内容提要

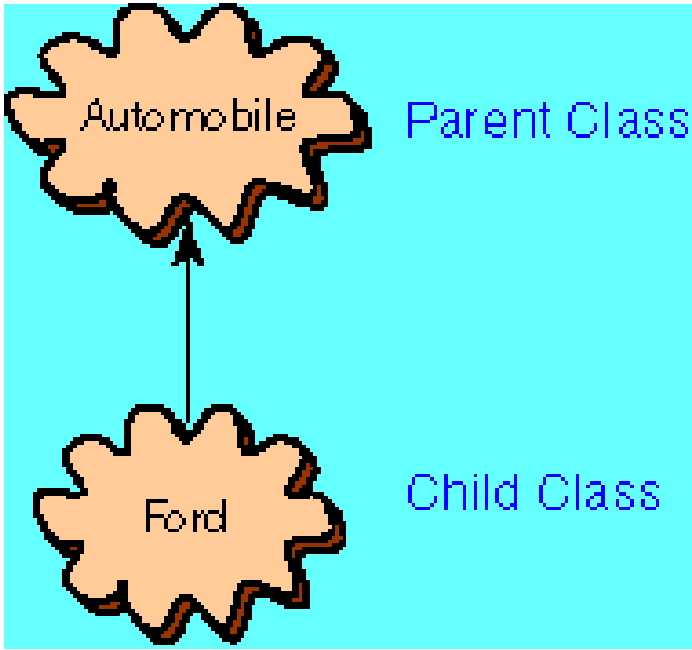
- 继承
- 终结类与终结方法
- 抽象类与抽象方法
- 接口
- 多态

类的继承

- 一种由已有的类创建新类的机制，是面向对象程序设计的基石之一
- 通过继承，可以根据已有类来定义新类，新类拥有已有类的所有功能
- Java只支持类的单继承，每个子类只能有一个直接父类
- 父类是所有子类的公共属性及方法的集合，子类则是父类的特殊化
- 继承机制可以提高程序的抽象程度，提高代码的可重用性

类的继承：概念

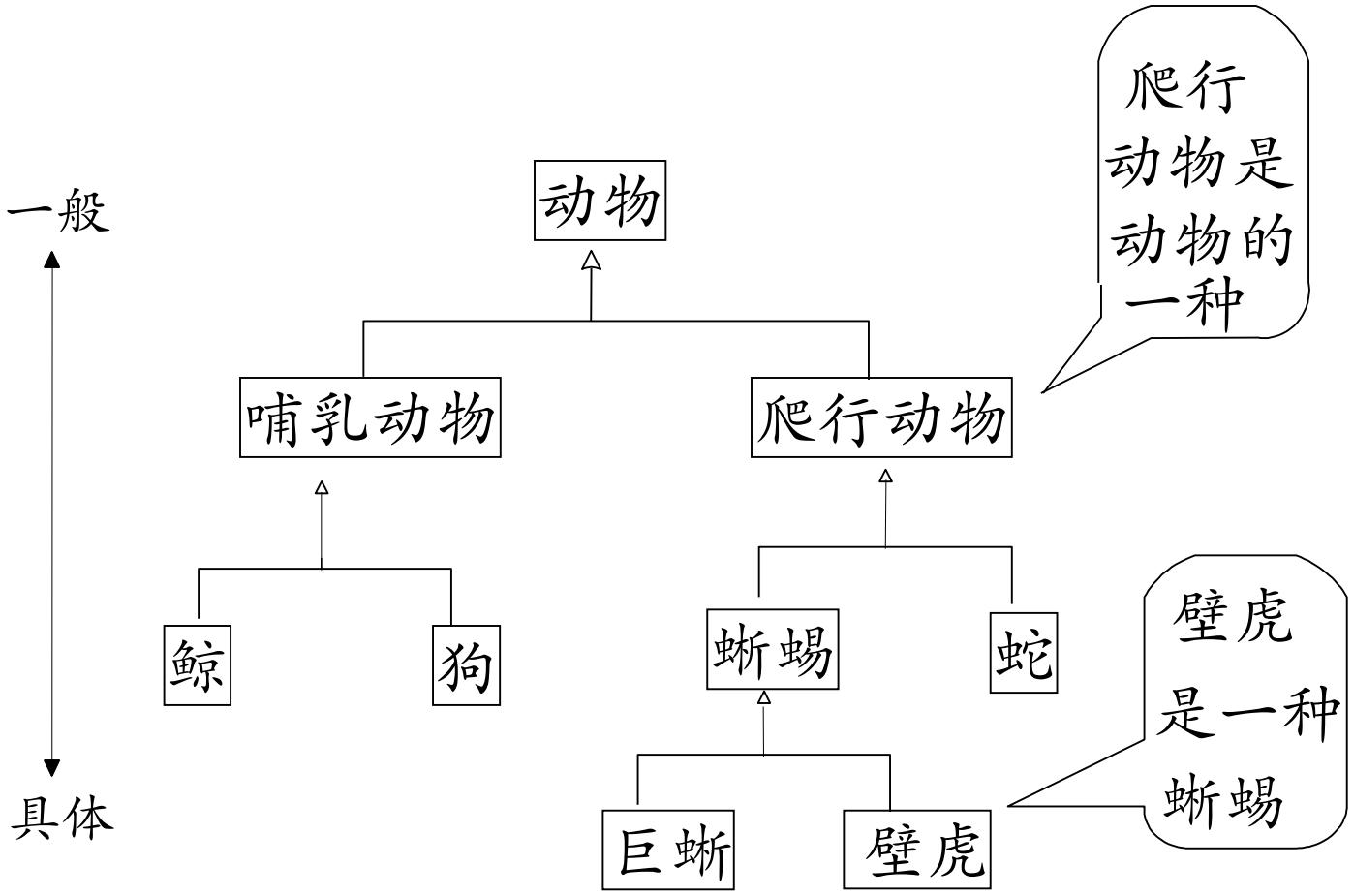
- 基类(base class)
  - 也称超类(superclass)
  - 是被直接或间接继承的类
- 派生类(derived-class)
  - 也称子类 (subclass)
  - 继承其他类而得到的类
  - 继承所有祖先的状态和行为
  - 派生类可以增加变量和方法
  - 派生类也可以覆盖(override)继承的方法



is_a（is a kind of）关系

类的继承：概念

- 动物类层次举例

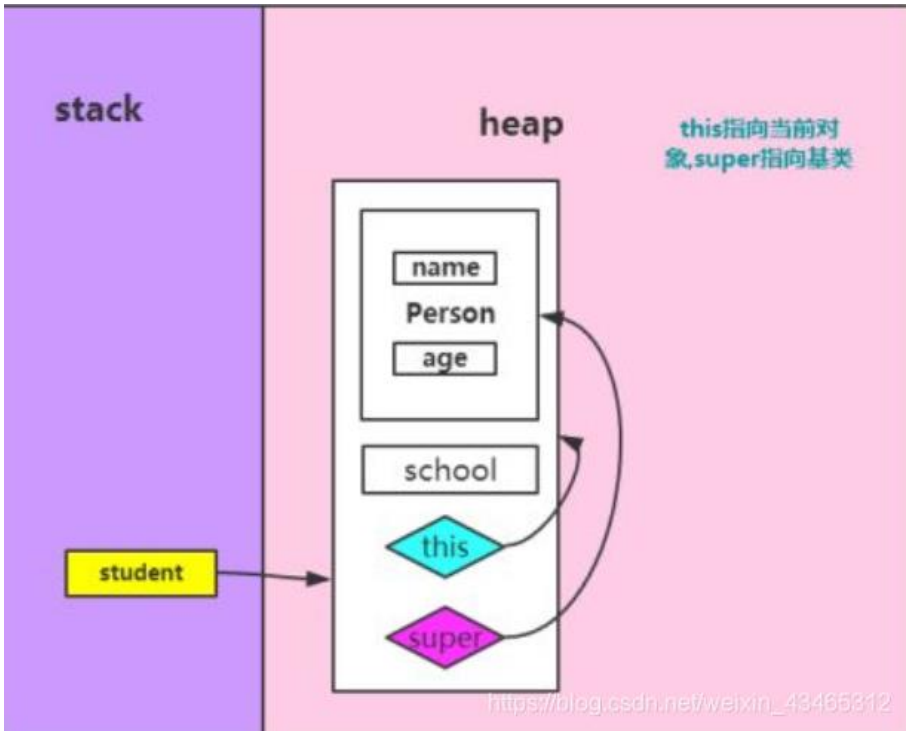


类的继承：概念

- 派生类产生的对象
  - 从外部来看，它应该包括
    - 与基类相同的接口
    - 可以具有更多的方法和数据成员
  - 其内包含着一个基类类型的子对象

父类为Person类

子类为Student类



类的继承：语法

- 继承语法

```
class childClass extends parentClass
{  //子类类体}
```

- 举例：在一个公司中，有普通员工及管理人员两类人员
  - 职员对象（Employees）可能有的属性信息包括
    - 员工号，姓名，地址，电话号码
  - 管理人员（Managers）除具有普通员工的属性外，还可能具有下面的属性
    - 职责，所管理的职员

类的继承：语法

```
//父类Employee
class Employee
{
    int  employeeNumbe ;
    String  name, address, phoneNumber ;
}

//子类Manager
class  Manager extends Employee
{
    //子类增加的数据成员
    String responsibilities, listOfEmployees;
}
```

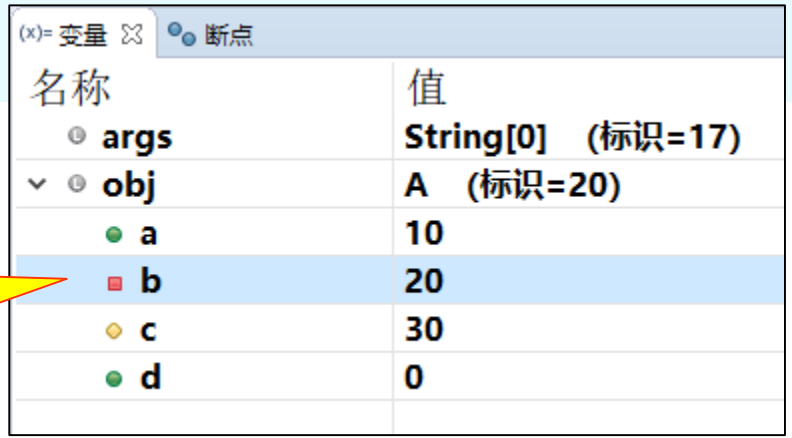
类的继承：语法

- 子类不能直接访问从父类中继承的私有属性及方法，但可使用公有（及保护）方法进行访问。例如：

```
public class B {
   public int a = 10;
   private int b = 20;
   protected int c = 30;
   public int getB()  {
       return b;
   }
}
```

```
public class A extends B {
    public int d;
    public void tryVariables() {
        System.out.println(a);          //允许
        // System.out.println(b);       //不允许
        System.out.println(getB());     //允许
        System.out.println(c);          //允许
    }
    public static void main(String[] args) {
        A  obj = new A();
    }
}
```

子类继承了私有属性，但是不允许直接访问。



类的继承：隐藏和覆盖

- 子类对从父类继承来的属性变量及方法可以重新定义

```
class Parent {
    float aNumber;
}
```

```
class Child extends Parent
{
    float aNumber;
}
```

- 属性的隐藏
  - 子类中声明了与父类中相同的成员变量名，则从父类继承的变量将被隐藏
  - 子类拥有了两个相同名字的变量，一个继承自父类，另一个由自己声明
  - 当子类执行继承自父类的操作时，处理的是继承自父类的变量，而当子类执行它自己声明的方法时，所操作的就是它自己声明的变量

类的继承：隐藏和覆盖

```
class Parent {
    float aNumber;
    public void p_display(){
        System.out.println(aNumber);
    }
}
```

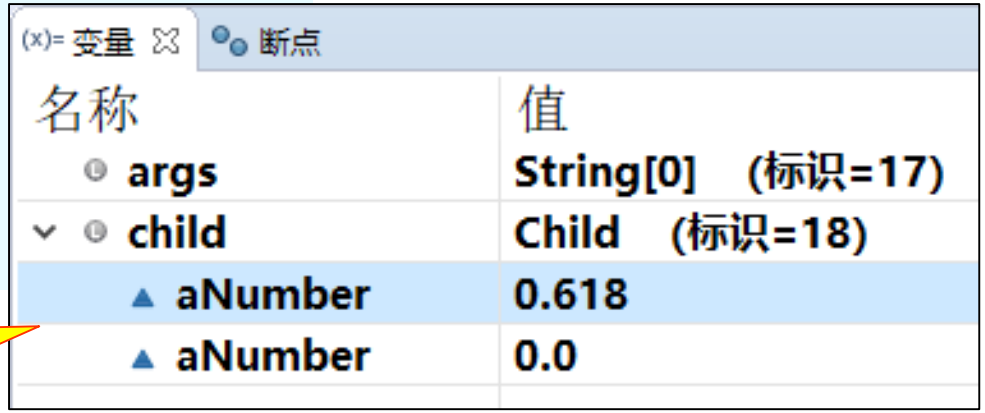
```
public class Child extends Parent{
    float aNumber=0.618F;
    public void c_display(){
        System.out.println(aNumber);
    }
    public static void main(String[] args){
        Child child = new Child();
        child.p_display();
        child.c_display();
    }
}
```

运行结果：

0.0

0.618

子类拥有两个同名变量

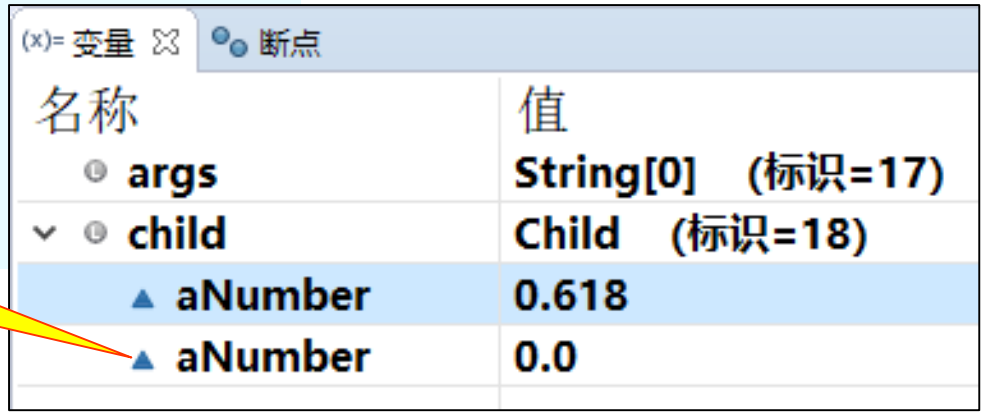


类的继承：隐藏和覆盖

- 如何访问被隐藏的父类属性
  - 调用从父类继承的方法，则操作的是从父类继承的属性
  - 使用super.属性（注意：子类对象无法访问被隐藏的属性）

```
public class Child extends Parent{
   float aNumber=0.618F;
   public void c_display(){
      System.out.println(aNumber);
      System.out.println(super.aNumber);
   }
   public static void main(String[] args){
      Child child = new Child();
      child.p_display();
      child.c_display();
   }
}
```

super.aNumber



类的继承：隐藏和覆盖

【例】属性的隐藏举例

```
class A1{
   int x = 2;
    public void setx(int i)  {  x = i; }
    void printa(){
        System.out.println(x);
    }
}
```

```
class B1 extends A1{
   int x=100;
   void printb() {
      super.x = super.x +10 ;
      System.out.println("super.x="
         +  super.x + "  x= " + x);
   }
}
```

```
public class Exam4_4Test {
  public static void main(String[] args){
      A1 a1 = new A1();
      a1.setx(4);
      a1.printa();

      B1 b1 = new B1();
      b1.printb();
      b1.printa();

      b1.setx(6);  // 将继承的x值设置为6
      b1.printb();
      b1.printa();
      a1.printa();
   }
}
```

x= 2

x= 4

super.x= 2
x=100

super.x=12
x=100

super.x=6
x=100

super.x=16
x=100

运行结果：

```
4
super.x= 12  x= 100
12
super.x= 16  x= 100
16
4
```

注意，对象a1与b1各自占有独立内存空间，互不影响

类的继承：隐藏和覆盖

- 父类静态属性将由父类对象和子类对象共享

```
class A1{
   static int x = 2;
    public void setx(int i)
    {   x = i;    }
    void printa()
    { System.out.println(x); }
}
class B1 extends A1{
   int x=100;
   void printb() {
      super.x = super.x +10 ;
      System.out.println("super.x="
         +  super.x + "  x= " + x);
    }
}
```

A1.x= 2

```
public class Exam4_4Test {
  public static void main(String[] args){
      A1 a1 = new A1();
       a1.setx(4);
       a1.printa();

       B1 b1 = new B1();
       b1.printb();
       b1.printa();
       b1.setx(6);  // 将共享属性x值设置为6
       b1.printb();
       b1.printa();
       a1.printa();
   }
}
```

A1.x= 4

A1.x= 4
x=100

A1.x=14
x=100

A1.x=6
x=100

A1.x=16
x=100

运行结果:

```
4
super.x= 14  x= 100
14
super.x= 16  x= 100
16
16
```

类的继承：隐藏和覆盖

- 方法覆盖
  - 如果子类不需使用从父类继承来的方法的功能，则可以声明自己的同名方法，称为方法覆盖（Override）
  - 覆盖方法的返回类型，方法名称，参数的个数及类型必须和被覆盖的方法一模一样
  - 只需在方法名前面使用不同的类名或不同类的对象名即可区分覆盖方法和被覆盖方法
  - 覆盖方法的访问权限可以比被覆盖的宽松，但是不能更为严格

类的继承：隐藏和覆盖

- 方法覆盖的应用场合
  - 派生类必须覆盖基类中的抽象的方法，否则派生类自身也成为抽象类
  - 子类中实现与父类相同的功能，但采用不同的算法或公式
  - 在名字相同的方法中，要做比父类更多的事情
  - 在子类中需要“取消”从父类继承的方法（让其失效）
- 子类中调用被覆盖的方法（注意：子类对象无法调用它）

  super.overriddenMethodName();
- 不能覆盖（重写）的方法
  - 声明为 final 的终结方法，例如Collections.emptyList( )
  - 声明为 static 的静态方法，例如Math.sin( )等方法。

补充：关于final方法

- 使用final方法的两个原因

① 为方法“上锁”，防止任何继承类改变它的本来含义。若希望方法的行为在继承期间保持不变（即不被子类覆盖或改写），就可将其设计为final方法。

② 提高程序执行的效率。方法体积较小时，编译器会将一个final方法变成内联方法，即在编译时它会用方法主体内实际代码的一个副本来替换方法调用。

- 通常，只有在方法的代码量非常少，或者想明确禁止方法被覆盖的时候，才应考虑将一个方法设为final。

类的继承：子类的构造方法

- 有继承时的构造方法遵循以下的原则
  - 子类不能从父类继承构造方法，只可以通过super关键字调用父类的构造方法。
  - 好的程序设计方法是在子类的构造方法中调用某一个父类构造方法，调用语句必须出现在子类构造方法的第一行，可使用super关键字
  - 如果子类没有声明构造方法或构造方法的声明中没有明确调用父类构造方法，则系统在执行子类的构造方法时会自动调用父类的无参的构造方法，相当于super( )。

继承原则：

- 继承父类的成员属性，包括实例变量和类变量。
- 继承除构造方法和私有方法之外的所有方法，包括实例方法和类方法。

类的继承：子类的构造方法

【例】父类子类（人—职工）的构造方法：

```
public class Person {
    protected String name, phoneNumber, address;
    public Person() {
        //this("", "", ""); //构造方法调用必须是构造函数中的第一个语句
      System.out.println("Initialize members of Person in Person ().");
    }
    public Person(String aName, String aPhoneNumber, String anAddress){
        name = aName;
        phoneNumber = aPhoneNumber;
        address = anAddress;
        __.println("Initialize members of Person in Person (…) .");
    }
}
```

类的继承：子类的构造方法

【例】父类子类（人—职工）的构造方法：

```
public class Employee extends Person {
    protected int employeeNumber;
    protected String workPhoneNumber;
    public Employee() {
      // 此处隐含调用构造方法 Person()，也可以显示调用: super();
      _.println("Initialize new members of Employee in Employee().");
    }
    public Employee(String aName, String aPhoneNumber, String
    anAddress, int aNumber, String aWorkPhoneNumber) {
        super(aName, aPhoneNumber, anAddress); // 调用父类构造方法
        employeeNumber = aNumber;
        workPhoneNumber = aPhoneNumber;
    _.println("Initialize new members of Employee in Employee(…).");
    }
}
```

类的继承：子类的构造方法

【例】父类子类（人—职工—专家）的构造方法：

```
Employee e1=new Employee("Tom","7391234","Qingdao",1001,"");
Employee e2=new Employee();
```

运行结果：

```
Initialize members of Person in Person (...).
Initialize new members of Employee in Employee(...).
Initialize members of Person in Person ( ).
Initialize new members of Employee in Employee( )
```

如果修改Employee的无参构造方法为：

```
public Employee() {
    this("", "", "",0, "");//调用有参构造方法Employee(...)
  _.println("Initialize new members of Employee in Employee().");
}
```

运行结果：

```
Initialize members of Person in Person ( ).
Initialize new members of Employee in Employee(...)
Initialize new members of Employee in Employee( )
```

终结类与终结方法

- 被final修饰符修饰的类和方法
- 终结类不能被继承
- 终结方法不能被当前类的子类重写
- 终结类存在的理由
  - 安全: 黑客用来搅乱系统的一个手法是建立一个类的派生类，然后用他们的类代替原来的类
  - 设计: 你认为你的类是最好的或从概念上该类不应该有任何派生类
  - 例如： **public final class System**　　**public final class Math**
    public final class String　　**public final class Integer**
- 终结方法存在的理由
  - 对于一些比较重要且不希望子类进行更改的方法，可以声明为终结方法。可防止子类对父类关键方法的错误重写，增加了代码的安全性和正确性。
  - 例如： 根类Object中的 public final Class<?> **getClass**()
    Collections类中的 public static final <T> List<T> **emptyList**()

终结类与终结方法

- 终结类的例子

```
final class A {  . . .  }

class B extends A{  . . .  }
```

- 终结方法的例子

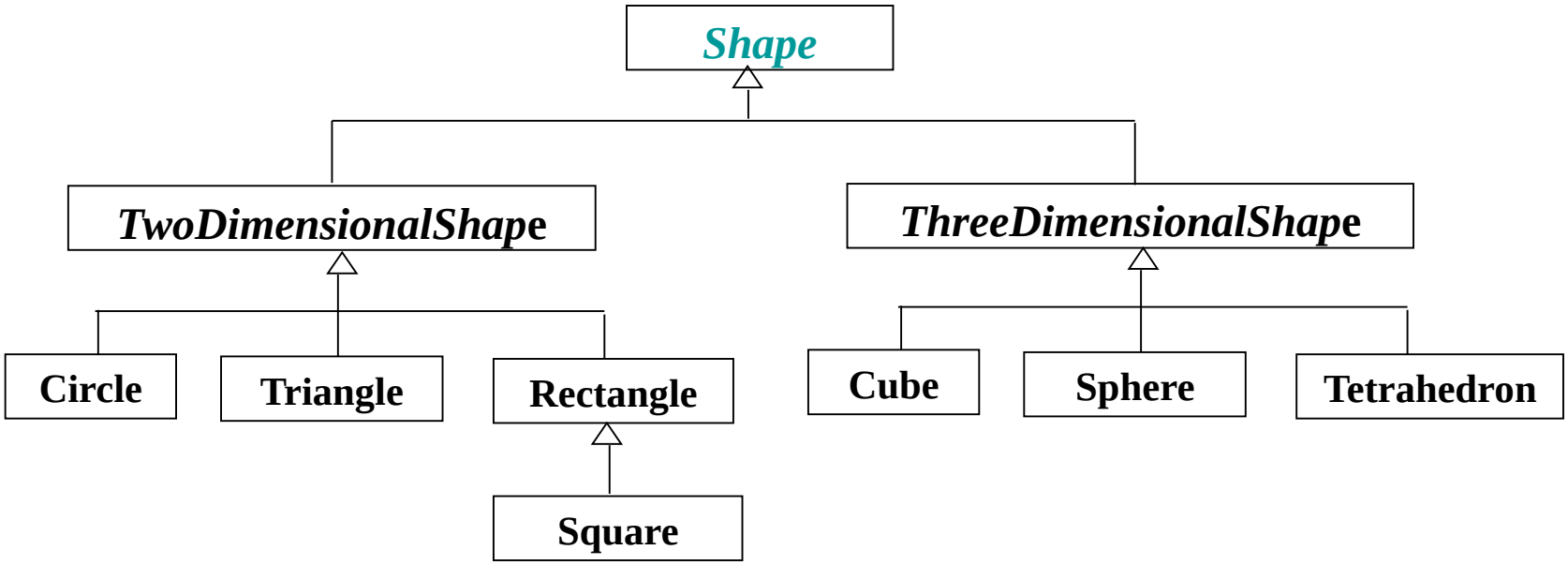
```
class Parent{
   public Parent() {  }
   final int getPI() { return Math.PI; } //终结方法
}
class Child extends Parent{
   public Child() {  }
   @Override
   int getPI() { return 3.14; } //重写父类中的终结方法，不允许
}
```

抽象类与抽象方法

- 抽象类
  - 代表一个抽象概念的类
  - 没有具体实例对象的类，不能使用new方法进行实例化
  - 类前需加修饰符abstract
  - 可包含常规类能够包含的任何东西，例如构造方法，非抽象方法
  - 也可包含抽象方法（仅有方法的声明，而没有实现）
- 抽象类存在意义
  - 抽象类是类层次中较高层次的概括，抽象类的作用是让其他类来继承它的抽象化的特征
  - 抽象类中可以包括被它的所有子类共享的公共属性和公共行为
  - 在用户生成实例时强迫用户生成更具体的实例，保证代码的安全性

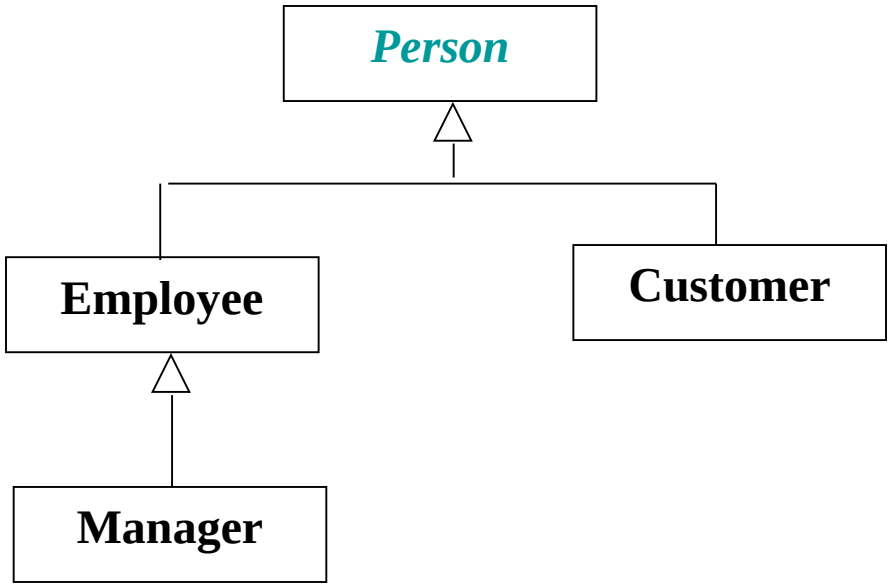
抽象类与抽象方法

- 几何形状的例子
  - 将所有图形的公共属性及方法抽象到抽象类Shape。再将2D及3D对象的特性分别抽取出来，形成两个抽象类TwoDimensionalShape及ThreeDimensionalShape



抽象类与抽象方法

- 人员的例子
  - 如果在应用系统中涉及到的人员只包括Customers、Employees 及 Managers，则Person 类的子类对象涵盖了应用中的对象，因此可以将Person 类声明为抽象类



抽象类与抽象方法

- 抽象类声明的语法形式为

```
abstract class Shape {
    属性;
    构造方法;
     一般方法;
     抽象方法;
}
```

  - 如果写：

```
new Shape();
```

抽象类与抽象方法

- 抽象方法
  - 抽象类中才可以声明抽象方法，换言之，含有抽象方法的类必须为抽象类。
  - 声明的语法形式为

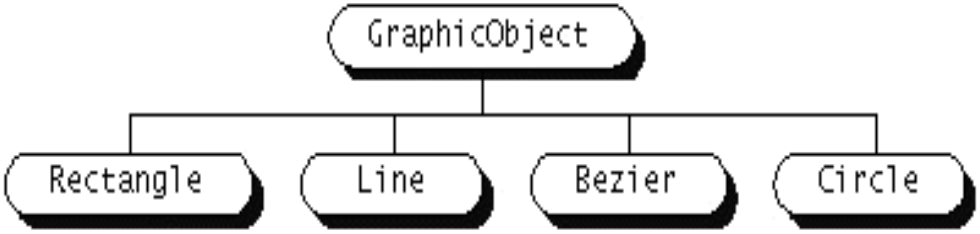
    public abstract <returnType> <methodName>(...);
  - 仅有方法头，而没有方法体和操作实现
  - 具体实现由当前类的不同子类在它们各自的类声明中完成

抽象类与抽象方法

- 抽象方法
  - 需注意的问题
    - 一个抽象类的子类如果不是抽象类，则它必须为父类中的所有抽象方法书写方法体，即重写父类中的所有抽象方法；否则子类也必须是抽象类。
    - 只有抽象类才能具有抽象方法，即如果一个类中含有抽象方法，则必须将这个类声明为抽象类
  - 抽象方法的优点
    - 隐藏具体的细节信息，所有的子类使用的都是相同的方法头，其中包含了调用该方法时需要了解的全部信息
    - 强迫子类完成指定的行为（动作），规定其子类需要用到的“标准”行为（动作）

抽象类与抽象方法

- 一个绘图的例子



– 各种图形都需要实现绘图方法，可以在它们的抽象父类中声明一个draw抽象方法

```
abstract class GraphicObject {
    int x, y;
    void moveTo(int newX, int newY) { . . . }
    abstract void draw();
}
```

– 各派生的图形子类都实现抽象的绘图方法draw，例如：

```
class Circle extends GraphicObject {
    void draw() { . . . }
}
```

类的组合

- 面向对象编程的一个重要思想就是用软件对象来模仿现实世界的对象
  - 现实世界中，大多数对象由更小的对象组成
  - 与现实世界的对象一样，软件中的对象也常常是由更小的对象组成
- Java的类中可以有其他类的对象作为成员，这便是类的组合
- 组合与继承的比较
  - “包含”关系用组合来表达，可以用“has a”描述
  - “属于”关系用继承来表达，可以用“is a”描述
  - 许多时候都要求将组合与继承两种技术结合起来使用，创建一个更复杂的类

类的组合

- 例子：一条线段包含两个端点

```
class Point {   //点类
   private int x, y;  //coordinate
   public Point(int x, int y) { this.x = x; this.y = y;}
   public int GetX()  {  return x; }
   public int GetY()  {  return y; }
}

class Line {  //线段类
   private Point  p1, p2;     // 两端点
   public Line(Point a, Point b) {
      p1 = new Point(a.GetX(), a.GetY());
      p2 = new Point(b.GetX(), b.GetY());
   }
    public double Length() {
      return Math.sqrt(Math.pow(p2.GetX() - p1.GetX(), 2)
                       + Math.pow(p2.GetY() - p1.GetY(), 2));
    }
}
```

- 组合类的构造方法该如何编写？
- 深拷贝 or 浅拷贝？

习 题

1. 设计图形类Shape及其子类：

① Shape类具有表示图形中心坐标的受保护的centerX和centerY属性，且规定其子类要具有计算并返回b图形面积的方法area。

② 其子类包括圆类Circle和矩形类Rectangle。

③ 为这些类设计恰当的属性、构造方法和set/get方法。

④ 对你设计的类进行测试，其中包括创建若干个圆和矩形对象，然后计算它们的总面积。（提示：创建Shape类型的数组s，数组元素可以引用Shape类的子类对象，例如s[0]=new Circle(5.0)或s[0]=new Circle(10.0, 20.0, 5.0)，计算面积则调用s[0].area( )）

谢谢大家！