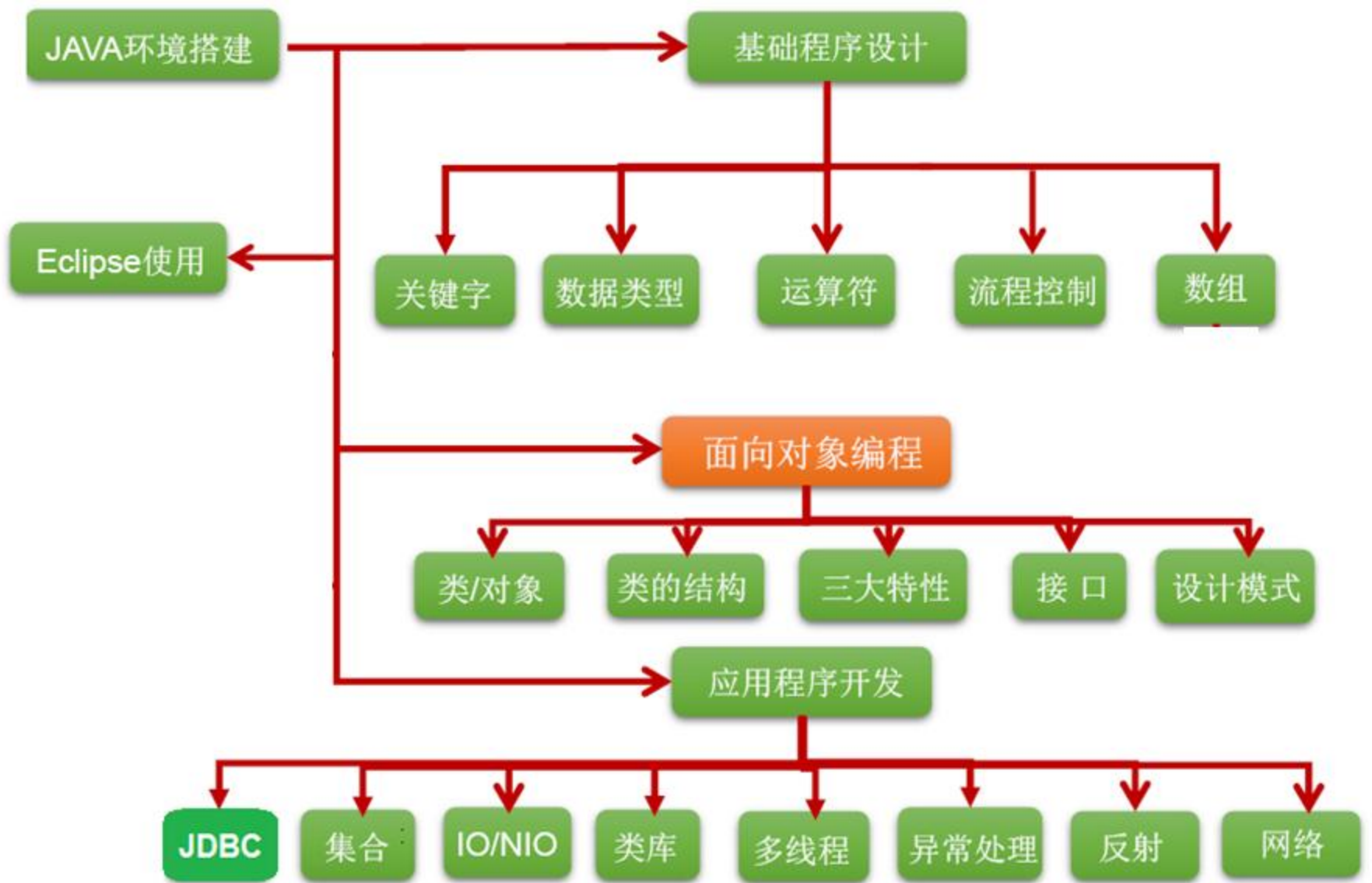


Java语言与系统设计

第6讲 继承与多态

- ❑ 继承
 - ❑ 重写
 - ❑ 访问控制
 - ❑ super关键字
 - ❑ 子类对象
 - ❑ 多态
 - ❑ Object类
 - ❑ 包装类
-



1. 继承

- 类是一种抽象数据类型，是对具有共同属性和行为的对象（事物）的抽象描述，事物的属性被描述为类的数据成员，事物的行为被描述为类的函数成员（成员函数）。
 - 能否扩展或修改一种类而得到另一种类？
-

1. 继承

- Java规定一个新类可以继承另一个类。这个新类被称为继承类、派生类或子类，而被继承的类称之为基类或父类。
- 继承类能够继承基类的全部属性和行为，在基类被定义之后，再利用基类定义派生类就很容易了，只要定义派生类中所特有属性和行为即可。
- 可以理解为：“子类is a 父类”

1. 继承

- 例如，可以用一个类来描述一般的人员，它包含姓名、性别等属性，但若定义大学生类，则仅有这些一般的属性是不够的，还要有学校、专业等属性。所以，可以把描述人员的类作为父类，把描述大学生的类作为子类。
 - 这样在描述一般人员的类之后，让大学生类继承一般人员的类，就自动把一般人员类中的所有数据成员和函数成员都继承下来，另外在大学生类中再定义大学生所特有的数据成员和函数成员即可。
-

1. 继承

- 类的继承能够充分利用已有的类，在已有类的基础上进行必要的扩充和修改就可以得到具有新功能和用途的类，从而大大简化了程序设计：
 - 继承的出现减少了代码冗余，提高了代码的复用性、可维护性和可靠性。
 - 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。
-

1. 继承

- 类就是程序设计中的组件，通过类的继承能够采用工程化的方法进行程序设计。
 - 进行程序设计（软件开发）的过程就象建造一项工程一样，能够采用一般通用的工程管理方法和制度，在总体设计的基础上，进行分工合作和流水线作业，生产出合格的软件产品投入市场。
-

1. 继承

类继承的语法规则：

<修饰词> class <派生类名> extends <父类名> {<派生类成员表>}

- <派生类名>是为新定义的类所起的名字，它是一个合法的标识符，习惯要求是第1个字母要大写。
- extends <父类名>这是定义派生类时的必选项，定义一般类（非派生类）时此项（子句、短语）被省略，<父类名>是一个已定义类的标识符，它可以是Java系统类库中的类，也可以是用户自己已经编译过的类

1. 继承

类继承的语法规则：

<修饰词> class <派生类名> extends <父类名> {<派生类成员表>}

- <派生类成员表>是新定义的派生类的类体，它包括数据成员和函数成员的定义，它可以对基类中已有成员进行重新定义，赋予新的含义和功能，也可以定义基类中不存在的成员，以反映派生类所特有的属性和行为，从而扩充基类的功能，实现派生类定义的目的。
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”。

1. 继承

派生类定义举例：

```
class Circle {                                //圆类
    double radius;                             //半径
    void setRadius(double r) {radius=r;}
    //设置半径值
    double area() {    //返回圆面积
        return Math.PI*radius*radius;}
}
```

1. 继承

在派生类球类中共包含有5个成员，其中3个从圆类继承而来，2个是新定义的成员，但可访问4个成员，因为新定义的area()成员覆盖了基类中的对应成员，使得无法访问基类中的同一成员。注意，同名属性不能覆盖，只会在子类中增加新的属性。

派生类定义举例：

```
class Sphere extends Circle {  
    //球类为圆类的派生类，圆类为基类  
    double area() { //返回表面积，重写基类的函数  
        return 4*Math.PI*radius*radius;  
    }  
    double volume() { //返回体积，新定义的函数  
        return 4*Math.PI*Math.pow(radius,3)/3;  
    } }
```

1. 继承

- 子类获取父类所有的属性和方法，但子类不能直接访问父类中私有的(private)的成员变量和方法。



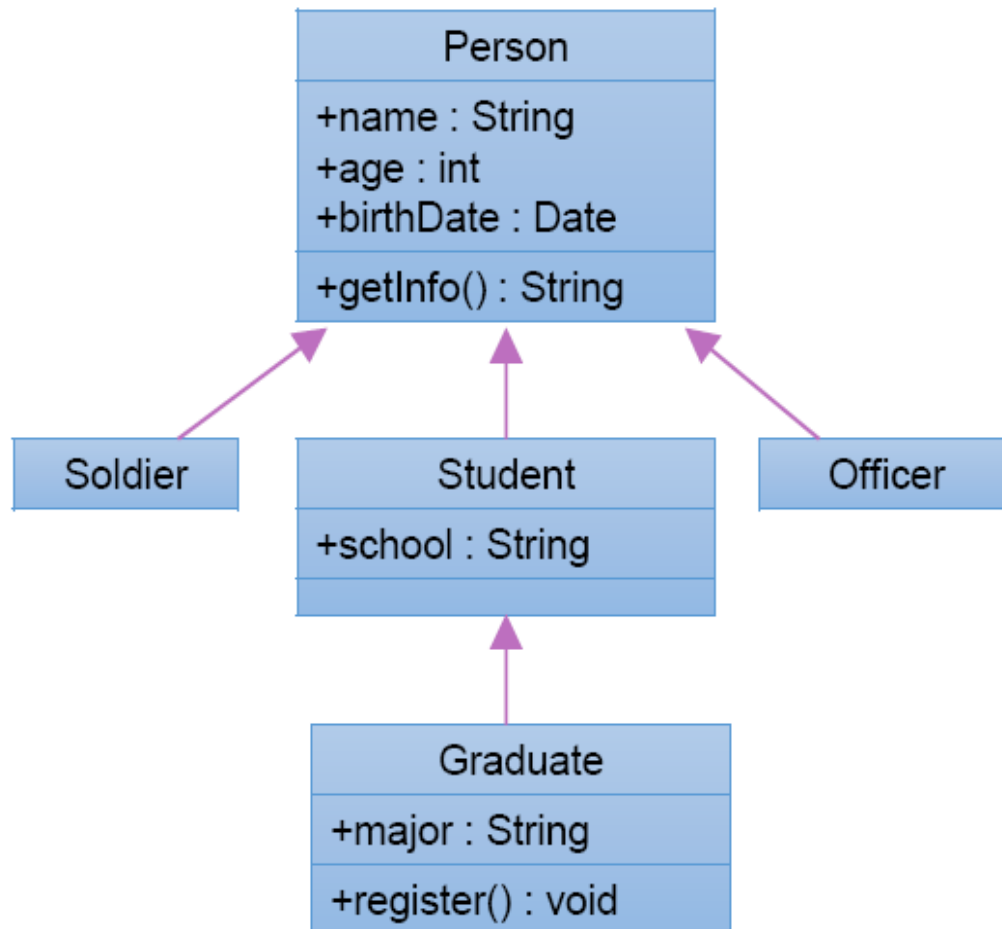
1. 继承

- ❑ 子类只能直接继承一个父类，而不允许继承多个父类，Java的类支持单继承和多层继承，不允许多重继承。

`class SubDemo extends Demo {} //ok`

`class SubDemo extends Demo1,Demo2... //error`

- ❑ 但接口可以支持多继承
-



superclass

subclass / superclass

subclass

1. 继承

- ❑ Java类的继承结构为树状结构（即层次结构），Java系统类库中的`java.lang.Object`类为整个树状结构类图的、最顶层的树根节点。
 - ❑ 若定义的是不包含有继承子句的一般类，则隐含着继承系统提供的`java.lang`包中的`Object`类。
 - ❑ 如上面举例圆类`Circle`，隐含着继承了`Object`类，而球类`Sphere`直接继承了圆类，间接继承了`Object`类。在`Object`类中提供的成员都可以在圆类和球类中使用。
-

练习

```
class A {  
    int a = 1;  
    double d = 2.0;  
    void show() {  
        System.out.println("Class A: a=" + a + "\td=" + d);  
    }  
}  
  
class B extends A {  
    float a = 3.0f;  
    String d = "Java program.";  
    void show() {  
        super.show();  
        System.out.println("Class B: a=" + a + "\td=" + d);  
    }  
}
```

(1) 若在应用程序的main 方法中有以下语句：
A a=new A();
a.show();
(2) 若在应用程序的main 方法中有以下语句：
B b=new B();
b.show();

2. 方法重写

方法的重写(override/overwrite)

- ❑ 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。
 - ❑ 需要注意的问题是：子类在重新定义父类已有的方法时，应保持与父类完全相同的方法头部声明，即应与父类有完全相同的方法名和参数列表。否则就不是方法的覆盖，而是在子类中定义了与父类无关的方法，父类的方法仍存在于子类中。
-

2. 方法重写

```
class A{  
double f(double x, double y){  
    return x+y;}  
}
```

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。

意义在于：通过隐藏成员变量和覆盖方法可以把父类的状态和行为改为自身的状态和行为，而对外又有统一的名字与接口，不失其继承性。

```
public static void main(String args [ ]) {  
A a = new A(); B b = new B();  
System.out.println(a.f(6,4)+", "+b.f(6,4));  
}}
```

2. 方法重写

方法重写的规定

- ❑ 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
 - ❑ 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
-

2. 方法重写

方法重写的规定

- ❑ 父类被重写方法返回值类型是void，子类重写方法的返回值类型也是void
 - ❑ 父类被重写方法返回值类型是A类型，子类重写方法的返回值类型是A类或A类子类
 - ❑ 父类被重写方法返回值类型是基本数据类型，子类重写方法的返回值类型是相同的基本数据类型
 - ❑ 静态static方法不能被重写
-

2. 方法重写

```
class A{
double f(double x, double y){ // 2.private ?
    return x+y;}
}

class B extends A {
double f(double x , double y){ // 1.private ? //3.int ?
    return x-y;}
}

public class TestAB {
    public static void main(String args [ ]) {
A a = new A(); B b = new B();
System.out.println(a.f(6,4)+", "+b.f(6,4));
}}
```

2. 方法重写

方法重写的总结

- 一个方法只能有一个名称，但可以有許多形态，也就是程序中可以定义多个同名的方法。多态提供了“接口与实现的分离”，将“是什么”（what）与“怎么做”分离，改善程序的组织架构及可读性，提高程序的可扩展性。
-

练习

重载 (overload)和重写(override) 的区别?

方法的重写和重载是Java多态性的不同表现。重写是父类与子类之间多态性的一种表现，重载是一个类中多态性的一种表现。如果在子类中定义某方法与其父类有相同的名称和参数，我们说该方法被重写。子类的对象使用这个方法时，将调用子类中的定义，对它而言，父类中的定义如同被"屏蔽"了。如果在一个类中定义了多个同名的方法，它们或有不同的参数个数或有不同的参数类型，则称为方法的重载。

练习

写出程序结果:

```
class Super {  
    public int get() {return 4;}  
}  
class Demo extends Super {  
    public long get() {return 5;}  
    public static void main(String[] args) {  
        Demo s = new Demo();  
        System.out.println(s.get());  
    }  
}
```

练习

```
class Demo{  
int show(int a,int b){return 0;}  
}
```

下面那些函数可以存在于Demo的子类中。

A. `public int show(int a,int b){return 0;}` //可以，覆盖。

B. `private int show(int a,int b){return 0;}` //不可以，权限不够。

C. `private int show(int a,long b){return 0;}` //可以，和父类不是一个函数。没有覆盖，相当于重载。

D. `public short show(int a,int b){return 0;}` //不可以，父类返回基本数据类型，子类重写方法的返回值类型是相同的基本数据类型。

3. 访问控制

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。

3. 访问控制

派生类对基类成员的访问限制：

- 由类成员的访问属性可知：若派生类继承的是不同包中的父类，则只能直接访问父类中属于公有(public)和保护(protected)属性的成员，不能访问私有和缺省访问属性的成员；若派生类继承的是同一包中的父类，则能够直接访问父类中属于公有(public)、保护(protected)和缺省访问属性的成员，不能访问私有成员。
-

3. 访问控制

派生类对基类成员的访问限制：

- 总之，不管继承的父类来自哪里，同派生类属于同一个包，还是不同的包，派生类中的成员函数都不能够直接访问父类中属于私有(private)的成员，包括私有的数据成员和函数成员。
 - 所以要保护类中的数据，不被外部类任意修改，使外部类只能通过构造函数初始化和专门提供的成员函数进行访问，应把类中的数据成员设置为私有访问属性，这就是类的封装性（隐藏性）的特性。
-

3. 访问控制

派生类对基类成员的访问限制:

```
public class Base {  
    int b1; //缺省  
    private int b2; //私有  
    protected int b3; //保护  
    public int b4; //公有  
}
```

```
public class Derive  
    extends Base {  
    // Derive继承Base  
    int d1; //缺省  
    private int d2; //私有  
    protected int d3; //保护  
    public int d4; //公有  
    public void fun() {...}  
    //成员函数, 访问属性任意}
```

3. 访问控制

派生类对基类成员的访问限制：

- 基类Base中的成员函数能够访问本类中定义的所有数据成员和函数成员，派生类Derive中的成员函数不仅能够访问本类中定义的所有数据成员和函数成员，而且能够把基类Base中的非私有成员都继承下来，但对基类的私有数据成员b2不能继承，对其他非私有成员既继承又能够访问。所以Derive中的成员函数能够访问自己类中的d1、d2、d3和d4数据成员以及基类Base中的b1、b3和b4数据成员。
-

3. 访问控制

```
class A{
int idA=1; //缺省、public、protected、private
int getA(){ return idA;} }
class B extends A {
}

public class TestAB {
public static void main(String args [ ]) {
B b = new B();
System.out.println(b.idA); //getA()
}}
```

4. super 关键字

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可用于在子类构造函数中调用父类的构造函数
 - ❑ 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
-

```
class A{
double f(double x, double y){ return x+y;} }
class B extends A {
double f(double x , double y){ return x-y;}
double f1(double x , double y){ return super.f(x,y);}
}
public class TestAB {
public static void main(String args [ ]) {
B b = new B();
System.out.println(b.f(6,4)+","+b.f1(6,4));
}}
```

4. super 关键字

super在子类构造函数中调用父类的构造函数

- ❑ 在派生类的构造函数包括无参构造函数和带参构造函数。
 - ❑ 在有参数构造函数中首行必须用super(参数)语句，实现对基类数据成员初始化的任务。
 - ❑ 若无参数构造函数，则可以省略super()语句，因为系统会隐含调用基类的无参构造函数，自动实现对基类成员的初始化。
-

4. super 关键字

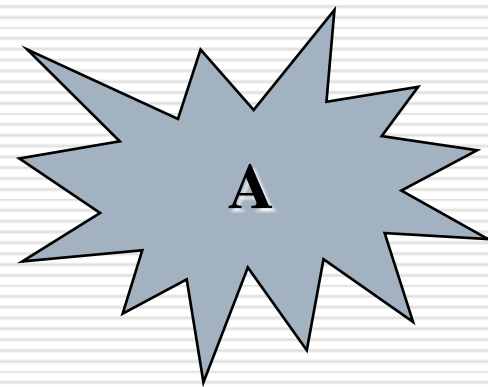
- this代表本类对象的引用，super代表父类的内存空间的标识

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

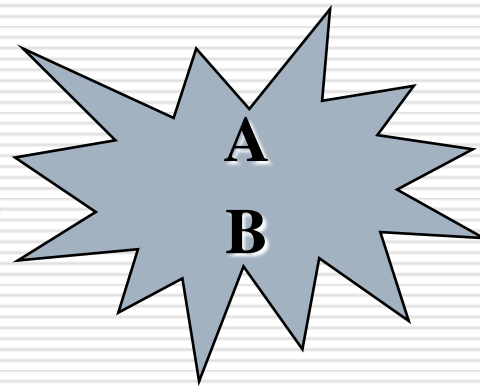
如果子类自己没有构造方法，则它将继承父类的无参数构造方法作为自己的构造方法；

```
class A{
    public A( ) {System.out.println("A");}
}
class B extends A {
}
public class test {
    public static void main(String args[ ]){
        B b = new B( );
    }
}
```



如果子类自己定义了构造方法，则在创建新对象时，它将先执行继承自父类的无参数构造方法，然后再执行自己的构造方法。

```
class A{  
    public A( ) {System.out.println("A");}  
}  
class B extends A {  
    public B( ) {System.out.println("B");}  
}  
public class test {  
    public static void main(String args[ ]){  
        B b = new B( );  
    }  
}
```



对于圆和球类定义如下:

```
class Circle {                                //圆类
    double radius;                            //半径
    Circle(){radius=0.0;}                    //无参构造函数
    Circle(double r){radius=r;}             //带参构造函数
    void setRadius(double r) {radius=r;}
    //设置半径值
    double area() {                          //返回圆面积
        return Math.PI*radius*radius;
    }
}
```

```
class Sphere extends Circle {    //球类
    Sphere() {super();} //无参构造函数
    Sphere(double r) {super(r);} //带参构造函数
    double area() {//返回表面积
        return 4*Math.PI*radius*radius;
    }
    double volume() {//返回体积
        return 4*Math.PI*Math.pow(radius,3)/3;
    }
}
```

假定采用下面的主类程序来调试上面定义的圆类和球类。

```
public class Test
{ public static void main(String[] args) {
    Sphere s=new Sphere(4);    //假定半径为4
    double x=s.area();          //计算球体表面积
    double y=s.volume();        //计算球体的体积
    System.out.println(x);      //输出表面积
    System.out.println(y);      //输出体积
  }
}
```

```
201.06192982974676  //表面积
268.082573106329    //体积
```

- 1.子类增加一个新的数据成员d
- 2.父类构造函数super是必须的吗?
- 3.能用Circle代替super吗?

```
class Sphere extends Circle {    //球类
    double d;
    Sphere() {super(); d=1;} //无参构造函数
    Sphere(double r) {super(r);d=1;}
    //带参构造函数
    double area() {//返回表面积
        return 4*Math.PI*radius*radius*d;
    }
    double volume() {//返回体积
        return 4*Math.PI*Math.pow(radius,3)*d/3;
    }
}
```

5. 子类对象的实例化

Student类的父类是Person类。创建子类对象，会继承父类的属性和方法。

栈 (stack)

student

堆 (heap)

Object()

Person()

Student()

Name
Age

StudentID



5. 子类对象的实例化

- ❑ 子类在继承父类过程中，继承父类属性和方法。创建子类对象，在堆空间中，会加载父类属性。
 - ❑ 如果有多层继承，创建子类时，子类会直接或间接调用多层父类构造函数，直至最上层的父类 `java.lang.Object` 类的空参构造函数。
 - ❑ 但注意调用了多层父类的构造函数，但始终只创建子类的一个对象。
-

```
public class FatherClass{
    public FatherClass(){System.out.println("FatherClass Create");
    }
}

public class ChildClass extends FatherClass{
    public ChildClass(){System.out.println("ChildClass Create");
    }

    public static void main(String[] args) {
        FatherClass fc = new FatherClass();
        ChildClass cc = new ChildClass();
    }
}
```

FatherClass Create
FatherClass Create
ChildClass Create

编译失败。子类应该通过super语句指定要调用的父类中的带参构造函数，或父类中增加空参数的构造函数。

```
class Super {  
    int i = 0;  
    public Super(String s) {i = 1;}  
}  
  
class Demo extends Super {  
    public Demo(String s) {i = 2;}  
    public static void main(String[] args) {  
        Demo d = new Demo("yes");  
        System.out.println(d.i);  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Base b1 = new Base();  
        Base b2 = new Sub();  
    }  
}  
  
class Base{  
    Base(){method(100);}  
    public void method(int i){System.out.println("base : " + i);}  
}  
  
class Sub extends Base{  
    Sub(){super.method(70);}  
    public void method(int j){System.out.println("sub : " + j);}  
}
```

base : 100

sub : 100

base : 70

6. 多态

对象的多态性 (Polymorphism)

- ❑ 定义：父类的引用指向子类的对象
 - ❑ 实现：当调用父类同名同参数的方法时，实际执行的是子类重写父类方法，称为虚拟方法调用。
 - ❑ 实例：Person、Man和Woman类。
-

```
class Person{ int id=100;
public void eat(){System.out.println("人吃饭");}
}
class Man extends Person{ int id=200; int manAge=20;
public void eat(){System.out.println("男人吃两碗饭");}
public void play(){System.out.println("男人踢足球");}
}
class Woman extends Person{
public void eat(){System.out.println("女人吃一碗饭");}
public void shop(){System.out.println("女人购物");}
}
```

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

```
        Person p1=new Person();
```

```
        p1.eat();
```

```
        Man man=new Man();
```

```
        man.eat();
```

```
        man.play();
```

```
        Person p2= new Man();
```

```
        p2.eat();
```

```
        Person p3= new Woman();
```

```
        p3.eat();
```

```
    }  
}
```

正常方法调用:

```
Man p2= new Man();
```

```
p2.eat();
```

```
Woman p3= new Woman();
```

```
p3.eat();
```

原因是p2定义为Person类型

```
System.out.println(p2.id);
```

```
p2.play();
```

```
System.out.println(p2.manAge);
```

6. 多态

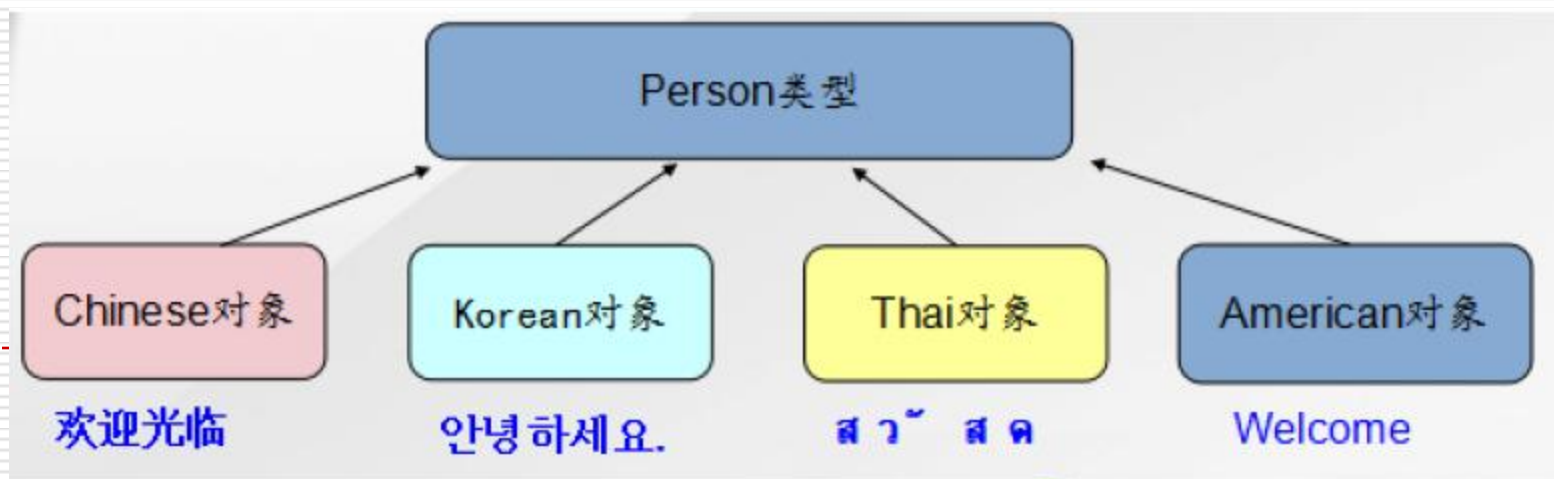
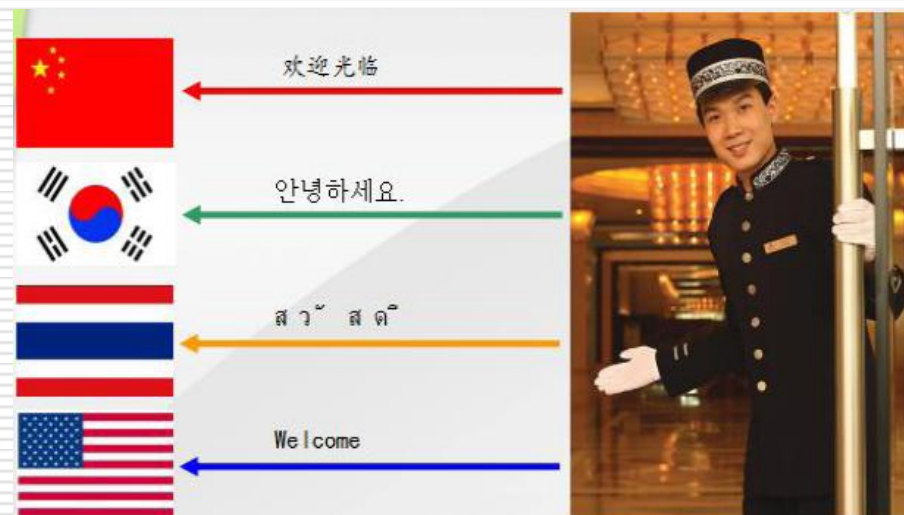
□ 多态的使用（虚拟方法调用）：

□ **编译的时候**只能调用父类中声明的方法。一个引用类型变量如果声明为父类类型，但实际引用的是子类对象，那么该变量编译时不能访问子类中添加的属性和方法。

□ **运行的时候**执行的是子类重写父类的方法。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，只能在运行时确定的，也称为动态绑定。

□ 对象的多态只适用与方法，不适用于属性。

- 虚拟方法调用举例：
- Person类中定义了welcome()方法，各个子类重写welcome()。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。



6. 多态

□ 多态的要求:


- (1) 类的继承关系
- (2) 子类重写父类的方法

□ 多态的应用方法: 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 void func(Man man)
{man.eat();}
public void func(Woman woman)
{woman.eat();}
```

```
public class PersonTest{
```

```
public void func(Person person){
person.eat();
}
```



```
public static void main(String [] args){
PersonTest test=new PersonTest();
test.func(new Man());
test.func(new Woman());}}
```

6. 多态

□ 重载与多态：

□ 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。

□ 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**”；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**”。

6. 多态

实例：如何证明多态是一个运行时的，而不是编译时的行为？

代码：InterviewTest

6. 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，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。
-

```
class A {private int a;
    public void setA(int a) {this.a = a;}
    public int getA() {return a;}}
class B extends A {private int a;
    public void setA(int a) {this.a = a;}
    // public int getA(){return a;}}
public class PersonTest {
    public static void main(String[] args) {
        A c = new B();
        c.setA(5);
        System.out.println(c.getA());}
}
```

```
class Fu {
    boolean show(char a) {System.out.println(a);
        return true;
    }
}

class Demo extends Fu {
    public static void main(String[] args) {
        int i = 0;
        Fu f = new Demo();
        Demo d = new Demo();
        for (f.show('A'); f.show('B') && (i < 2); f.show('C')) {
            i++;
            d.show('D');
        }
    }
    boolean show(char a) {
        System.out.println(a);
        return false;
    }
}
```

A

B

```
class Fu {
    int num = 4;
    void show() {System.out.println("showFu");}
}

class Zi extends Fu {
    int num = 5;
    void show() {System.out.println("showZi");}
}

class T {
    public static void main(String[] args) {
        Fu f = new Zi();
        Zi z = new Zi();
        System.out.println(f.num);
        System.out.println(z.num);
        f.show();
        z.show();
    }
}
```

4

5

showZi

showZi

6. 多态

□ 类型转换:

Man类的构造函数实例化完整对象，但p2是Person对象，只能访问父类属性和方法。强制转化为Man类对象后可以访问子类特有属性和方法。

栈 (stack)

m1

p2

堆 (heap)

Object()

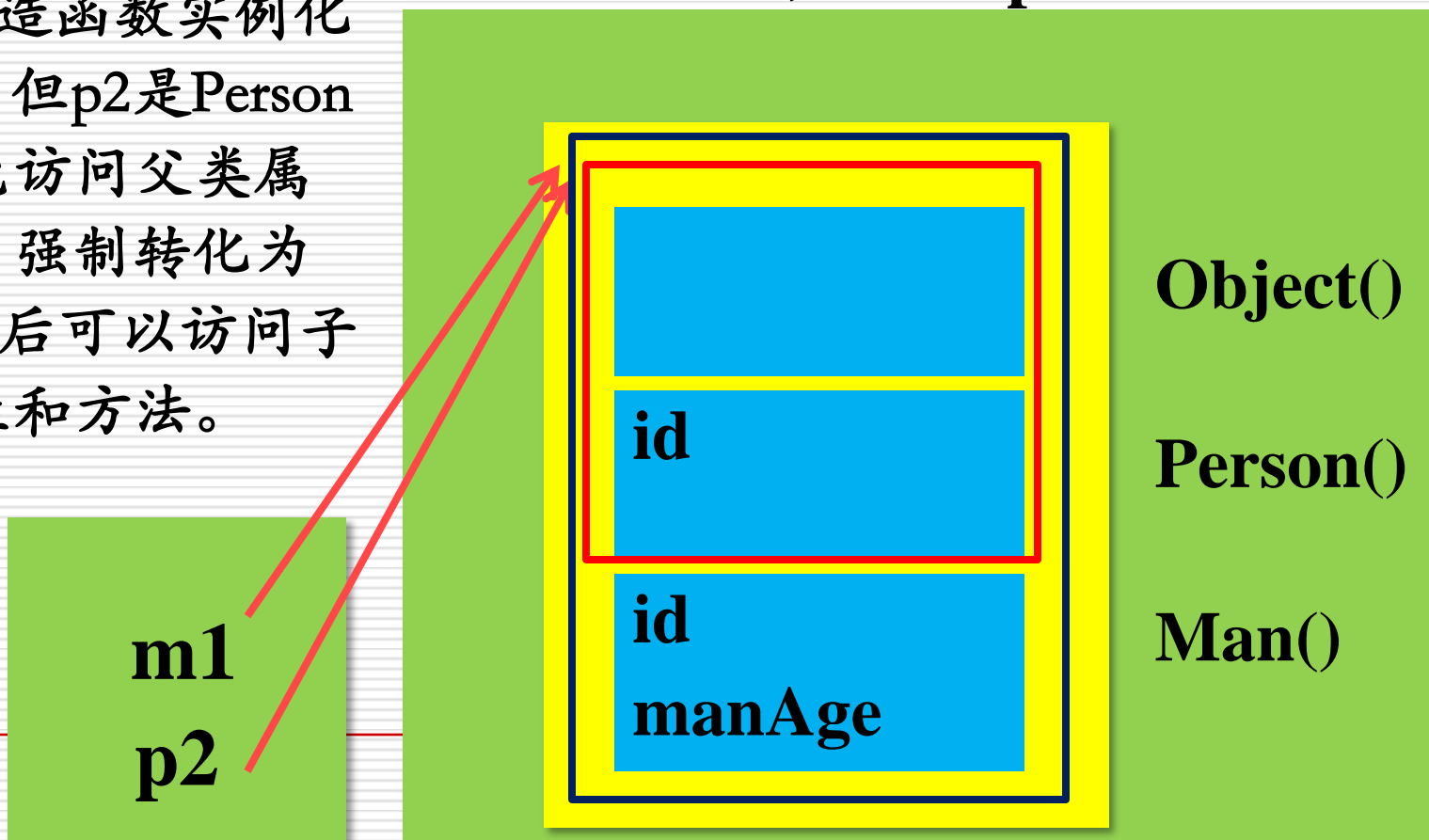
Person()

Man()

id

id

manAge



6. 多态

□ 强制类型转换

多态后，内存中实际上是加载了子类所特有的属性和方法，但由于变量声明为父类类型，导致编译时只能调研父类声明的属性和方法，子类特有的属性和方法不能调用。

□ 如何能调用子类特有的属性和方法？

答案：使用强制类型转换，将父类对象转换为子类对象，也叫向下转型。

例如： Man m1=(Man)p2;

```
public class PersonTest
public static void main
Person p1=new Person();
p1.eat();
Man man=new Man();
man.eat();
man.play();
Person p2=new Man();
Man m1=(Man)p2;
m1.play();
System.out.println(m1)
System.out.println(m1)
```

```
Woman w1= Woman(p2);
w1.shop();//报错
```

强制转换可能出现异常，为了解决此问题，引入instanceof

```
if (p2 instanceof Woman){
Woman w1=(Woman)p2;
w1.shop();}
```

//判断为false，避免运行异常

```
p2 instanceof Man
p2 instanceof Person
```

//判断都返回true，可以转型

6. 多态

instanceof关键字

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


6. 多态

对象类型转换(Casting)

□ 基本数据类型的Casting:

- 自动类型转换: 小数据类型自动转换成大数据类型
- 强制类型转换: 把大数据类型强制转换成小的数据类型

□ 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 (**实际就是多态**)
- 从父类到子类的类型转换须通过造型(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

较高级的基本数据类型

强制类型转化

自动类型提升

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：多态

子类（如： **Student**）

6. 多态

以下代码对吗？

```
Person p1=new Person();
```

```
Man m1=(Man)p1;
```

- ❑ 编译通过但运行不通过。虽然语法没问题，但P1是Person对象，没有加载Man的属性，不能强制转换成Man对象。
 - ❑ 注意，向下转型中new创建的应该是子类对象或其子孙对象，而不是父类对象。
-

6. 多态

以下代码对吗？

```
Object obj=new Man();
```

```
Person p=(Person)obj;
```

- ❑ 编译通过但运行也通过。Man对象包含子类和父类属性，第一句是向上自动转型为object对象。第二句向下强制转型时，由于obj包含了Person的属性，可以转换。
-

6. 多态

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

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

- ◆ 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。
- ◆ 对于属性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属性变量，这个属性变量依然不可能覆盖父类中定义的属性变量。

```
System.out.println(b == s); true。——应用于引用数据类型，比较是地址  
System.out.println(b.count); 10，多态性，  
b.display(); 20，重写，虚拟方法调用  
}
```

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java.lang.Object类。
 - ❑ Object类没有定义属性，只有空参构造函数和其他方法。其中主要方法包括：
 - clone：复制当前对象
 - finalize:垃圾回收器调用以回收对象所占的空间
 - getClass：返回对象的类
 - equals方法
 - toString方法
-

7. Object 类

==运算符

- 基本类型比较值：只要两个变量的值相等，即为 true，类型可以不同。

```
int i=10;
```

```
int j=10;
```

```
char c=10;
```

```
double d=10.0;
```

```
i==j //true
```

```
i==c //true
```

```
i==d //true
```

7. Object 类

==运算符

- 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。

```
String str1=new String("hello");
```

```
String str2=new String("hello");
```

```
str1==str2//false
```

- 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。
-

7. Object 类

equals 方法

- ❑ 所有类都继承了Object，也就获得了equals()方法。equals()应用于引用数据类型，其作用与“==”相同，比较是否指向同一个对象。
 - ❑ 格式:obj1.equals(obj2)
 - ❑ 当用equals()方法进行比较时，对类File、String、Date及包装类来说，是比较类型及内容而不考虑引用的是否是同一个对象。
-

7. Object 类

equals 方法和==操作符区别

- ❑ == 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
 - ❑ equals属于java.lang.Object类的方法，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。具体要看自定义类里有没有重写Object的equals方法来判断。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。
-

7. Object 类

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

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

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

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

```
System.out.println("hello" == new java.util.Date());
```

7. Object 类

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());
```

7. Object 类

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);
```

举例-学生类

```
public class Student
{
    //3个数据成员，2个构造函数，11个一般成员函数
    private String numb; //学号，私有数据成员
    private String name; //姓名，私有数据成员
    private double grade; //成绩，私有数据成员
    public Student() { //无参构造函数
        numb=null; name=null; grade=0.0;
    }
}
```

```
public Student(String numb, String name, double grade)
{ //带参构造函数
this.numb=numb; this.name=name; this.grade=grade;
}
public String getNumb() {return numb;} //返回学生号
public String getName() {return name;} //返回学生姓名
public double getGrade() {return grade;} //返回学生成绩
public Student getRecord() {return this;} //返回当前学生记录
```

```
public void updateGrade(double x) {grade=x;} //更新学生成绩
public void updateRecord(Student r) {//更新学生记录
    numb=r.numb; name=r.name; grade=r.grade;
} //用参数r的学生记录值修改调用（当前）对象的记录值
public void deleteRecord() {//清除当前记录内容
    numb=null; name=null; grade=0.0;
}
public boolean isPass() {//判断学生成绩是否及格
    return grade>=60;
}
```

```
public boolean isNumb(String numb) {    //判断学号相等
    return this.numb==numb;    //调用对象的学号与参数比较
}
public double compare(Student stu) {    //以成绩比较记录大小
    return grade-stu.grade;
}    //当前调用对象的成绩较大返正数，较小返负数，相等返0
public String toString() { //定义用来输出的成员函数toString()
    return numb+" "+name+" "+grade;    //返回学生值字符串
}    //由3个成员值连接而成的字符串
}
```

利用下面定义的主类Example_6来使用学生类:

```
public class Example_6
{
    public static void main(String[] args) {
        Student s1=new Student();
        Student s2=new Student("jsj-2","ningchen",86);
        System.out.println(s2); //输出r2, 自调toString()
        s2.updateGrade(92);
        System.out.println(s1); //输出s1
        System.out.println(s2); //输出s2
    }
}
```

```
s1.updateRecord(new Student("jsj-3","wangqi",75));
System.out.println(s1);           //输出s1
System.out.println(s2);           //输出s2
if(s1.isPass()) System.out.println(s1+" "+"通过");
    else System.out.println(s1+" "+"不通过");
double x=s1.compare(s2);
if(x>0.0) System.out.println("s1>s2");
    else if(x<0.0) System.out.println("s1<s2");
    else System.out.println("s1=s2");
    }
}
```

jsj-2	ningchen	86.0	//第3条: 输出学生对象s2的值
null	null	0.0	//第5条: 输出学生对象s1的值
jsj-2	ningchen	92.0	//第6条: 输出学生对象s2的新值
jsj-3	wangqi	75.0	//第8条: 输出学生对象s1的新值
jsj-2	ningchen	92.0	//第9条: 再输出学生对象s2的值
jsj-3	wangqi	75.0	//第10条: 表明s1的成绩大于等于60
s1<s2			//第12条: 表明s1的成绩小于s2的成绩

如果没有重写toString()会有什么结果?
如何改写equals()判断成绩相等?

问题

1. `public void println(char[] x)`, 打印字符数组
2. `println()`对一般对象是打印其地址
3. `Arrays.toString(cArr)` 输出数组的元素值

```
public class toStringTest
{
    public static void main(String[] args) {
        char[] cArr ={'北', '京'};
        System.out.println(cArr); //北京
        System.out.println(Arrays.toString(cArr)); //[北, 京]
        int[] iArr = {1,2};
        System.out.println(iArr); //[I@6aa553e2
        System.out.println(Arrays.toString(iArr)); [1, 2]
        String[] sArr = {"人","民"};
        System.out.println(sArr); [Ljava.lang.String;@c265121
        System.out.println(Arrays.toString(sArr)); [人, 民]
    }
}
```

□ 为什么要重写toString() 方法?

答：重写toString() 方法，可以用以返回对象中存储的信息内容，而不是对象的地址。

□ 用 “==” 比较两个String总是false，但是它们明明都是"abc"！

答：比较String一定要使用equals或equalsIgnoreCase方法，不要使用 == 。

==比较的是两个引用（变量）是否指向了同一个对象，而不是比较其内容。

8. 包装类 (Wrapper)

- ❑ 针对八种基本数据类型定义相应的引用类型：包装类
- ❑ 基本数据类型转换成包装类，就有了类的特点，才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	包装类	父类: Number
byte	Byte	
short	Short	
int	Integer	
long	Long	
float	Float	
double	Double	父类: Number
boolean	Boolean	
char	Character	

8. 包装类(Wrapper)

❑ 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
```

```
Boolean bobj = new Boolean("true");
```

❑ 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();
```

8. 包装类(Wrapper)

❑ 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:

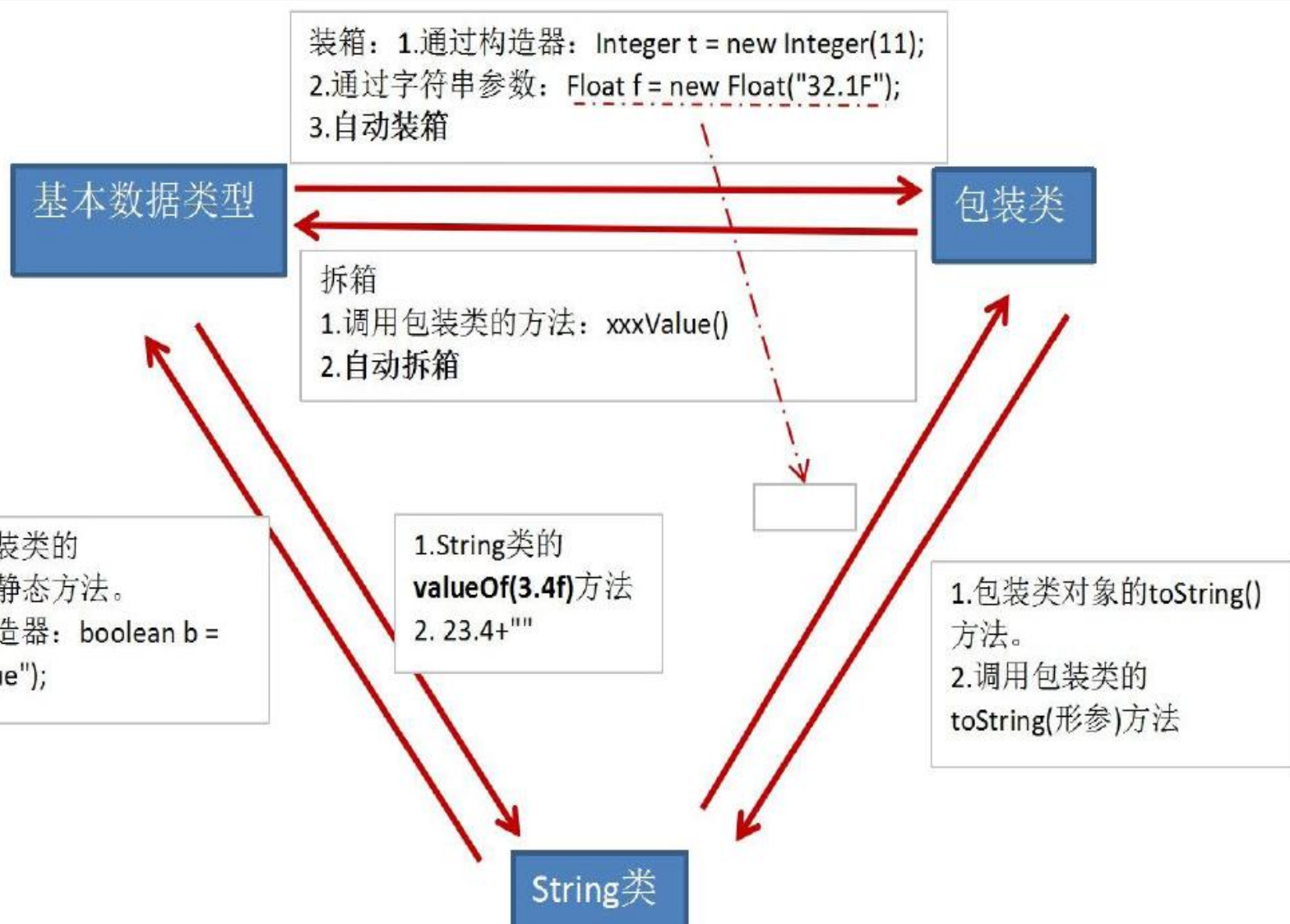
```
Float f = Float.parseFloat("12.1");
```

❑ 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);
```

■ 或者String intStr= 5 + ""



JDK 5后，基本数据类型和包装类之间可以自动装箱和自动拆箱

8. 包装类 (Wrapper)

❑ 下面的输出结果是什么？

```
Object o1 = true ? new Integer(1) : new Double(2.0);  
System.out.println(o1); // 1.0, int类型转成double
```

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

8. 包装类 (Wrapper)

❑ 下面的输出结果是什么？

```
public void method1() {  
    Integer i = new Integer(1);  
    Integer j = new Integer(1);  
    System.out.println(i == j); //false  
  
    Integer m = 1;  
    Integer n = 1;  
    System.out.println(m == n); //true, integer内部定义整形  
                                   Cache, 缓存 -128~127, 如  
                                   果自动装箱数在这个范围,  
                                   就不调用new, 直接写数组  
  
    Integer x = 128;  
    Integer y = 128;  
    System.out.println(x == y); // false, 自动装箱调new  
}
```

```
public static void main(String[] args) {  
    Integer i1 = 128;  
    Integer i2 = 128;  
    int i3 = 128;  
    int i4 = 128;  
    System.out.println(i1 == i2);           false  
    System.out.println(i3 == i4);           true  
    System.out.println(i1 == i3);           true  
}
```

```
public static void main(String[] args) {  
    double a = 2.0;  
    double b = 2.0;  
    Double c = 2.0;  
    Double d = 2.0;  
    System.out.println(a == b);           true  
    System.out.println(c == d);           false  
    System.out.println(a == d);           true  
}
```

int 和 Integer 有什么区别

答：Java 提供两种不同的类型：引用类型和原始类型（或内置类型）。

int是java的原始数据类型，Integer是java为int提供的封装类。Java为每个原始类型提供了封装类。对象引用实例变量的缺省值为 null，而原始类型实例变量的缺省值与它们的类型有关。
