概述 & 封装隐藏

面向对象程序设计

第 5 讲 Java OOP- 继承 & 多态

刘进

2230652597@qq.com

OOP 教辅 2022 秋季 QQ 群：305915615

群名称：OOP教辅2022秋季
群　号：305915615

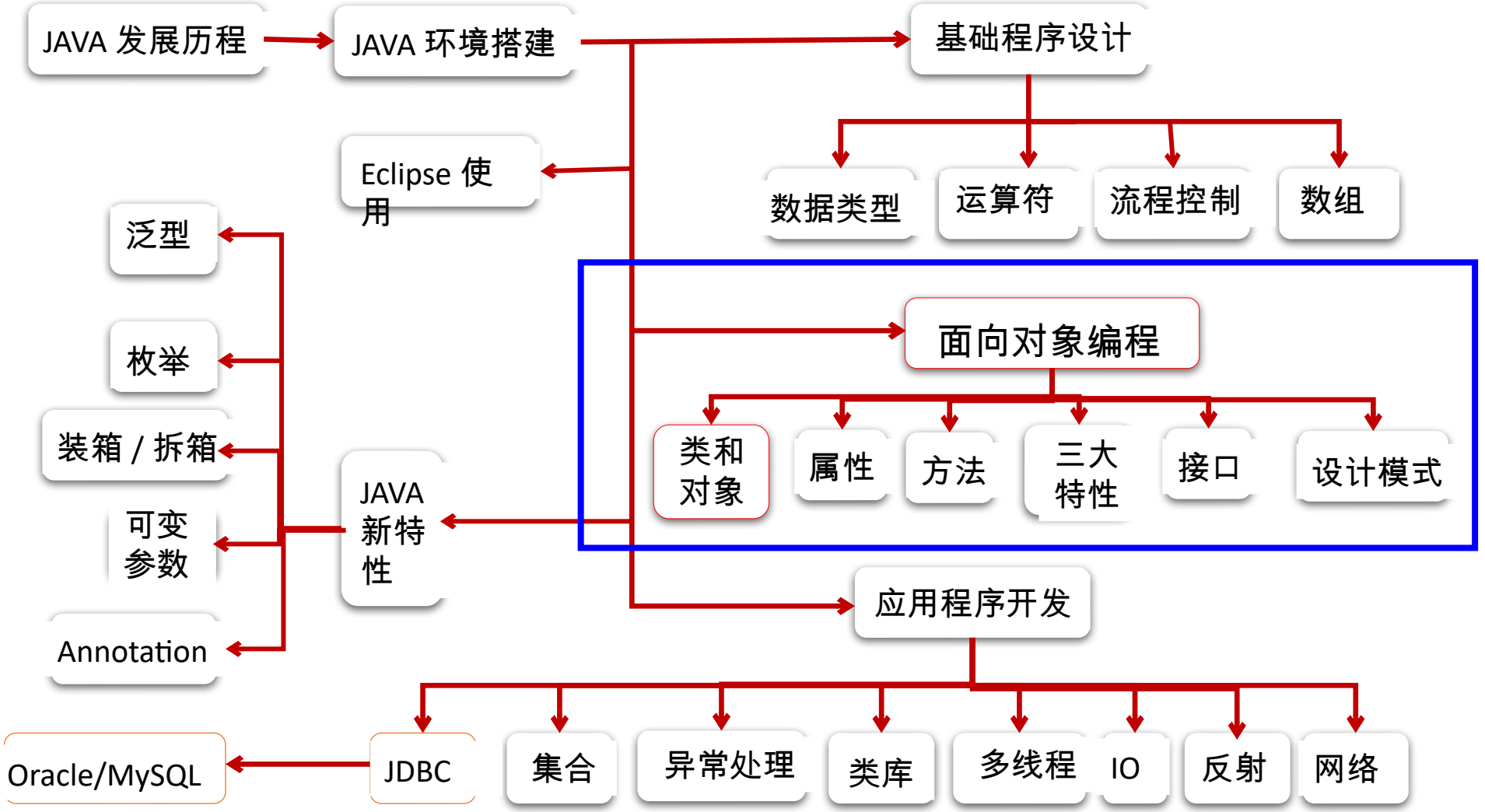
第 5 讲 Java 面向对象编程 -2

继承

5.1 面向对象特征之继承

5.2 面向对象特征之多态

Java 基础知识图解


JAVA 发展历程
JAVA 环境搭建
基础程序设计
Eclipse 使用
数据类型
运算符
流程控制
数组
泛型
枚举
装箱 / 拆箱
可变参数
Annotation
JAVA 新特性
面向对象编程
类和对象
属性
方法
三大特性
接口
设计模式
应用程序开发
Oracle/MySQL
JDBC
集合
异常处理
类库
多线程
IO
反射
网络

本章内容

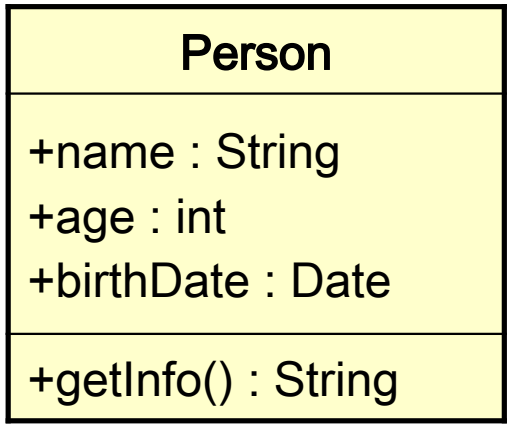
5.1 **面向对象特征之继承**
5.2 方法的重写 (override)
5.3 四种访问权限修饰符
5.4 关键字 super
5.5 子类对象实例化过程
5.6 **面向对象特征之多态**
5.7 Object 类、包装类

重点
继承 & 多态

5.1 面向对象特征之二：继承

- 为描述和处理个人信息，定义类 Person:

抽象

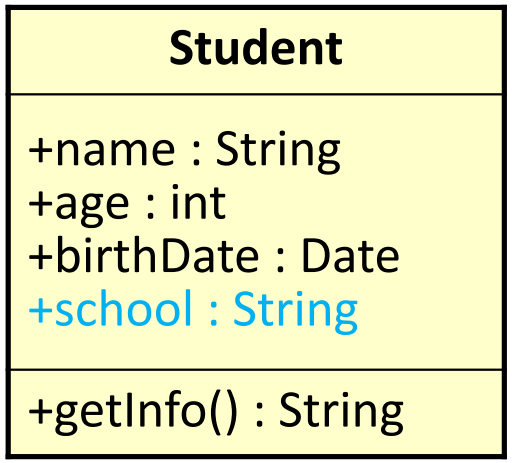



```
public class Person {
    public String name;
    public int age;
    public Date birthDate;

    public String getInfo()
    {...}
}
```

继 承 (1)

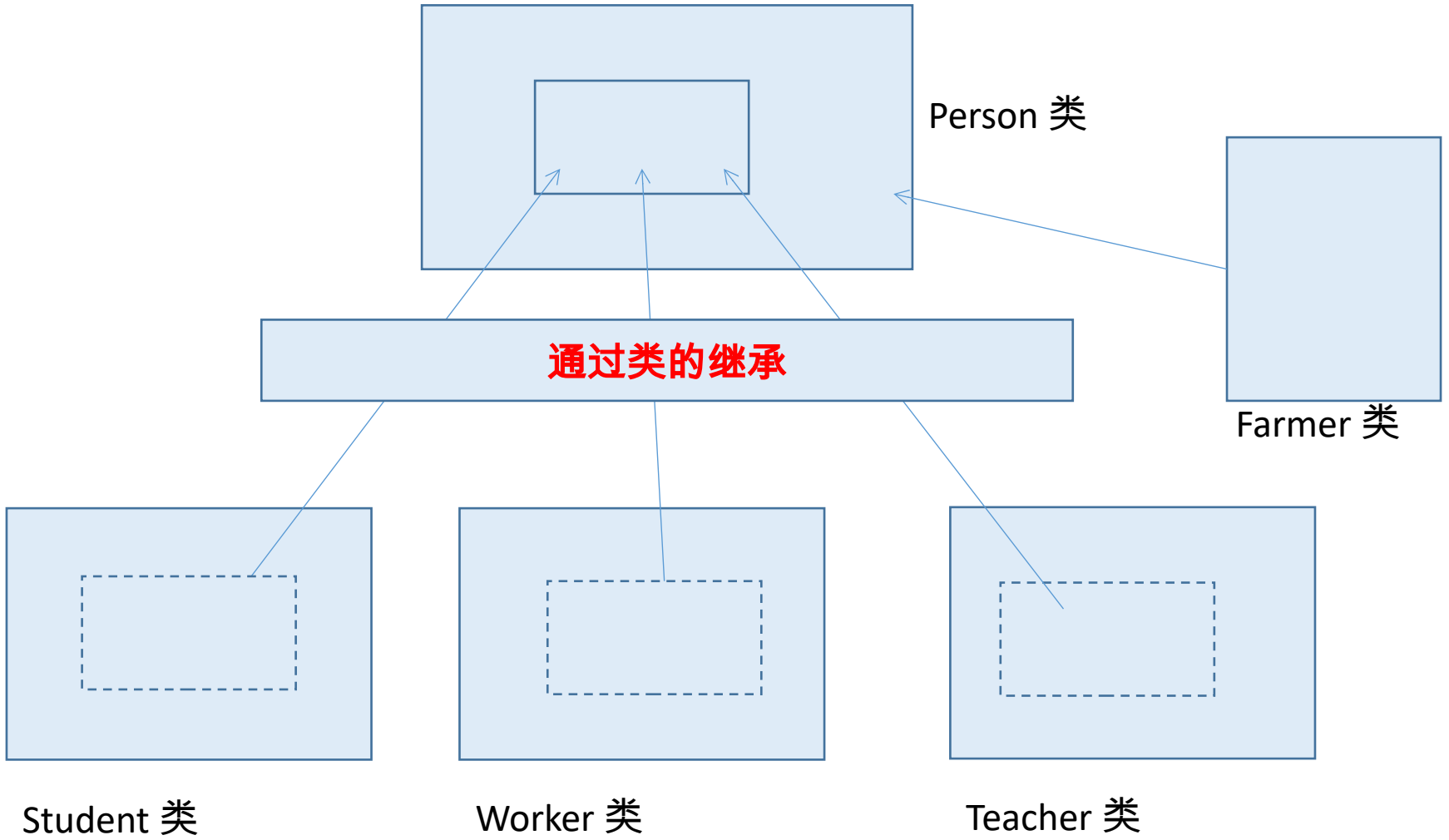
- 为描述和处理学生信息，定义类 Student:

更具体

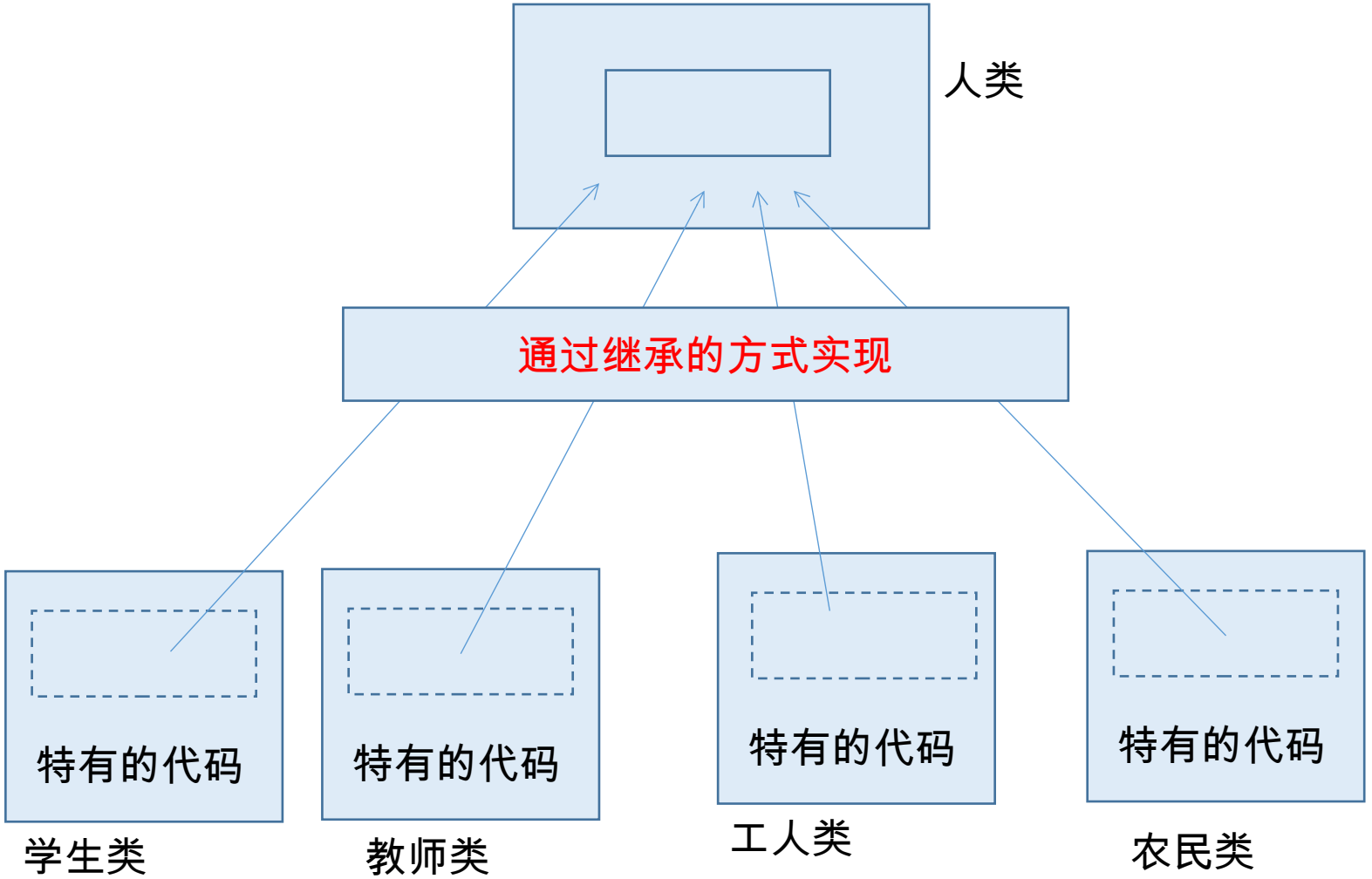



```
public class Student {
    public String name;
    public int age;
    public Date birthDate;
    public String school;

    public String getInfo()
   {...}
}
```

由抽象到具体



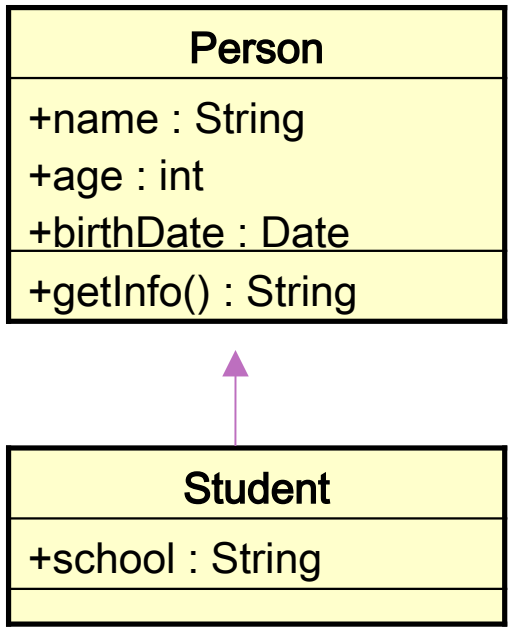
由抽象到具体



继 承 (2)

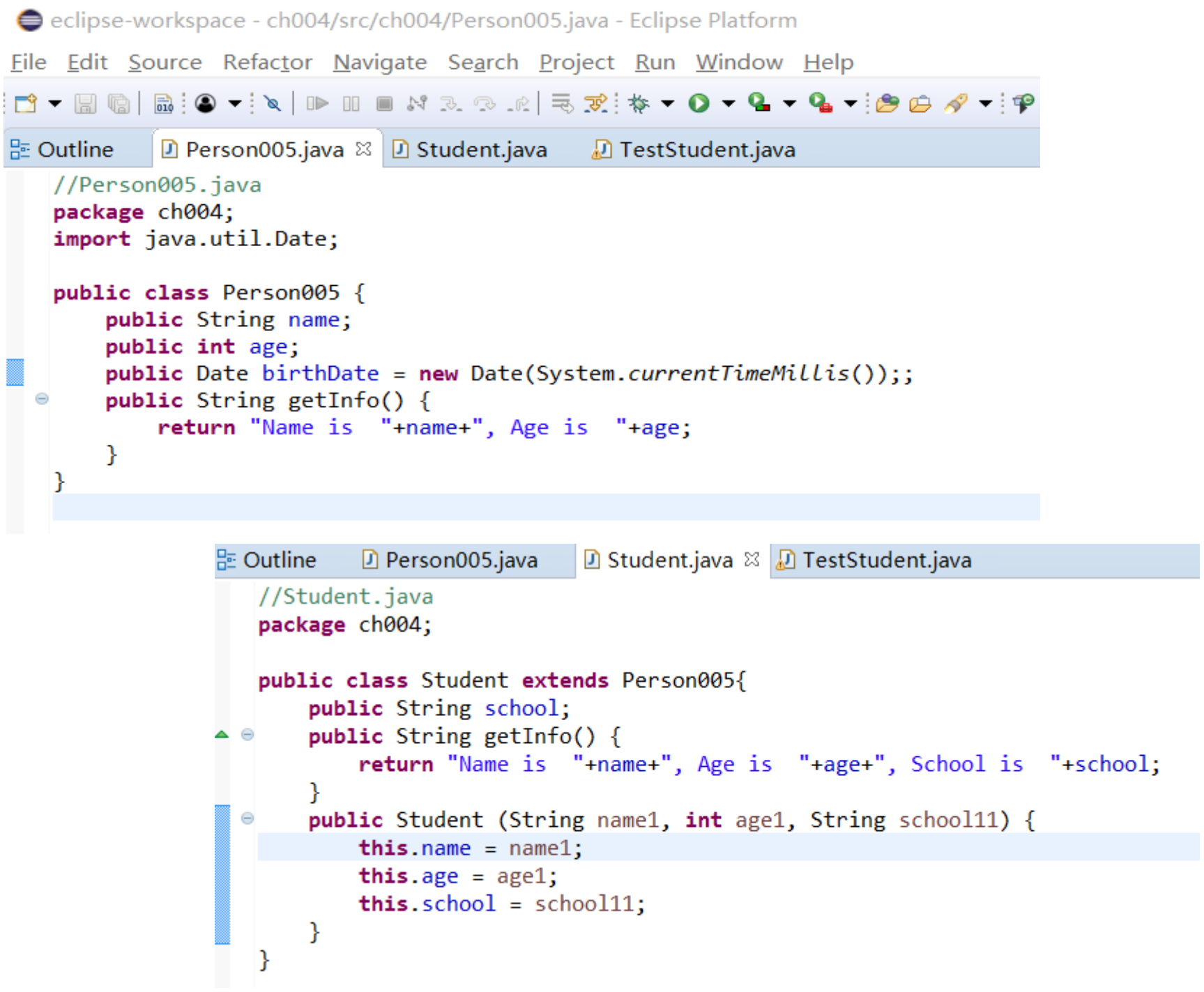
由抽象到具体

- 通过继承，简化 Student 类的定义：



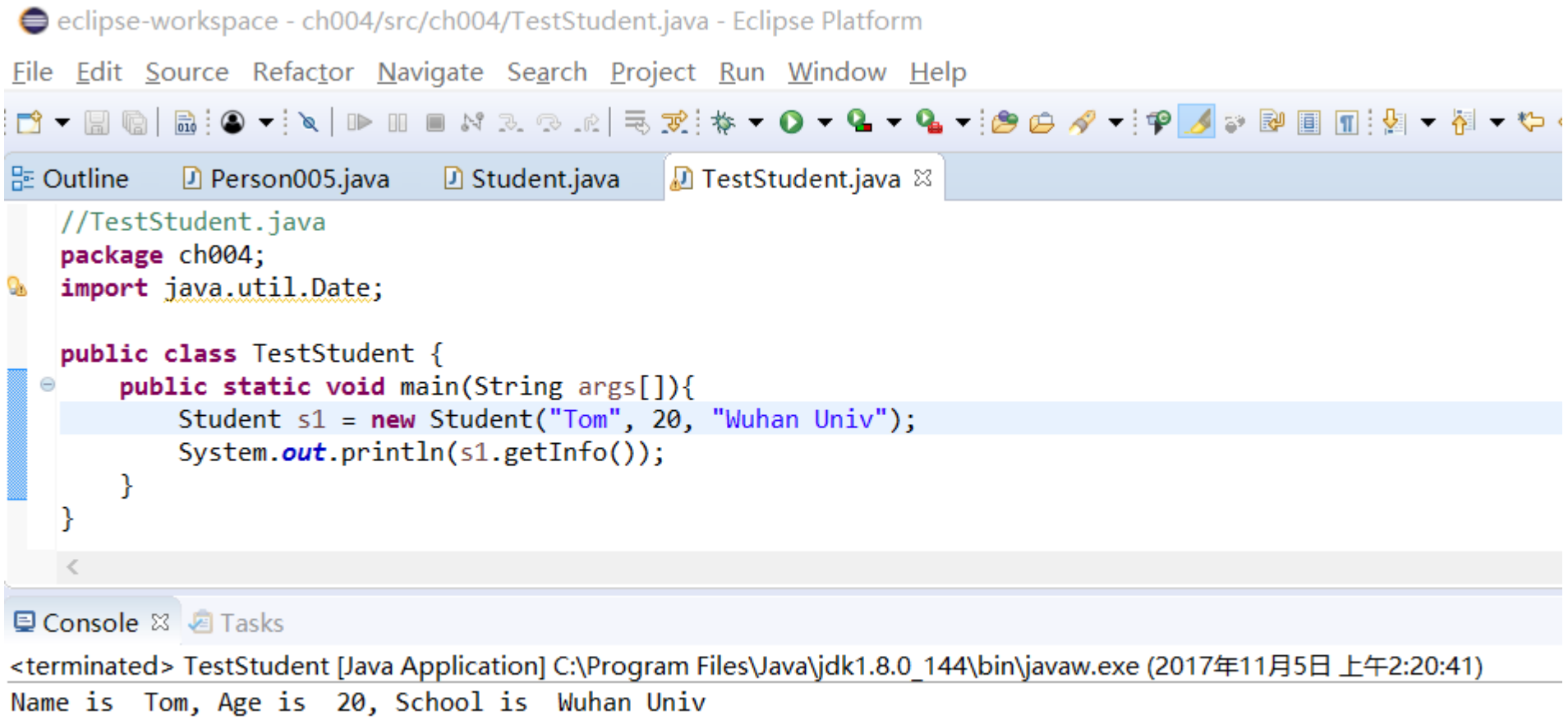

```
public class Person {
    public String name;
    public int age;
    public Date birthDate;
    public String getInfo() {...}
}

public class Student extends Person{
    public String school;
}
```

//Student 类继承了父类 Person 的所有属性和方法，并增加了一个属性 school 。 Person 中的属性和方法 ,Student 都可以利用。


eclipse-workspace - ch004/src/ch004/Person005.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Outline
Person005.java
Student.java
TestStudent.java
//Person005.java
package ch004;
import java.util.Date;
public class Person005 {
public String name;
public int age;
public Date birthDate = new Date(System.currentTimeMillis());;
public String getInfo() {
return "Name is "+name+", Age is "+age;
}
}
Outline
Person005.java
Student.java
TestStudent.java
//Student.java
package ch004;
public class Student extends Person005{
public String school;
public String getInfo() {
return "Name is "+name+", Age is "+age+", School is "+school;
}
public Student (String name1, int age1, String school11) {
this.name = name1;
this.age = age1;
this.school = school11;
}
}

由抽象
到具体

由抽象
到具体


eclipse-workspace - ch004/src/ch004/TestStudent.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Outline
Person005.java
Student.java
TestStudent.java
//TestStudent.java
package ch004;
import java.util.Date;
public class TestStudent {
public static void main(String args[]){
Student s1 = new Student("Tom", 20, "Wuhan Univ");
System.out.println(s1.getInfo());
}
}
Console
Tasks
<terminated> TestStudent [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (2017年11月5日 上午2:20:41)
Name is Tom, Age is 20, School is Wuhan Univ

继 承 (3)

由抽象到具体

- **为什么要有继承？**
  - 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，只要继承那个类即可。

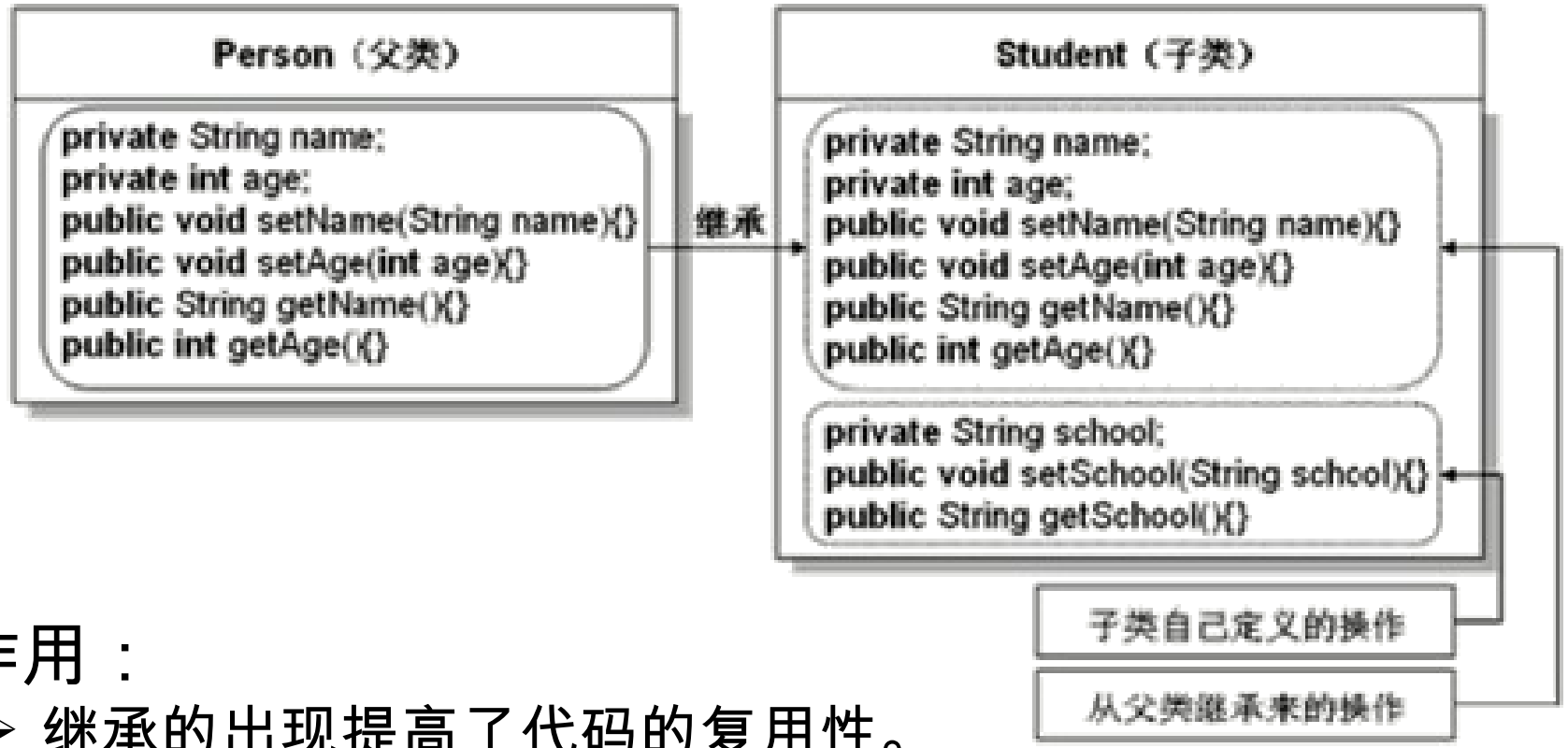
- 此处的多个类称为**子类**，单独的这个类称为父类（基类或超类）。可以理解为：“子类 is a 父类”

- 类继承语法规则：

  class Subclass extends Superclass{ }

继 承 (4)

由抽象到具体



- 作用：
  - 继承的出现提高了代码的复用性。
  - 继承的出现让类与类之间产生了关系，提供了多态的前提。
  - 不要仅为了获取其他类中某个功能而去继承

类的继承（5)

由抽象到具体

- 子类继承了父类，就继承了父类的方法和属性。
- 在子类中，可以使用父类中定义的方法和属性，也可以创建新的数据和方法。
- 在 Java 中，继承的关键字用的是“ extends ”，即子类不是父类的子集，而是对父类的“扩展”。

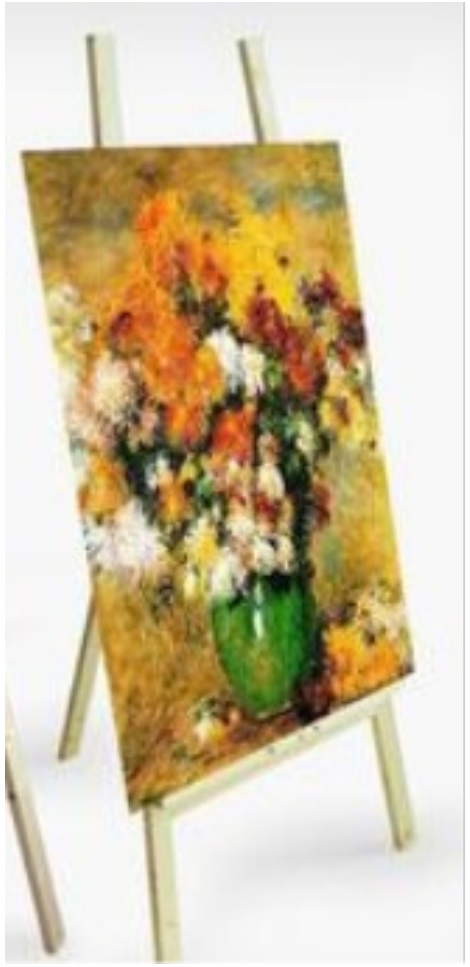
关于继承的规则：

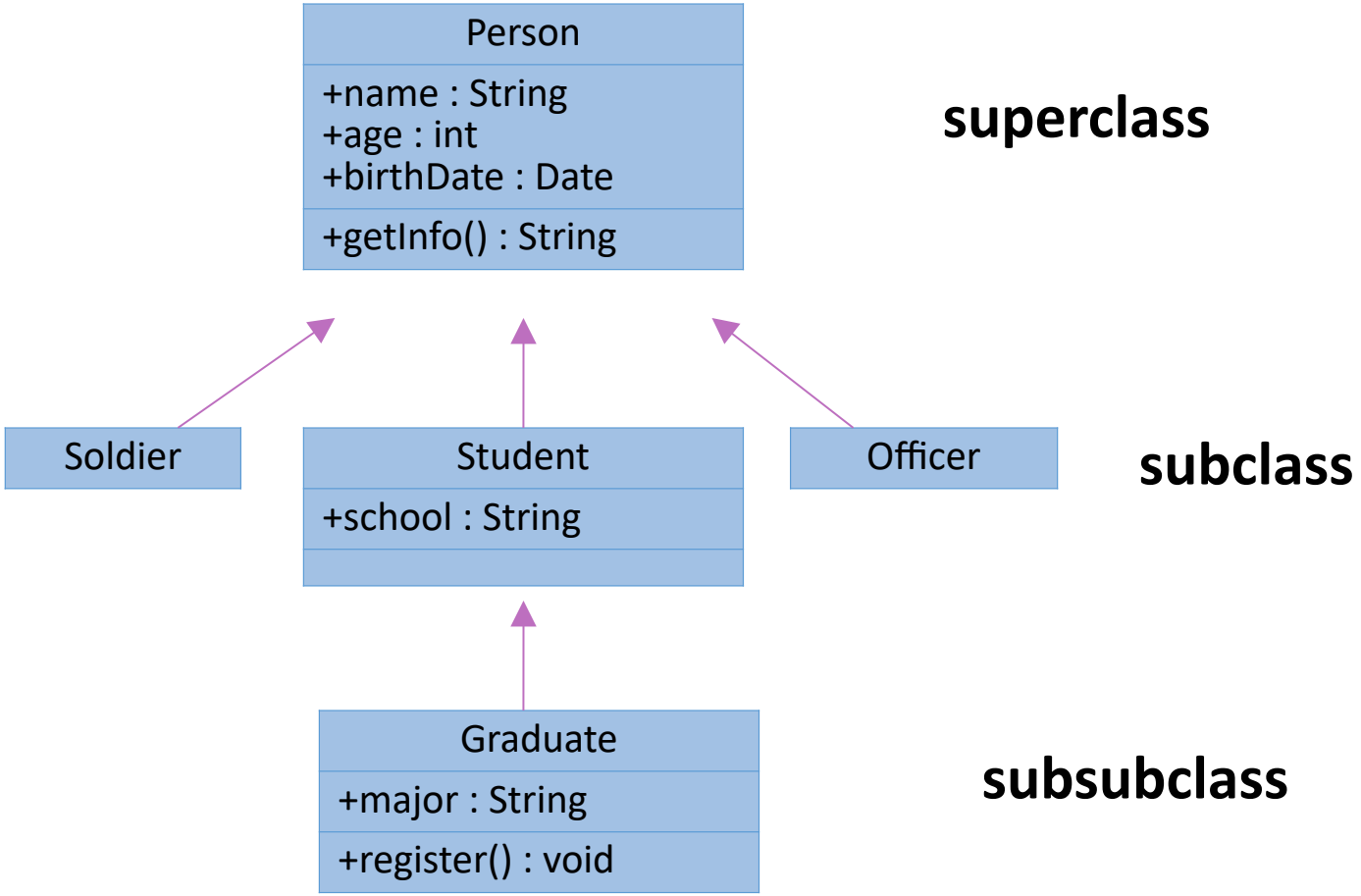
- **子类不能直接访问父类中私有的 (private) 的成员变量和方法。**

单继承举例

由抽象
到具体

从画板
到画作





《席凡宁根海滩》中的搁浅鲸鱼

画中画

这幅描绘席凡宁根海滩景色的画作是由亨德里克·万·雅培于 1641 年创作的，而后在 1873 年被捐赠给菲茨威廉博物馆。画面描绘了一片冬日宁静的海滩，画面上的人们无缘由的聚集在一起。虽然看上去让人觉得很莫名其妙，但也还算正常。直到有一天，一个学生管理员被派去清理这幅画上泛黄的部分，才发现了藏身在这幅画中的秘密。画面上的海平面附近出现了一处和船帆形状很相似的图案。随着进一步的清理，管理人员又在图中发现了一只搁浅的鲸鱼。而之前的“帆“其实就是鲸鱼的鱼鳍。然而这幅画最初被移交到菲茨威廉博物馆时，画面上并没有这只鲸鱼。经过分析得知，遮住鲸鱼的颜料大概是 18 到 19 世纪被画家加上去的。专家还认为这只鲸之所以被隐藏起来，是由于那时候的人们认为死去的动物出现在画作上很令人反感，又或者是因为去掉这只鲸之后，这幅画会卖的更好。

世界名画中的画中画

画中画

伦勃朗《穿军装的老人》

纽厄尔·康弗斯·韦思《全家福》

梵高“牧场花地和玫瑰静物”

预制件

全集

预制件

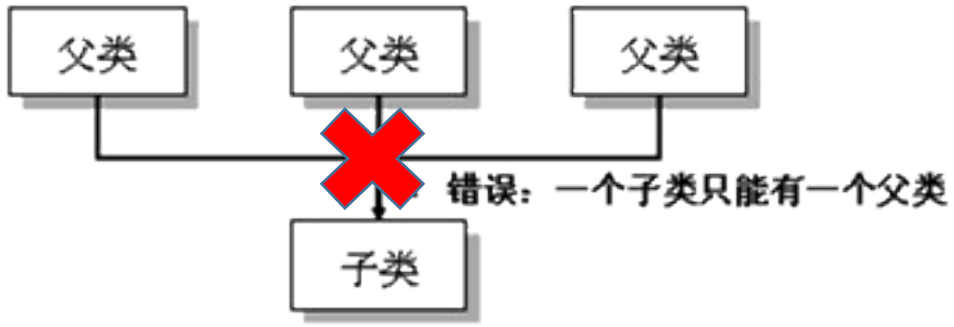
全集

子集

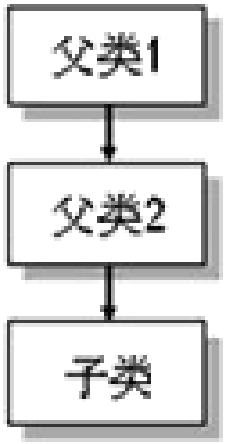
类的继承（6)

- **Java 只支持单继承，不允许多重继承**
  - 一个子类只能有一个父类
  - 一个父类可以派生出多个子类
    - ✓ class SubDemo extends Demo{ } //ok
    - ✓ class SubDemo extends Demo1,Demo2...//error

单继承
多级继承



多重继承



多层继承

△5.2 方法的重写 (override)

override
方法重写

在画板上涂抹

- **定义**：在子类中可以根据需要对从父类中继承来的方法进行改造，也称方法的重置、覆盖。在程序执行时，子类的方法将覆盖父类的方法。
- **要求**：
  - **重写方法必须和被重写方法具有相同的方法名称、参数列表和返回值类型。**
  - **重写方法不能使用比被重写方法更严格的访问权限。**
  - **重写和被重写的方法须同时为 static 的，或同时为非 static 的**
  - **子类方法抛出的异常不能大于父类被重写方法的异常**

重写方法举例(1)

举例

```
public class Person {
        public String name;
        public int age;
        public String getInfo() {
                return "Name: "+ name + "\n" +"age: "+ age;
        }
}
public class Student extends Person {
        public String school;
        public String getInfo() {      // 重写方法
                return  "Name: "+ name + "\nage: "+ age
                   + "\nschool: "+ school;
         }
        public static void main(String args[]){
                Student s1=new Student();
                s1.name="Bob";
                s1.age=20;
                s1.school="school2";
                System.out.println(s1.getInfo());  //Name:Bob  age:20  school:school2
        }
}
```

```
Person p1=new Person();
p1.getInfo();
// 调用 Person 类的 getInfo() 方法
Student s1=new Student();
s1.getInfo();
// 调用 Student 类的 getInfo() 方法
```

这是一种“多态性”：同名的方法，用不同的对象来区分调用的是哪一个方法。

重写方法举例(2)

举例

```
class Parent {
        public void method1() {}
}
```

```
class Child extends Parent {
        private void method1() {}
// 非法，子类中的 method1() 的访问权限 private 比被覆盖方法的访问权限 public 弱
}
```

```
public class UseBoth {
        public static void main(String[] args) {
                Parent p1 = new Parent();
                Child p2 = new Child();
                p1.method1();
                p2.method1();
        }
}
```

5.3 四种访问权限修饰符

Java 权限修饰符 public 、 protected 、 private 置于**类的成员**定义前，用来限定对象对该类对象成员的访问权限。

修饰符	类内部	同一个包	子类	任何地方
private	Yes			
default	Yes	Yes		
protected	Yes	Yes	Yes	
public	Yes	Yes	Yes	Yes

对于 class 的权限修饰只可以用 public 和 default 。

- public 类可以在任意地方被访问。
- default 类只可以被同一个包内部的类访问。

访问控制举例

Outline | *Parent.java | Child.java

```java
//Parent.java
package ch004;

public class Parent {
    private int f1 = 1;
            int f2 = 2;
    protected  int f3 = 3;
    public  int f4 = 4;

    private  void  fm1() {
        System.out.println("in fm1() f1=" + f1);
    }

         void fm2() {
        System.out.println("in fm2() f2=" + f2);
    }

    protected  void  fm3() {
        System.out.println("in fm3() f3=" + f3);
    }

    public void fm4() {
        System.out.println("in fm4() f4=" + f4);
    }
}
```

访问控制举例

//设父类和子类在同一个包内

权限

Outline | *Parent.java | Child.java

```
//Child.java
package ch004;

public class Child extends Parent{
    private int c1 = 21;
    public  int c2 = 22;
    private void cm1(){System.out.println("in cm1() c1=" + c1);}
    public  void cm2(){System.out.println("in cm2() c2=" + c2);}

    public static void main(String args[]){
        int i;
        Parent  p = new Parent();
        i = p.f2;           //  2 i = p.f3;     i = p.f4;

        Child  c = new Child();
        i = c.f2;           //  ??? i = c.f3;       i = c.f4;
        i = c.c1;           //  i = c.c2;
        c.cm1();        // c.cm2();    c.fm2();   c.fm3();   c.fm4()
    }
}
```

2

2

21

21

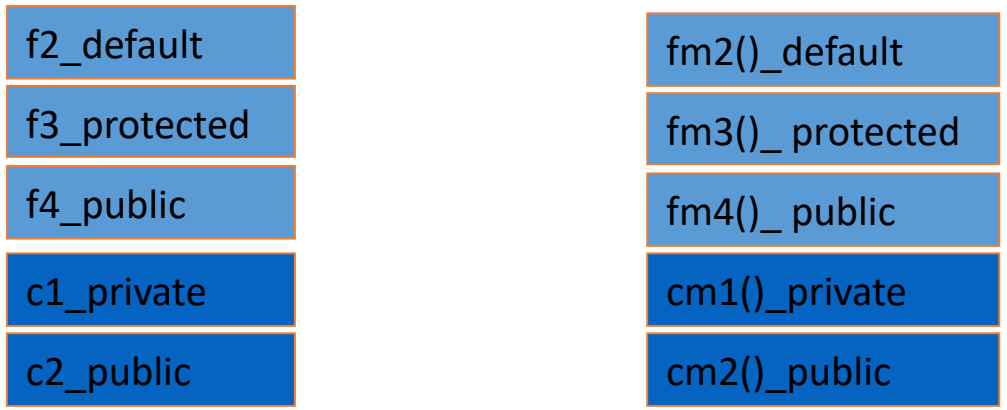
Problems | Console | Debug

<terminated> Child [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (2017

in cm1() c1=21

访问控制分析

父类 Parent 和子类 Child 在同一包中定义时：



子类对象可以访问的数据

子类的对象可以调用的方法

5.4 关键字 super （ vs. this ）

super

- 在 Java 类中使用 super 来调用父类中的指定操作：
  - super 可用于访问父类中定义的属性
  - super 可用于调用父类中定义的成员方法
  - super 可用于在子类构造方法中调用父类的构造器
- 注意：
  - 尤其当子父类出现同名成员时，可以用 super 进行区分
  - super 的追溯不仅限于直接父类
  - super 和 this 的用法相像， this 代表本类对象的引用， super 代表父类的内存空间的标识

关键字 super 举例

super

Outline | Person006.java | Student01.java | TestSt

```java
//Person006.java
package ch004;
import java.util.Date;

public class Person006 {
    public String name;
    public int age;
    public String getInfo() {
        return "Name is  "+name+"\nAge is  "+age;
    }
}
```

Outline | Person006.java | Student01.java | TestStuder

```java
package ch004;

public class Student01 extends Person006 {
    protected String name = "李四";
    private String school = "New Oriental";
    public String getSchool(){ return school; }
    public String getInfo() {
        return super.getInfo() +"\nschool: " +school;
    }
}
```

关键字 super 举例

super

Outline | Person006.java | Student01.java | TestStudent001.java

```
//TestStudent001.java
package ch004;

public class TestStudent001 {
    public static void main(String[] args){
        Student01 st = new Student01();
        System.out.println(st.getInfo());
    }
}
```

Problems | Console | Debug

<terminated> TestStudent001 [Java Application] C:\Program Files\Java\jdk1.8.0_144\

```
Name is  null
Age is  0
school: New Oriental
```

调用父类的构造器

- 子类中所有的构造器**默认**都会访问父类中**空参数**的构造器
- 当父类中没有空参数的构造器时，子类的构造器必须通过 **this(参数列表)** 或者 **super(参数列表)** 语句指定调用本类或者父类中相应的构造器，且必须放在构造器的第一行
- 如果子类构造器中既未显式调用父类或本类的构造器，且父类中又没有无参的构造器，则**编译出错**

调用父类构造器举例

```
1    public class Person {
2
3        private String name;
4        private int age;
5        private Date birthDate;
6
7        public Person(String name, int age, Date d) {
8            this.name = name;
9            this.age = age;
10           this.birthDate = d;
11       }
12       public Person(String name, int age) {
13           this(name, age, null);
14       }
15       public Person(String name, Date d) {
16           this(name, 30, d);
17       }
18       public Person(String name) {
19           this(name, 30);
20       }
21       // ……
22   }
```

```java
//Person003.java
package ch004;
import java.util.Date;

public class Person003 {
    private String name;
    private int age;
    private Date birthDate;

    public void Person(String name, int age, Date d) {
        this.name = name;
        this.age = age;
        this.birthDate = d;
    }
    public void Person(String name, int age) {
        //this(name, age, null);
        this.name=name; this.age=age; this.birthDate=null;
    }

    public void Person(String name, Date d) {
        //this(name, 30, d);
        this.name=name; this.age=30; this.birthDate=d;
    }

    public void Person(String name) {
        //this(name, 30);
        this.name=name; this.age=30;
    }
}
```

调用父类构造器举例

```
1   public class Student extends Person {
2           private String school;
3
4           public Student(String name, int age, String s) {
5                   super(name, age);
6                   school = s;
7           }
8           public Student(String name, String s) {
9                   super(name);
10                  school = s;
11          }
12    public Student(String s) { //  编译出错 : no super(), 系统将调用父
                                  // 类无参数的构造方法。
13                  school = s;
14          }
15  }
```

区别

this 和 super 的区别

No.	区别点	this	super
1	访问属性	访问本类中的属性，如果本类没有此属性则从父类中继续查找	访问父类中的属性
2	调用方法	访问本类中的方法	直接访问父类中的方法
3	调用构造器	调用本类构造器，必须放在构造器的首行	调用父类构造器，必须放在子类构造器的首行
4	特殊	表示当前对象	无此概念

思考

问题

如果现在父类的一个方法定义成 private 访问权限，在子类中将此方法声明为 default 访问权限，那么这样还叫重写吗？

(NO)

我的理解：应该不算。子类中访问不了 private 的方法，也不能改权限。写一个同名方法，相当于新方法。

第 5 讲 Java 面向对象编程 -2

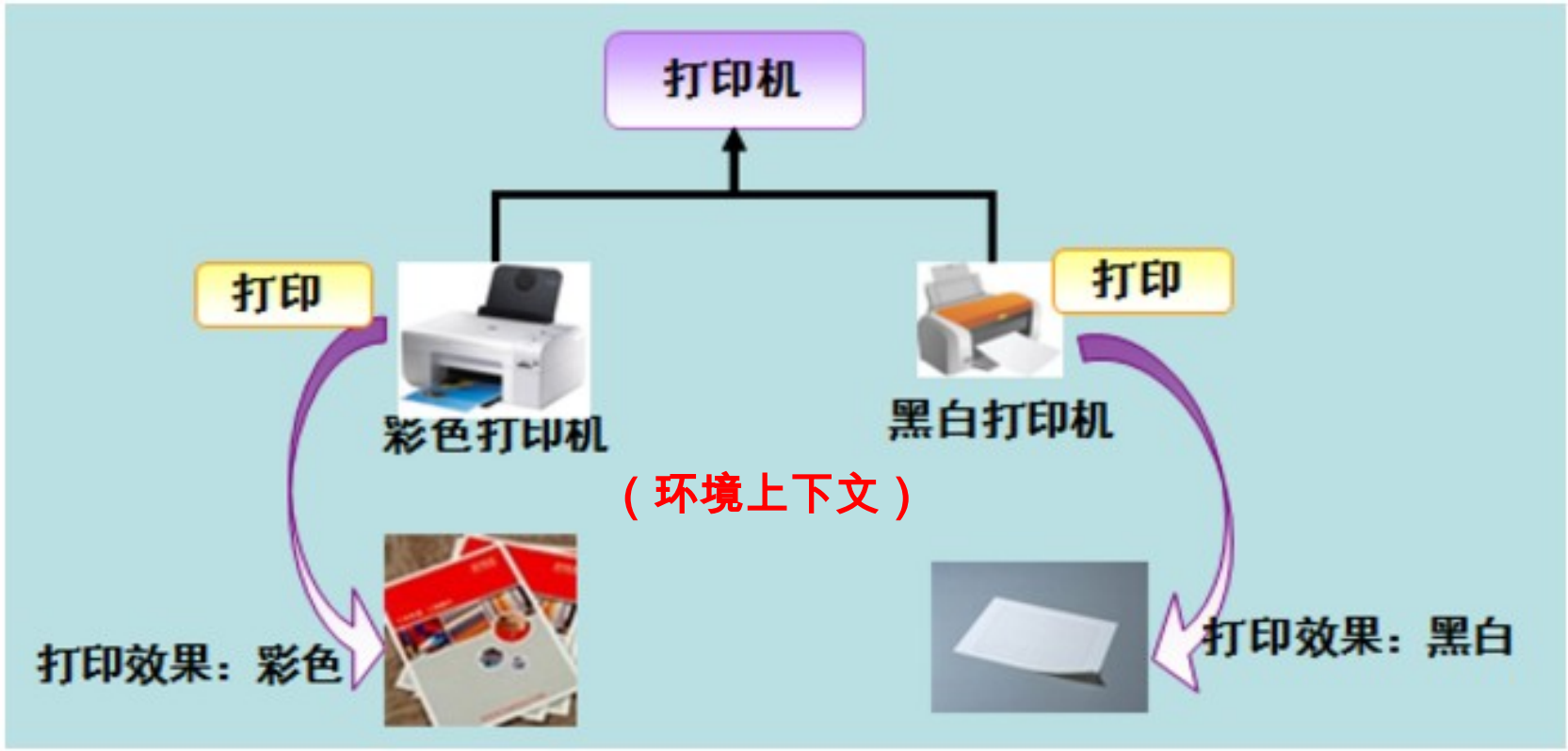
5.1 面向对象特征之继承

5.2 面向对象特征之多态

5.6 面向对象特征之三： Java 多态性

多态是同一个行为具有多个不同表现形式或形态的能力。

多态就是同一个接口，使用不同的实例而执行不同操作，如图所示：



举例
一个行为定义，
多个实现方案
所以是指的方法

多态性是对象多种表现形式的体现。

现实中，比如我们按下 F1 键这个动作：

- 如果当前在 Flash 界面下弹出的就是 AS 3 的帮助文档；
- 如果当前在 Word 下弹出的就是 Word 帮助；
- 在 Windows 下弹出的就是 Windows 帮助和支持。

同一个事件发生在不同的对象上会产生不同的结果。

（类型 / 环境上下文）

行为抽象
对应不同情景下的解决方案

多态的优点

- 1. 消除类型之间的耦合关系
- 2. 可替换性
- 3. 可扩充性
- 4. 接口性
- 5. 灵活性
- 6. 简化性

（类型 / 环境上下文）

多态存在的三个必要条件

- 继承
- 重写
- 父类引用指向子类对象

比如：

```
Parent p = new Child();
```

多态的好处：

- 可以使程序有良好的扩展
- 可以对所有类的对象进行通用处理。

当使用多态方式调用方法时，

- 首先检查父类中是否有该方法，如果没有，则编译错误；
- 如果有，再去调用子类的同名方法。

Java 多态性

多态：
编译时与运行时
类型不一致

- 多态性，是面向对象中最重要的概念，在 java 中有两种体现：
  1. 方法的重载 (overload) 和重写 (overwrite) 。
  2. **对象的多态性（环境上下文）** ——可以直接应用在抽象类和接口上。

- Java 引用变量有两个类型：**编译时类型（类型上下文）**和**运行时类型（环境上下文）**。编译时类型由声明该变量时使用的类型决定，**运行时类型由实际赋给该变量的对象（环境上下文）决定**。
  - **若编译时类型和运行时类型不一致，就出现多态（ Polymorphism ）**

多态性(2)

向上转型

- 对象的多态 —在 Java 中，子类的对象可以替代父类的对象使用
  - 一个变量只能有一种确定的数据类型
  - 一个引用类型变量可能指向（引用）多种不同类型的对象

```
Person p = new Student();
Object o = new Person();//Object 类型的变量 o ，指向 Person 类型的对象
o = new Student(); //Object 类型的变量 o ，指向 Student 类型的对象
```

◆子类可看做是特殊的父类，所以父类类型的引用可以指向子类的对象：向上转型 (upcasting) 。

向上转型

全集

子集

一旦向上转型，子类中原本特有的方法就不能再被调用了

多态性 (3)

- 一个引用类型变量如果声明为父类的类型，但实际引用的是子类对象，那么该变量就不能再访问子类中添加的属性和方法

```
Student m = new Student();
m.school = “ pku ”;      // 合法 ,Student 类有 school 成员变量
Person e = new Student();
e.school = “ pku ”;      // 非法 ,Person 类没有 school 成员变量
```

属性是在编译时确定的，编译时 e 为 Person 类型，没有 school 成员变量，因而编译错误。

Animal.java

```java
//Animal.java
package ch006;

abstract class Animal {
    abstract void eat();
}
```

Cat.java

```java
//Cat.java
package ch006;

class Cat extends Animal {
    public void eat() {
        System.out.println("吃鱼");
    }
    public void work() {
        System.out.println("抓老鼠");
    }
}
```

Dog.java

```java
//Dog.java
package ch006;

class Dog extends Animal {
    public void eat() {
        System.out.println("吃骨头");
    }
    public void work() {
        System.out.println("看家");
    }
}
```

Test.java

```java
//Test.java
package ch006;

public class Test {
    public static void main(String[] args) {
        System.out.println("show(new Cat())");
        show(new Cat());  // 以 Cat 对象调用 show 方法
        System.out.println("show(new Cat())");
        show(new Dog());  // 以 Dog 对象调用 show 方法

        Animal a = new Cat();  // 向上转型
        System.out.println("a.eat()-向上转型");
        a.eat();               // 调用的是 Cat 的 eat
        System.out.println("c.work()-向下转型");
        Cat c = (Cat)a;        // 向下转型
        c.work();              // 调用的是 Cat 的 work
    }

    public static void show(Animal a)  {
        a.eat();
        // 类型判断
        if (a instanceof Cat)  {  // 猫做的事情
            Cat c = (Cat)a;
            c.work();
        } else if (a instanceof Dog) { // 狗做的事情
            Dog c = (Dog)a;
            c.work();
        }
    }
}
```

编译时类型和运行时类型不一致

```
show(new Cat())
吃鱼
抓老鼠
show(new Cat())
吃骨头
看家
a.eat()-向上转型
吃鱼
c.work()-向下转型
抓老鼠
```

虚拟方法调用 (Virtual Method Invocation)

观察子类中被重写的方法的行为怎样影响多态性。

已经讨论了方法的重写，也就是子类能够重写父类的方法。

- 当子类对象调用重写的方法时，调用的是子类的方法，而不是父类中被重写的方法。
- 要想调用父类中被重写的方法，则必须使用关键字 super 。

虚拟方法调用 (Virtual Method Invocation)

- 正常的方法调用

```
Person e = new Person();
e.getInfo();
Student e = new Student();
e.getInfo();
```

- 虚拟方法调用 (多态情况下)

```
Person e = new Student();
e.getInfo();     // 调用 Student 类的 getInfo() 方法
```

- 编译时类型和运行时类型

动态绑定

编译时 e 为 Person 类型，而方法的调用是在运行时确定的，所以调用的是 Student 类的 getInfo() 方法。——动态绑定

Employee.java

```java
//Employee.java
package ch006;

public class Employee {
    private String name;
    private String address;
    private int number;

    public Employee(String name, String address, int number) {
        System.out.println("Employee 构造函数");
        this.name = name;
        this.address = address;
        this.number = number;
    }

    public void mailCheck() {
        System.out.println("邮寄支票给: "+ this.name+" "+ this.address);
    }

    public String toString() {
        return name +" "+ address + " " + number;
    }

    public String getName() {
        return name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String newAddress) {
        address = newAddress;
    }

    public int getNumber() {
        return number;
    }
}
```

Employee.java | Salary.java

```java
//Salary.java
package ch006;

public class Salary extends Employee
{
    private double salary; // 全年工资

    public Salary(String name, String address, int number, double salary) {
        super(name, address, number);
        setSalary(salary);
    }

    public void mailCheck() {
        System.out.println("Salary 类的 mailCheck 方法 ");
        System.out.println("邮寄支票给：" + getName() + ", 工资为：" + salary);
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double newSalary) {
        if(newSalary >= 0.0) {
            salary = newSalary;
        }
    }

    public double computePay() {//按周付薪（返回周薪）
        System.out.println("计算工资，付给：" + getName());
        return salary/52;
    }
}
```

继承

Employee.java | Salary.java | VirtualDemo.java

```java
//VirtualDemo.java
package ch006;

public class VirtualDemo {
    public static void main(String [] args) {
        Salary s = new Salary("员工 A", "北京", 3, 3600.00);
        Employee e = new Salary("员工 B", "上海", 2, 2400.00);
        System.out.println("使用 Salary 的引用调用 mailCheck --");
        s.mailCheck();
        System.out.println("\n使用 Employee 的引用调用 mailCheck--");
        e.mailCheck();
    }
}
```

Problems | Javadoc | Declaration | Console

```
<terminated> VirtualDemo [Java Application] C:\Pro
Employee 构造函数
Employee 构造函数
使用 Salary 的引用调用 mailCheck --
Salary 类的 mailCheck 方法
邮寄支票给：员工 A，工资为：3600.0

使用 Employee 的引用调用 mailCheck--
Salary 类的 mailCheck 方法
邮寄支票给：员工 B，工资为：2400.0
```

多态存在的三个必要条件

- 继承
- 重写
- 父类引用指向子类对象

比如：

```
Parent p = new Child();
```

多态

多态

Employee.java | Salary.java | VirtualDemo.java

```
//VirtualDemo.java
package ch006;

public class VirtualDemo {
    public static void main(String [] args) {
        Salary s = new Salary("员工 A", "北京", 3, 3600.00);
        Employee e = new Salary("员工 B", "上海", 2, 2400.00);
        System.out.println("使用 Salary 的引用调用 mailCheck --");
        s.mailCheck();
        System.out.println("\n使用 Employee 的引用调用 mailCheck--");
        e.mailCheck();
    }
}
```

```
public void mailCheck() {
    System.out.println("Salary 类的 mailCheck 方法");
    System.out.println("邮寄支票给：" + getName() + ", 工资为：" + salary);
}
```

Problems | Javadoc | Declaration | Console

<terminated> VirtualDemo [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.ex

```
Employee 构造函数
Employee 构造函数
使用 Salary 的引用调用 mailCheck --
Salary 类的 mailCheck 方法
邮寄支票给：员工 A，工资为：3600.0

使用 Employee 的引用调用 mailCheck--
Salary 类的 mailCheck 方法
邮寄支票给：员工 B，工资为：2400.0
```

Employee.java | Salary.java | VirtualDemo.java

```java
//VirtualDemo.java
package ch006;

public class VirtualDemo {
    public static void main(String [] args) {
        Salary s = new Salary("员工 A", "北京", 3, 3600.00);
        Employee e = new Salary("员工 B", "上海", 2, 2400.00);
        System.out.println("使用 Salary 的引用调用 mailCheck --- ");
        s.mailCheck();
        System.out.println("\n使用 Employee 的引用调用 mailCheck--");
        e.mailCheck();
    }
}
```

```java
public void mailCheck() {
    System.out.println("Salary 类的 mailCheck 方法 ");
    System.out.println("邮寄支票给: " + getName() + ", 工资为: " + salary);
}
```

多态

Problems @ Javadoc Declaration

<terminated> VirtualDemo [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.ex

```
Employee 构造函数
Employee 构造函数
使用 Salary 的引用调用 mailCheck --
Salary 类的 mailCheck 方法
邮寄支票给：员工 A，工资为：3600.0

使用 Employee 的引用调用 mailCheck--
Salary 类的 mailCheck 方法
邮寄支票给：员工 B，工资为：2400.0
```

多态引用时，构造子类对象时的构造方法的调用顺序

1，先调用超类的构造方法，多重超类首先调用最远超类的方法；

2，然后再执行当前子类的构造方法；

调用时需要谨慎处理调用方法

Employee.java | Salary.java | VirtualDemo.java

```java
//VirtualDemo.java
package ch006;

public class VirtualDemo {
    public static void main(String [] args) {
        Salary s = new Salary("员工 A", "北京", 3, 3600.00);
        Employee e = new Salary("员工 B", "上海", 2, 2400.00);
        System.out.println("使用 Salary 的引用调用 mailCheck --");
        s.mailCheck();
        System.out.println("\n使用 Employee 的引用调用 mailCheck--");
        e.mailCheck();
    }
}
```

多态

例子解析

- 实例中，实例化了两个 Salary 对象：一个使用 Salary 引用 s，另一个使用 Employee 引用 e。
- 当调用 s.mailCheck() 时，编译器在编译时会在 Salary 类中找到 mailCheck()，执行过程 JVM 就调用 Salary 类的 mailCheck()。
- 因为 e 是 Employee 的引用，所以调用 e 的 mailCheck() 方法时，编译器会去 Employee 类查找 mailCheck() 方法 。
- 在编译的时候，编译器使用 Employee 类中的 mailCheck() 方法验证该语句， 但是在运行的时候，Java虚拟机(JVM)调用的是 Salary 类中的 mailCheck() 方法。

以上整个过程被称为虚拟方法调用，该方法被称为虚拟方法。

通常是在上层的宽泛数据类型

Java中所有的方法都能以这种方式表现，因此，重写的方法能在运行时调用，不管编译的时候源代码中引用变量是什么数据类型。

通常是在下层的具体数据类型 - 重现了父辈的方法

多态小结

多态

- **前提：**
  - 需要存在继承或者实现关系
  - 要有覆盖操作

- **成员方法：**
  - 编译时：要查看引用变量所属的类中是否有所调用的方法。
  - 运行时：调用实际对象所属的类中的重写方法。
- **成员变量：**
  - 不具备多态性，只看引用变量所属的类。

●子类继承父类

- 若子类重写了父类方法，就意味着子类里定义的方法彻底覆盖了父类里的同名方法，系统将不可能把父类里的方法转移到子类中
- 对于实例变量则不存在这样的现象，即使子类里定义了与父类完全相同的实例变量，这个实例变量依然不可能覆盖父类中定义的实例变量

多态
和方法有关
和属性无关

instanceof 操作符

instance of

x instanceof A ：检验 x 是否为类 A 的对象，返回值为 boolean 型。

- 要求 x 所属的类与类 A 必须是子类和父类的关系，否则编译错误。
- 如果 x 属于类 A 的子类 B ， x instanceof A 值也为 true 。

```
public class Person extends Object {…}
public class Student extends Person {…}
public class Graduate extends Person {…}
-----------------------------------------------------------------
public void method1(Person e) {
        if (e instanceof Person)
                // 处理 Person 类及其子类对象
        if (e instanceof Student)
                // 处理 Student 类及其子类对象
        if (e instanceof Graduate)
                // 处理 Graduate 类及其子类对象
}
```

多态的实现方式

方式一：重写：

这个内容已经在上一章节详细讲过，就不再阐述，详细可访问：Java 重写(Override)与重载(Overload)。

方式二：接口

- 1. 生活中的接口最具代表性的就是插座，例如一个三接头的插头都能接在三孔插座中，因为这个是每个国家都有各自规定的接口规则，有可能到国外就不行，那是因为国外自己定义的接口类型。
- 2. java中的接口类似于生活中的接口，就是一些方法特征的集合，但没有方法的实现。具体可以看 java接口 这一章节的内容。

方式三：抽象类和抽象方法

详情请看 Java抽象类 章节。

对于多态，可以总结以下几点：

一、使用父类类型的引用指向子类的对象；

二、该引用只能调用父类中定义的方法和变量；

三、如果子类中重写了父类中的一个方法，那么在调用这个方法的时候，将会调用子类中的这个方法；（动态连接、动态调用）；

四、变量不能被重写（覆盖），"重写"的概念只针对方法，如果在子类中"重写"了父类中的变量，那么在编译时会报错。

类的属性变量是能重写（覆盖）

多态

Animal01.java | Dog01.java | Cat01.java | TestOverride.java

```java
//Animal01.java
package ch006;

public class Animal01 {
    public int age;
    public void move(){
        System.out.println("动物可以移动");
    }
}
```

Animal01.java | Dog01.java | Cat01.java | Te

```java
//Dog01.java
package ch006;

class Dog01 extends Animal01 {
    public double age;

    public void move(){
        age = 10.00;
        System.out.println("狗可以跑和走");
    }

    public void bark(){
        System.out.println("狗可以吠叫");
    }
}
```

Animal01.java | Dog01.java | Cat01.java

```java
//Cat01.java
package ch006;

class Cat01 extends Animal01 {
    public void move(){
        super.age = 3;
        System.out.println("猫可以跳");
    }
}
```

Animal01.java | Dog01.java | Cat01.java | TestOverride.java

```java
//TestOverride.java
package ch006;

public class TestOverride {
    public static void main(String args[]){
        Animal01 a = new Animal01(); // Animal 对象
        Animal01 b = new Dog01(); // Dog 对象
        Dog01 c = new Dog01();
        Cat01 d = new Cat01();

        a.move();// 执行 Animal 类的方法
        b.move();//执行 Dog 类的方法
        c.move();//执行 Dog 类的方法
        d.move();//执行 Cat 类的方法
        Object aValue = a.age;
        Object bValue = b.age;
        Object cValue = c.age;
        System.out.println("The type of "+a.age+" is "+(aValue instanceof Double ? "double" : (aValue instanceof Integer ? "int" : "")));
        System.out.println("The type of "+b.age+" is "+(bValue instanceof Double ? "double" : (bValue instanceof Integer ? "int" : "")));
        System.out.println("The type of "+c.age+" is "+(cValue instanceof Double ? "double" : (cValue instanceof Integer ? "int" : "")));// 覆盖age属性
        System.out.println("The age of cat  is "+d.age);
    }
}
```

多态

```
动物可以移动
狗可以跑和走
狗可以跑和走
猫可以跳
The type of 0 is int
The type of 0 is int
The type of 10.0 is double
The age of cat  is 3
```

课堂练习

```
class Person {
        protected String name="person";
        protected int age=50;
        public String getInfo() {
                return "Name: "+ name + "\n" +"age: "+ age;
        }
}

class Student extends Person {
        protected String school="pku";
        public String getInfo() {
                return  "Name: "+ name + "\nage: "+ age
                + "\nschool: "+ school;
        }
}

class Graduate extends Student{
        public String major="IT";
        public String getInfo()
        {
                return  "Name: "+ name + "\nage: "+ age
                + "\nschool: "+ school+"\nmajor:"+major;
        }
}
```

建立 TestInstance 类，在类中定义方法 method1(Person e);

在 method1 中： **覆盖**

(1) 根据 e 的类型调用相应类的 getInfo() 方法。

(2) 根据 e 的类型执行：

如果 e 为 Person 类的对象，输出：“ a person ”;

如果 e 为 Student 类的对象，输出

“a student”

“a person ”

如果 e 为 Graduate 类的对象，输出：

“a graduated student”

“a student”

“a person”

对象类型转换 (Casting)

- **基本数据类型的 Casting ：**
  - **自动类型转换**：小的数据类型可以自动转换成大的数据类型

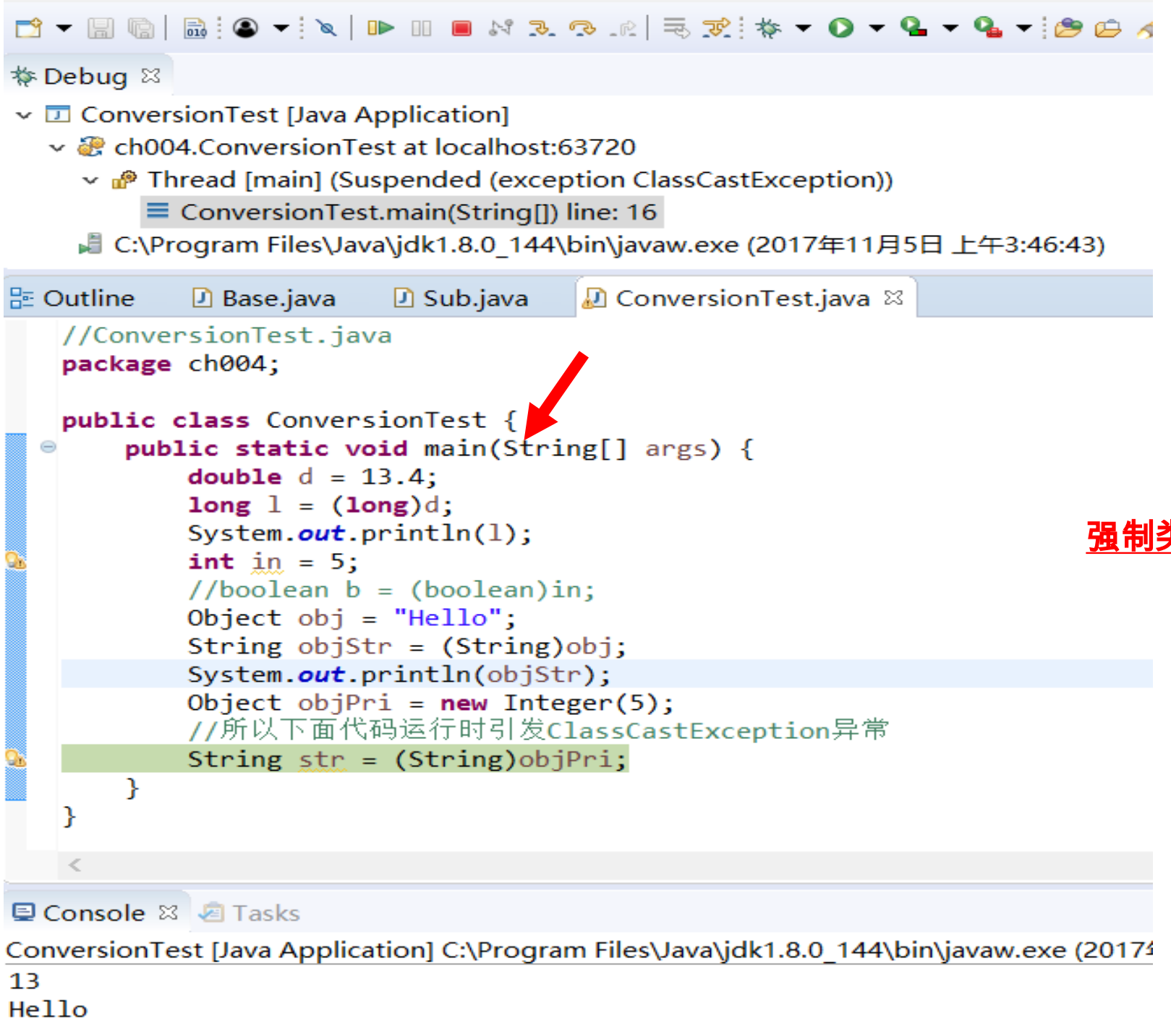
    如 long g=20;　　double d=12.0f
  - **强制类型转换**：可以把大的数据类型强制转换 (casting) 成小的数据类型

    如　float f=(float)12.0;　int a=(int)1200L
- **对 Java 对象的强制类型转换称为造型**
  - **从子类到父类的类型转换可以自动进行**
  - **从父类到子类的类型转换必须通过造型 Casting (强制类型转换) 实现****强制类型转换**
  - **无继承关系的引用类型间的转换是非法的**
  - **在造型前可以使用 instanceof 操作符测试一个对象的类型**

对象类型转换举例

```
public class ConversionTest{
   public static void main(String[] args) {
            double d = 13.4;
            long l = (long)d;
            System.out.println(l);
            int in = 5;
            //boolean b = (boolean)in;
            Object obj = "Hello";
            String objStr = (String)obj;
            System.out.println(objStr);
            Object objPri = new Integer(5);
            // 所以下面代码运行时引发 ClassCastException 异常
            String str = (String)objPri;
   }
}
```

强制类型转换


Debug
ConversionTest [Java Application]
ch004.ConversionTest at localhost:63720
Thread [main] (Suspended (exception ClassCastException))
ConversionTest.main(String[]) line: 16
C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (2017年11月5日 上午3:46:43)
Outline
Base.java
Sub.java
ConversionTest.java
//ConversionTest.java
package ch004;
public class ConversionTest {
public static void main(String[] args) {
double d = 13.4;
long l = (long)d;
System.out.println(l);
int in = 5;
//boolean b = (boolean)in;
Object obj = "Hello";
String objStr = (String)obj;
System.out.println(objStr);
Object objPri = new Integer(5);
//所以下面代码运行时引发ClassCastException异常
String str = (String)objPri;
}
}
Console
Tasks
ConversionTest [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (2017
13
Hello

强制类型转换

对象类型转换举例

```
public class Test{
        public void method(Person e) {          // 设 Person 类中没有 getschool() 方法
          // System.out.pritnln(e.getschool());   // 非法，编译时错误

           if(e  instanceof  Student){
                Student me = (Student)e;        // 将 e 强制转换为 Student 类型
                System.out.pritnln(me.getschool());
           }
    }

     public static  void main(Stirng args[]){
           Test t = new Test();
           Student m = new Student();
           t.method(m);
     }
}
```

对象类型转换

类型转换

较高级的基本数据类型

强制类型转化

自动类型转化

较低级的基本数据类型

父类（如： Person ）

向下转型

使用 instanceof
进行判断

向上转型

子类（如： Student ）

5.7 Object 类

- Object 类是所有 Java 类的根父类
- 如果在类的声明中未使用 extends 关键字指明其父类，则默认父类为 Object 类

```
public class Person {
...
}
```

Object 类

等价于：

```
public class Person extends Object {
...
}
```

- 例：

```
method(Object obj){...}// 可以接收任何类作为其参数
Person o=new Person();
method(o);
```

Object 类中的主要方法

Object 类

NO.	方法名称	类型	描述
1	public Object()	构造	构造方法
2	public boolean equals(Object obj)	普通	对象比较
3	public int hashCode()	普通	取得 Hash 码
4	public String toString()	普通	对象打印时调用

当前正在操作本方法的对象称为当前对象。

Person004.java | TestPerson.java

```java
//Person004.java
package ch004;

public class Person004 {
    String name;
    Person004(String name){
        this.name = name;
    }
    public void getInfo(){
        System.out.println("Person类 --> " + this.name) ;
    }
    public boolean compare(Person004 p){
        return this.name==p.name;
    }
}
```

this 当前对象

当前正在操作本方法
的对象称为当前对象。

TestPerson.java

```
//TestPerson.java
package ch004;

public class TestPerson {
    public static void main(String args[]){
        Person004 per1 = new Person004("张三");
        Person004 per2 = new Person004("李四");
        per1.getInfo();    //当前调用getInfo()方法的对象是per1
        per2.getInfo();    //当前调用getInfo()方法的对象是per2
        boolean b = per1.compare(per2);
        if (b) System.out.println("They are the same guys.");
        else System.out.println("They are not the same guys.");
    }
}
```

Console

```
<terminated> TestPerson [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\ja
Person类 --> 张三
Person类 --> 李四
They are not the same guys.
```

TestPerson.java

```
//TestPerson.java
package ch004;

public class TestPerson {
    public static void main(String args[]){
        Person004 per1 = new Person004("张三");
        Person004 per2 = new Person004("张三");
        per1.getInfo();    //当前调用getInfo()方法的对象是per1
        per2.getInfo();    //当前调用getInfo()方法的对象是per2
        boolean b = per1.compare(per2);
        if (b) System.out.println("They are the same guys.");
        else System.out.println("They are not the same guys.");
    }
}
```

Console

```
<terminated> TestPerson [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\
Person类 --> 张三
Person类 --> 张三
They are the same guys.
```

== 操作符与 equals 方法

- **==** ：
  - 基本类型比较值：只要两个变量的值相等，即为 true.

```
int a=5; if(a==6){…}
```

  - 引用类型比较引用（是否指向同一个对象）: 只有指向同一个对象时，== 才返回 true.

```
Person p1=new Person();
Person p2=new Person();
if (p1==p2){…}
```

    - ✓用“ == ”进行比较时，符号两边的**数据类型必须兼容**（可自动转换的基本数据类型除外），否则编译出错；

== 操作符与 equals 方法

- equals() ：所有类都继承了 Object ，也就获得了 equals() 方法。还可以重写。
  - 只能比较引用类型，其作用与“ == ”相同，比较是否指向同一个对象。
  - 格式 :obj1.equals(obj2)
- 特例：当用 equals() 方法进行比较时，对类 File 、 String 、 Date 及包装类（ Wrapper Class ）来说，是比较类型及内容而不考虑引用的是否是同一个对象；
  - 原因：在这些类中重写了 Object 类的 equals() 方法。

推荐练 习

```
int it = 65;
float fl = 65.0f;
System.out.println( “ 65 和 65.0f 是否相等？” + (it == fl)); //true
```

```
char ch1 = 'A'; char ch2 = 12;
System.out.println("65 和 'A' 是否相等？ " + (it == ch1));//true
System.out.println( “ 12 和 ch2 是否相等？ " + (12 == ch2));//true
```

```
String str1 = new String("hello");
String str2 = new String("hello");
System.out.println("str1 和 str2 是否相等？ "+ (str1 == str2));//false
```

```
System.out.println("str1 是否 equals str2 ？ "+(str1.equals(str2)));//true
```

```
System.out.println( “ hello ” == new java.sql.Date()); // 编译不通过
```

推荐练 习

```
Person p1 = new Person();
p1.name = "atguigu";

Person p2 = new Person();
p2.name = "atguigu";

System.out.println(p1.name .equals( p2.name));//true
System.out.println(p1.name == p2.name);//true
System.out.println(p1.name == "atguigu");

String s1 = new String("bcde");

String s2 = new String("bcde");
System.out.println(s1==s2);//false
```

```
class TestEquals {
        public static void main(String[] args) {
                MyDate m1 = new MyDate(14, 3, 1976);
                MyDate m2 = new MyDate(14, 3, 1976);

                if ( m1 == m2 ) {
                    System.out.println("m1==m2");
                } else {
                    System.out.println("m1!=m2"); //m1 != m2
                }

                if ( m1.equals(m2) ) {
                    System.out.println("m1 is equal to m2");
                  // m1 is equal to m2
                } else {
                    System.out.println("m1 is not equal to m2");
                } } }
```

toString() 方法

- toString() 方法在 Object 类中定义，其返回值是 String 类型，返回类名和它的引用地址。
- 在进行 String 与其它类型数据的连接操作时，自动调用 toString() 方法

  Date now=new Date();

  System.out.println(“ now= ” +now); 相当于

  System.out.println(“now=”+now.toString());
- 可以根据需要在用户自定义类型中重写 toString() 方法

  如 String 类重写了 toString() 方法，返回字符串的值。

  s1=“hello”;

  System.out.println(s1);// 相当于 System.out.println(s1.toString());
- 基本类型数据转换为 String 类型时，调用了对应包装类的 toString() 方法
  - int a=10; System.out.println(“a=”+a);

5.7 包装类 (Wrapper)

- 针对八种基本定义相应的引用类型—包装类（封装类）
- 有了类的特点，就可以调用类中的方法。

基本数据类型	包装类
boolean	Boolean
byte	Byte
short	Short
int	Integer
long	Long
char	Character
float	Float
double	Double

包装类

- **基本数据类型包装成包装类的实例 --- 装箱** 装箱
  - 通过包装类的构造器实现：

    int i = 500; Integer t = new Integer(i);
  - 还可以通过字符串参数构造包装类对象：

    Float f = new Float("5.56");

    Long l = new Long("asdf"); //NumberFormatException
- **获得包装类对象中包装的基本类型变量 --- 拆箱** 拆箱
  - 调用包装类的 .xxxValue() 方法：

    boolean b = bObj.booleanValue();
- JDK1.5 之后，支持自动装箱，自动拆箱。但类型必须匹配。

●字符串转换成基本数据类型

- 通过包装类的构造器实现：

```
int i = new Integer(“12”);
```

- 通过包装类的 parseXxx(String s) 静态方法：

```
Float f = Float.parseFloat(“12.1”);
```

●基本数据类型转换成字符串

- 调用字符串重载的 valueOf() 方法：

```
String fstr = String.valueOf(2.34f);
```

- 更直接的方式：

```
String intStr = 5 + “”
```

包装类用法举例

包装类举例

```
int i = 500;

Integer t = new Integer(i);
```

装箱：包装类使得一个基本数据类型的数据变成了类。

有了类的特点，可以调用类中的方法。

```
String s = t.toString(); // s = “ 500 “ ,t 是类，有 toString 方法

String s1 = Integer.toString(314); // s1= “ 314 “ 将数字转换成字符串。

String s2=“5.56”;

double ds=Double.parseDouble(s2);  // 将字符串转换成数字
```

在 Java 中有两种变量 (Variable) ，一种是基本型别 (primitive type) 的变量，还有一种是类性别 (class type) 的变量，即对象与非对象两种。 Java 为每一个基本型别都提供了一个相应的包装类 (Wrapper) 对其进行包装，例如 int 对应 Integer ， double 对应 Double ， char 对应 Character 等等。可以把某个包装类的一个对象当成其对应的基本型别的一个值来看，所不同的是它现在是一个对象了，可以利用 reference 来进行操作了。 Integer i=new Integer(3) ，这样就创建了一个包装类，如果想得到它所代表的值就可以通过 i.intValue() 来获得。包装类的一个对象一经创建，其所代表的值将不再变化，也就是说不能通过某个指向 (refer to) 它的 reference 来改变它的值了，直至它被垃圾回收器回收。

包装类的用法举例

包装类举例

- **拆箱：将数字包装类中内容变为基本数据类型。**

```
int j = t.intValue();      // j = 500 ， intValue 取出包装类中的数据
```

- 包装类在实际开发中用的最多的在于字符串变为基本数据类型。

```
String str1 = "30" ;
String str2 = "30.3" ;
int x = Integer.parseInt(str1) ; //  将字符串变为 int 型
float f = Float.parseFloat(str2) ; //  将字符串变为 int 型
```

总结—抽象

抽象：通过特定的实例抽取出共同的特征以后形成的概念的过程，它强调主要特征和忽略次要特征。

抽象存在的意义，一方面是提高代码的可扩展性、维护性，修改实现不需要改变定义，减少代码的改动范围；另一方面，它也是处理复杂系统的有效手段，能有效地过滤掉不必要关注的信息。

总结—封装

总结

封装：把对象的属性和操作 (或服务) 结合为一个独立的整体，并尽可能隐藏对象的内部实现细节。

封装的优点：

1 、隐藏实现细节。

2 、安全性。

比如在程序中私有化了 age 属性，并提供了对外的 get 和 set 方法，当外界 使用 set 方法为属性设值，可以在 set 方法里面做个 if 判断，把值设值在 0-80 岁，使得不能随意赋值。

3 、增加代码复用性。

比如在工具中封装的各种方法，可以任意调用，而不用每处去实现细节。

4 、模块化。

封装分为封装属性，方法，类等等。有利于代码调试，相互配合。

总结—继承

总结

继承：子类继承父类的特征和行为，使得子类对象（实例）具有父类的实例域和方法，或子类从父类继承方法，使得子类具有父类相同的行为。当然，如果在父类中拥有私有属性 (private 修饰），则子类是不能被继承的。

继承的特定：

（1）注意事项：

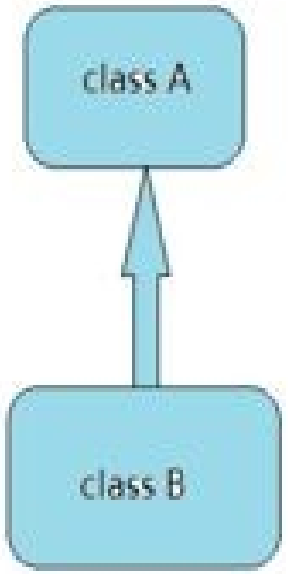
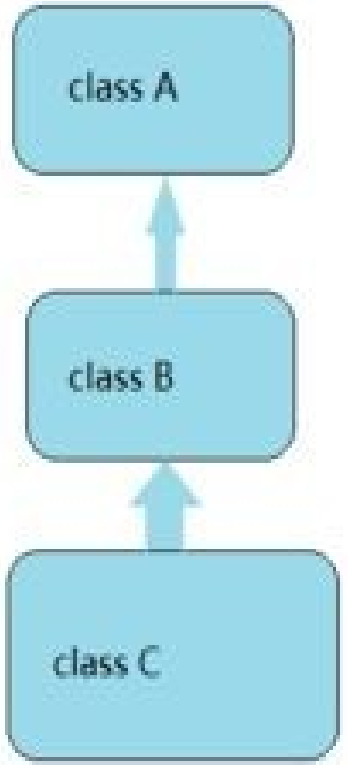
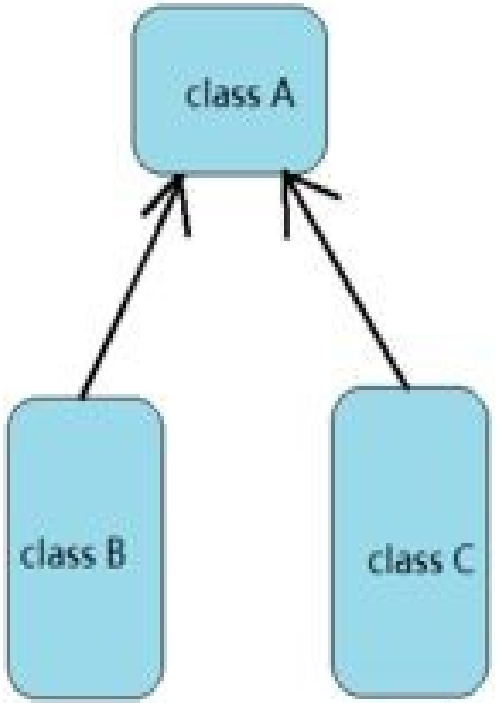
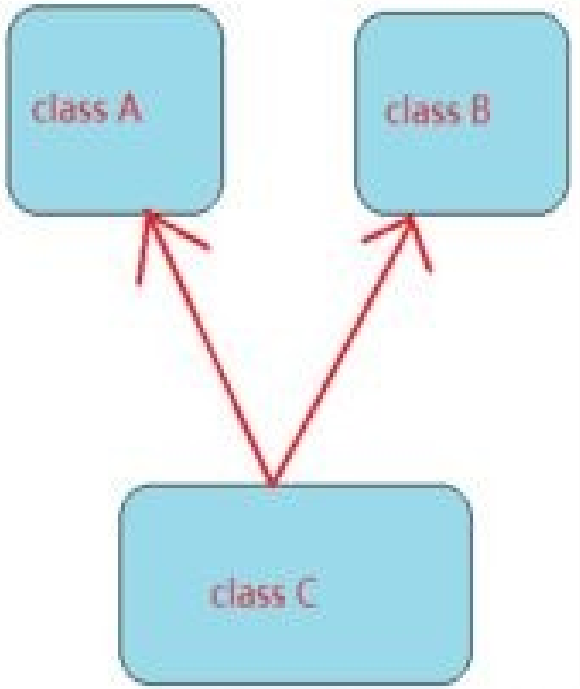
- 只支持单继承，即一个子类只允许有一个父类，但是可以实现多级继承，及子类拥有唯一的父类，而父类还可以再继承。
- 子类可以拥有父类的属性和方法。
- 子类可以拥有自己的属性和方法。
- 子类可以重写覆盖父类的方法。
- 子类对父类的允许访问的方法的实现进行重写 (override)，返回值和形参都不能改变。即外壳不变，核心重写！

（2）特点：

- 提高代码复用性。
- 父类的属性方法可以用于子类。
- 可以轻松的定义子类。
- 使设计应用程序变得简单。

总结—继承

总结

单继承	多级继承	同一个父类多个子类	多继承
public class A{}	public class A{}	public class A{}	public class A{}
public class B extends B{}	public class B extends A{}	public class B extends A{}	public class B{}
	public class C extends B{}	public class C extends A{}	public class C extends A,B{}
			错误，Java不支持多继承 

总结—多态

总结

多态：是同一个行为具有不同表现形式或形态的能力。

多态的特点：
1 ）消除类型之间的耦合关系，实现低耦合。
2 ）灵活性
3 ）可扩充性
4 ）可替换性

多态的条件：
1 ）存在继承
2 ）“父类型引用” 指向“子类型实例”　　Animals dog = new Dog();
3 ）重写
注意：在多态中，编译看左边，运行看右边。
4. 向上转型
格式：父类名称 对象名 = new 子类名称 ();
含义：右侧创建一个子类对象，把它当作父类来使用。
注意：向上转型一定是安全的。
缺点：一旦向上转型，子类中原本特有的方法就不能再被调用了。

总结—多态

总结

Animals dog = new Dog();

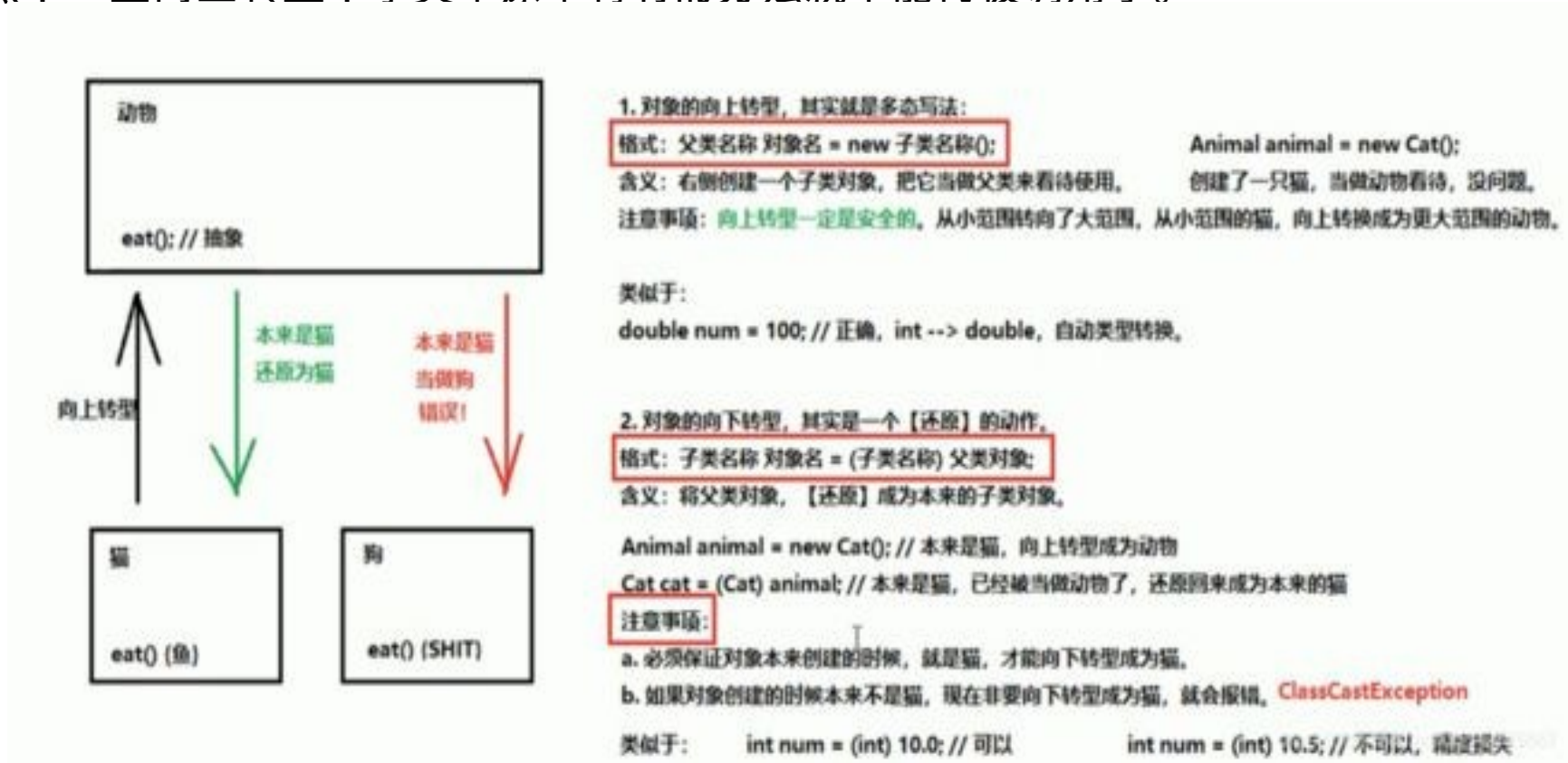
4. 向上转型

格式：父类名称 对象名 = new 子类名称 ();

含义：右侧创建一个子类对象，把它当作父类来使用。

注意：向上转型一定是安全的。

缺点：一旦向上转型，子类中原本特有的方法就不能再被调用了。



本章作业 -1

本章作业

1.(1) 定义一个 ManKind 类，包括

- 成员变量 int sex 和 int salary ；
- 方法 void manOrWorman() ：根据 sex 的值显示“ man ” (sex==1) 或者“ women ” (sex==0) ；
- 方法 void employeed() ：根据 salary 的值显示“ no job ” (salary==0) 或者“ job ” (salary!=0) 。

(2) 定义类 Kids 继承 ManKind ，并包括

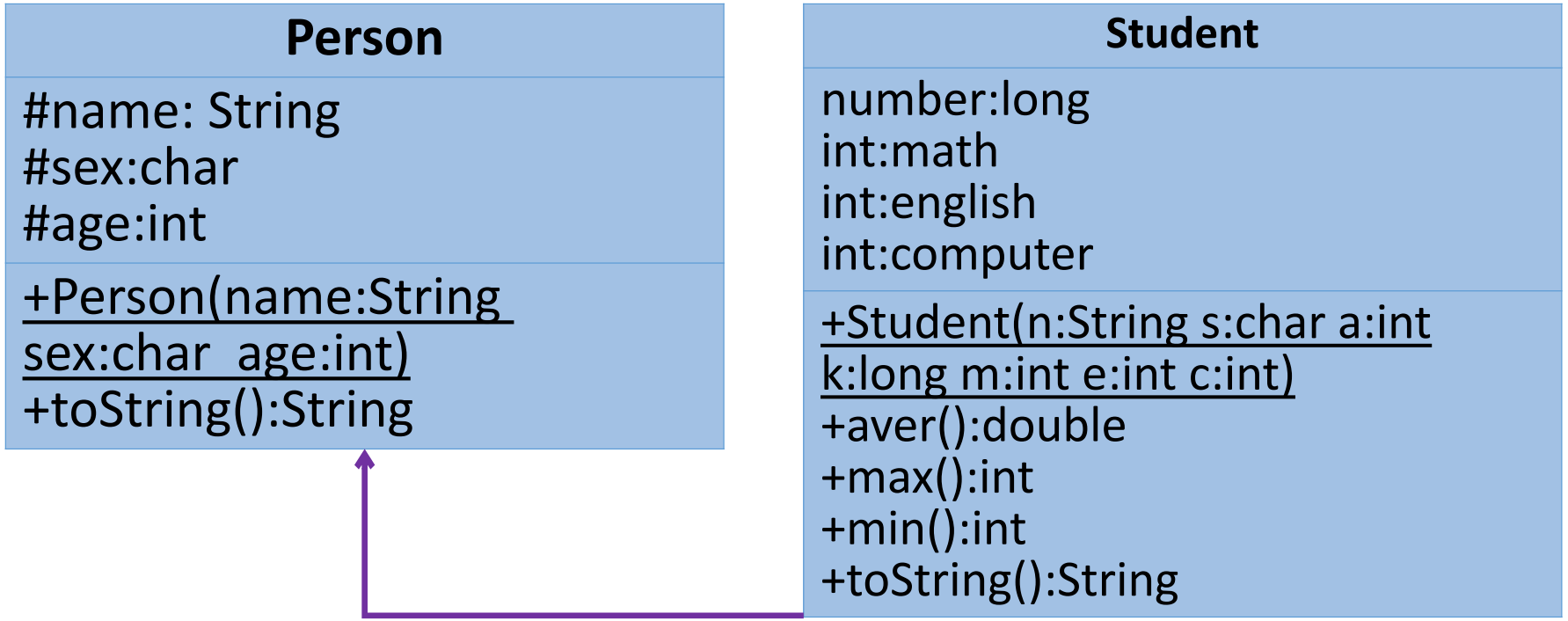
- 成员变量 int yearsOld ；
- 方法 printAge() 打印 yearsOld 的值。

(3) 在 Kids 类的 main 方法中实例化 Kids 的对象 someKid ，用该对象访问其父类的成员变量及方法。

本章作业 -2

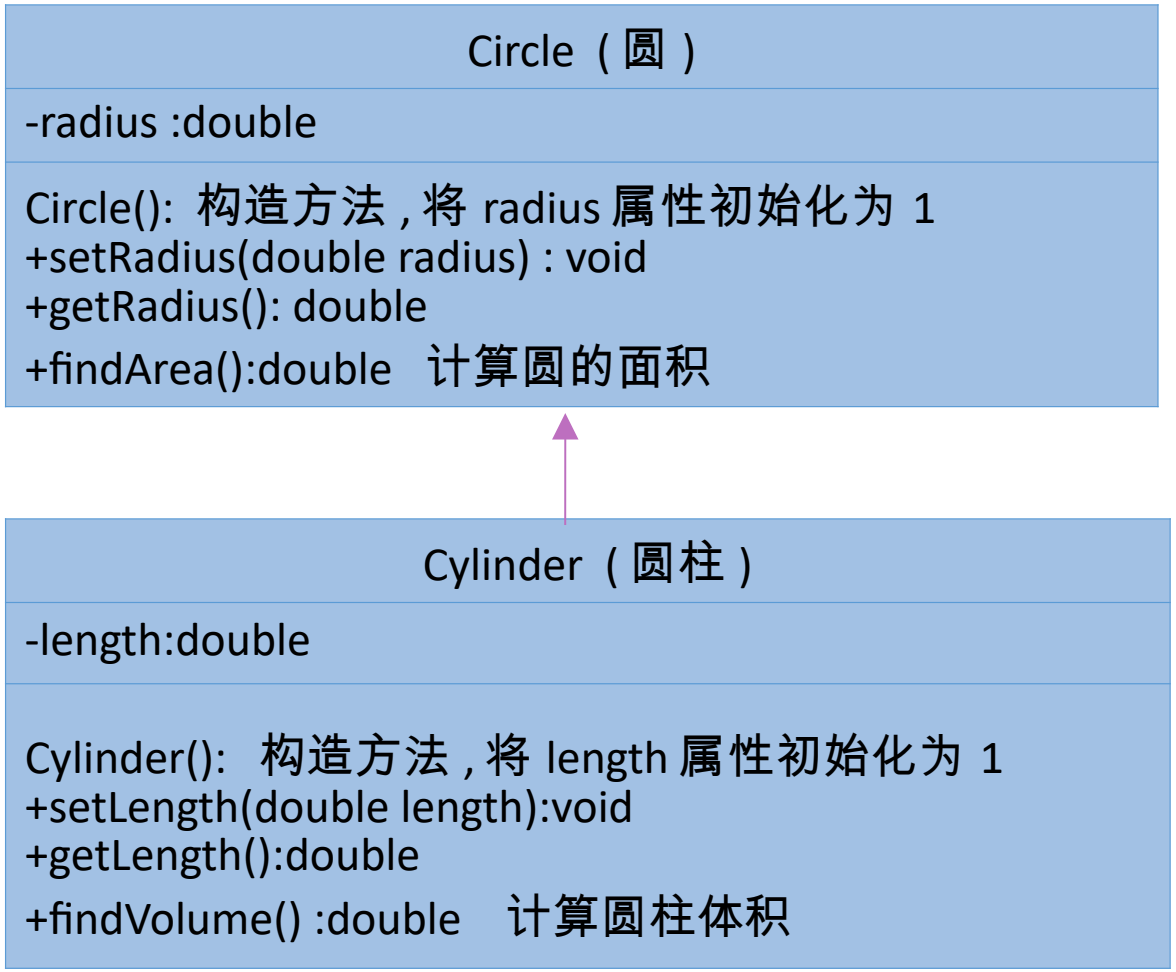
本章作业

2. 定义一个学生类 Student ，它继承自 Person 类



本章作业 -3

3. 根据下图实现类。在 TestCylinder 类中创建 Cylinder 类的对象，设置圆柱的底面半径和高，并输出圆柱的体积。



本章作业 -4

本章作业

. 修改之前练习中定义的类 Kids ，在 Kids 中重新定义 employed() 方法，覆盖父类 ManKind 中定义的 employed() 方法，输出“ Kids should study and no job. ”

本章作业 -5

1. 修改之前练习中定义的类 Kids 中 employed() 方法，在该方法中调用父类 ManKind 的 employed() 方法，然后再输出“ but Kids should study and no job. ”

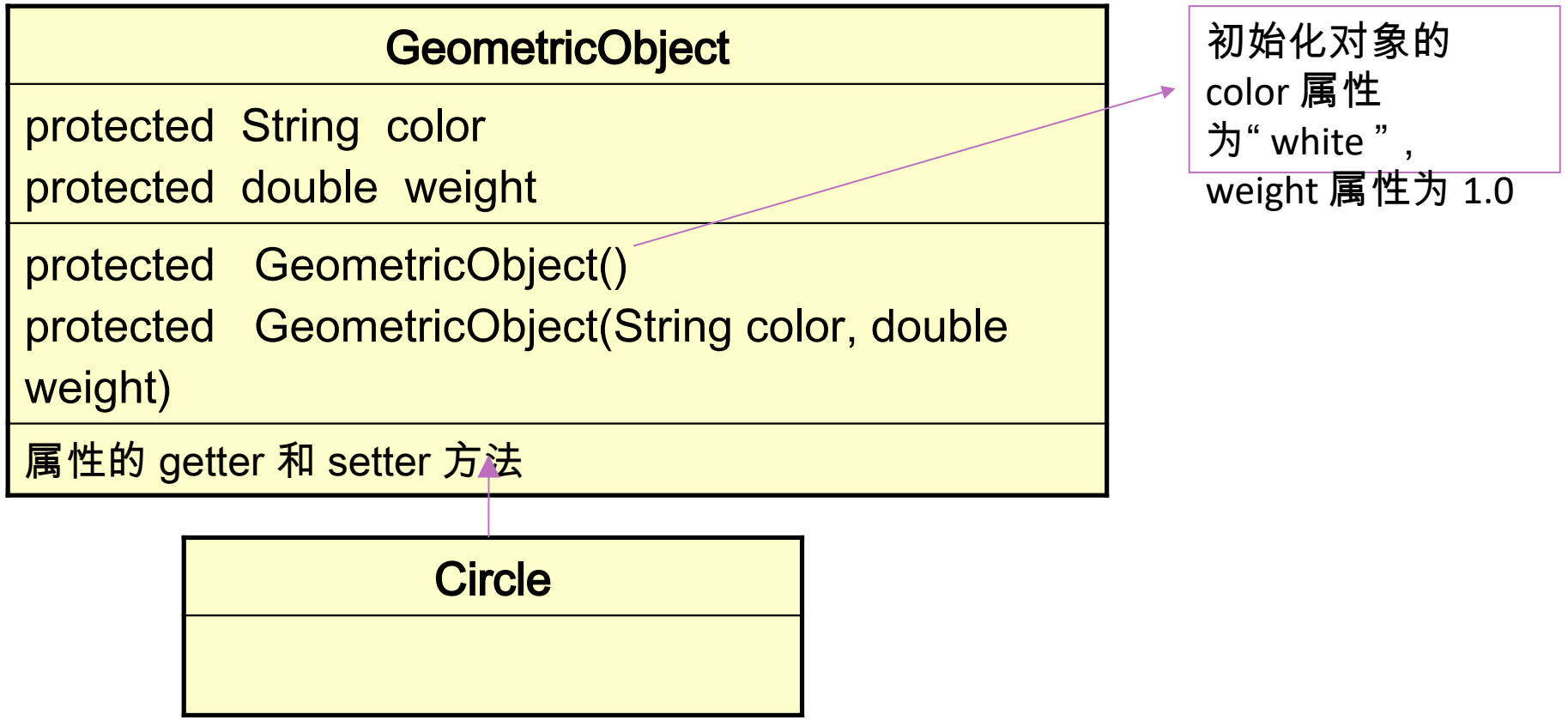
2. 修改练习 1.3 中定义的 Cylinder 类，在 Cylinder 类中覆盖 findArea() 方法，计算圆柱的表面积。考虑： findVolume 方法怎样做相应的修改？

在 TestCylinder 类中创建 Cylinder 类的对象，设置圆柱的底面半径和高，并输出圆柱的表面积和体积。

附加题：在 TestCylinder 类中创建一个 Circle 类的对象，设置圆的半径，计算输出圆的面积。体会父类和子类成员的分别调用。

本章作业 -5

●定义两个类，父类 GeometricObject 代表几何形状，子类 Circle 代表圆形。



本章作业 -6

GeometricObject
protected String color protected double weight

Circle
private double radius
public Circle() public Circle(double radius) public Circle(double radius,String color,double weight)
radius 属性的 setter 和 getter 方法 public double findArea() ：计算圆的面积 public boolean equals(Circle c)

初始化对象的 color 属性为“ white ”，weight 属性为 1.0 ， radius 属性为 0。

初始化对象的 color 属性为“ white ”，weight 属性为 1.0 ， radius 根据参数构造器确定。

重写 equals 方法，比较两个圆的半径是否相等，如相等，返回 true 。

重写 toString 方法，输出圆的半径。

写一个测试类，创建两个 Circle 对象，判断其颜色是否相等；利用 equals 方法判断其半径是否相等；利用 toString() 方法输出其半径。

本章作业 -7

本章作业

修改之前练习中定义的 Circle 类和 Cylinder 类的构造器，利用构造器参数为对象的所有属性赋初值。

作业（标注学号）发邮件到：
2230652597@qq.com