* 今天学习目标

* 能够理解面向对象抽象过程

* 发现种类，属性，行为

* 能够理解什么是UML

* 统一建模语言（图形）

* 能够使用PowerDesigner画类图

* 安装

* 画类图

* 能够掌握static关键字

* 类变量:

* 类方法（静态方法）

* Arrays,Math

* 静态代码块

* 能够掌握this关键字

* 调用本类构造器(必须在第一行)

* 能够编写神兽青龙Dragon和神兽白虎Tiger类并输出宠物神兽信息

* 实体

* 能够理解面向对象封装过程

* 该隐藏的隐藏，该公开的公开

* private，set,get

* JavaBean的规范

* 无参数的构造器

* set/get

* 能够理解面向对象继承

* extends

* abstract(没有方法体，子类必须强制实现)

* 成员变量：重名,super

* 成员方法：方法重写和方法重载的区别

* 构造器：super():父类的构造器先执行

---

* 能够理解面向对象抽象过程

* 面向对象抽象

* 发现种类：Dragon

* 发现属性：id，masterId，name，grade（等级），健康值,亲密度....

* 发现行为：作战（Fight）

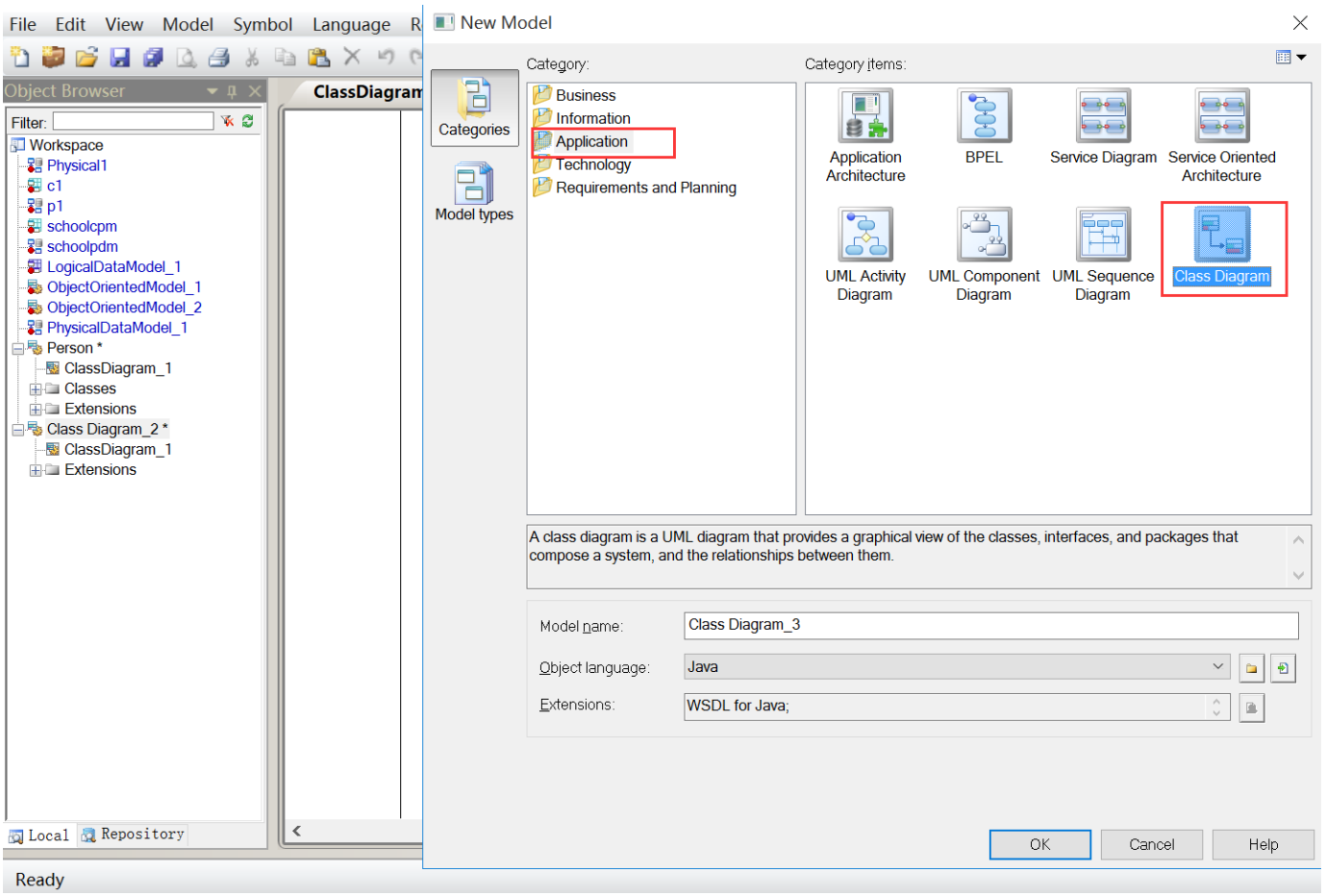
```
package com.lg.test;

public class Dragon {
    //id，masterId，name，grade（等级），健康值,亲密度....
    public long id;// 宠物id
    public long masterId;// 宠物主人id
    public String name;// 宠物名字
    public String grade;// 宠物的等级
    public int health;// 宠物的健康值
    public int love;// 宠物与主人亲密度

    //作战（Fight）
    public void fight(String name) {
        System.out.println(this.name+"与"+name+"大战");
    }
}

package com.lg.test;

public class Test1 {
    public static void main(String[] args) {
        Dragon dragon=new Dragon(); // 发现种类
        dragon.id=10001;
        dragon.masterId=101;
        dragon.name="小青龙";
        dragon.grade="一爪金龙";
```

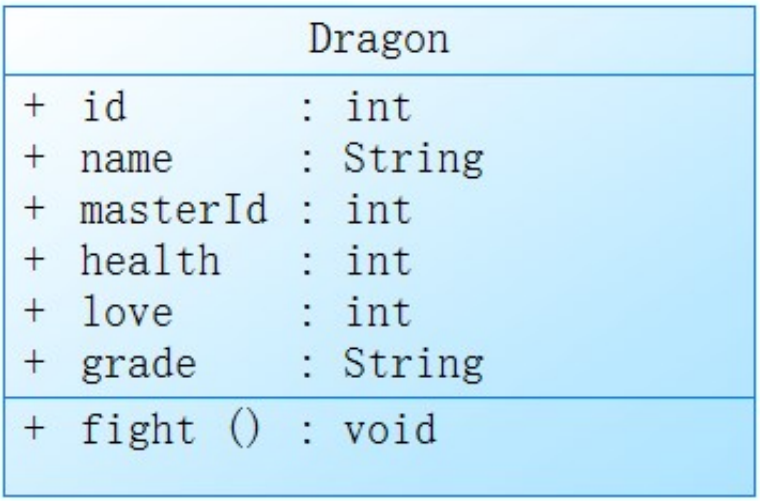
```
        dragon.love=10;
        dragon.health=100; // 发现属性
        dragon.fight("小白虎"); // 发现方法
    }
}

结果：小青龙与小白虎大战
```

* 能够理解什么是UML
    * UML(Unified Modeling Language)统一建模语言
    * 作用：为所有的开发语言创建统一的项目模型
* 能够使用PowerDesigner画类图
    * 简介

```
PowerDesigner最初由Xiao-Yun Wang（王晓昀）在SDP Technologies公司开发完成。
PowerDesigner是Sybase的企业建模和设计解决方案，采用模型驱动方法，将业务与IT结合起来，
可帮助部署有效的企业体系架构，并为研发生命周期管理提供强大的分析与设计技术。
```

    * PowerDesigner安装（参考安装文档）PowerDesigner安装
    * 画类图





* 能够掌握static关键字

* 概述:static可以用来修饰类的成员方法、类的成员变量，另外可以编写static代码块来优化程序性能。

* 类变量： 当 static 修饰成员变量时，该变量称为类变量。

* 静态变量被所有的对象所共享，在内存中只有一份，它当且仅当在类初次加载时会被初始化。

* 类方法（静态方法）：当 static 修饰成员方法时，该方法称为类方法

* 静态方法可以直接访问类变量和静态方法。

* 静态方法不能直接访问普通成员变量或成员方法。

* 成员方法可以直接访问类变量或静态方法。

* 静态方法中，不能使用this关键字。

* 温馨提示：静态方法只能访问静态成员。

```
2
3  public class Utils {
4      public static String str1="xx";
5      public String str2="xxx";
6      //静态方法
7      public static void sayHello() {
8          System.out.println(str1);
9          System.out.println(str2);
10         say();
11     }
12
13     // 非静态方法
14     public void say() {
15         System.out.println(str1);
16         System.out.println(str2);
17         sayHello();
18     }
19 }
20
```

静态的成员变量

静态的方法

在静态方法里面，不能调用非静态的方法和变量

* 静态代码块：

* static块可以置于类方法外的任何地方，类中可以有多个static块。在类初次被加载的时候，会按照static块的顺序来执行每个static块，并且只会执行一次。

* static代码块优先于main方法和构造方法的执行

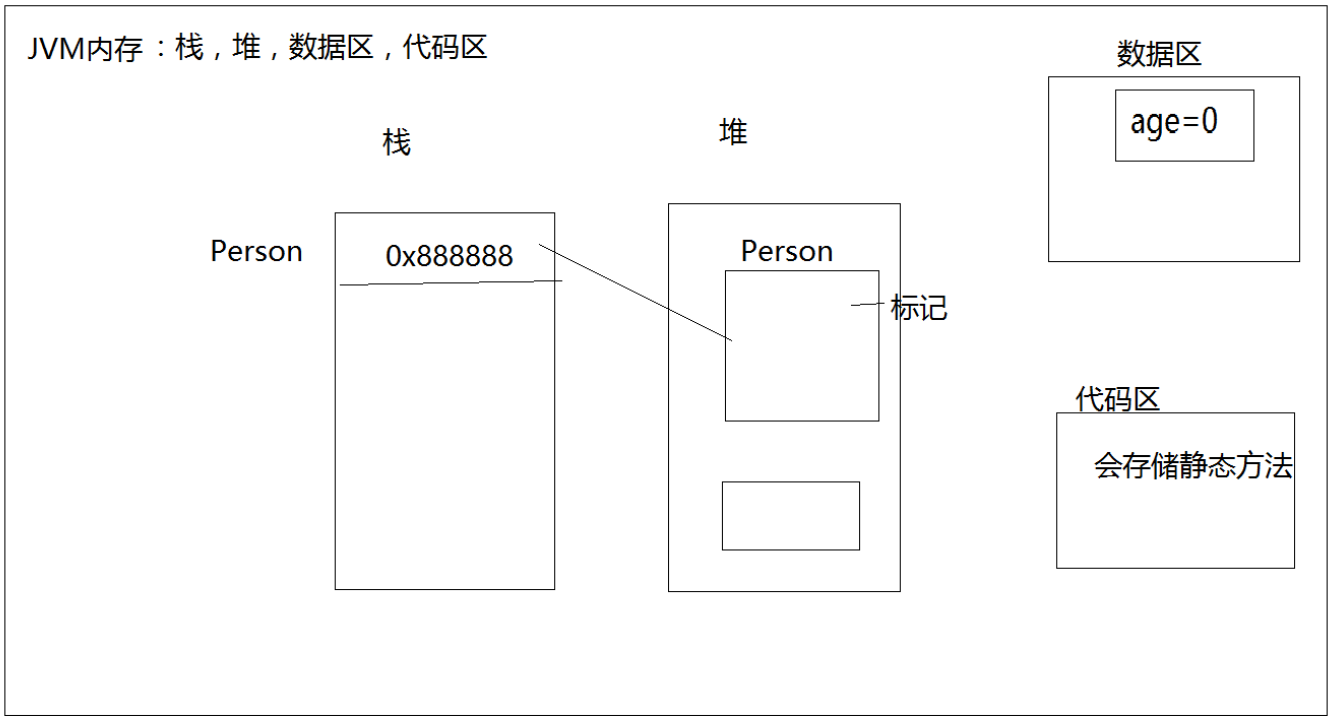
```java
public static Customer customers[]=new Customer[100];//用户集合
public static Goods goods[]=new Goods[100];//商品集合
public static Manager manager=new Manager();
// 静态代码块
static {
    //1 初始化用戶信息
        initCustomers();
    //2 初始化商品信息
        initGoods();
    //3 初始化管理员
        initManager();
}
```

* static在内存存放位置

  * JVM内存总共有4部分：堆，栈，数据区，代码区

  * static修饰变量存储在数据区

  * static修饰方法存储在代码区



* 举例：Math，Arrays 用到类变量，静态方法

```java
double pi=Math.PI;
        double e=Math.E;
        System.out.println(Math.abs(-1));
```

```
        System.out.println(Math.pow(3, 2));//2 的三次方
        System.out.println(Math.sqrt(4));
        System.out.println(Math.min(2, 3));
        System.out.println(Math.max(3, 2));
        System.out.println((int)(Math.random()*10));
        // 天
        System.out.println(Math.ceil(3.1));
        // 地
        System.out.println(Math.floor(3.8));
        // 四舍五入
        System.out.println(Math.round(3.5));
}

public static void main(String[] args) {
        // 二分查找
        int[] arr= {1,4,8,99,158};
        int index = Arrays.binarySearch(arr, 99);
        System.out.println(index);
        System.out.println(Arrays.toString(arr));
}
```

* 能够掌握this关键字

  * 代表当前对象

  * 区分成员变量和局部变量

  * 调用本类的构造器

```
public Dragon(int id, int masterId, String name) {
    this(id,masterId,name,0,0);
}
public Dragon(int id, int masterId, String name, int health, int love) {
    super();
    this.id = id;
    this.masterId = masterId;
    this.name = name;
    this.health = health;
    this.love = love;
}
```

构造器的调用

```
public Dragon(int id, int masterId, String name) {
    System.out.println("");
    this(id,masterId,name,0,0);
}
```

this方法，需要放在第一行

```
public Dragon(int id, int masterId, String name, int health, int love) {
    super();
    this.id = id;
    this.masterId = masterId;
    this.name = name;
    this.health = health;
    this.love = love;
}
```

```
public Dragon() {
    this(0,0,null);
    System.out.println("Dragon()");
}
public Dragon(int id, int masterId, String name) {
    this(id,masterId,name,0,0);
    System.out.println("Dragon(int id, int masterId, String name)");
}

public Dragon(int id, int masterId, String name, int health, int love) {
    super();
    this.id = id;
    this.masterId = masterId;
    this.name = name;
    this.health = health;
    this.love = love;
    this.grade="一爪青龙";
    System.out.println("Dragon(int id, int masterId, String name, int health, int love)");
}
```

无论哪个构造器被调用，都会给青龙等级赋值

* 能够编写神兽青龙Dragon和神兽白虎Tiger类并输出宠物神兽信息

```
我是小青龙
神兽的自白:
我的名字叫小白虎，我的健康值是100，我和主人的亲密程度是0。
```

```
1 package com.lg.test;
2
3 public class Dragon {
4     //id，masterId，name，grade（等级），健康值,亲密度....
5     public long id;// 宠物id
```

```
    public long masterId;// 宠物主人id
    public String name;// 宠物名字
    public String grade;// 宠物的等级
    public int health;// 宠物的健康值
    public int love;// 宠物与主人亲密度
    // 假如没有写构造器，系统默认提供无参数构造器
    public Dragon() {
        // JavaBean
//      System.out.println("");
        this(10001,"青龙");// 调用本类构造器的时候，只能写在第一行
    }

    public Dragon(long id,String name) {
        this.id=id;
        this.name=name;
        System.out.println("Dragon(long id,String name)");
    }

    //作战（Fight）
    public void fight(String name) {
        System.out.println(this.name+"与"+name+"大战");
    }

    public void printPetInfo() {
         System.out.println("我是"+this.name );
    }
}

package com.lg.test;

public class Tiger {
    public long id;// 宠物id
    public long masterId;// 宠物主人id
    public String name;// 宠物名字
    public int health;// 宠物的健康值
    public int love;// 宠物与主人亲密度
    public static final char SEX_MALE = '男';
    public static final char SEX_FEMALE = '女';
    public static char sex = SEX_MALE;

```

```
    public Tiger() {
        this(0,0,"");
    }

    public Tiger(long id,long masterId,String name) {
        this(id,masterId,name,100,0);
    }

    public Tiger(long id,long masterId,String name,int health,int love) {
        this.id=id;
        this.masterId=masterId;
        this.name=name;
        this.health=health;
        this.love=love;
    }



    public void printPetInfo() {
        System.out.println("神兽的自白：\n我的名字叫" + this.name + "，我的健康值是
    }


}

public class Test4 {
    public static void main(String[] args) {
        Dragon dragon=new Dragon();
        Tiger tiger=new Tiger();
        dragon.printPetInfo();
        tiger.printPetInfo();
    }
}

结果：
我是青龙
神兽的自白：
我的名字叫，我的健康值是100，我和主人的亲密程度是0。

```

* 能够理解面向对象封装过程

    * 对象中的成员该隐藏的隐藏、该公开的要公开

```java
package com.lg.test;

public class Dragon {
    //id，masterId，name，grade（等级），健康值,亲密度....
    private long id;// 宠物id
    private long masterId;// 宠物主人id
    private String name;// 宠物名字
    private String grade;// 宠物的等级
    private int health;// 宠物的健康值
    private int love;// 宠物与主人亲密度
    // 假如没有写构造器，系统默认提供无参数构造器
    public Dragon() {
        // JavaBean
//      System.out.println("");
        this(10001,"青龙");// 调用本类构造器的时候，只能写在第一行
    }

    public Dragon(long id,String name) {
        this.id=id;
        this.name=name;
    }

//  // get,set
//  public long getId() {
//      return this.id;
//  }
//  public void setId(long id) {
//      this.id=id;
//  }



    //作战（Fight）
```

```
34      public void fight(String name) {
35          System.out.println(this.name+"与"+name+"大战");
36      }
37
38      public long getId() {
39          return id;
40      }
41
42      public void setId(long id) {
43          this.id = id;
44      }
45
46      public long getMasterId() {
47          return masterId;
48      }
49
50      public void setMasterId(long masterId) {
51          this.masterId = masterId;
52      }
53
54      public String getName() {
55          return name;
56      }
57
58      public void setName(String name) {
59          this.name = name;
60      }
61
62      public String getGrade() {
63          return grade;
64      }
65
66      public void setGrade(String grade) {
67          this.grade = grade;
68      }
69
70      public int getHealth() {
71          return health;
72      }
73
```

```
    public void setHealth(int health) {
        this.health = health;
    }

    public int getLove() {
        return love;
    }

    public void setLove(int love) {
        this.love = love;
    }

    public void printPetInfo() {
         System.out.println("我是"+this.name );
    }
}

package com.lg.test;

public class Tiger {
    private long id;// 宠物id
    private long masterId;// 宠物主人id
    private String name;// 宠物名字
    private int health;// 宠物的健康值
    private int love;// 宠物与主人亲密度
    public static final char SEX_MALE = '男';
    public static final char SEX_FEMALE = '女';
    private  char sex = SEX_MALE;

    public Tiger() {
        this(0,0,"");
    }

    public Tiger(long id,long masterId,String name) {
        this(id,masterId,name,100,0);
    }

    public Tiger(long id,long masterId,String name,int health,int love) {
        this.id=id;
        this.masterId=masterId;
```

```
114         this.name=name;
115         this.health=health;
116         this.love=love;
117     }
118 
119 
120 
121     public void printPetInfo() {
122         System.out.println("神兽的自白：\n我的名字叫" + this.name + "，我的健康值是
123     }
124 
125     public long getId() {
126         return id;
127     }
128 
129     public void setId(long id) {
130         this.id = id;
131     }
132 
133     public long getMasterId() {
134         return masterId;
135     }
136 
137     public void setMasterId(long masterId) {
138         this.masterId = masterId;
139     }
140 
141     public String getName() {
142         return name;
143     }
144 
145     public void setName(String name) {
146         this.name = name;
147     }
148 
149     public int getHealth() {
150         return health;
151     }
152 
153     public void setHealth(int health) {
```

```
154         this.health = health;
155     }
156 
157     public int getLove() {
158         return love;
159     }
160 
161     public void setLove(int love) {
162         this.love = love;
163     }
164 
165     public char getSex() {
166         return sex;
167     }
168 
169     public void setSex(char sex) {
170         this.sex = sex;
171     }
172 
173 }
174 
```

* JavaBean的规范

* JavaBean 是 Java语言编写类的一种标准规范。符合 JavaBean 的类，要求类必须是具体的和公共的，并且具有无参数的构造方法，提供用来操作成员变量的 set 和 get 方法。

* 提供无参数构造器

* 提供get/set

* ...

```
1 public class ClassName{
2     //成员变量
3     //构造方法
4     //无参构造方法【必须】
5     //有参构造方法【建议】
6     //成员方法
7     //getXxx()
```

```
    //setXxx()
}
```

* 能够理解面向对象继承

    * 修改Dragon和Tiget

```
package com.lg.test;

public abstract class Pet {
    protected long id;// 宠物id
    protected long masterId;// 宠物主人id
    protected String name;// 宠物名字
    protected int health;// 宠物的健康值
    protected int love;// 宠物与主人亲密度

    public abstract void printPetInfo();// 抽象方法，没有方法体
}

package com.lg.test;

public class Dragon extends Pet{
    //id，masterId，name，grade（等级），健康值,亲密度....
    private String grade;// 宠物的等级
    // 假如没有写构造器，系统默认提供无参数构造器
    public Dragon() {
        // JavaBean
//      System.out.println("");
        this(10001,"青龙");// 调用本类构造器的时候，只能写在第一行
    }

    public Dragon(long id,String name) {
        this.id=id;
        this.name=name;
    }

//  // get,set
//  public long getId() {
//      return this.id;
```

```
//  }
//  public void setId(long id) {
//      this.id=id;
//  }


    //作战（Fight）
    public void fight(String name) {
        System.out.println(this.name+"与"+name+"大战");
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public long getMasterId() {
        return masterId;
    }

    public void setMasterId(long masterId) {
        this.masterId = masterId;
    }

    public String getName() {
        return name;
    }

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

    public String getGrade() {
        return grade;
    }

```

```
    public void setGrade(String grade) {
        this.grade = grade;
    }

    public int getHealth() {
        return health;
    }

    public void setHealth(int health) {
        this.health = health;
    }

    public int getLove() {
        return love;
    }

    public void setLove(int love) {
        this.love = love;
    }

    public void printPetInfo() {
         System.out.println("我是"+this.name );
    }
}

package com.lg.test;

public class Tiger extends Pet{
    public static final char SEX_MALE = '男';
    public static final char SEX_FEMALE = '女';
    private  char sex = SEX_MALE;

    public Tiger() {
        this(0,0,"");
    }

    public Tiger(long id,long masterId,String name) {
        this(id,masterId,name,100,0);
    }

```

```
113     public Tiger(long id,long masterId,String name,int health,int love) {
114         this.id=id;
115         this.masterId=masterId;
116         this.name=name;
117         this.health=health;
118         this.love=love;
119     }
120
121
122
123     public void printPetInfo() {
124         System.out.println("神兽的自白：\n我的名字叫" + this.name + "，我的健康值
125     }
126
127     public long getId() {
128         return id;
129     }
130
131     public void setId(long id) {
132         this.id = id;
133     }
134
135     public long getMasterId() {
136         return masterId;
137     }
138
139     public void setMasterId(long masterId) {
140         this.masterId = masterId;
141     }
142
143     public String getName() {
144         return name;
145     }
146
147     public void setName(String name) {
148         this.name = name;
149     }
150
151     public int getHealth() {
152         return health;
```

```
153     }
154 
155     public void setHealth(int health) {
156         this.health = health;
157     }
158 
159     public int getLove() {
160         return love;
161     }
162 
163     public void setLove(int love) {
164         this.love = love;
165     }
166 
167     public char getSex() {
168         return sex;
169     }
170 
171     public void setSex(char sex) {
172         this.sex = sex;
173     }
174 
175 }
176 
177 package com.lg.test;
178 
179 public class Tortoise extends Pet{
180 
181     @Override
182     public void printPetInfo() {
183         System.out.println("I am "+this.name);
184     }
185 
186 }
187 
188 package com.lg.test;
189 
190 public class Test4 {
191     public static void main(String[] args) {
192         Pet dragon=new Dragon();
```

```
193         Pet tiger=new Tiger();
194         dragon.printPetInfo();
195         tiger.printPetInfo();
196         Pet tortoise=new Tortoise();
197         tortoise.name="小玄武";
198         tortoise.printPetInfo();
199         // 多态
200     }
201 }
202
203
```

* 生活中继承：

* 子类，父类（超类，基类）

* 继承描述的是事物之间的所属关系，这种关系是： is-a 的关系。例如，图中兔子属于食草动物，食草动物属于动 物。可见，父类更通用，子类更具体。

* 概述

* 多个类中存在相同属性和行为时，将这些内容抽取到单独一个类中，那么多个类无需再定义这些属性和行为，只要继承那一个类即可。

* 定义

* 子类继承父类的属性和行为，使得子类对象具有与父类相同的属性、相同的行为。子类可以直接 访问父类中的非私有的属性和行为。

* 好处

* 提高代码的复用性。

* 类与类之间产生了关系，是多态的前提。

* 语法

* 通过 extends 关键字，可以声明一个子类继承另外一个父类

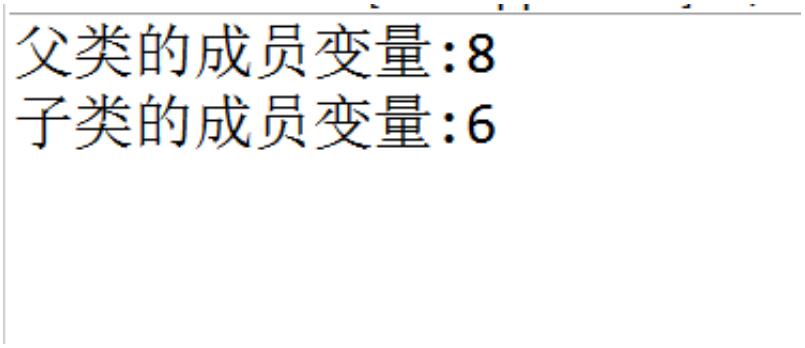
```
class Parent {
    ...
 }
class Son extends Parent {
    ...
 }
```

* 继承之后之成员变量

* 成员变量不重名：如果子类父类中出现不重名的成员变量，这时的访问是没有影响的。

```
package com.lg.test;

public class Parent {
    int count=8;// 父类的成员变量
}

package com.lg.test;

public class Son extends Parent{
    int count2=6;// 子类的成员变量
    public void show() {
        System.out.println("父类的成员变量:"+count);
```

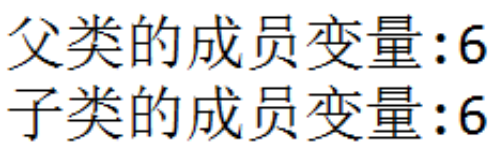
```
        System.out.println("子类的成员变量:"+count2);
    }
}

package com.lg.test;

public class Test {
    public static void main(String[] args) {
        Son son=new Son();
        son.show();
    }
}

```



* 成员变量重名:如果子类父类中出现重名的成员变量，这时的访问是有影响的

```
package com.lg.test;

public class Parent {
    int count=8;// 父类的成员变量
}

package com.lg.test;

public class Son extends Parent{
    int count=6;// 子类的成员变量
    public void show() {
        System.out.println("父类的成员变量:"+count);
        System.out.println("子类的成员变量:"+count);
    }
}
```

```
16
17 package com.lg.test;
18
19 public class Test {
20     public static void main(String[] args) {
21         Son son=new Son();
22         son.show();
23     }
24 }
25
26
```



* super:子父类中出现了同名的成员变量时，在子类中需要访问父类中非私有成员变量时，需要使用 super 关键字，修饰父类成员变量

```
1  package com.lg.test;
2
3  public class Parent {
4      int count=8;// 父类的成员变量
5  }
6
7  package com.lg.test;
8
9  public class Son extends Parent{
10     int count=6;// 子类的成员变量
11     public void show() {
12         System.out.println("父类的成员变量:"+super.count);
13         System.out.println("子类的成员变量:"+this.count);
14     }
15 }
16
17 package com.lg.test;
```

```
18
19 public class Test {
20     public static void main(String[] args) {
21         Son son=new Son();
22         son.show();
23     }
24 }
25
26
```

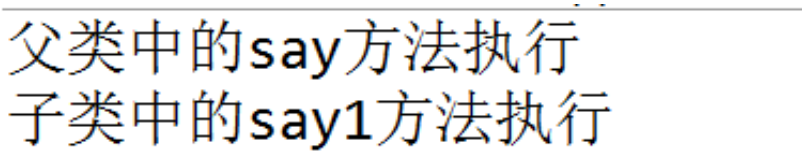
```
父类的成员变量:8
子类的成员变量:6
```

* 继承之后之成员方法

  * 成员方法不重名：如果子类父类中出现不重名的成员方法，这时的调用是没有影响的

```
1  package com.lg.test;
2
3  public class Parent {
4      int count=8;// 父类的成员变量
5
6      public void say() {
7          System.out.println("父类中的say方法执行");
8      }
9  }
10
11 package com.lg.test;
12
13 public class Son extends Parent{
14     int count=6;// 子类的成员变量
15     public void show() {
16         System.out.println("父类的成员变量:"+super.count);
```

```
        System.out.println("子类的成员变量:"+this.count);
    }

    public void say1() {
        System.out.println("子类中的say1方法执行");
    }
}

package com.lg.test;

public class Test2 {
    public static void main(String[] args) {
        Son son=new Son();
        son.say();
        son.say1();
    }
}


```



* 成员方法重名：如果子类父类中出现重名的成员方法，这时的访问是一种特殊情况，叫做方法重写 (Override)

* 方法重写 ：子类中出现与父类一模一样的方法时（返回值类型，方法名和参数列表都相同），会出现覆盖效果，也称为重写或者复写。声明不变，重新实现。

```

package com.lg.test;

public class Parent {
    int count=8;// 父类的成员变量

    public void say() {

```

```
        System.out.println("父类中的say方法执行");
    }
}

package com.lg.test;

public class Son extends Parent{
    int count=6;// 子类的成员变量
    public void show() {
        System.out.println("父类的成员变量:"+super.count);
        System.out.println("子类的成员变量:"+this.count);
    }

    public void say() {
        System.out.println("子类中的say方法执行");
    }
}

package com.lg.test;

public class Test2 {
    public static void main(String[] args) {
        Son son=new Son();
        son.say();
    }
}

```

* 区别方法重写和方法重载

    * 方法重写（Override）：子类继承父类，返回值，方法名，参数列表一样

    * 方法重载：在一个类中：可以写多个同名的方法，方法参数不一样

* 继承之后之构造方法

    * 构造方法的名字是与类名一致的。所以子类是无法继承父类构造方法的。

    * 构造方法的作用是初始化成员变量的。所以子类的初始化过程中，必须先执行父类的初始化动作。子类的构 造方法中默认有一个 super() ，表示调用父类的构造方法，父类成员

变量初始化后，才可以给子类使用。

```
package com.lg.test;

public class Parent {

    public Parent() {
        System.out.println("执行父类的构造方法");
    }
    int count=8;// 父类的成员变量

    public void say() {
        System.out.println("父类中的say方法执行");
    }
}

package com.lg.test;

public class Son extends Parent{
    public Son() {
        super();// 必须写在第一行，不能与this()一起使用,不能默认会加上
        System.out.println("执行子类的构造方法");
    }
    int count=6;// 子类的成员变量
    public void show() {
        System.out.println("父类的成员变量:"+super.count);
        System.out.println("子类的成员变量:"+this.count);
    }

    public void say() {
        System.out.println("子类中的say方法执行");
    }
}

package com.lg.test;

public class Test3 {
    public static void main(String[] args) {
```

```
        Son son=new Son();
    }
}


package com.lg.test;

public class Parent {
    int count=8;
    /*public Parent() {
        System.out.println("Parent ...");
    }*/

    public Parent(int count) {
        System.out.println("Parent ...");
        this.count=count;
    }

    public void say() {
        System.out.println("父亲说活...");
    }
}

package com.lg.test;

public class Son extends Parent{

    public Son() {
        super(18);
        System.out.println("Son ...");
    }
    int count=6;
    public void show() {
        System.out.println("父类count的值: "+super.count);
        System.out.println("子类count的值: "+this.count);
    }

    public void say() {
        System.out.println("子类说...");
    }
```

```
77
78      public void say(int age) {// 方法重载
79
80      }
81  }
82
83
```

```
执行父类的构造方法
执行子类的构造方法
```

* super 与 this

* 在每次创建子类对象时，先初始化父类空间，再创建其子类对象本身。目的在于子类对象中包含了其对应的父类空间，便可以包含其父类的成员，如果父类成员非private修饰，则子类可以随意使用父类成员。代码体现在子类的构造方法调用时，一定先调用父类的构造方法。

* super ：代表父类的存储空间标识(可以理解为父亲的引用)。

* this ：代表当前对象的引用(谁调用就代表谁 ）。