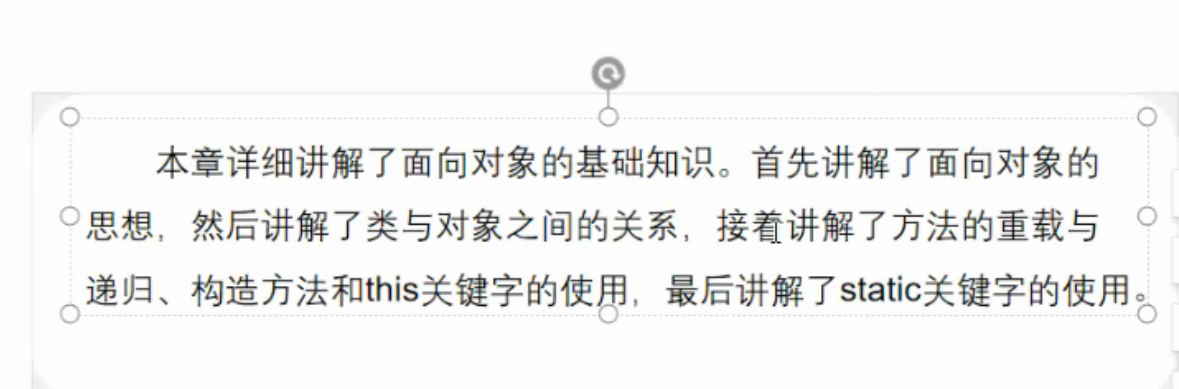
第三章



构造方法

【构造符（例如public）】 方法名（）{

//方法体

}

注意方式：

1.无返回值，void也不能写

2.方法名和类名相同

3.构造方法里面也没return也没

```
//无参数构造
public Person(){

}

//有参数构造
pubic Person(String n,int a){
    name=n;
    age = a;
}
```

注意方式2（总结）：

1、如果我们写任何构造方法，系统会给我一个提供一个构造方法，以供我们创建对象使用。

2.如果我们定义了有参数的，则不会给我们提供

this关键字

1、可以区分成员变量和局部变量同名的问题

2、代表的是当前对象的引用，谁调用我，我代表谁

this 调用成员方法

```
    //成员变量
    public void speake() {
        System.out.println("我在吃饭");
        this.eat();
    }
    public void eat() {
        System.out.println("开始吃饭");
    }
```

this 调用成员变量

```
private int age; //成员变量

    private String name;
    private char sex;


    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public char getSex() {
        return sex;
    }
```

this调用构造方法

3.6 this关键字

3. 通过this关键字调用构造方法。

```
class Person {
    public Person() {
    }
    public Person(int age) {
        this();              // 调用无参的构造方法
    }
}
```

说明：构造方法是在实例化对象时被Java虚拟机自动调用

用其他方法一样去调用构造方法，但可以在一个构造方法

参数2...])”的形式来调用其他的构造方法。

注意

3.6 this关键字

在使用this调用类的构造方法时，应注意以下几点：

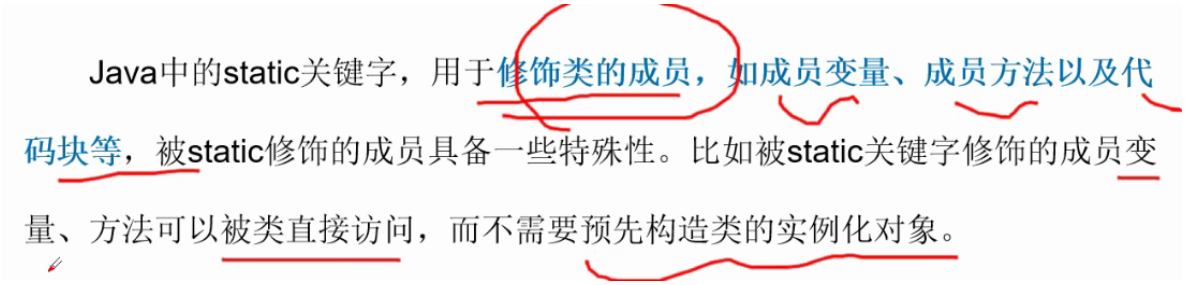
① 只能在构造方法中使用this调用其他的构造方法，不能在成员方法中使用。

② 在构造方法中，使用this调用构造方法的语句必须是该方法的第一条执行语句，且只能出现一次。

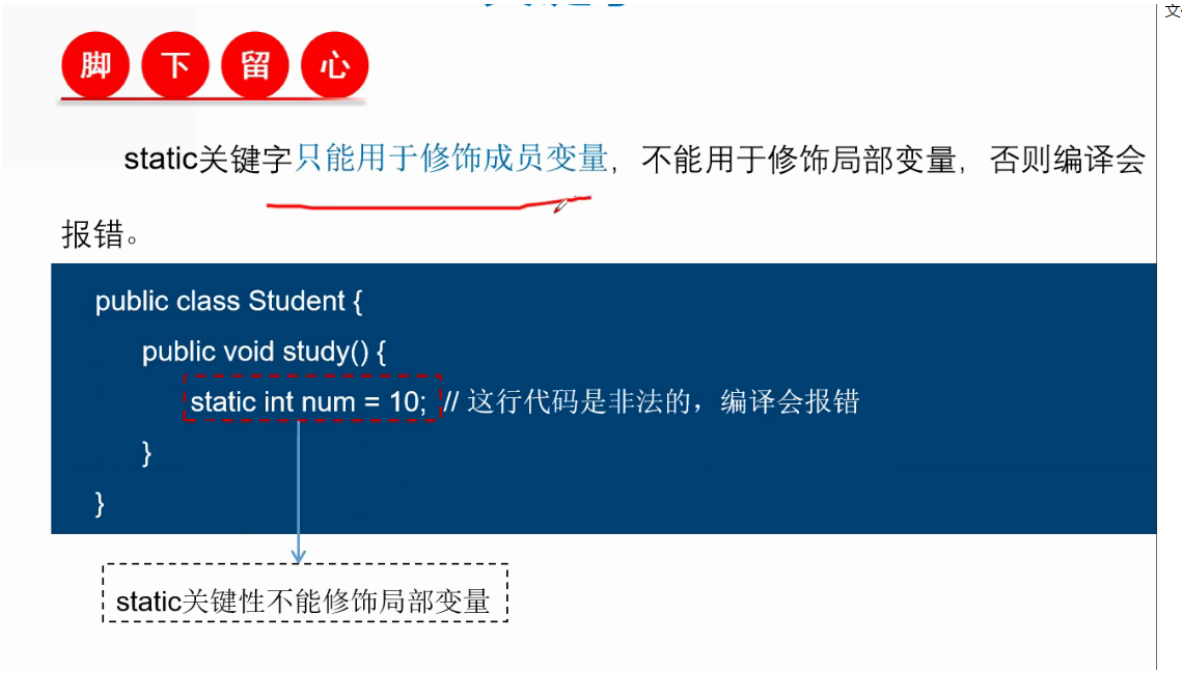
③ 不能在一个类的两个构造方法中使用this互相调用。

快捷键 **alt+shift+s** 可以调set get 方法 以及带不带参数的构造方法 以及tostring方法等

static 关键字



修饰成员变量 不能修饰局部变量



static 关键字

修饰变量。变量可以被所以的对象共享。

什么时候需要将变量定义为静态变量呢？当这个变量被类中所以变量共享的时候

静态方法

静态方法：

可以直接被类名使用；

静态方法属于类的。随着类的加载而加载

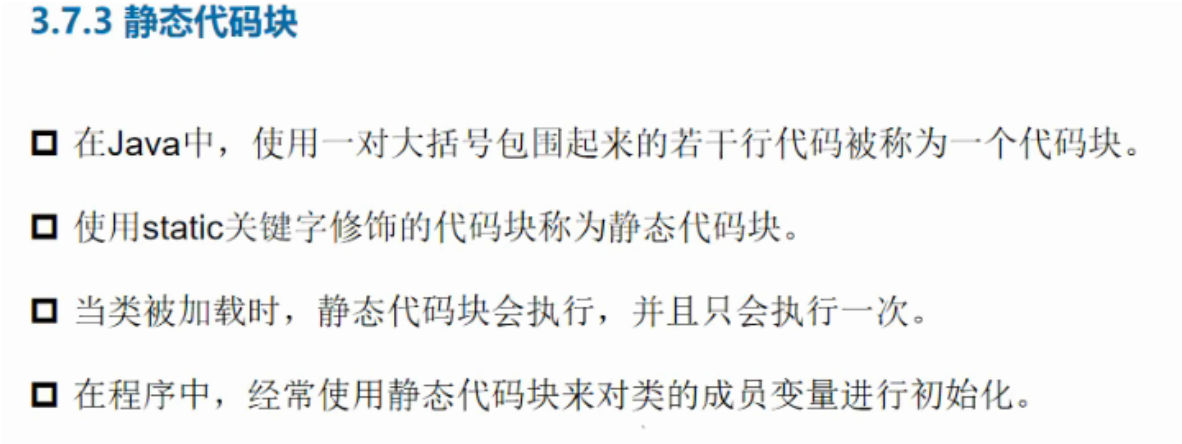
非静态方法属于对象，随着对象而对象

静态和非静态特点

静态只能访问静态的（先存在）

非静态可以访问静态，也可以访问非静态的。（后存在）

静态代码块



当类加载的时候就会执行

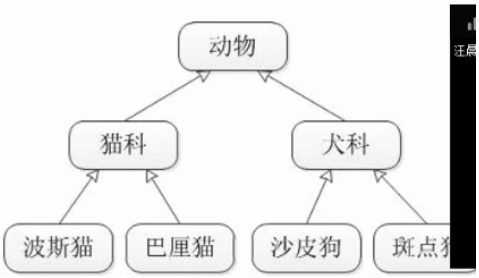
只会执行一次

作用：注册驱动、加载资源

第四章

4.1类的继承

例如猫和狗都属于动物，程序中便可以描述为猫和狗继承自动物；同理，波斯猫和巴厘猫继承自猫，而沙皮狗和斑点狗继承自狗。



定义：在Java中，类的继承是指在一个现有类的基础上去构建一个新的类，构建出来的新类被称作子类，现有类被称作父类或基类，子类会自动拥有父类所有可继承的属性和方法。

[修饰符] class 子类名 extends 父类名

public class Cat extends Animal

特点：

1、什么是继承：让类与类之间产生关系，这种关系就是子父类

2、继承的关键字：子类 extends 父类

3、Java中继承的特点：只支持单继承，但可以多层继承

4、继承父类的公共成员

5、继承的好处和弊端：

好处：提高了代码的复用性，提高了代码的维护性

弊端：类与类之间的耦合性太强，子类不想要的也会被继承

4.2 重写父类方法

定义：在继承关系中，子类会自动继承父类中公共的方法，但有时在子类中需要对继承的方法进行一些修改，即对父类的方法进行重写。

注意：

①子类中重写的方法需要和父类被重写的方法**具有相同的方法名、参数列表以及返回值类型。**

②子类重写父类方法时，**不能使用比父类中被重写的方法更严格的访问权限。**

4.3 super关键字

问题：在继承关系中，当子类重写父类的方法后，子类对象将无法直接访问父类被重写的方法。

解决方法：**在Java中专门提供了一个super**关键字来访问父类的成员，**例如访问父类的成员变量、成员方法和构造方法**

4.1.3 super关键字——具体使用

① 使用super关键字调用父类的成员变量和成员方法。

```
super.成员变量
super.成员方法([参数1,参数2...])
```

② 使用super关键字调用父类的构造方法。

```
super([参数1,参数2...])
```

4.4 Object类

1. 在Java中提供了一个Object类，**它是所有类的父类**，即每个类都直接或间接继承自该类。
2. Object类通常被称之为超类、基类或根类。
3. 当定义一个类时，如果没有使用extends关键字为这个类显示地指定父类，那么该类会默认继承Object类。

4.1.4 Object类——常用方法

方法声明	功能描述
boolean equals(Object obj)	判断某个对象与此对象是否相等
final Class<?> getClass()	返回此Object的运行时类
int hashCode()	返回该对象的哈希码值
String toString()	返回该对象的字符串表示
void finalize()	垃圾回收器调用此方法来清理没有被任何引用变量所引用对象的资源

➢ Object类的toString()方法中输出信息格式及说明：


```
getClass().getName() + "@" + Integer.toHexString(hashCode());
```

- getClass().getName()：代表返回对象所属类的类名，即包名+类名的全限定名称。
- hashCode()：代表返回该对象的哈希值。
- Integer.toHexString(hashCode())：代表将对象的哈希值用16进制表示。其中，hashCode()是Object类中定义的一个方法，这个方法将对象的内存地址进行哈希运算，返回一个int类型的哈希值。

4.5 final关键字

final关键字可用于修饰类、变量和方法，它有“不可更改”或者“最终”的含义。因此被final修饰的类、变量和方法将具有以下特性。

注意：

- final修饰的类不能被继承；
- final修饰的方法不能被子类重写；
- final修饰的变量（成员变量和局部变量）是常量，只能赋值一次。

4.6 抽象类

定义理解：

4.3.1 抽象类

问题：例如前面在定义Animal类时，shout()方法用于表示动物的叫声，但是不同的动物，叫声也是不同的，因此在shout()方法中无法准确描述动物的叫声。如何能使Animal类中既包含shout()方法，又无需提供其方法的实现呢？

解决方法：Java提供了抽象方法来满足这种需求。抽象方法必须使用abstract关键字来修饰，并且在定义方法时不需要实现方法体。当一个类中包含了抽象方法，那么该类也必须使用abstract关键字来修饰，这种使用abstract关键字修饰的类就是抽象类。

```
// 定义抽象类
 [修饰符] abstract class 类名 {
    // 定义抽象方法
    [修饰符] abstract 方法返回值类型 方法名([参数列表]);
    // 其他方法或属性
}
```

注意：包含抽象方法的类必须定义为抽象类，但**抽象类中可以不包含任何抽象方法。**另外，抽象类是不可以被实例化的，如果想调用抽象类中定义的抽象方法，需要创建一个子类，在子类中实现抽象类中的抽象方法。

4.6.1 接口

接口中除了抽象方法外，还可以有默认方法和静态方法（也叫类方法），默认方法使用default修饰，静态方法使用static修改，并且这两种方法都允许有方法体。

```
    //JDK 8接口定义的语法格式：

    [修饰符] interface 接口名 [extends 父接口1,父接口2,...] {
          [public] [static] [final] 常量类型 常量名 = 常量值;
          [public] [abstract] 方法返回值类型 方法名([参数列表]);
          [public] default 方法返回值类型 方法名([参数列表]){
    // 默认方法的方法体
          }
          [public] static 方法返回值类型 方法名([参数列表]){
    // 类方法的方法体
          }
    }
```