* 今天学习目标

* 能够掌握抽象类

* 抽象方法：没方法体，

* 抽象类：拥有抽象方法的类

* 抽象类不能实例化

* 能够理解Java类加载和执行顺序

* 类加载时机：new,继承（new Son）,JVM标明启动类（类名与文件名相同）,访问静态变量，静态方法，Class.forName("...")，static final（看成常量，不加载）:

* 类加载过程：加载（class文件-->加载到内存,java.lang.Class）--->连接（验证-->准备-->解析）--->类的初始化(静态（父->子）--> 父类（非静态代码块--构）-->子类（非静态代码块--构）)-- 使用-->卸载

* 类加载器（(根--jre/lib)-->（扩--jre/lib/ext/）-->（系统--classpath）-- 自定义类加载器

双亲委派机制（坑爹机制）

* 能够掌握四个修饰符（public，private，default，protected）的区别

* private

* public

* 默认

* protected

* 能够理解面向对象的多态

* 必要条件：继承，重写方法，向上转型

* 父类类型 变量=new 子类型();

变量.方法（）；

* 作为方法参数时候，尽量用父类型（100-->1）

* instanceOf

---

* 回顾

* 面向对象抽象过程

* 发现种类，属性，行为

* UML：统一建模语言

* PowerDesigner（starUml）:画类图

* static(静态)

  * 类变量：静态变量

  * 类方法（静态方法）：Math，Arrays

  * 静态代码块

  * JVM内存：堆，栈，数据区，代码区

* this

  * 区分成员变量和局部变量

  * 代表当前对象

  * 调用本类构造器（只能写在第一行）

* Dragon，Tiger

* 封装：该隐藏隐藏，该公开公开

  * private，get/set

  * JavaBean 的规范：无参数构造器，set/get

 * 继承：Pet，abstract

  * 成员变量：有没有重名，假如有重名，访问的时候有影响，没有重名没有影响

    * super.name

    * this.name

  * 成员方法：有没有重名，假如有重名，访问的时候有影响，方法重写（子类会覆盖父类的方法），没有重名没有影响

    * 方法重写（子父之间）和方法重载（本类之间同名方法不同参数列表）

  * 构造器

    Son son=new Son();

    super();

   * 父构造器是由于子类构造执行

* 能够掌握抽象类

* 概述：父类中的方法，被它的子类们重写，子类各自的实现都不尽相同。我们把没有方法主体的方法称为抽象方法。Java语法规定，包含抽象方法的类就是抽象类。

* 定义：

* 抽象方法 ：没有方法体的方法

* 抽象类：包含抽象方法的类。

* 使用abstract关键字进行就行修饰

```
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;
    }
```

```
70
71      public void setGrade(String grade) {
72          this.grade = grade;
73      }
74
75      public int getHealth() {
76          return health;
77      }
78
79      public void setHealth(int health) {
80          this.health = health;
81      }
82
83      public int getLove() {
84          return love;
85      }
86
87      public void setLove(int love) {
88          this.love = love;
89      }
90
91      public void printPetInfo() {
92           System.out.println("我是"+this.name );
93      }
94  }
95
96  package com.lg.test;
97
98  public class Tiger extends Pet{
99      public static final char SEX_MALE = '男';
100     public static final char SEX_FEMALE = '女';
101     private  char sex = SEX_MALE;
102
103     public Tiger() {
104         this(0,0,"");
105     }
106
107     public Tiger(long id,long masterId,String name) {
108         this(id,masterId,name,100,0);
109     }
```

```
110
111     public Tiger(long id,long masterId,String name,int health,int love) {
112         this.id=id;
113         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 
175 package com.lg.test;
176 
177 public class Tortoise extends Pet{
178 
179     @Override
180     public void printPetInfo() {
181         System.out.println("I am "+this.name);
182     }
183 
184 }
185 
```

* 温馨提示

  * 抽象类不能创建对象，如果创建，编译无法通过而报错。只能创建其非抽象子类的对

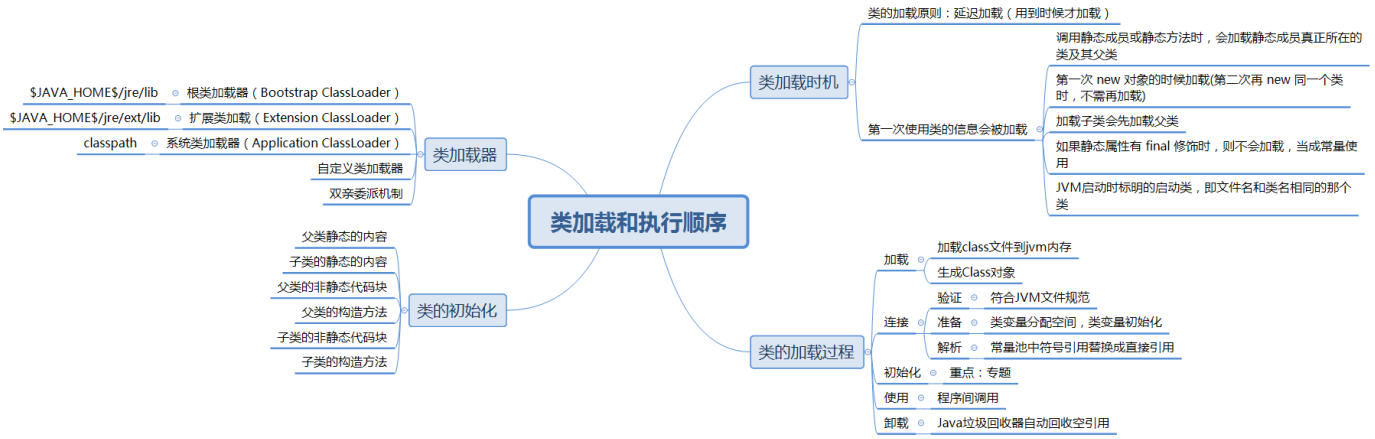
象。

* 抽象类中，可以有构造方法，是供子类创建对象时，初始化父类成员使用的。

* 抽象类中，不一定包含抽象方法，但是有抽象方法的类必定是抽象类。

* 抽象类的子类，必须重写抽象父类中所有的抽象方法，否则，编译无法通过而报错。除非该子类也是抽象类。

* 能够理解Java类加载和执行顺序



* 类加载时机

  * 类的加载原则：延迟加载（用到时候才加载）。

  * 类什么时候会被加载： 第一次使用类的信息会被加载

    * 调用静态成员或静态方法时，会加载静态成员或静态方法真正所在的类及其父类

    * 第一次 new 对象的时候加载(第二次再 new 同一个类时，不需再加载)

    * 加载子类会先加载父类

    * 如果静态属性有 final 修饰时，则不会加载，当成常量使用

    * JVM启动时标明的启动类，即文件名和类名相同的那个类

    * 反射：Class.forName(""):

    * 测试类的加载

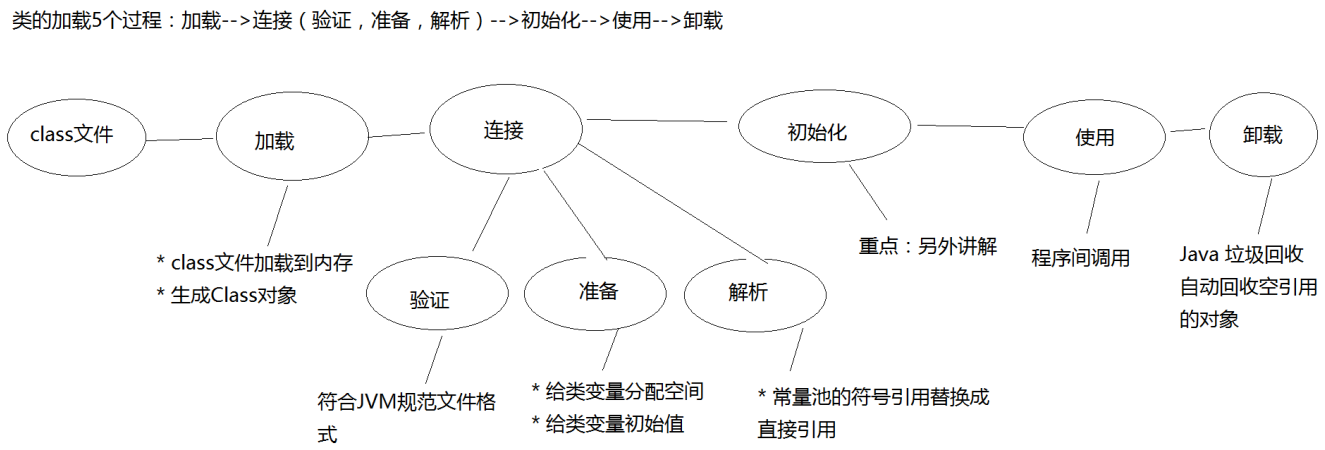
```
1 public class Parent {
2     int age;
3     static {
```

```
        System.out.println("加载了Parent...");
    }
}

public class Son extends Parent{
    static {
        System.out.println("加载了Son...");
    }
    public static void sayHello() {
            System.out.println("HelloWorld");
    }
}

public class Person {
    public static final char SEX_MALE='男';
    public static int age;
    static {
        System.out.println("加载Person了...");
    }
}

*验证：调用静态成员或静态方法时，会加载静态成员或静态方法真正所在的类及其父类
代码：System.out.println(Person.age);
结果：加载Person了...
     0
代码：Son.sayHello();
结果：加载了Parent...
    加载了Son...
    HelloWorld

* 验证：第一次 new 对象的时候加载(第二次再 new 同一个类时，不需再加载)
代码：Parent parent=new Parent();
    parent=new Parent();
结果：加载了Parent...

* 验证：加载子类会先加载父类
代码：Son son=new Son();
结果：加载了Parent...
     加载了Son...
```

```
*验证：如果静态属性有 final 修饰时，则不会加载，当成常量使用
代码： System.out.println(Person.SEX_MALE);
结果：男

* 验证：JVM启动时标明的启动类，即文件名和类名相同的那个类
 代码：
    public class Test1 {
    static {
        System.out.println("加载了Test1...");
    }
    public static void main(String[] args) {
    }
  }
  结果：加载了Test1...
```

* 类的加载过程

  * 加载-->连接（验证，准备，解析）-->初始化-->使用-->卸载



* 解析:符号引用解析为直接引用，因为对一些类和类的字段，方法的引用，在编译时不知道其具体的位置，所以会使用符号引用，在加载时再具体的解析

```
String s1="abc";
String s2=new String("abc");
String s3=new String("abc");
String s4=new String("abc").intern();
System.out.println(s1==s2);
```

```
System.out.println(s3==s4);
System.out.println(s2==s4);
System.out.println(s1==s4);
* 结果：
false
false
false
true
* 编译时候，产生4个符号引用 s1，s2，s3，s4和一个字面量“abc”,到连接解析的时候替换成直接
```

* 类的初始化

    * 静态：父类静态代码块-->子类静态代码块

    * 父类：父类非静态代码块-->父类的构造方法

    * 子类：子类非静态代码块-->子类的构造方法

```
public class Parent {
    static {
        System.out.println("1 父类静态的内容");
    }
    {
        System.out.println("3 父类的非静态代码块");
    }
    public Parent() {
        System.out.println("4 父类的构造方法");
    }
}

public class Son extends Parent{
    static {
        System.out.println("2 子类的静态的内容");
    }
    {
        System.out.println("5 子类的非静态代码块");
```

```
    }
    public Son() {
        System.out.println("6 子类的构造方法");
    }
}


public class Test1 {
    public static void main(String[] args) {
        Son son=new Son();
    }
}

* 结果：
1 父类静态的内容
2 子类的静态的内容
3 父类的非静态代码块
4 父类的构造方法
5 子类的非静态代码块
6 子类的构造方法

```

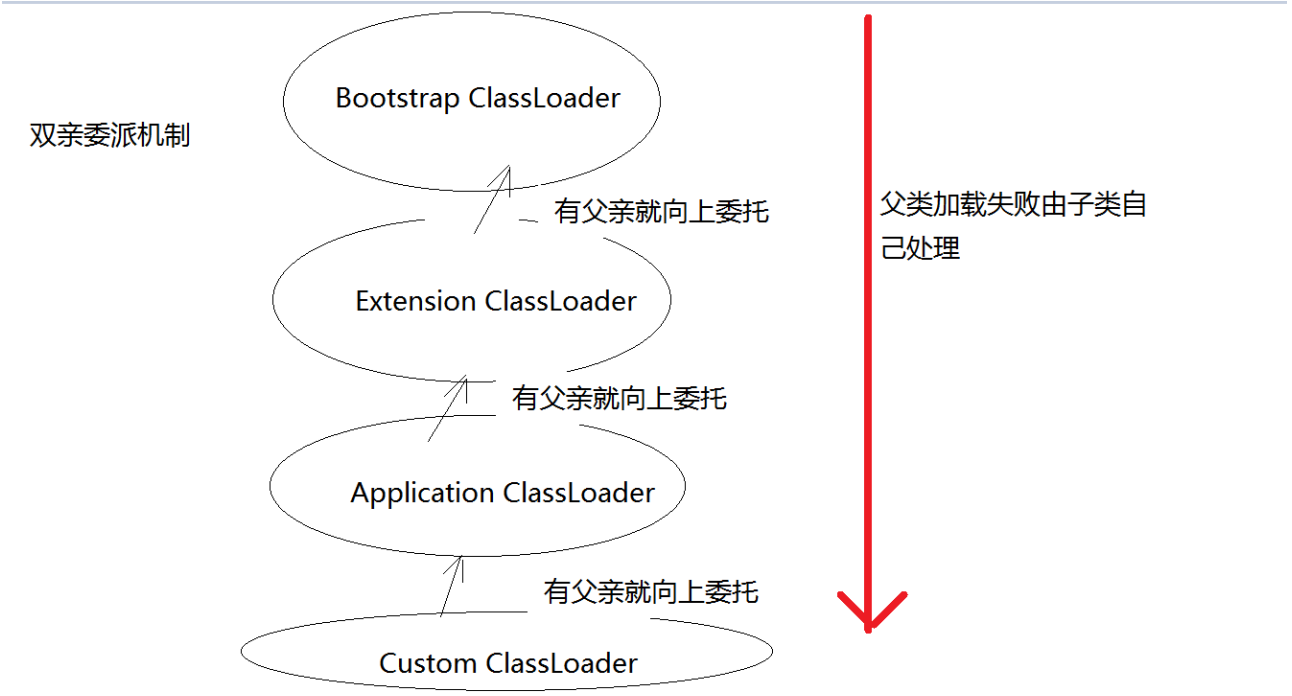
* 类的初始化顺序案例

```
public class A {
    D d;
    static {
        System.out.println("A1");
    }

    {
        System.out.println("A2");
        d=new D();
    }

    public A() {
        System.out.println("A3");
    }
}
```

```
16
17
18 public class B extends A{
19     static C c=new C();
20     static {
21         System.out.println("B1");
22     }
23
24     {
25         System.out.println("B2");
26     }
27
28     public B() {
29         System.out.println("B3");
30     }
31 }
32
33
34 public class C {
35     public C() {
36         System.out.println("C");
37     }
38 }
39
40 public class D extends C{
41     public D() {
42         System.out.println("D");
43     }
44 }
45
46 public class Test1 {
47     public static void main(String[] args) {
48         B b=new B();
49     }
50 }
51
52 结果:
53 A1
54 C
55 B1
```

```
A2
C
D
A3
B2
B3
```

* 类加载器（了解）（JVM）
  * 根类类加载器（Bootstrap ClassLoader）---%JAVA_HOME%/jre/lib
  * 扩展类加载器 (Extension ClassLoader) ----%JAVA_HOME%/jre/ext
  * 系统类加载器（Application ClassLoader）----ClassPath
  * 自定义类加载--->路径都不符合项目要求
  * 双亲委派机制



* 能够掌握四个修饰符（public，private，default，protected）的区别

访问权限	当前类	当前包	派生类(不在当前包中)	其他包
私有的private	可以	不	不	不

默认的(没有访问权限修饰符)	可以	可以	不	不
受保护的(protected)	可以	可以	可以	不
公共的public	可以	可以	可以	可以

* 演示

* 使用原则:遵循权限最小化原则（封装一个思想）

* 应用场景:

    * 如果想在任意位置都能使用到, 就用public 修饰

    * 如果想在当前类和派生类(子类)中直接使用, 用protected修饰

    * 如果想在当前包中使用, 不使用任何权限修饰符

    * 如果只想在当前类中使用, 用private修饰

* 能够理解面向对象的多态

* 生活中多态：

    * 跑的动作，小猫、小狗和大象，跑起来是不一样的。

    * 飞的动作，昆虫、鸟类和飞机，飞起来 是不一样的。

    * 美女笑的动作，古典美女，现代美女，笑起来是不一样的。

    * 温馨提示：同一行为，通过不同的事物，可以体现出来的不同的形态。多态，描述的就是这样的状态。

    * 业务需求：

    * 青龙法力不足，白虎也是法力不足

    * 主人

        * 给青龙输送法力（power）

            * 健康值低于50，给输送法力--->90

* 给白虎输送法力（power）

* 健康值低于50,，给输送法力--->80

* 给...输送法力

```
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 );
    }

    @Override
    public int getPower() {
        if(health<50) {
            power+=90;
            System.out.println(this.name+"获得法力...");
        }
        return power;
    }
}

package com.lg.test7;

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);
    }

    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;
    }

```

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

```
    }

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

    public char getSex() {
        return sex;
    }

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

    @Override
    public int getPower() {
        if(health<50) {
            power+=80;
            System.out.println(this.name+"获得法力...");
        }
        return power;
    }

}

package com.lg.test7;

public class Tortoise extends Pet{

    @Override
    public void printPetInfo() {
        System.out.println("I am "+this.name);
    }

    @Override
    public int getPower() {
        System.out.println(this.name+"获得法力...");
        return 0;
    }

```

```java
}

package com.lg.test7;

public class Master {
    private int id;
    private String name;
    public Master() {
    }
    public Master(int id, String name) {
        super();
        this.id = id;
        this.name = name;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    // 给青龙输送法力
//  public void givePower(Dragon dragon) {
//      dragon.getPower();
//  }
//
//  // 给白虎输送法力
//  public void givePower(Tiger tiger) {
//      tiger.getPower();
//  }
//  // 给玄武输送法力
//  public void givePower(Tortoise tortoise) {
//      tortoise.getPower();
//  }

```

```
    public void givePower(Pet pet) {
        pet.getPower();
    }

    // 给100宠物法力,上面的方法，要写100个
    // 思考：能不能通过一种技术：解决问题(只要一个方法)-->面向对象特征之一：多态
}

package com.lg.test7;

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
    protected int power;// 法力

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

    public abstract int getPower();
}

package com.lg.test7;

public class Test {
    public static void main(String[] args) {
        // 多态代码体现形式
        // 父类 引用=new 子类();
        // 父类 引用=new 子类();
        // 父类 引用=new 子类();
        // 青龙
        Pet dragon=new Dragon();
        dragon.name="小青龙";
        dragon.health=30;

        // 白虎
        Pet tiger=new Tiger();
        tiger.name="小白虎";
        tiger.health=45;
```

```
276
277         // 主人
278         Master master=new Master();
279         master.givePower(dragon);
280         master.givePower(tiger);
281     }
282 }
283
284 结果：
285 小青龙获得法力...
286 小白虎获得法力...
287
```

* 多态的定义：

  * 是指同一行为，具有多个不同表现形式

* 多态的作用：

  * 消除类型之间的耦合关系

    * Tiger，Dragon ---> Pet

* 实现多态有三个必要条件:

  * 继承

  * 重写

  * 向上转型

* 多态体现的格式：

  * 父类类型 变量名 = new 子类对象；

    变量名.方法名();

* 多态温馨提示：

  * 指向子类的父类引用由于向上转型了,它只能访问父类中拥有的方法和属性,而对于子类中存在而父类中不存在的方法,该引用是不能使用的。

* 用了多态之后

  * Parent p=new Son();

  * instatnceOf

```
Dragon:
    public void fly() {
        System.out.println("小青龙飞呀飞呀...");
    }
Tiger:
    public void run() {
        System.out.println("小白虎跑啊跑啊。。。");
    }
Master:
public void playWith(Pet pet) {
        if(pet instanceof Dragon) {
            //青龙--->fly
            Dragon dragon=(Dragon)pet;
            dragon.fly();
        }else if(pet instanceof Tiger) {
            //白虎--->run
            Tiger tiger=(Tiger)pet;
            tiger.run();
        }

    }
测试:
public static void main(String[] args) {
        Pet dragon=new Dragon();
        dragon.name="小青龙";
        dragon.health=30;
        Pet tiger=new Tiger();
        tiger.name="小白虎";
        tiger.health=45;
        Master master=new Master();
        master.playWith(dragon);
        master.playWith(tiger);
    }

结果:
小青龙飞呀飞呀...
小白虎跑啊跑啊。。。
```