

第五章 消息、继承与多态

本章主要内容

1. 消息
2. 访问控制
3. 多态机制
4. 继承机制
5. 抽象类、接口与包

1 消息

● 消息的概念

在面向对象的系统中，把“请求”或“命令”抽象成“消息”，对象之间的联系是通过消息传递来实现的。

当系统中的其他对象请求当前对象执行某个服务时，它就响应这个请求，完成指定的服务。

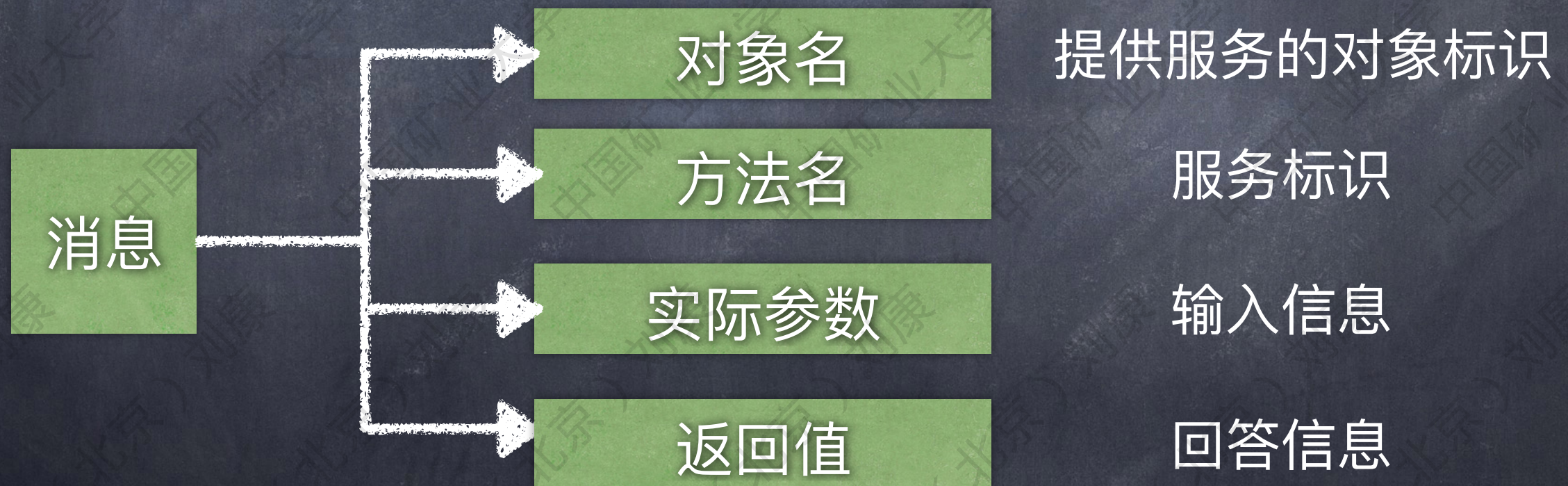
发送消息的对象
(发送者)



接收消息的对象
(接收者)

消息的实质

向对象发出服务请求，是对数据成员和成员方法的引用。



● 消息具有三个性质：

- (1) 同一对象可接收不同形式的多个消息，产生不同的响应。
- (2) 相同形式的消息可以发送给不同对象，所做出的响应可以是截然不同的。
- (3) 消息的发送可以不考虑具体的接收者，对象可以响应消息，也可以对消息不予理会，对消息的响应并不是必须的。

消息的三种形式

1. **公有消息**：由外界对象直接发送给这个对象的消息。



2. **私有消息**：对象自己发送给本身的消息。



3. 特定于对象的消息

将所有能支持此对象可接受消息的方法集中在一起，形成一个大消息。

特定于对象的消息分为三类：

(1) 可以返回对象内部状态的消息。

(2) 可以改变对象内部状态的消息。

(3) 可以改变系统状态的消息。

2 访问控制

- 一个类总能够访问自己的数据成员和成员方法。
- 其他类是否能访问这个类的数据成员或成员方法，是由类的访问控制符及该类数据成员和成员方法的访问控制符共同决定。

类	public	缺省
数据成员与方法	public	protected
public	所有类	包中类(含当前类)
protected	包中类, 所有子类	包中类
缺省	包中类	包中类
private	当前类本身	当前类本身

2.1 public访问控制符

- 一个类被声明为 `public` 时，只要在其他包的程序中使用 `import` 语句引入这个 `public` 类。
- 访问权限：
 1. 访问和引用这个类
 2. 创建类的对象
 3. 访问类内部可见的数据成员
 4. 引用类的可见的方法。

2.2 缺省访问控制符

- 如果一个类没有访问控制符，说明它具有缺省的访问控制特性，称为“友好访问”。
- 友好访问规定只有在同一个包中的对象才能访问和引用这些类，又称为包访问性。
- 类内的数据成员和成员方法，如果没有访问控制符来限定，也具有“友好访问”的特性，也具有包访问性，可以被同一个包中的其他类所访问和引用。

2.3 `private`访问控制符

`private`修饰的数据成员或成员方法只能被该类自身所访问和修改，而不能被任何其他类（包括该类的子类）来访问和引用。

提供**最高的保护级别**。当其他类希望获取或修改私有成员时，需要借助于类的方法来实现。

2.4 `protected`控制符

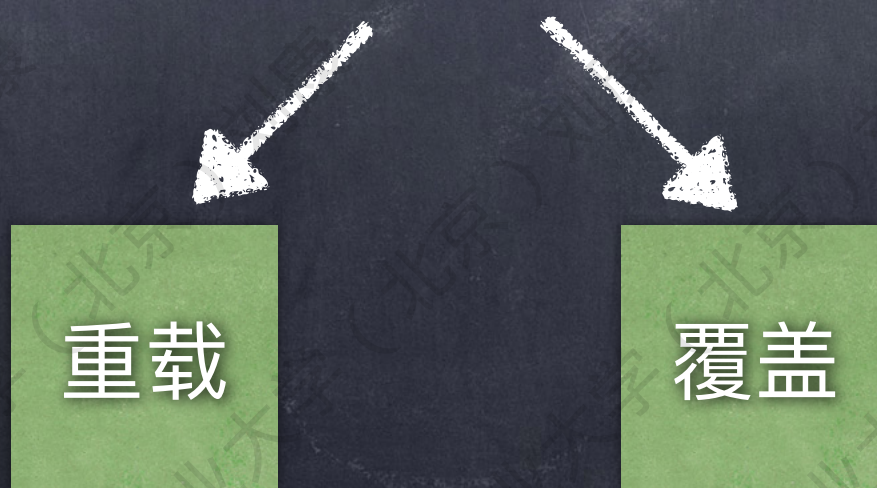
- 用`protected`修饰的成员变量可以被三类引用：`该类自身`、`与它在同一个包中的其他类`、`在其他包中的该类的子类`。
- 使用`protected`修饰符的主要作用是允许其他包中的它的子类来访问父类的特定属性。

3 多态机制

● 多态的概念

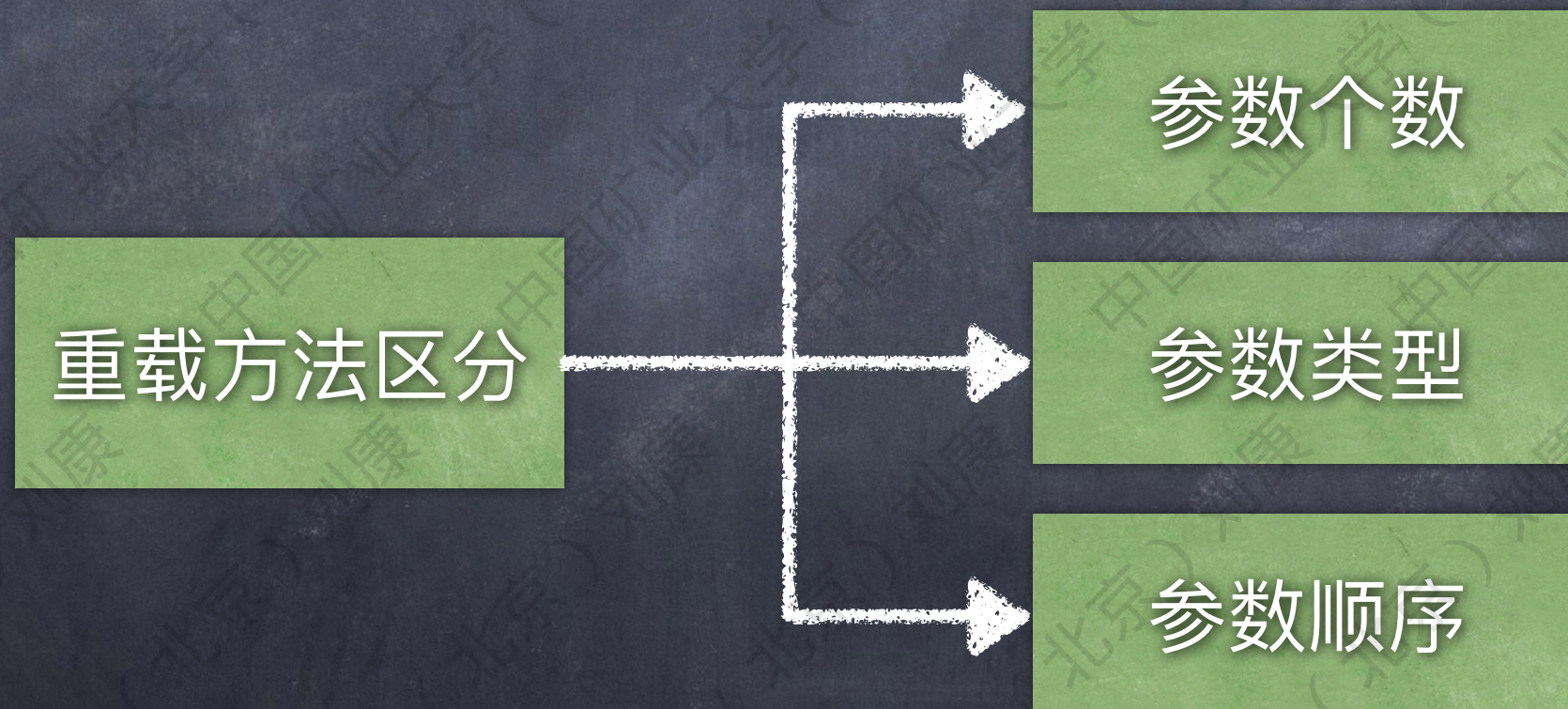
面向对象系统中的又一重要特性，**同名方法**可以根据发送消息的对象传送**参数的不同**，采取不同的行为方式的特性。

● Java提供两种多态机制



3.1 方法重载

- 在**同一类中**定义了多个**同名而不同内容**的成员方法时，我们称这些方法是**重载(override)**。



【例5-1】 加法重载的例子。

```
import java.awt.*;
import java.applet.*;
public class c5_5 extends Applet
{
    int add(int a,int b)    //重载的方法1
    { return(a+b); }
    double add(double x,double y) //重载的方法2
    { return(x+y); }
    double add(double x,double y, double z) //重载的方法3
    { return(x+y+z); }

    public void paint(Graphics g)
    { g.drawString("Sum is:"+add(8.5,2.3),5,10);
      g.drawString("Sum is:"+add(21,38),5,30);
      g.drawString("Sum is:"+add(8.5,2.3,8.5+2.3),5,50);
    }
}
```


3.2 方法覆盖

- 面向对象的继承机制，子类可以继承父类的方法。为了体现子类的个性，在子类中定义与父类中已定义的相同名而内容不同的方法。称为覆盖 (overload)。
- 由于覆盖的同名方法是存在于子类对父类的关系中，所以只需在方法引用时，指明引用的是父类的方法还是子类的方法，就可以很容易地区分。

4 继承机制

● 继承的概念

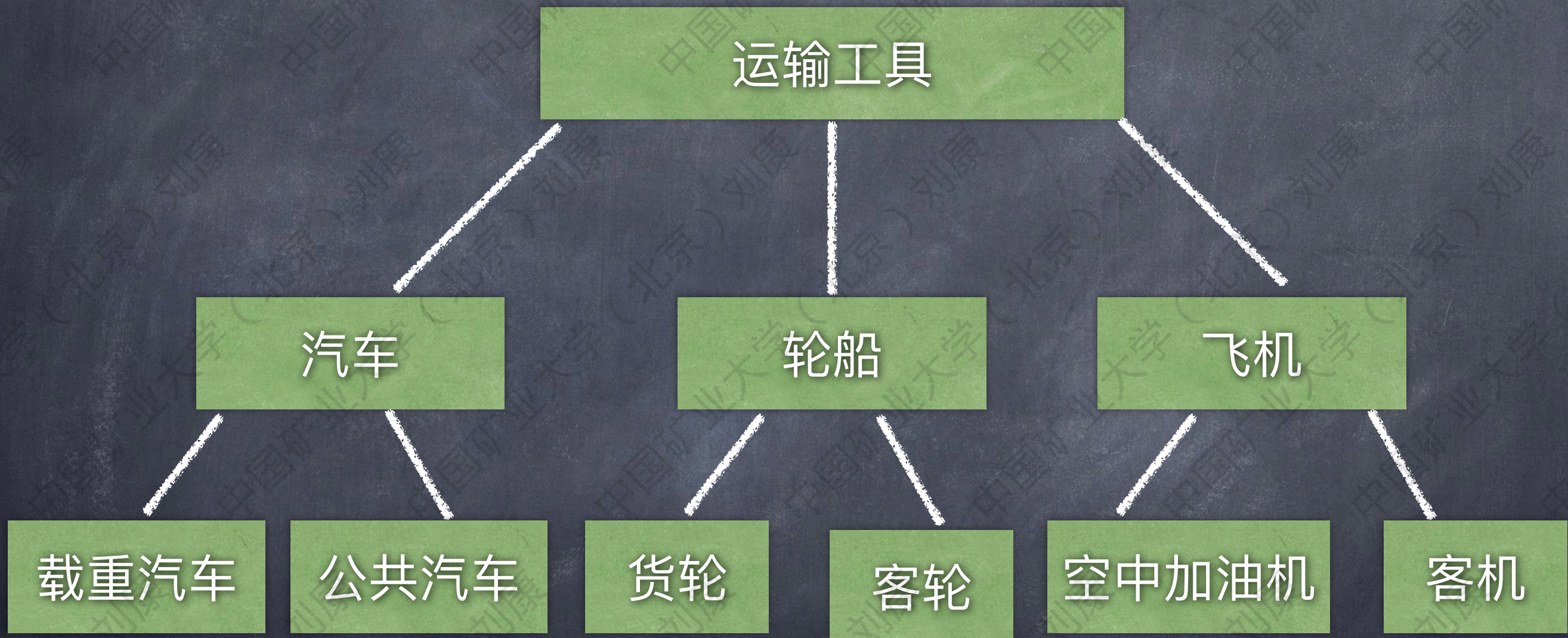
继承是一种对象的类之间相交关系，使得某类对象可以继承另外一类对象的数据成员和成员方法。

若类B继承类A时，则属于B的对象便具有类A的全部或部分性质(数据属性)和功能(操作)。

被继承的类A为基类或父类

继承类B为A的派生类或子类

父类与子类的层次关系



👁 继承的特征

1. 继承关系是**传递**的。
2. 继承简化了人们对事物的认识和描述，能清晰体现相关类间的**层次结构关系**。
3. 提供**软件复用**功能。通过增强一致性来减少模块间的接口和界面，大大增加程序的易维护性。
4. 通过增强一致性来减少模块间的接口和界面，大大增加程序的**易维护**性。
5. 提供**多重继承机制**。

学生

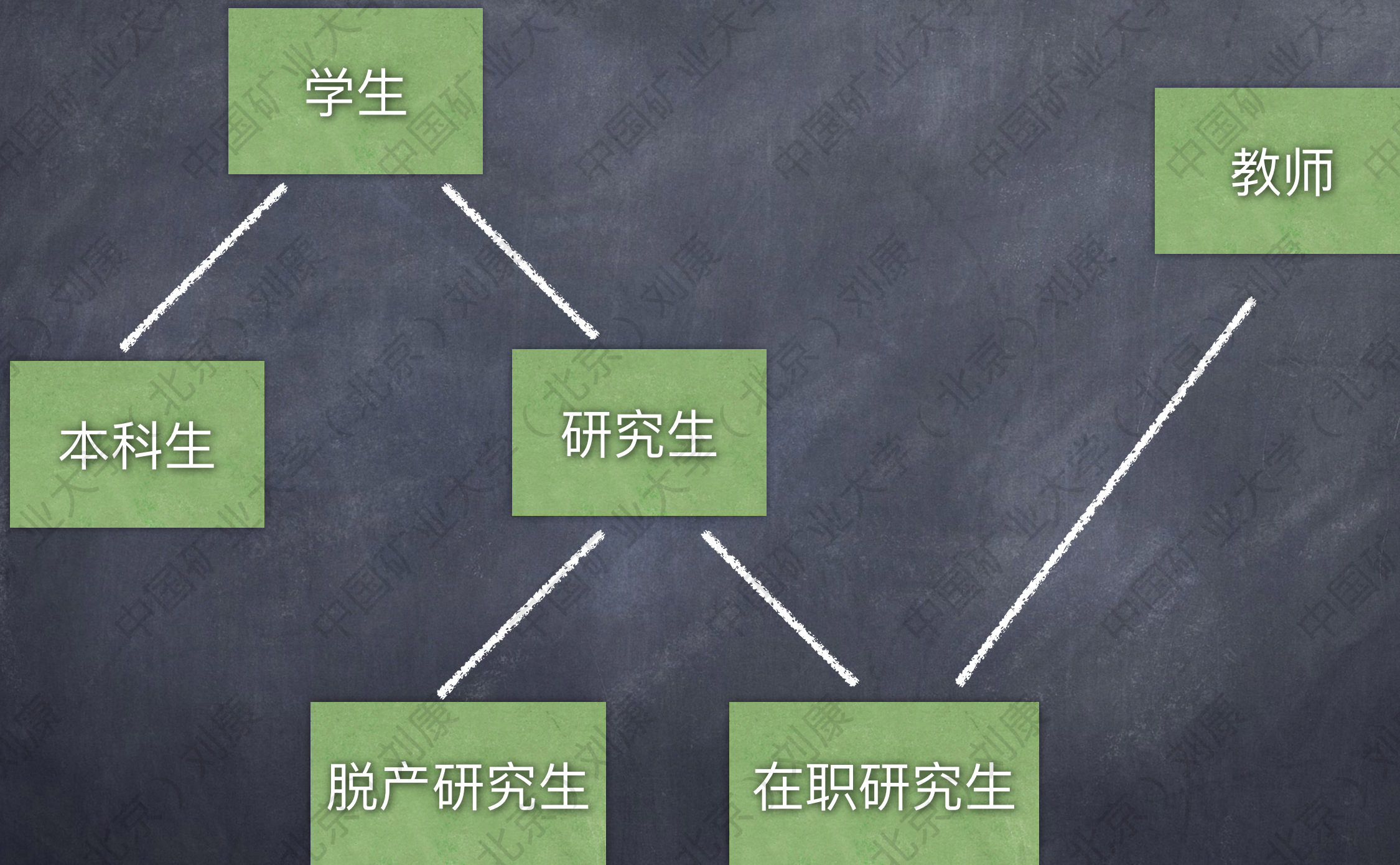
教师

本科生

研究生

脱产研究生


在职研究生



4.2 extends指明继承关系

- 在Java中，继承是通过`extends`关键字实现。
- 在定义类时，使用`extends`关键字指明新定义类的父类，新定义的类型称为指定父类的子类，在两个类之间建立了继承关系。

● 继承机制分为四个部分

1. 数据成员的继承
 2. 数据成员的隐藏
 3. 成员方法的继承
 4. 成员方法的覆盖
- 
- The diagram consists of two right-facing square brackets. The top bracket is purple and groups items 1 and 2. The bottom bracket is blue and groups items 3 and 4.

1 数据成员的继承

子类可以继承父类的所有非私有的数据成员

【例5-1】

```
class a1
{
    int x=25;
    private int z;
}
```

```
class e5_1 extends a1
{
    public static void main(String[] args)
    {
        e5_1 p = new e5_1();
        System.out.println("p.x="+p.x);
        System.out.println("p.z="+p.z);
    }
}
```

运行结果:

p.x=25

2 数据成员的隐藏

在子类中重新定义一个与父类中已定义的数据成员名完全相同的数据成员。

子类拥有了两个相同名字的数据成员：一个是继承父类的，另一个是自己定义的。

当子类引用这个同名的数据成员时，默认操作是它自己定义的数据成员，而把从父类继承的数据成员“隐藏”起来。

【例5-2】

```
class a1
{ int x=8; }

class e5_2 extends a1
{ int x=24;
  public static void main(String[] args)
  { int s1,s2;
    a1 p= new a1( );
    e5_2 p1= new e5_2( );
    s1=p.x;
    s2=p1.x;
    System.out.println("s1="+s1);
    System.out.println("s2="+s2);
  }
}
```

运行结果：

s1=8 s2=24

3 成员方法的继承

子类可以继承父类的非私有的成员方法

【例5-3】

```
class a1
{
    int x=0, y=1;
    void Myp( )
    {
        System.out.println("x="+x+" y="+y);
    }
    private void Printme( )
    {
        System.out.println("x="+x+" y="+y);
    }
}
```

```
public class e5_3extends a1
{
    public static void main(String arg[ ])
    {
        int z=3;
        e5_3 p1=new e5_3( );
        p1.Myp( );
        p1.Printme( );
    }
}
```


4 成员方法覆盖

子类可以重新定义与父类同名的成员方法，实现对父类方法的覆盖。

方法的覆盖与数据成员的隐藏的不同之处在于：

1. 子类**隐藏**父类的数据成员只是使之不可见，父类同名的数据成员在子类对象中**仍占有独立的内存空间**；
2. 子类方法对父类同名方法的**覆盖**将**清除父类方法占用的内存**，从而使父类方法在子类对象中不复存在。

【例e5_4】

```
class a1
{
    int x=10; int y=31;
    public void Printme( )
    {
        System.out.println("x="+x+" y="+y);
    }
}

public class e5_4 extends a1
{
    int z=35;
    public void Printme( )
    {
        System.out.println(" z="+z);
    }

    public static void main(String arg[ ])
    {
        a1 p2= new a1( );
        e5_4 p1= new e5_4( );
        p1.Printme( );
        p2.Printme( );
    }
}
```

z=35

x=10 y=31

4.3 this与super

1. this的概念

this代表了当前对象的一个引用，可将其理解为当前对象的另一个名字，通过这个名字可以顺利地访问对象、修改对象的数据成员、调用对象的方法。

• `this`的使用场合有下述三种：

1. 用来访问当前对象的数据成员

`this.数据成员`

2. 用来访问当前对象的成员方法

`this.成员方法(参数)`

3. 当有重载的构造方法时，引用同类的其它构造方法

`this(参数)`

2. `super`的概念

当前对象的**直接父类**对象，是当前对象的直接父类对象的引用。

若子类的数据成员或成员方法名与父类的数据成员或成员方法名相同时，当要调用父类的同名方法或使用父类的同名数据成员，则可用关键字`super`来指明父类的数据成员和方法。

● `super`的使用场合有下述三种：

1. 用来访问直接父类隐藏的数据成员

`super.数据成员`

2. 用来调用直接父类中被覆盖的成员方法

`super.成员方法(参数)`

3. 用来调用直接父类的构造方法

`super.(参数)`

✗
=01

4.4 构造方法的重载与继承

1. 构造方法的重载

- A. 一个类的若干个构造方法之间可以相互调用。
- B. 当一个构造方法a调用构造方法b时，使用关键字**this**，同时这个调用语句应该是整个构造方法的第一个可执行语句。

2. 构造方法的继承

子类可以继承父类的构造方法，继承遵循以下的原则：

- 1) 子类无条件地继承父类的**不含参数**的构造方法。
- 2) 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
- 3) 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
- 4) 对于父类的含参数构造方法，**子类**可以通过在自己的构造方法中**使用super**关键字来调用它，但这个**调用语句**必须是子类构造方法的第一个可执行语句

4.5 向方法传递对象

- 传递给方法的参数可以是表达式、对象等。
- 传递给方法的参数若是变量，则只能由实参传递给形参，而不能由形参带回，它是一种单向值传递。
- 传递给方法的参数若是对象，则方法可以对其做永久性修改。

4.6 类转换

- 类转换是指父类对象与子类对象之间在一定条件下的相互转换。
- 父类对象与子类对象之间的相互转换规则如下：
 1. 父类对象与子类对象之间可以隐式转换，也可显式转换
 2. 处于相同类层次的类的对象不能进行转换
 3. 子类对象可转换成父类对象，但对数据成员和成员方法的引用必须使用强制转换（重点）

4.7 继承与封装的关系

- 在面向对象系统中，封装性主要指的是**对象的封装性**，即将属于某一类的一个具体的对象封装起来，使其数据和操作成为一个整体。
- 在引入了继承机制的面向对象系统中，对象依然是封装得很好的实体，其他对象与它进行通讯的途径仍然只有一条，那就是发送消息。

● 继承和封装机制还具有一定的相似性

继承是一种**静态共享代码**的手段，通过派生类对象的创建，可以接受某一消息，启动其基类所定义的代码段，从而使基类和派生类共享了这一段代码。

封装机制所提供的是一种**动态共享代码**的手段，通过封装，我们可将一段代码定义在一个类中，在另一个类所定义的操作中，我们可以通过创建该类的实例，并向它发送消息而启动这一段代码，同样也达到共享的目的。

5 抽象类、接口与包

5.1 抽象类

圆类

圆心坐标
半径

计算面积
计算周长

三角形类

底边长
高

计算面积
计算周长

矩形类

长
宽

计算面积
计算周长

抽象类

抽象方法

圆类

圆心坐标
半径

计算面积
计算周长

三角形类

底边长
高

计算面积
计算周长

矩形类

长
宽

计算面积
计算周长

梯形类

上、下底
边长高

计算面积
计算周长

👁 抽象类的概念

抽象类是所有子类的公共属性的集合，是包含一个或多个抽象方法的类。

抽象类描述共有行为的特征，并通过继承机制传递给派生类。

● 抽象类与抽象方法的限制如下：

1. 凡是用`abstract`修饰的类是抽象类，修饰的成员方法是抽象方法
2. 抽象类可有零个或多个抽象方法，也可包含非抽象方法
3. 抽象类中可没有抽象方法，有抽象方法的类必是抽象类
4. 抽象方法之定义方法体，不写实现代码
5. 抽象类可派生子类，派生的子类必须实现抽象类中所有抽象方法

6. 抽象类不能创建对象，仅能由抽象类派生的子类创建
7. 如果父类中已有同名的`abstract`方法，则子类中不能再有同名的抽象方法
8. `abstract`不能与`final`并列修饰同一个类
9. `abstract`不能与`private`, `static`, `final`或`native`并列修饰同一个方法
10. `abstract`类中不能有`private`的数据成员或成员方法

5.2 接口

- 在面向对象的程序设计语言中，有些语言提供了多继承机制。而Java出于安全性、简化程序结构的考虑，不支持类间的多继承而只支持单继承。
- 为了使Java程序的类间层次结构更加合理，更符合实际问题的本质，Java语言提供接口来实现多重继承机制。

1. 声明接口

声明接口的格式如下：

[修饰符] **interface** 接口名 [extends 父接口名列表]

{

常量数据成员声明

抽象方法声明

}

2. 定义接口的注意事项:

(1) 接口定义用关键字 `interface`

(2) 接口中定义的数据成员全是 `final static`,
即常量

(3) 接口中没有自身的构造方法, 所有成员方法
是抽象方法

(4) 接口也具有继承性, 可通过 `extends` 关键字
声明该接口的父接口

3. 类实现接口的注意事项

- 1) 在类中，用`implements`关键字就可以调用接口。
一个类若要调用多个接口时，可在`implements`后用逗号隔开多个接口的名字。
- 2) 如果实现某接口的类不是`abstract`的抽象类，则在类的定义部分必须实现指定接口的所有抽象方法，即为所有抽象方法定义方法体，而且方法头部分应该与接口中的定义完全一致，即有完全相同的返回值和参数列表。

- 3) 如果实现某接口的类是`abstract`的抽象类，则它可以不实现该接口所有的方法。但是对于这个抽象类的任何一个非抽象的子类而言，它们的父类所实现的接口中的所有抽象方法都必须有实在的方法体。这些方法体可以来自抽象的父类，也可以来自子类自身，但是不允许存在未被实现的接口方法。这主要体现了非抽象类中不能存在抽象方法的原则。
- 4) 接口的抽象方法的访问限制符都已指定为`public`，所以类在实现方法时，必须显式地使用`public`修饰符，否则将被系统警告为缩小了接口中定义的方法的访问控制范围。

5.3 包与程序复用

- Java语言提供包：Java.io、Java.awt、Java.lang等，包中存放着一些常用的基本类，如System类、String类、Math类等，称为Java类库中的包。
- 在许多场合反复使用那些早已编写好的，且经过严格测试的的技术被称为软件复用，在面向对象的程序设计中称为对象复用。

- 包是接口和类的集合，使用包有利于实现不同程序间类的重用。`Java`语言为编程人员提供了自行定义包的机制。

- 包的作用有两个：

1. 划分类名空间
2. 控制类之间的访问

● 创建包(package)

包的创建就是将源程序文件中的接口和类纳入指定的包。

在一般情况下Java源程序的构成由四部分组成：

1. 一个包说明语句(可选项)。其作用是将本源文件中的接口和类纳入指定包。源文件中若有包说明语句，必须是第一个语句。
2. 若干个(import)语句(可选项)。其作用是引入本源文件中需要使用的包。
3. 一个public的类声明。在一个源文件中只能有一个public类。
4. 若干个属于本包的类声明(可选)。

包的声明语句格式:

`package` 包名;

- 利用这个语句就可以创建一个具有指定名字的包，当前 `java` 文件中的所有类都被放在这个包中。
- 创建包就是在当前文件夹下创建一个子文件夹，存放这个包中包含的所有类的 `.class` 文件。
- 若源文件中未使用 `package`，则该源文件中的接口和类位于 `Java` 的无名包中(无名包又称缺省包)，它们之间可以相互引用非 `private` 的数据成员或成员方法。无名包中的类不能被其他包中的类引用和复用。

● 建立包的步骤:

1. 建立源文件，文件的接口与类都属于包，将文件存入当前文件夹
2. 编译源文件
3. 包引用: `import` 包名;