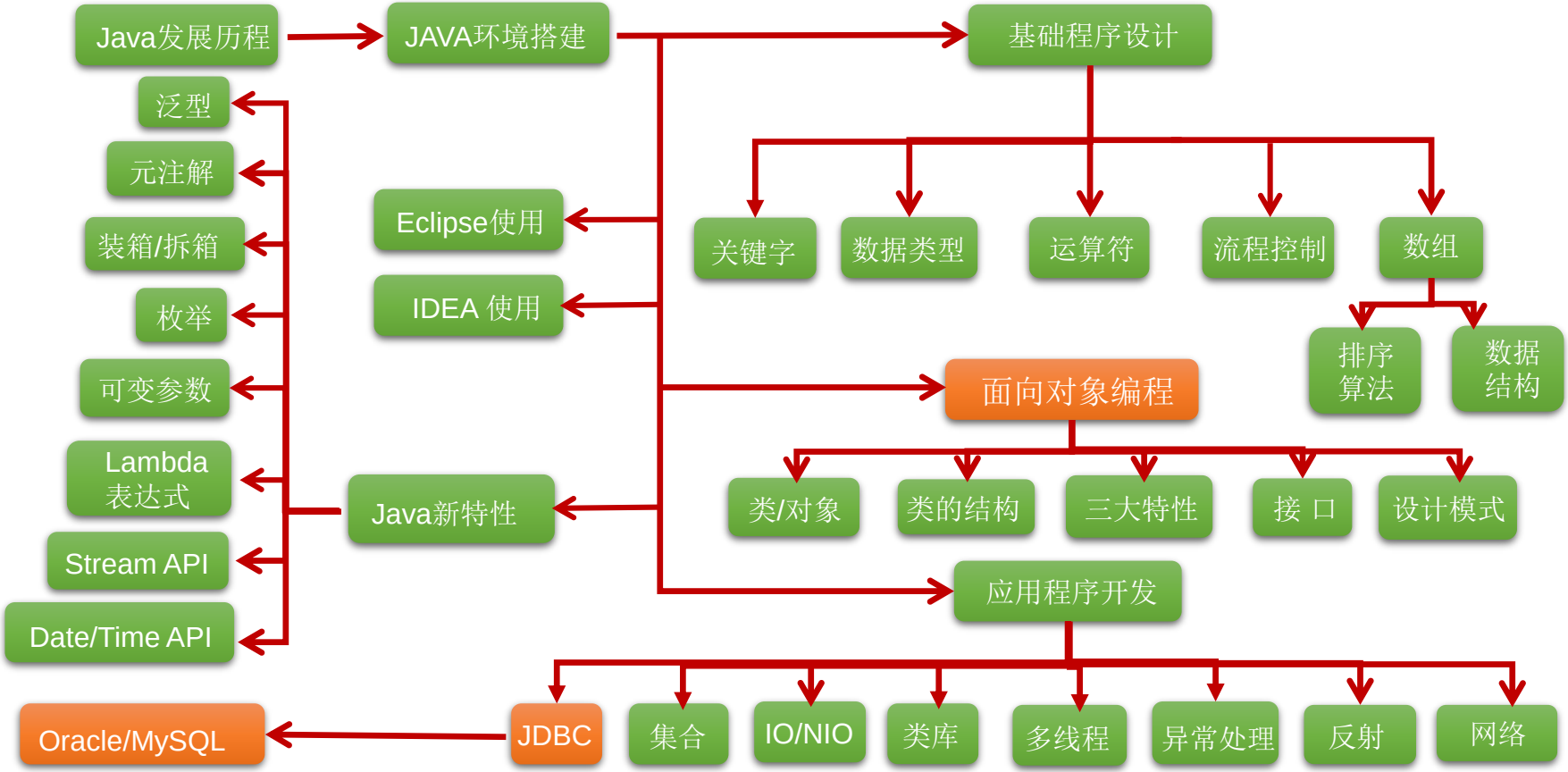
第5章
面向对象编程(中)

讲师：宋红康
新浪微博：尚硅谷-宋红康

Java基础知识图解



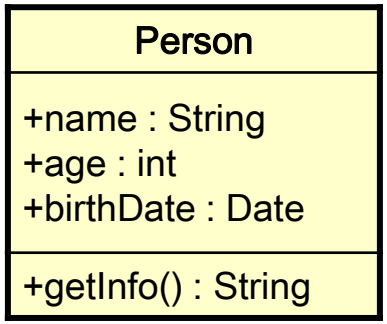
目录

尚硅谷

目录

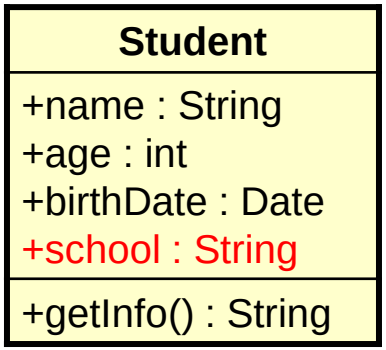
5-1 面向对象特征之二：继承性

- 为描述和处理个人信息，定义类Person:




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

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

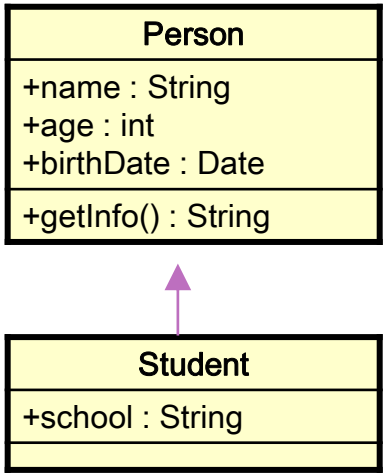
- 为描述和处理学生信息，定义类Student:




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

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

5.1 面向对象特征之二：继承性(inheritance)

- 通过继承，简化Student类的定义:

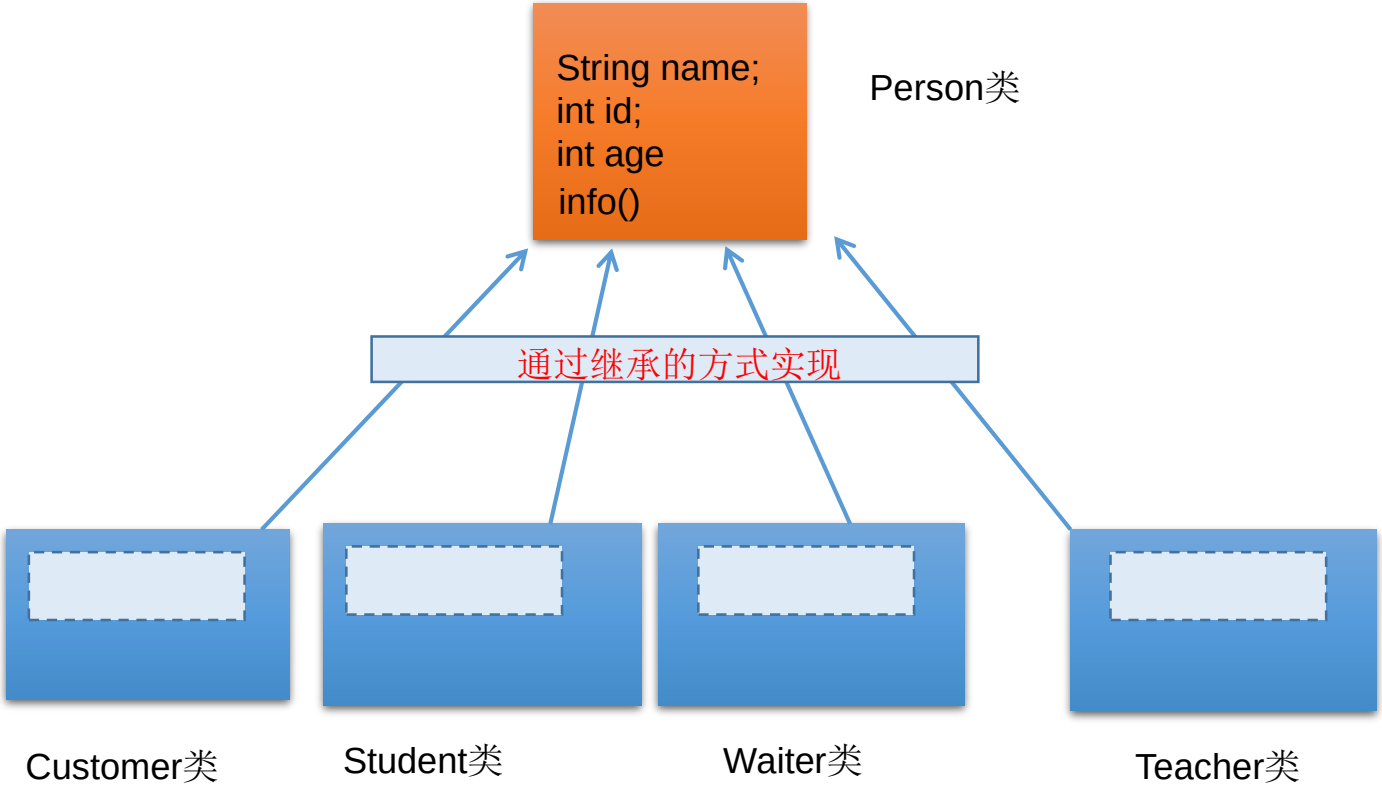



```
class Person {
    public String name;
    public int age;
    public Date birthDate;

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

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

注意，继承性保证了子类可以继承父类的所有属性与方法，包括private，但是封装性又保证了子类无法调用private方法与属性

- 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都可以使用。

5.1 面向对象特征之二：继承性(inheritance)



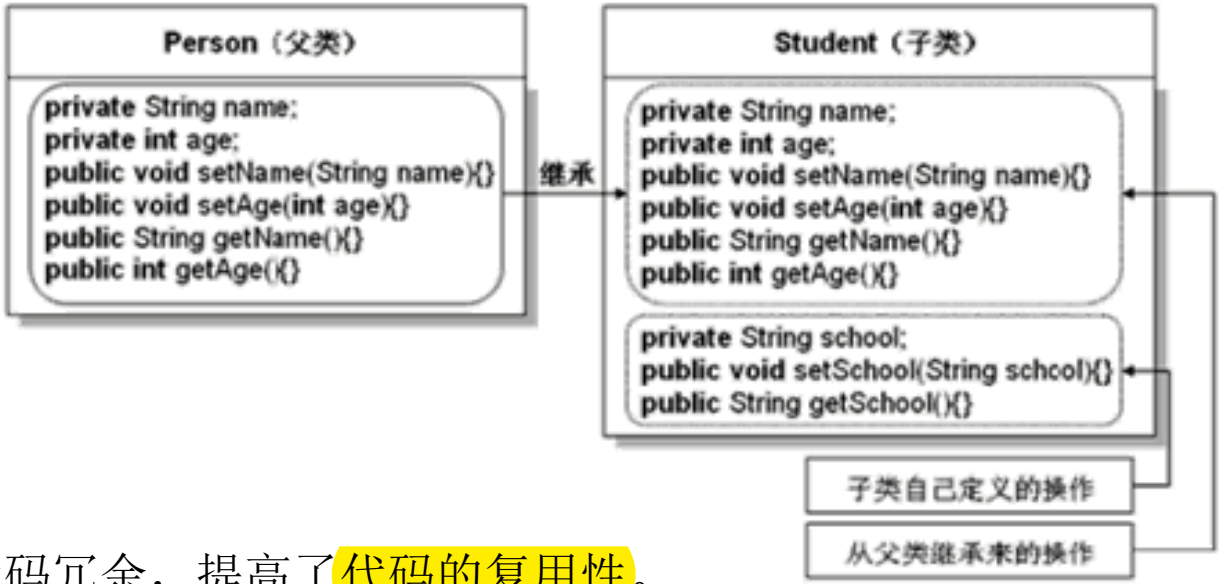
- 为什么要有继承？
  - 多个类中存在相同属性和行为时，将这些内容抽取到单独一个类中，那么多个类无需再定义这些属性和行为，只要继承那个类即可。

- 此处的多个类称为子类(派生类)，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{ }
```

5.1 面向对象特征之二：继承性(inheritance)



- 作用：
  - 继承的出现减少了代码冗余，提高了代码的复用性。
  - 继承的出现，更有利于功能的扩展。
  - 继承的出现让类与类之间产生了关系，提供了多态的前提。

- 注意：不要仅为了获取其他类中某个功能而去继承

5.1 面向对象特征之二：继承性(inheritance)

- 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”。

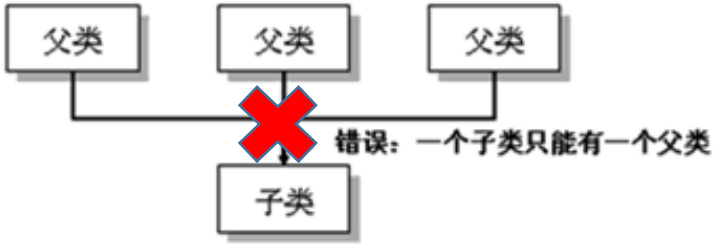
关于继承的规则：

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

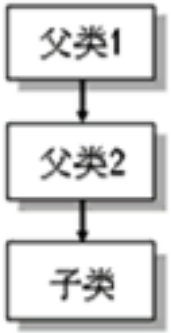
5.1 面向对象特征之二：继承性(inheritance)

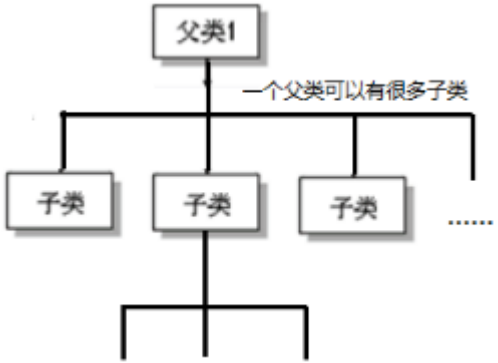
- Java只支持单继承和多层继承，不允许多重继承
  - 一个子类只能有一个父类
  - 一个父类可以派生出多个子类
    - ✓ class SubDemo extends Demo{ } //ok
    - ✓ class SubDemo extends Demo1,Demo2...//error

子类可以获取直接父类和间接父类的所有属性与方法，可以直接调用



多重继承

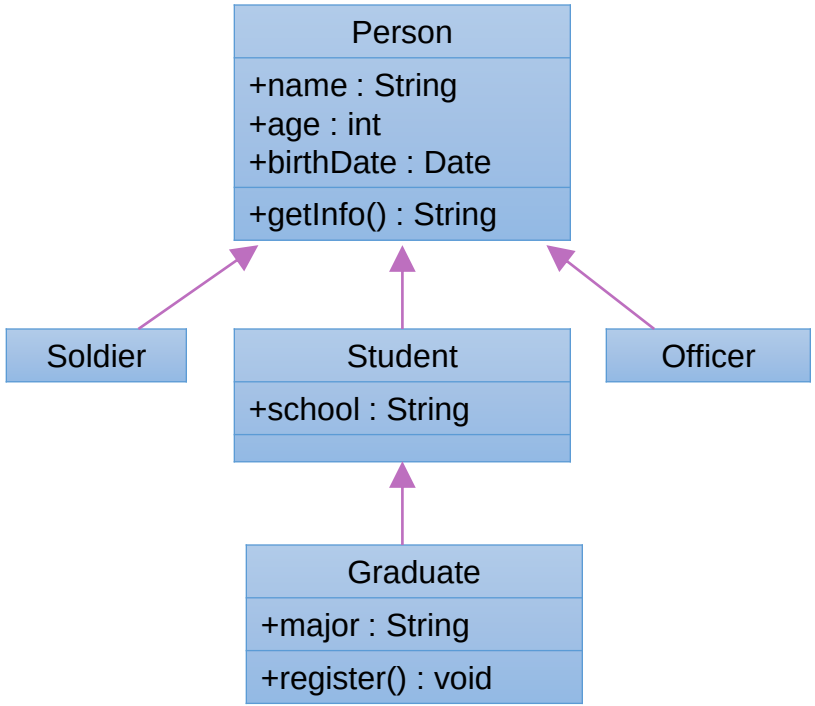




多层继承

单继承与多层继承举例

如果一个类没有显示继承父类，则它会默认继承java.lang.Object类



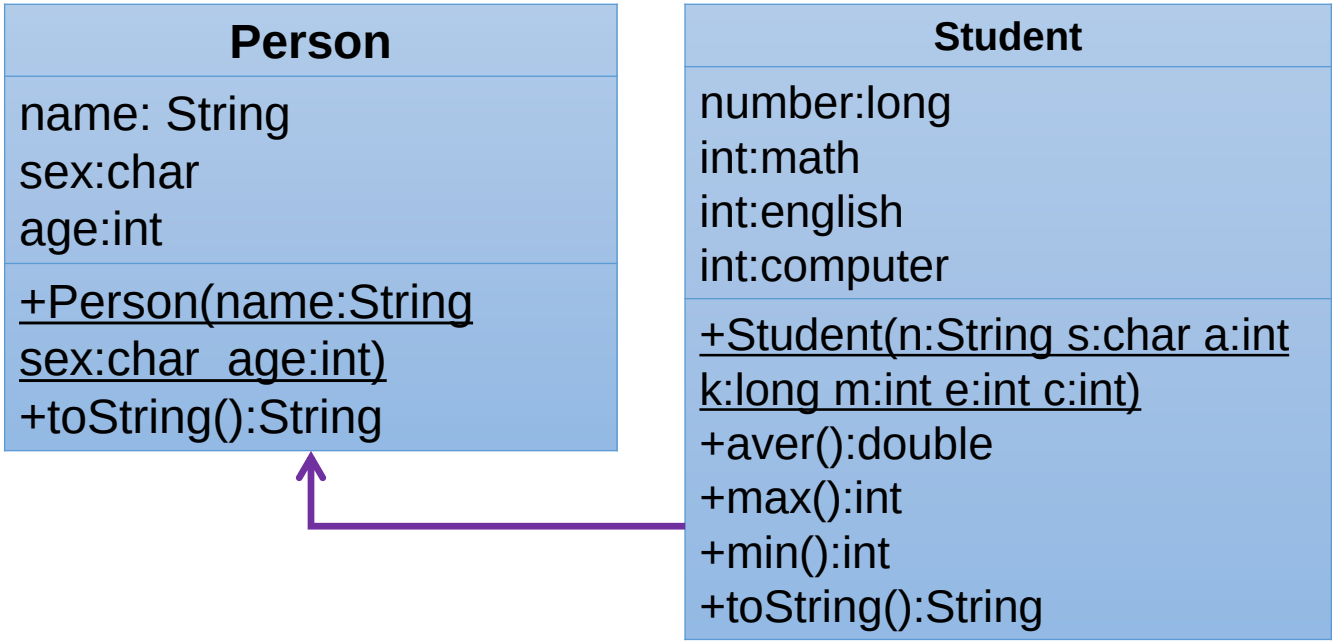
superclass

subclass / superclass

subclass

练习1

1. 定义一个学生类Student，它继承自Person类



练习1

2. (1)定义一个ManKind类，包括

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

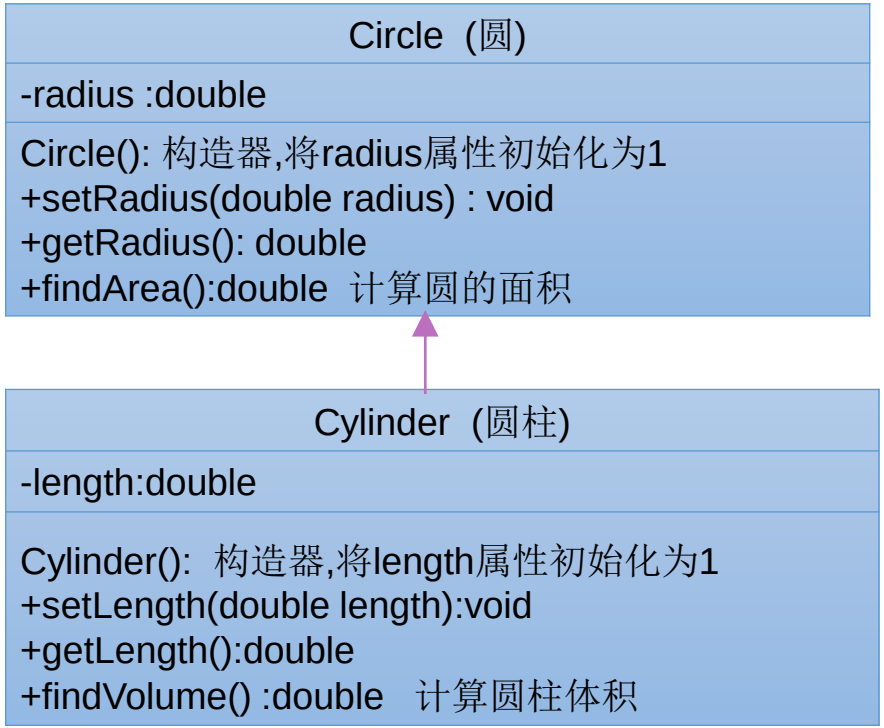
(2)定义类Kids继承ManKind，并包括

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

(3)定义类KidsTest，在类的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。

练习1

3. 根据下图实现类。在**CylinderTest**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。



5-2 方法的重写 (override/overwrite)

5.2 方法的重写(override/overwrite)

- **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，也称为方法的重置、覆盖。在程序执行时，子类的方法将覆盖父类的方法。
- **要求**：
  1. 子类重写的方法必须和父类被重写的方法具有相同的方法名称、参数列表
  2. 子类重写的方法的返回值类型不能大于父类被重写的方法的返回值类型
  3. 子类重写的方法使用的访问权限不能小于父类被重写的方法的访问权限
     - 子类不能重写父类中声明为private权限的方法**理解为privatte权限过小**
  4. 子类方法抛出的异常不能大于父类被重写方法的异常

返回值类型必须是父类返回值类型或者它的子类

如果返回值是基本数据类型，则必须一致

- **注意**：

子类与父类中同名同参数的方法必须同时声明为非static的(即为重写)，或者同时声明为static的（不是重写）。因为static方法是属于类的，子类无法覆盖父类的方法。

因为static方法不能被覆盖，它是类的方法，随着类的加载而加载

重写方法举例(1)

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

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)

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

class Child extends Parent {
//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小
        private void method1() {}
}

public class UseBoth {
        public static void main(String[] args) {
                Parent p1 = new Parent();
                Child c1 = new Child();
                p1.method1();
                c1.method1();
        }
}
```

练习2

1.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)

2. 修改练习1.2中定义的类Kids，在Kids中重新定义employeed()方法，覆盖父类ManKind中定义的employeed()方法，输出“Kids should study and no job.”

5-3 四种访问权限修饰符

5.3 四种访问权限修饰符

Java权限修饰符public、protected、 (缺省)、 private置于类的成员定义前，用来限定对象对该类成员的访问权限。

修饰符	类内部	同一个包	不同包的子类	同一个工程
private	Yes			
(缺省)	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(缺省)。

- 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

```
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

```
class Child extends Parent{              //设父类和子类在同一个包内
        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;          //     i = p.f3;              i = p.f4;
                p.fm2();         //       p.fm3();   p.fm4();
                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()
        }
}
```

访问控制分析

父类Parent和子类Child在同一包中定义时：

f2_default	fm2()_default
f3_protected	fm3()_ protected
f4_public	fm4()_ public
c1_private	cm1()_private
c2_public	cm2()_public

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

- 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可用于在子类构造器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表明调用的是父类中的成员
  - 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

对于重写方法，super会调用父类中被重写的方法

关键字super举例

```
class Person {
    protected String name = "张三";
    protected int age;
    public String getInfo() {
        return "Name: " + name + "\nage: " + age;
    }
}
class Student extends Person {
    protected String name = "李四";
    private String school = "New Oriental";
    public String getSchool() {
        return school;
    }
    public String getInfo() {
        return super.getInfo() + "\nschool: " + school;
}}
public class StudentTest {
    public static void main(String[] args) {
        Student st = new Student();
        System.out.println(st.getInfo());
}}
```

调用父类的构造器

super()

- 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。同时，只能”二选一”，且必须放在构造器的首行

  否则，会默认调用super()父类空参构造器

- 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**

在子类的多个构造器中，至少有一个构造器调用了父类构造器，不可能都是调用this构造器，因为总有一个不能调用this构造器，不然就会出现调用环

调用父类构造器举例

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

    public Person(String name, int age, Date d) {
        this.name = name;
        this.age = age;
        this.birthDate = d;
    }
    public Person(String name, int age) {
        this(name, age, null);
    }
    public Person(String name, Date d) {
        this(name, 30, d);
    }
    public Person(String name) {
        this(name, 30);
    }
}
```

调用父类构造器举例

```
public class Student extends Person {
    private String school;
    public Student(String name, int age, String s) {
        super(name, age);
        school = s;
    }
    public Student(String name, String s) {
        super(name);
        school = s;
    }
    // 编译出错: no super(),系统将调用父类无参数的构造器。
    public Student(String s) {
        school = s;
    }
}
```

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	直接访问父类中的属性
2	调用方法	访问本类中的方法，如果本类没有此方法则从父类中继续查找	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

练习3

1.修改练习1.2中定义的类Kids中employeed()方法，在该方法中调用父类ManKind的employeed()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CylinderTest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。

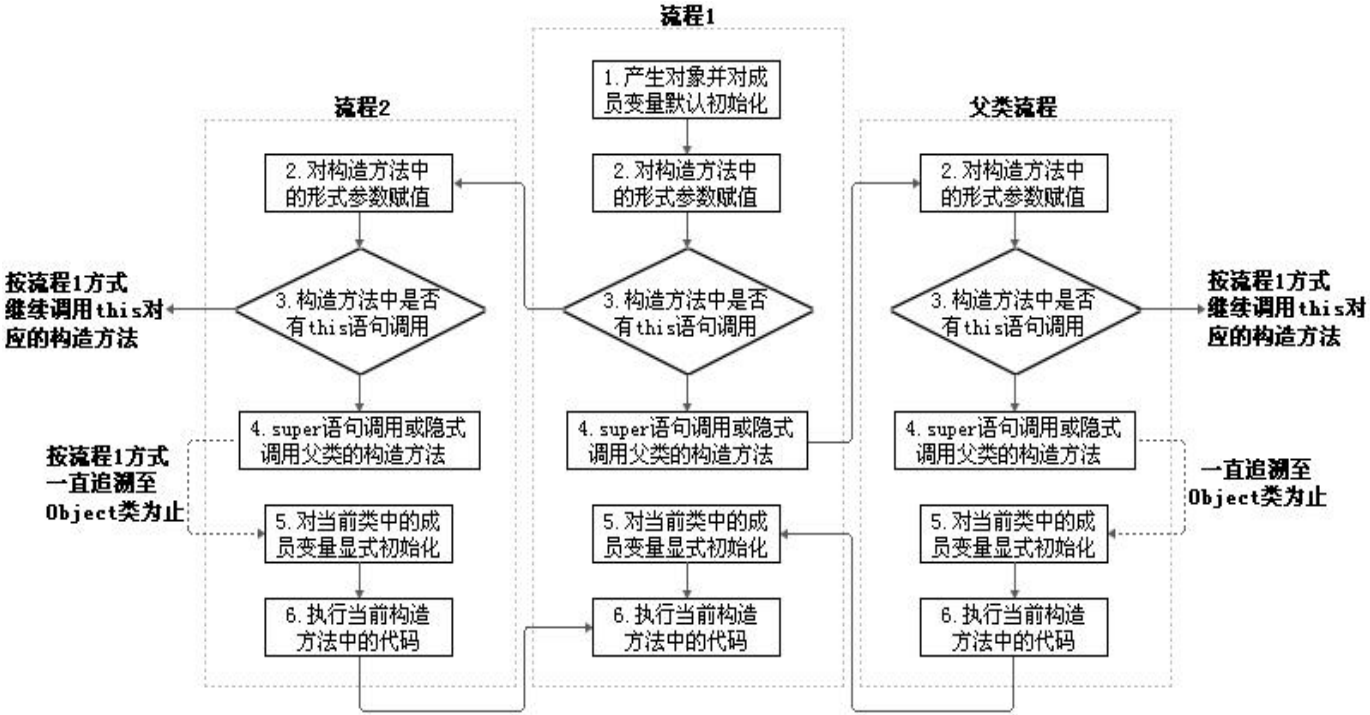
附加题：在CylinderTest类中创建一个Circle类的对象，设置圆的半径，计算输出圆的面积。体会父类和子类成员的分别调用。

课后实验题：

实验-继承&super.doc

5-5 子类对象实例化过程

5.5 子类对象的实例化过程

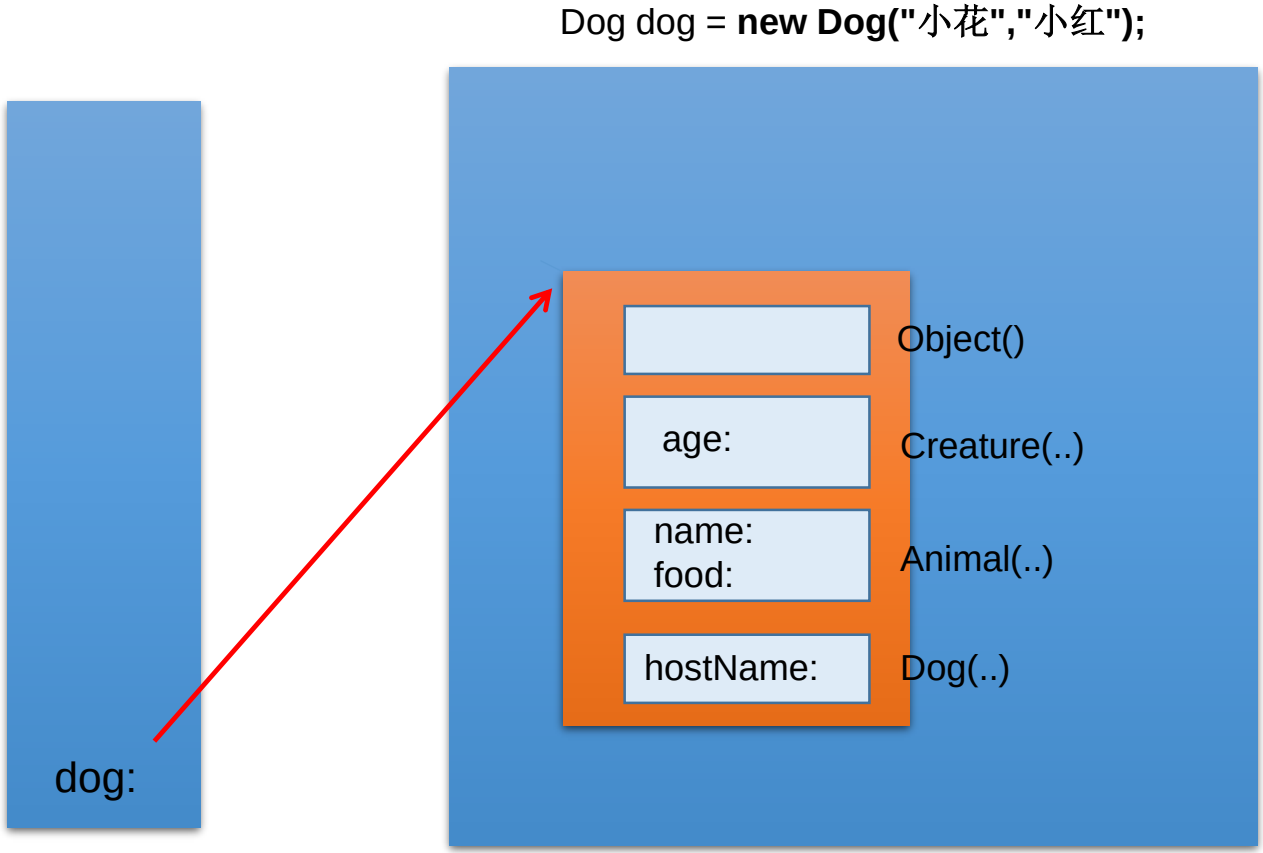


思考：

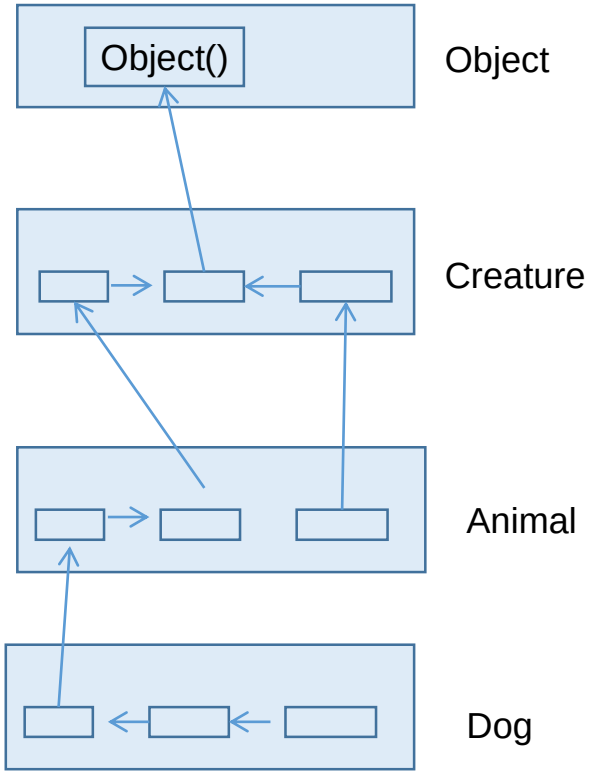
1).为什么super(…)和this(…)调用语句不能同时在一个构造器中出现？

2).为什么super(…)或this(…)调用语句只能作为构造器中的第一句出现？

5.5 子类对象的实例化过程



5.5 子类对象的实例化过程



this(…) 或 super(…) 或 super()

子类对象在实例化过程中，不管是调用自己的构造器方法还是调用父类的构造器方法，最终都会一层层地调用父类的构造器方法，直到最后的Object类的空参构造器中，这样，就会将所有父类的属性和方法加载到子类对象的内存中，但是并没有创建父类对象。

```
class Creature {
    public Creature() {
        System.out.println("Creature无参数的构造器");
}}
class Animal extends Creature {
    public Animal(String name) {
        System.out.println("Animal带一个参数的构造器，该动物的name为" + name);
    }
    public Animal(String name, int age) {
        this(name);
        System.out.println("Animal带两个参数的构造器，其age为" + age);
}}
public class Wolf extends Animal {
    public Wolf() {
        super("灰太狼", 3);
        System.out.println("Wolf无参数的构造器");
    }
    public static void main(String[] args) {
        new Wolf();
}}
```

练习4

修改练习1.3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。

5-6 面向对象特征之三：多态性

5.6 面向对象特征之三：多态性

- 多态性，是面向对象中最重要的概念，在Java中的体现：

  对象的多态性：父类的引用指向子类的对象

  - 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**和**运行时类型**。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决定。简称：编译时，看左边；运行时，看右边。

  - 若编译时类型和运行时类型不一致，就出现了对象的多态性(Polymorphism)
  - 多态情况下，“看左边”：看的是父类的引用（父类中不具备子类特有的方法）

    “看右边”：看的是子类的对象（实际运行的是子类重写父类的方法）

对于属性而言是没有多态性的，声明什么类型就是什么类型，声明了父类，就只能使用父类的属性

5.6 面向对象特征之三：多态性

- 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)。**

- 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。

多态性应用举例

- 方法声明的形参类型为父类类型，可以使用子类的对象作为实参调用该方法

```
public class Test {
    public void method(Person e) {
        // ……
        e.getInfo();
    }
    public static void main(Stirng args[]) {
        Test t = new Test();
        Student m = new Student();
        t.method(m); // 子类的对象m传送给父类类型的参数e
    }
}
```

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();
```

多态情况下，只能调用重写的方法

- 虚拟方法调用(多态情况下)

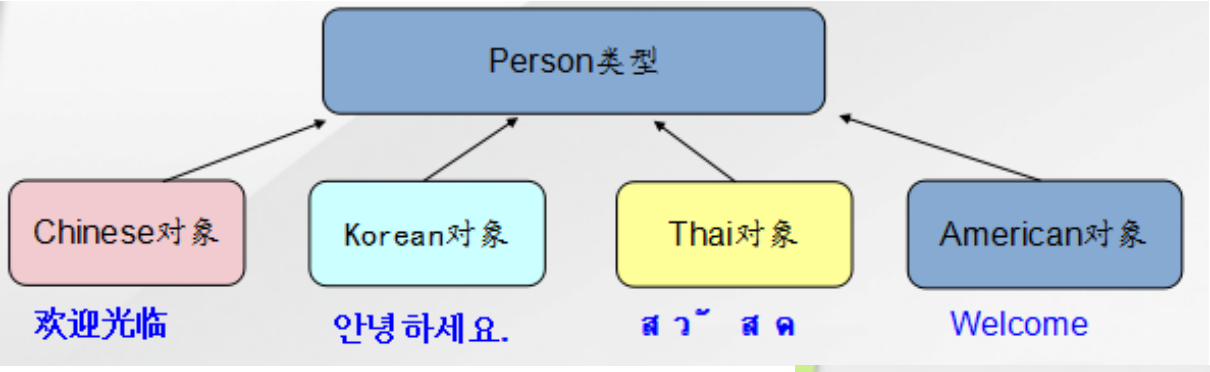
子类中定义了与父类同名同参数的方法，在多态情况下，将此时父类的方法称为虚拟方法，父类根据赋给它的不同子类对象，动态调用属于子类的该方法。这样的方法调用在编译期是无法确定的。

```
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

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

虚拟方法调用举例：



前提：

Person类中定义了welcome()方法，各个子类重写了welcome()。

执行：

多态的情况下，调用对象的welcome()方法，实际执行的是子类重写的方法。

小结：方法的重载与重写

1. 二者的定义细节：略

重载没有多态性，重写具有多态性

2. 从编译和运行的角度看：

重载，是指允许存在多个同名方法，而这些方法的参数不同。编译器根据方法不同的参数表，对同名方法的名称做修饰。对于编译器而言，这些同名方法就成了不同的方法。它们的调用地址在编译期就绑定了。Java的重载是可以包括父类和子类的，即子类可以重载父类的同名不同参数的方法。

所以：对于重载而言，在方法调用之前，编译器就已经确定了所要调用的方法，这称为“早绑定”或“静态绑定”；

而对于多态，只有等到方法调用的那一刻，解释运行器才会确定所要调用的具体方法，这称为“晚绑定”或“动态绑定”。

引用一句Bruce Eckel的话：“不要犯傻，如果它不是晚绑定，它就不是多态。”

多态小结

- **多态作用：**
  - 提高了代码的通用性，常称作接口重用
- **前提：**
  - 需要存在继承或者实现关系
  - 有方法的重写
- **成员方法：**
  - 编译时：要查看引用变量所声明的类中是否有所调用的方法。
  - 运行时：调用实际new的对象所属的类中的重写方法。
- **成员变量：**
  - 不具备多态性，只看引用变量所声明的类。

instanceof 操作符

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
}
```

使用场景：
在向下转型之前，首先，使用instanceof判断要转型的x类与要强转为的类型，如果返回true，则进行强转

```
if(x instanceof Woman){ p2 = (Woman)x;}
```

即防止向下转型时，子类中特有方法调用出现异常

instanceof操作符的出现是为了防止强转类型的异常，因为在强制类型转换时，直接将该对象的类型A转换为类型B，此时是不会报异常的，因为它所做的操作只是将内存地址进行了修改，内存地址由两部分组成，前一部分是类型，后一部分是地址。强转之后，当我们调用强转后的类型中特有的方法或属性时，就会出现异常。因此使用instanceof来检查类型，避免强转后可能的异常

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(强制类型转换)实现
  - 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

关于是否可以强制类型转换，在构建该对象时右侧使用的类型A是否加载了后续需要进行强制类型转换类型B的属性和方法，即A是否是B以及B的子类，如果是则可以进行类型转换。

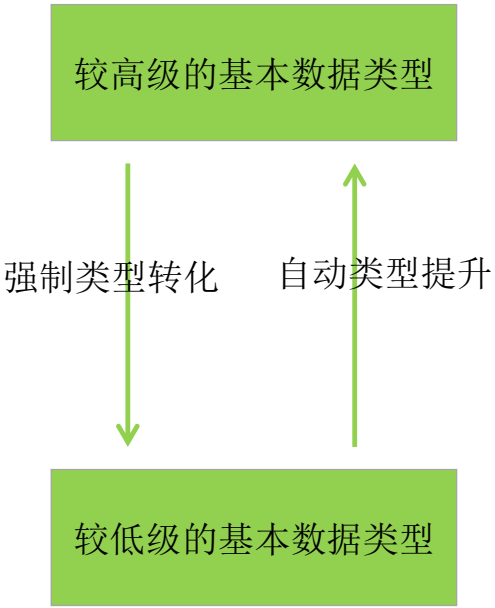
对象类型转换举例

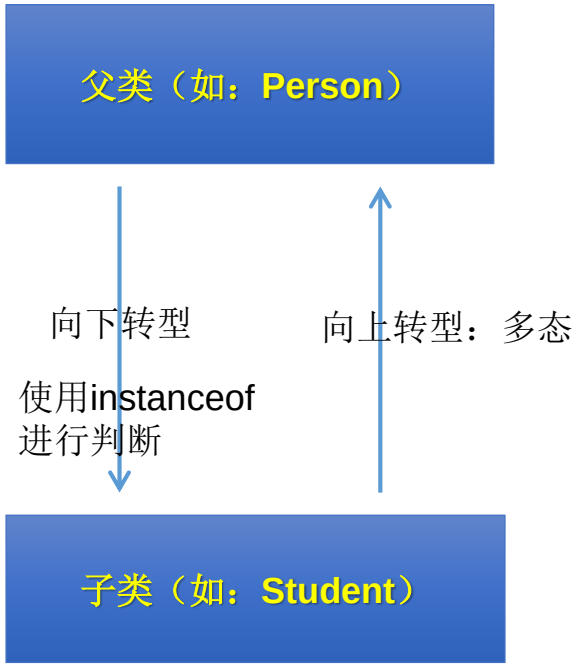
```
public class ConversionTest {
    public static void main(String[] args) {
        double d = 13.4;
        long l = (long) d;
        System.out.println(l);
        int in = 5;
        // boolean b = (boolean)in;
        Object obj = "Hello";
        String objStr = (String) obj;
        System.out.println(objStr);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

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(String[] args){
        Test t = new Test();
        Student m = new Student();
        t.method(m);
    }
}
```

5.6 面向对象特征之三：多态性





练习：继承成员变量和继承方法的区别

```java
class Base {
    int count = 10;
    public void display() {
        System.out.println(this.count);
    }
}
```

```java
class Sub extends Base {
    int count = 20;
    public void display() {
        System.out.println(this.count);
    }
}
```

```java
public class FieldMethodTest {
    public static void main(String[] args){
        Sub s = new Sub();
        System.out.println(s.count);
        s.display();
        Base b = s;
        System.out.println(b == s);
        System.out.println(b.count);
        b.display();
    }
}
```

●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

练习5

```
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InstanceTest 类，在类中定义方法method(Person e);

在method中:

(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：

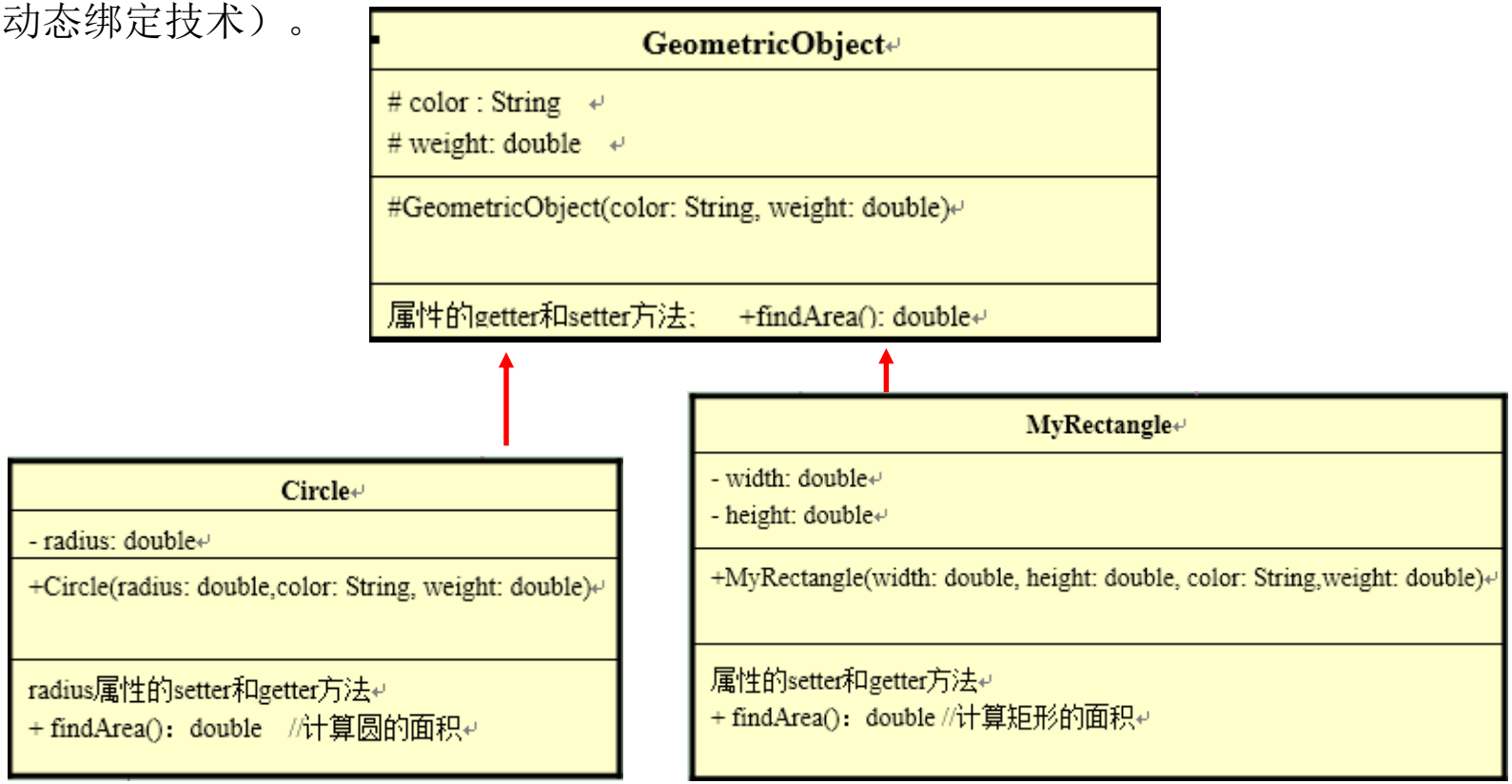
“a graduated student”

“a student”

“a person”

练习6

定义三个类，父类GeometricObject代表几何形状，子类Circle代表圆形，MyRectangle代表矩形。定义一个测试类GeometricTest，编写equalsArea方法测试两个对象的面积是否相等（注意方法的参数类型，利用动态绑定技术），编写displayGeometricObject方法显示对象的面积（注意方法的参数类型，利用动态绑定技术）。



面试题：

多态是编译时行为还是运行时行为？

如何证明？

证明见：InterviewTest.java

拓展问题：InterviewTest1.java

5-7 Object类的使用

5.7 Object 类的使用

- 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java.lang.Object类

```
public class Person {
…
}
```

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类中的主要结构

NO.	方法名称	类型	描述
1	public Object()	构造	构造器
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

finalize()方法在该对象被GC之前调用，当我们显示通过System.gc()进行垃圾回收时，会通知被回收对象调用finalize()方法，因此我们可以重写该方法，在回收前进行日志记录等。

==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。 即比较引用的地址

```
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

Object中的equals方法就是==比较，如果自定义的类中没有重写equals方法，那么它使用的就是Object类中的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。
- 当自定义使用equals()时，可以重写。用于比较两个对象的“内容”是否都相等

String类中首先比较地址是否相等，再比较内容是否相等

Date类中直接比较两个时间是否一致

File类是比较文件的字节内容是否相同

注意String常量池是共享的
但是 new String()变量则不是,
因此String常量使用==是true,
String变量使用==则是false

重写equals()方法的原则

- 对称性：如果x.equals(y)返回是“true”，那么y.equals(x)也应该返回是“true”。
- 自反性：x.equals(x)必须返回是“true”。
- 传递性：如果x.equals(y)返回是“true”，而且y.equals(z)返回是“true”，那么z.equals(x)也应该返回是“true”。
- 一致性：如果x.equals(y)返回是“true”，只要x和y内容一直不变，不管你重复x.equals(y)多少次，返回都是“true”。
- 任何情况下，x.equals(null)，永远返回是“false”；x.equals(和x不同类型的对象)永远返回是“false”。

面试题：==和equals的区别

- 从我面试的反馈，85%的求职者“理直气壮”的回答错误...

1 == 既可以比较基本类型也可以比较引用类型。对于基本类型就是比较值，对于引用类型就是比较内存地址

2 equals的话，它是属于java.lang.Object类里面的方法，如果该方法没有被重写过默认也是==;我们可以看到String等类的equals方法是被重写过的，而且String类在日常开发中用的比较多，久而久之，形成了equals是比较值的错误观点。

3 具体要看自定义类里有没有重写Object的equals方法来判断。

4 通常情况下，重写equals方法，会比较类中的相应属性是否都相等。

练 习7

```
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.util.Date()); //编译不通过
```

练 习8

1.编写Order类，有int型的orderId，String型的orderName，相应的getter()和setter()方法，两个参数的构造器，重写父类的equals()方法：public boolean equals(Object obj)，并判断测试类中创建的两个对象是否相等。

1.先用==比较地址是否相等
2. 再通过instanceof检查是否可以向下类型转换
3. 如果可以，则比较每个属性是否相等
4. 否则，返回true

2.请根据以下代码自行定义能满足需要的MyDate类,在MyDate类中覆盖equals方法，使其判断当两个MyDate类型对象的年月日都相同时，结果为true，否则为false。 public boolean equals(Object o)

```
public class EqualsTest {
    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。 getClass().getName() + "@" + Integer.toHexString(hashCode());
- 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());
```

输出一个对象的引用时，就是调用该对象的toString()方法

- 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());
```

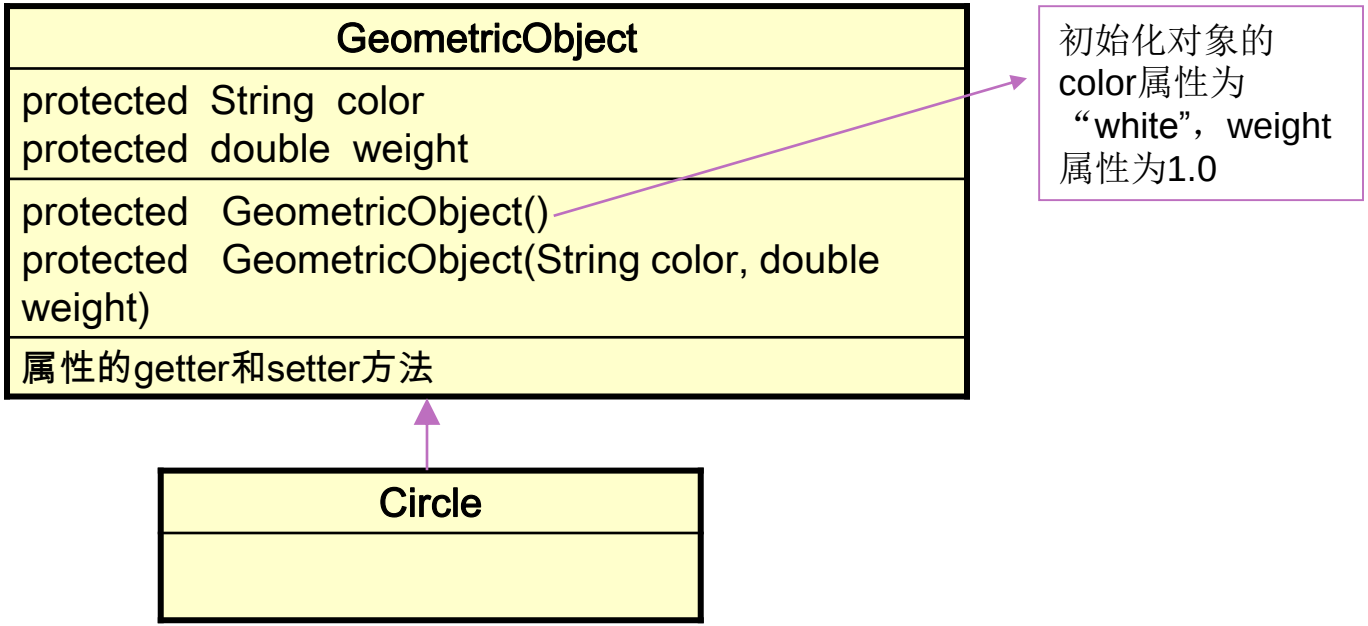
- 基本类型数据转换为String类型时，调用了对应包装类的toString()方法
  - int a=10; System.out.println("a="+a);

【面试题】

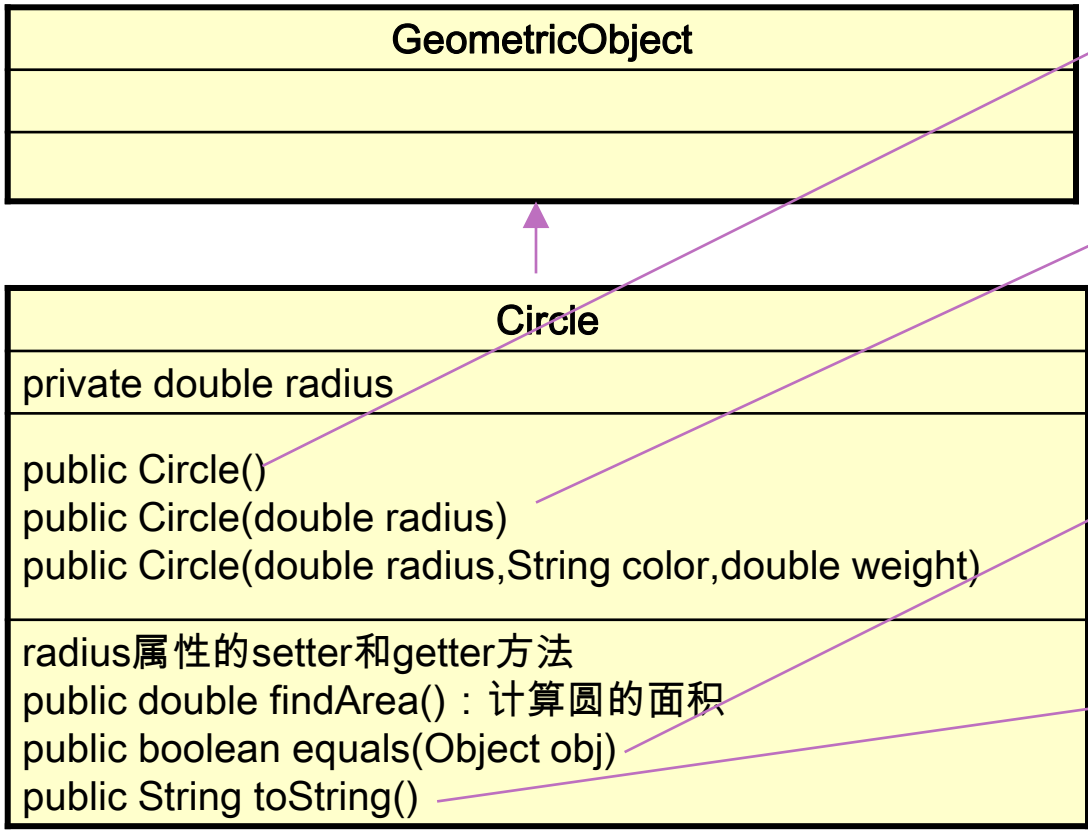
```
public void test() {
    char[] arr = new char[] { 'a', 'b', 'c' };
    System.out.println(arr);//

    int[] arr1 = new int[] { 1, 2, 3 };
    System.out.println(arr1);//

    double[] arr2 = new double[] { 1.1, 2.2, 3.3 };
    System.out.println(arr2);//
}
```

练习9

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



练习9



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。

5-8 包装类的使用

5.8 包装类(Wrapper)的使用

- 针对八种基本数据类型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，Java才是真正的面向对象

基本数据类型没有类的特点，很多情况无法使用，将它们封装为类之后，就可以将用在其他类中

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

数值型包装类都有一个共同的父类Number。

父类:Number

5.8 包装类(Wrapper)的使用

- **基本数据类型包装成包装类**的实例 ---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("4.56");

    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
  - 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。

5.8 包装类(Wrapper)的使用

- 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：

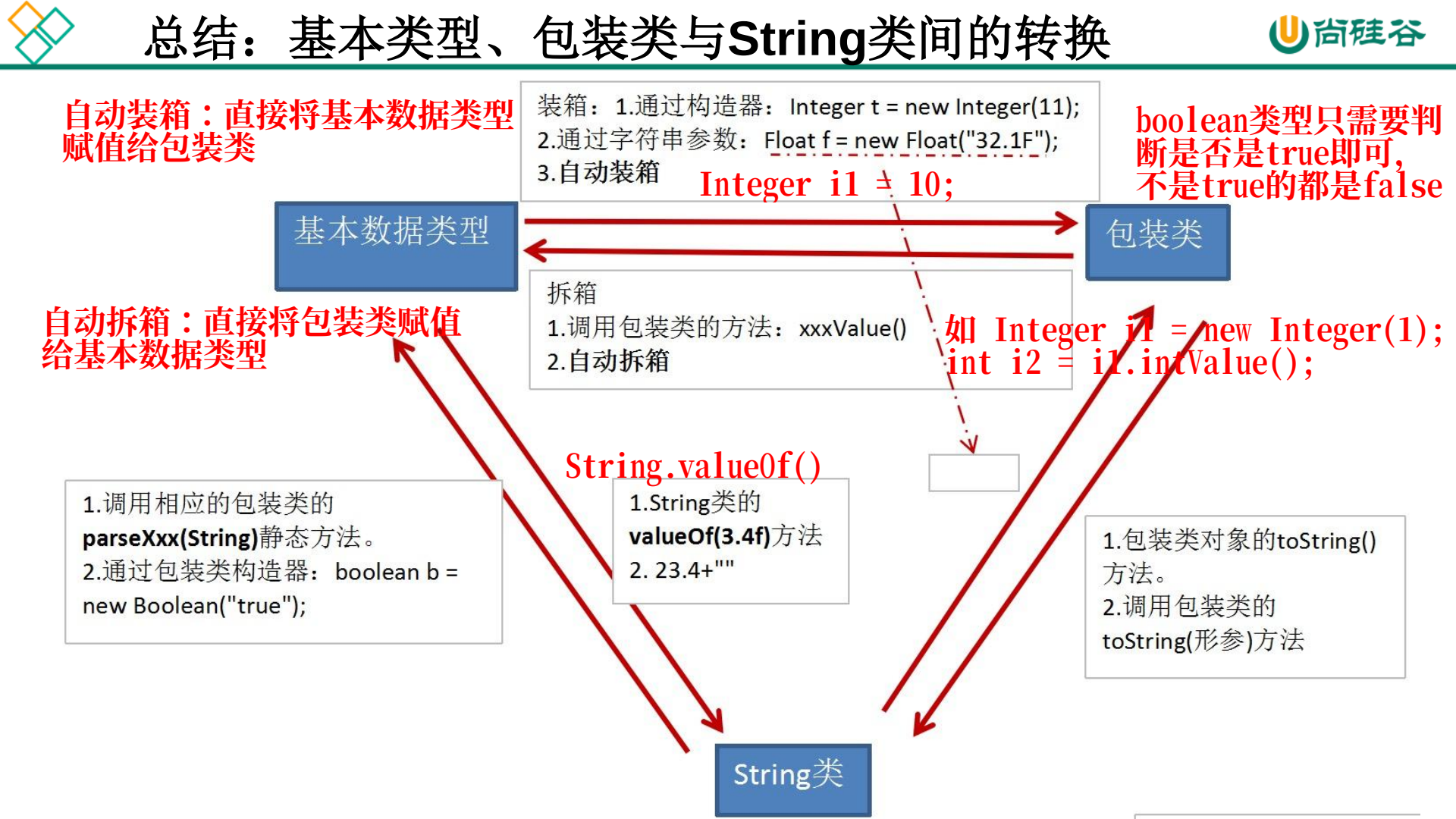
    Float f = Float.parseFloat(“12.1”);

- 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 + “”

总结：基本类型、包装类与String类间的转换



包装类用法举例

```
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=“4.56”;
double ds=Double.parseDouble(s2);   //将字符串转换成数字
```

包装类的用法举例

- **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
```

【面试题】

如下两个题目输出结果相同吗？各是什么：

三元运算符中类型必须相等，因此前面一个位Integer，后一个为Double,Integer需要提升为Double，因此结果是1.0

```
Object o1 = true ? new Integer(1) : new Double(2.0);
System.out.println(o1);//
```

```
Object o2;
if (true)
    o2 = new Integer(1);
else
    o2 = new Double(2.0);
System.out.println(o2);//
```

【面试题】

```
public void method1() {
    Integer i = new Integer(1);
    Integer j = new Integer(1);
    System.out.println(i == j);

    Integer m = 1;
    Integer n = 1;
    System.out.println(m == n);//

    Integer x = 128;
    Integer y = 128;
    System.out.println(x == y);//
}
```

true

在Integer中，有一个IntegerCache数组，缓存着-128-127，以供高效调用，如果是在这个区间中的数，则直接获取，不用新建，如果是区间之外的数，则需要新建。

false

练习10

利用Vector代替数组处理：从键盘读入学生成绩（以负数代表输入结束），找出最高分，并输出学生成绩等级。

- 提示：数组一旦创建，长度就固定不变，所以在创建数组前就需要知道它的长度。而向量类java.util.Vector可以根据需要动态伸缩。
- 创建Vector对象：Vector v=new Vector();
- 给向量添加元素：v.addElement(Object obj); //obj必须是对象
- 取出向量中的元素：Object obj=v.elementAt(0);
  - ✓注意第一个元素的下标是0，返回值是Object类型的。
- 计算向量的长度：v.size();
- 若与最高分相差10分内：A等；20分内：B等；
  30分内：C等；其它：D等


尚硅谷
让天下没有难学的的技术