继承

一 内容回顾（列举前一天重点难点内容）

1.1 教学重点

```
1.面向对象与面向过程的理解
2.类和对象的关系
3.熟练使用类创建对象
4.熟练掌握类成员的使用
5.掌握static的使用
6.熟练使用构造方法
7.熟练掌握继承特性
```

1.2 教学难点

```
1.面向对象和面向过程的深入理解
2.static的内存理解
```

二 教学目标

```
1.掌握继承的基本使用
2.掌握方法的重写
3.掌握继承中构造方法的使用
4.掌握final关键字的基本使用
5.掌握Object类的常用方法使用
6.了解访问权限修饰符
7.了解空白final
```

三 教学导读

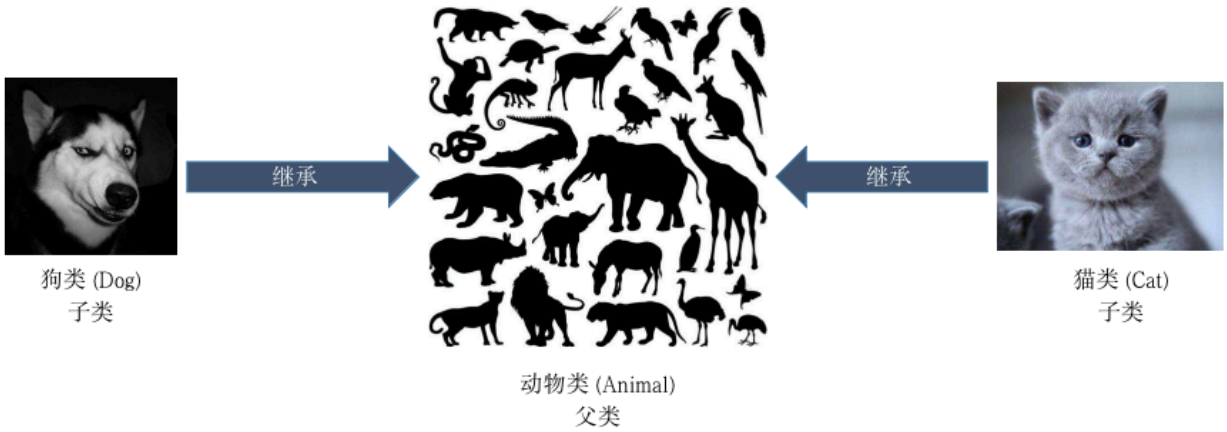
3.1. 程序中的继承

在现实生活中,我们与父母有继承的关系,在java中也存在继承的思想,来提高代码的复用性,代码的拓展性。

程序中的继承， 是类与类之间的特征和行为的一种赠予或获取。 一个类可以将自己的属性和方法赠予其他的类， 一个类也可以从其他的类中获取他们的属性和方法。

两个类之间的继承， 必须满足 **is a** 的关系。

两个类之间， A类将属性和特征赠予B类。 此时 A类被称为是**父类**， B类被称为是 **子类**， 两者之间的关系是 **子类继承自父类**



3.2. final关键字

是一个常用关键字,意思是最后的,最终的.在程序中只要被final修饰的内容是不能再被改变的.比如:

int a = 5;这是定义了一个int型的变量a赋值成5.如果我们给它加上修饰词final

final int a = 5; 这里a就相当于变成了常量,值一直是5,不能再被改变.

3.3. Object类

Object类:是java继承体系中所有类的父类,Object类没有父类.

作为所有类的父类,它里面有11个方法,相当于所有子类都默认有这些方法,所以都很重要.

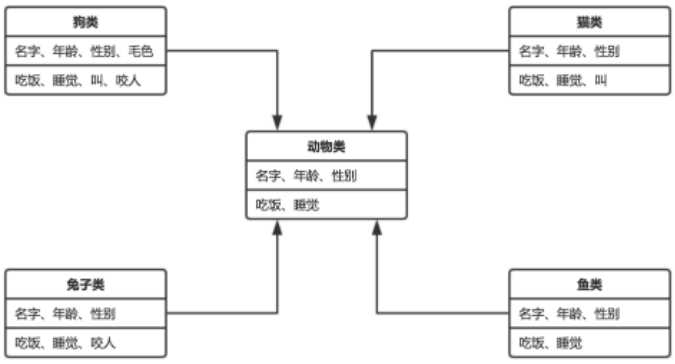
四 教学内容

4.1. 继承

4.1.1. 父类的抽取

在设计类的时候， 可以根据程序中需要使用到的多个具体的类， 进行共性的提取， 定义为父类。

将多个具体的类中相同的属性和行为提取出来到一个类中。



4.1.2. 继承的特点(重点)

- 产生继承关系后， 子类可以使用父类中的属性和方法， 也可以定义子类独有的属性和方法
- Java是单继承, 一个类有且只能有一个直接的父类,可以有若干个间接的父类,一个父类可以有0个或者多个子类,子类之间互不影响

- 使用继承,可以简化代码,提高代码的复用性,提高代码的拓展性,增加代码的健壮性,最重要的使类与类之间产生了继承的关系,是多态的前提

4.1.3. 继承的语法

- 继承， 需要使用关键字 extends。 在定义类的时候， 使用 子类 extends 父类 的方式描述继承。
- Object类:是java继承体系中所有类的父类,Object类没有父类.

```
//Phone类没有显示的父类,默认父类是Object类
class Phone {}
//Iphone类继承自Phone类
class Iphone extends Phone {
}
```

4.1.4. 不可以被继承

- 构造方法
  - 构造方法是为了创建当前类的对象的， 不可以继承给子类。
- 私有成员
  - 私有成员只能在当前的类中使用， 不可以继承给子类。
  - 注意:父类的私有成员,在子类中可见不可用
- 跨包子类
  - 默认权限的属性、方法， 不可以继承给跨包的子类。

4.1.5. 访问权限修饰符

访问权限修饰符， 就是修饰类、属性的访问级别。

	当前类	同包其他类	跨包子类	跨包其他类
private	√	×	×	×
default(不能写出， 不写权限， 默认就是这个权限)	√	√	×	×
protected	√	√	√	×
public	√	√	√	√

4.1.6. 示例代码

- 这里描述的是父类Phone类和子类Iphone类和HuaWei类的关系
- 对象调用方法的原理

  首先会在自己的内部找这个方法,找到了直接使用,找不到会继续去父类中找,同理,找到了使用,停止查找,找不到继续向上找,一直找得到Object类,如果还没有找到,就认为没有这个方法

```
1
2 public class Demo4 {
3     public static void main(String[] args) {
4         Iphone iphone = new Iphone();
5         iphone.fangShui();//调用自己的方法
6         iphone.model = 44;//调用父类的属性
7
8         iphone.callPhone();//调用父类的方法
```

```
    }
}

//Iphone类的父类是Phone类
class Iphone extends  Phone{
    String gui;

    public void  fangShui(){
        System.out.println("防水");
    }
}

class HuaWei extends  Phone{
    public void niu(){
        System.out.println("牛");
    }

    public Phone show(){
        return  null;
    }
}

//注意:如果一个类的后面没有写父类,默认的父类就是Object
class  Phone extends  Object{
    String color;
    int model;

    public  void  callPhone(){
        System.out.println("打电话");
    }
}

```

4.1.6. 方法重写(重点)

4.1.6.1. 重写原理

- 当子类希望实现父类方法功能之外的一些功能时,如何处理?

分析:

```
方法:callPhone
父类:Phone   打电话
子类:Iphone    打电话+唱歌
子类:Huawei    打电话+视频

对于子类Iphone:打电话时可以直接调用父类的callPhone方法,但是唱歌功能无法实现.
```

解决:

子类可以继承到父类中的属性和方法， 但是有些方法， 子类的实现与父类的方法可能实现的不同。 当父类提供的方法已经不能满足子类的需求时，子类中可以定义与父类相同的方法。

- 当出现同名方法后,方法如何调用?

这时子类方法完成对父类方法的覆盖， 又叫重写（Override）。

```
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程员大数据
 * @Description 方法的重写
 */
class  Phone extends  Object{
    String color;
    int model;
```

```
9
10      public  void  callPhone(){
11          System.out.println("打电话");
12      }
13
14      public void show(){
15          return  null;
16      }
17  }
18  class Iphone extends  Phone{
19      String gui;
20
21      public void  fangShui(){
22          System.out.println("防水");
23      }
24
25      @Override
26      public  void  callPhone(){
27          //公共的功能
28          //System.out.println("打电话");
29          //通过super调用父类的callPhone方法
30          super.callPhone();
31          //自己的功能
32          System.out.println("自己的功能");
33      }
34
35      //父类返回值是Phone,子类的show返回值是Iphone,重写时,程序
    允许返回值不同
36      //重写的注意点:父类和子类的重写方法返回值可以不同,但是方法
    名,参数必须相同.返回值不同时,
37      //必须保证父子类的重写方法的返回值有继承关系,并且子类中的返回
    值的类型是父类的返回值类型或者子类类型
```

```
38      //在同一个类中不允许出现方法名,参数都相同,但是返回值不同的情
    况.
39      public Iphone show(){
40          return  null;
41      }
42  }
```

4.1.6.2. @Override

一个注解， 进行重写前的校验。 校验这个方法是否是一个重写的方法。 如果不是重写的方法， 会直接报错。

4.1.6.3. 重写的注意事项

- 方法名字必须和父类方法名字相同
- 参数列表必须和父类一致
- 子类方法的访问权限需要大于等于父类方法的访问权限,父类不能是private
- 子类方法的返回值类型需要小于等于父类方法的返回值类型--这里说的是引用类型

注意:从上面代码段show方法上可以体现出来

4.1.6.4. super关键字

- 有时候,子类重写父类方法的时候,并不是要对父类的实现全盘推翻,而是对父类方法进行拓展。
- 父类方法中的实现仍然需要， 但是还需要在父类方法的基础上进行拓展的实现， 此时就需要使用super关键字调用父类的方法。
- this和super的对比

this:是一种引用类型,代表当前对象,保存的是当前对象的地址 super:代表的是当前对象的父类,可以调用父类的成员.但是他不是引用数据类型.

```
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description super
 */
class Student {
    public void work() {
        System.out.println("学生在学习");
    }
}
class Monitor extends Student {
    @Override
    public void work() {
        super.work(); // 调用父类的实现
        System.out.println("班长收作业");
    }
}
```

4.1.7. 继承中使用构造方法(重点)

4.1.7.1. 基本使用

- 子类对象在实例化的时候， 需要先实例化从父类继承到的部分。 此时通过super()默认调用父类中的无参构造方法。
- 注意:super()方法是默认的,不需要显示写出来.

示例代码

```
/**
```

```
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description super
 */
class Animal {
    public Animal() {
        System.out.println("父类中的构造方法执行");
    }
}
class Dog extends Animal {
    public Dog() {
        //这里不显示写,也会默认有super(),调用的是父类的空参构造方法
        super();
        System.out.println("子类中的构造方法执行");
    }
}
```

- 当我们创建构造方法的时候,为什么一定要调用super?

```
原因:父类中也有属性要进行初始化,而对象的属性必须由自己的构造方法进行初始化,所以必须调用super(),所以每个构造方法中都默认有一个super()
```

- 如果父类中没有无参构造方法， 对所有的子类对象实例化都会造成影响， 导致子类对象无法实例化!

(类中没有无参构造方法的情况:当我们在类中只写了有参构造方法时,系统不会再自动创建无参构造方法.)

解决这个问题， 有两种方案:

```
1.给父类添加无参构造方法。
2.在子类的构造方法中,使用 super(参数列表) 调用父类中的有参构造方法。

总结:在继承体系中,作为父类最好的办法就是将无参构造方法和有参构造方法都写了.
```

示例代码

```java
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description super
 */
class Animal {
    String name;
    int age;
    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    //1.给父类添加无参构造方法。
    public Animal(){
    }
}
class Dog extends Animal {
    public Dog(String name, int age) {
        super(name, age); // 2.调用父类中存在的构造方法，实例化父类部分
    }
}
```

4.1.7.2 填坑

为什么要将super放在构造方法的第一行?

> 在子类的构造方法中有可能用到父类的属性,而属性在使用之前必须先进行初始化,否则无法使用.super的作用是初始化父类的属性,如果在super之前调用了其他的代码,有可能造成在未对父类属性初始化时使用了属性的错误情况发生,所以系统默认super前面不能放任何代码,this()用于构造方法调用时也要放在第一行是一个道理.

示例代码

```
class  Animal{
    String name;
    //public Animal(){}
    public Animal(String name){
        super();
        this.name = name;
    }
}

class Dog extends  Animal{
    public Dog(){
        //super前不能写任何代码
        //System.out.println();
        super("bing");
    }
}
```

4.1.8. 类与类之间的关系

- 共有三种

```
1.继承---描述:谁是谁          例如:人是动物
2.组合---描述:谁属于谁        例如:狗属于人/人拥有狗
3.传参

注意:在实现继承或组合时,要符合现实的逻辑,我们写到代码是为人们的生活服务端
比如:语法中我们写成人属于狗也可以.但是不符合现实逻辑,现实中应该是够属于人.
```

- 实例代码

```
//创建测试类
public class Demo7 {
//人有一只狗
public static void main(String[] args) {
      Person per = new Person();
      per.name = "张三";
      per.dog = new Dog("拉布拉多");
    }

    public static void feedDog(Dog dog){//传参
      dog.eat();
    }
}

//创建的人类
class Person{
    String name;
    Dog dog;//组合
}
```

```
20
21 //创建动物类
22 class Animal{
23     String name;
24 }
25 //创建狗类
26 class  Dog extends Animal{//继承
27     String name;//组合
28     public Dog(String name){
29         this.name = name;
30     }
31
32     public void eat(){
33       System.out.println("吃骨头");
34     }
35 }
```

4.2 final关键字的使用(会)

4.2.1 定义

是一个关键字,最后的,最终的.被final修饰的内容是不能再被改变的.

4.2.2. 可以修饰的内容

- 1.类:final修饰的类不能有子类
- 2.成员变量:变量是一个终值,不能再被改变.所以在定义时必须先手动给一个值.
- 3.局部变量:被final修饰的局部变量是一个终值,不能再被改变
- 4.方法:final修饰的方法不允许重写

4.2.3. 示例代码

```
public class Demo6 {
    public static void main(String[] args) {
        Animal animal = new Animal();
        //name是常量,值不能改变.
        //animal.name = "chen";
    }
}
//1.类:final修饰的类不能有子类,外界只能用不能改
//final  class  Animal{
class  Animal{
    //2.成员变量:变量是一个终值,不能再被改变.所以在定义时必须先手动给一个值.
     final String name = "bing";

     //4.方法:final修饰的方法不允许重写
     public final   void  show(){
         //3.局部变量:被final修饰的局部变量是一个终值,不能再被改变
         final  int a = 4;
         //a++;
         System.out.println(a);
     }
}
```

4.2.4. 空白final(了解)

- 描述:修饰成员变量的一种形式
- 如果是自定义的类可以在定义时先不给值,但是必须在构造方法中给值,即必须保证属性是被初始化了的.这种情况叫空白final.
- 优点:空白final在final的使用上提供了更大的灵活性,因为一个类中的

final域可以做到根据对象而有所不同,却又保持其恒定不变的特性.

```
//接上面的代码
class  Bird extends  Animal{
  //这里暂时没有赋值
    final  String model;
    public Bird(String model){
        this.model = model;//这里完成赋值
    }
}
```

课上练习

求圆的面积

```
/*
 * double PI = 3.14
 */
public class Demo7 {
    public static void main(String[] args) {
        Circle circle = new Circle(2);
        double area = circle.getArea();
        System.out.println(area);
    }
}

class  Circle {
    final  double PI = 3.14;//定义符号常量
    double r;

    public Circle(double r) {
        this.r = r;
    }

```

```
20      public  double getArea(){
21          return  r*r*PI;
22      }
23
24      //举例--符号常量
25      final int HAPPY = 1;
26      final int UNHAPPY = 2;
27  }
```

4.3 Object类的使用(了解)

4.3.1 定义

Object类:是java继承体系中所有类的父类,Object类没有父类.

作为所有类的父类,它里面有11个方法,相当于所有子类都默认有这些方法,所以都很重要.

方法摘要	
protected Object	clone() 克隆 创建并返回此对象的一个副本。
boolean	equals(Object obj) 判断两个对象是否相等 指示其他某个对象是否与此对象“相等”。
protected void	finalize() 进行垃圾回收 当垃圾回收器确定不存在对该对象的更多引用时，由对象的垃圾回收器调用此方法。
Class<?>	getClass() 获取类的字节码文件对象 返回此 Object 的运行时类。
int	hashCode() 获取对象的哈希码值,默认是一个十六进制的数 返回该对象的哈希码值。
void	notify() 唤醒同一把锁下的任意一个线程 唤醒在此对象监视器上等待的单个线程。
void	notifyAll() 唤醒同一把锁下的所有线程 唤醒在此对象监视器上等待的所有线程。
String	toString() 返回字符串表示,经常需要重写 返回该对象的字符串表示。
void	wait() 线程等待 在其他线程调用此对象的 notify() 方法或 notifyAll() 方法前，导致当前线程等待。
void	wait(long timeout) 线程等待 在其他线程调用此对象的 notify() 方法或 notifyAll() 方法，或者超过指定的时间量前，导致当前线程等待。
void	wait(long timeout, int nanos) 线程等待 在其他线程调用此对象的 notify() 方法或 notifyAll() 方法，或者其他某个线程中断当前线程，或者已超过某个实际时间量前，导致当前线程等待。

4.3.2 常用方法讲解

注意:这里我们先讲下面的四个方法,其他的后面会涉及到.

- equals方法

使用场景:equals方法默认比较的是对象地址,在实际应用中没有太大的实际意义,我们经常需要重写equals方法,大部分系统类,自定义的类,都会通过重写equals,再按照自己制定的规则进行比较.比如:String字符串类,集合类等

```
public class Demo9 {
    public static void main(String[] args) {
        //判断两辆车的轮子个数多少
        Car car1 = new Car(3);
        Car car2 = new Car(3);
        boolean b = car1.equalsWithWheelNumber(car2);
        System.out.println(b);
        //用equals方法实现比较
        //默认规则:比较的是两个对象的地址      这里返回false
        //当我们重写了equals方法,改变了比较规则   这里返回ture
        boolean b1 = car1.equals(car2);
        System.out.println(b1);
    }
}


class  Car{
    int wheelNumber;
    public Car(int wheelNumber) {
        this.wheelNumber = wheelNumber;
    }

    //比较轮子个数
    public boolean equalsWithWheelNumber(Car car){
```

```
25          return this.wheelNumber ==  car.wheelNumber;
26      }
27
28      //重写equals方法
29      @Override
30      public boolean equals(Object o) {
31          if (this == o) return true;
32          if (o == null || getClass() != o.getClass())
    return false;
33          Car car = (Car) o;
34          return wheelNumber == car.wheelNumber;
35      }
36  }
```

- hashcode方法

使用场景:Object的hashcode方法默认返回的是对象哈希码值十六进制形式,在同一个工程内部,我们可以认为是唯一的,大部分情况会重写它,通过重写改变对象哈希值的计算方法.比如:集合类

```
1  public class Demo9 {
2      public static void main(String[] args) {
3          Car car1 = new Car(3);
4          Car car2 = new Car(3);
5          //重写hashcode方法
6          System.out.println(car1.hashCode());//460141958
    默认是哈希码值的十进制形式
7
    System.out.println(car2.hashCode());//1163157884
8
9
    System.out.println(Integer.toHexString(car1.hashCode())
    );  //1b6d3586    十六进制的哈希码值
```

```
        //因为重写了hashcode方法,此时的hashcode默认返回1,只要是Car类的对象都会返回1,所以结果true
        System.out.println(car1.hashCode() == car2.hashCode());
    }
}


class  Car{
    int wheelNumber;

    public Car(int wheelNumber) {
        this.wheelNumber = wheelNumber;
    }

    //重写hashcode方法
    @Override
    public int hashCode() {
        return 1;
    }
}
```

- getClass方法

使用场景:获取对象的字节码文件对象,每个类都有唯一的字节码文件对象.使用位置比如:反射

```
public class Demo9 {
    public static void main(String[] args) {
        //判断两辆车的轮子个数多少
        Car car1 = new Car(3);
        Car car2 = new Car(3);
```

```
6
7           Class  cla1 = car1.getClass();
8
System.out.println(cla1.getName());//com.qf.test.Car
9       }
10  }
11
12
13  class  Car{
14      int wheelNumber;
15      public Car(int wheelNumber) {
16          this.wheelNumber = wheelNumber;
17      }
18  }
```

- toString方法

使用场景:默认打印的是包名+类名+@+十六进制的哈希码值,这些内容对我们的作用也不大.

后期因为默认内容作用小,打印时直接写引用名字,让操作简单,我们会经常重写toString,用来展示对象属性的值.

```
1   public class Demo9 {
2       public static void main(String[] args) {
3           //判断两辆车的轮子个数多少
4           Car car1 = new Car(3);
5           Car car2 = new Car(3);
6
7           //当我们直接打印引用的时候,默认调用的时toString()方法
8
System.out.println(car1.toString());//com.qf.test.Car@1
```

```
        System.out.println(car1);

        //自己拼接toString的默认值

System.out.println(cla1.getName()+"@"+Integer.toHexStri
ng(car1.hashCode()));
    }
}


class  Car{
    int wheelNumber;

    public Car(int wheelNumber) {
        this.wheelNumber = wheelNumber;
    }

    //重写toString
    @Override
    public String toString() {
        return "Car{" +
                "wheelNumber=" + wheelNumber +
                '}';
    }
}
```