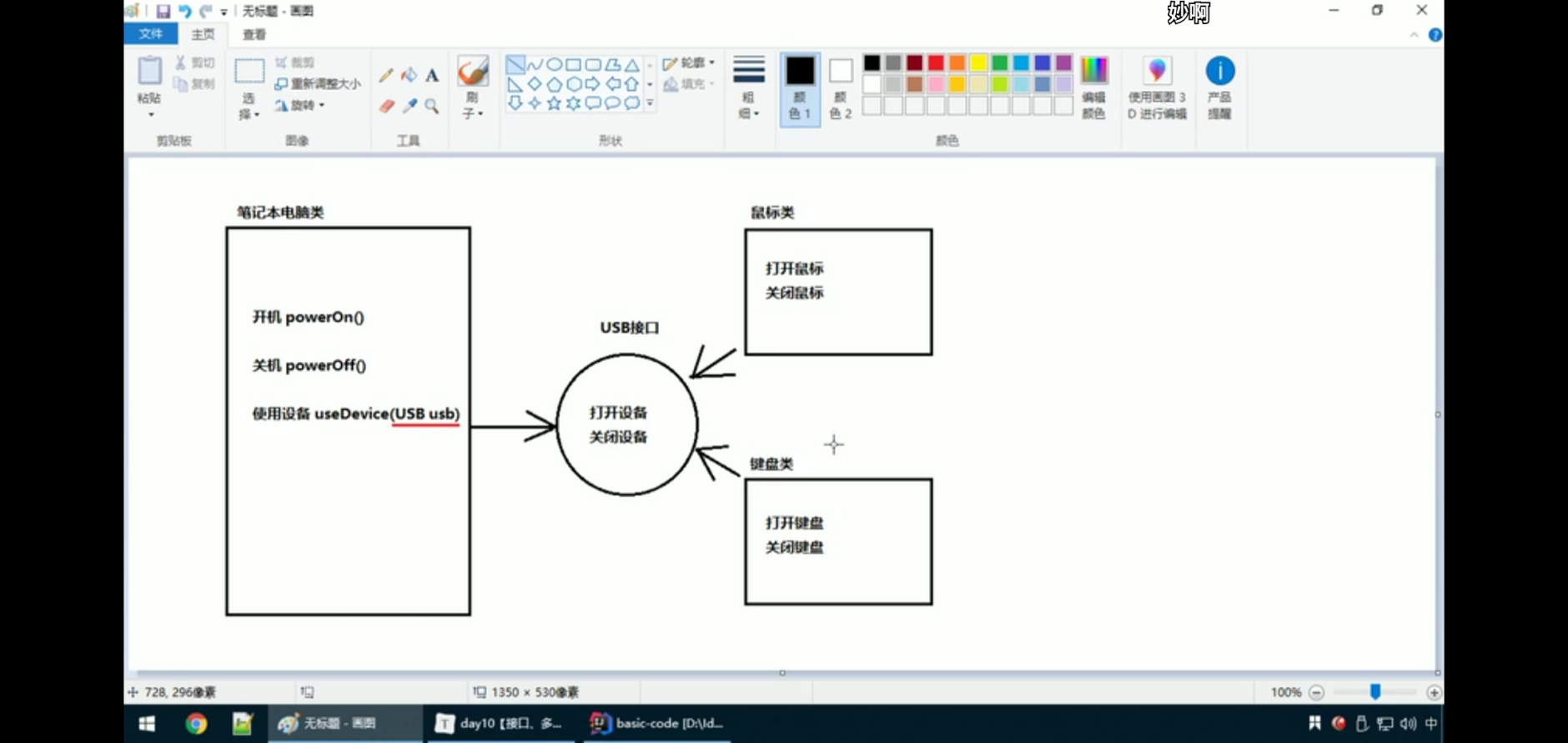
狗看家

狗吃SHIT

狗看家

猫抓老鼠

猫吃鱼

* + - * 1. 笔记本USB接口案例—分析
        2. 笔记本USB接口案例-实现

定义一个USB接口：  
public interface USB {  
 // 定义一个接口，编译两个抽象类  
 public abstract void open(); // 打开设备  
 public abstract void close(); // 关闭设备  
}

定义一台电脑：

public class Computer {  
 public void powerOn(){  
 System.out.println("笔记本电脑打开");  
 }  
 public void powerOff(){  
 System.out.println("笔记本电脑关闭");  
 }  
 // 使用USB设备的方法，使用接口作为方法的参数  
 public void useDevice(USB usb) {  
 usb.open(); // 打开设备  
 if(usb instanceof Mouse) { // 一定要先判断  
 Mouse mouse = (Mouse) usb; // 向下转型  
 mouse.click();  
 }else if(usb instanceof Keyboard){ // 一定要先判断  
 Keyboard keyboard = (Keyboard) usb; // 向下转型  
 keyboard.type();  
 }  
 usb.close(); // 关闭设备  
 }  
}

定义一个USB工具：键盘

// 键盘就是一个USB设备  
public class Keyboard implements USB {  
 @Override  
 public void open() {  
 System.out.println("打开键盘");  
 }  
 @Override  
 public void close() {  
 System.out.println("关闭键盘");  
 }  
 public void type(){  
 System.out.println("键盘输入");  
 }  
}

定义一个USB工具：鼠标

// 鼠标就是一个USB设备  
public class Mouse implements USB{  
 @Override  
 public void open() {  
 System.out.println("打开鼠标");  
 }  
 @Override  
 public void close() {  
 System.out.println("关闭鼠标");  
 }  
 public void click(){  
 System.out.println("鼠标点击");  
 }  
}

主函数：  
public class DemoMain {  
 public static void main(String[] args) {  
 // 首先创建一个笔记本电脑  
 Computer computer = new Computer();  
 computer.powerOn();  
  
 // 准备一个鼠标，供电脑使用  
// Mouse mouse = new Mouse(); z正确写法  
 // 首先进行向上转型  
 USB usbMouse = new Mouse(); // 多态写法  
 // 参数是USB类型，传递进去的就是USB鼠标  
 computer.useDevice(usbMouse);  
  
 // 创建一个USB键盘  
 Keyboard keyboard = new Keyboard(); // 没有使用多态写法。  
 // 方法参数是USB类型，传递进去的是实现类对象。  
 computer.useDevice(keyboard); // 正确写法!也发生了向上转型  
 // 使用子类对象，匿名对象，也可以：  
// computer.useDevice(new Keyboard()); // 正确写法  
 computer.powerOff();  
 }  
}

运行结果：

笔记本电脑打开

打开鼠标

鼠标点击

关闭鼠标

打开键盘

键盘输入

关闭键盘

笔记本电脑关闭