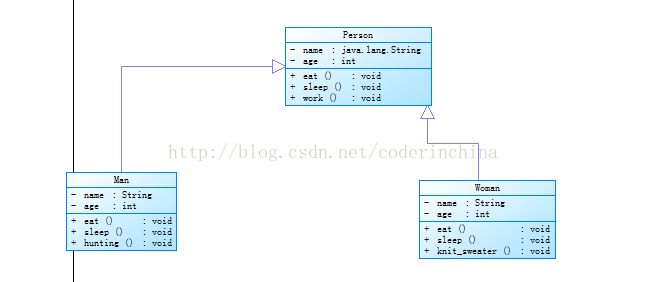
# Powerdesigner学习

在大型的项目中可能会有类图,因为有了它整个项目的类结构更加清晰,或者说一个刚入职进入你项目组的成员能更快的了解整个项目,画类图之前要明白类和类之间有几种关系,如果这几种关系都不知道,画图就是纸上谈兵,常见的类和类之间的关系有如下几种关系：

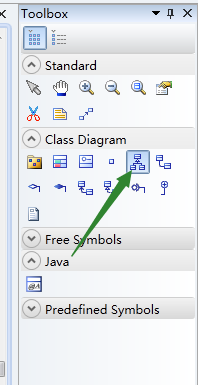
泛化（Generalization）

是一种继承关系, 表示一般与特殊的关系, 它指定了子类如何特化父类的所有非private修饰的特征和行为,比如说人有男人和女人,男人和女人即有共性的地方也有差异的地方

箭头指向:带三角箭头的实线，箭头指向父类



我们看到父类Person中有二个属性age和name,还有三个三个方法eat(),sleep(),work()方法,二个子类Man和Woman都继承了Person父类2个属性age,name和二个方法eat(),sleep(),这些都是二个子类共有的属性和父类,但是Man和Woman子类也有各自不同于父类的方法,在PowerDesigner中继承的图标是这个,如果不知道的话可以使用鼠标放在每个图标上,如果上面有Generalization单词就表示这是泛化关系。

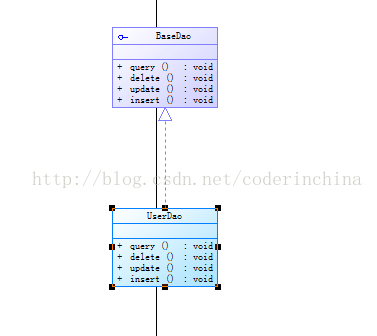


实现（Realization）

实现和我们java中的实现是一个意思,是一种类与接口的关系, 表示类是接口所有特征和行为的实现

箭头指向:带*三角箭头的虚线*，箭头指向接口

类图如下：



关联（Association)

是一种拥有的关系, 它使一个类知道另一个类的属性和方法,这个概念听起来感觉有点蒙,其实吧,就是一个对象A中持有对象B的引用,这样对象B中的属性和方法是不是A也知道了,就是这个意思,在实际的代码中体现在一个类上声明一个成员变量,类似这个

箭头及指向:*带普通箭头的实心线*，指向被拥有者

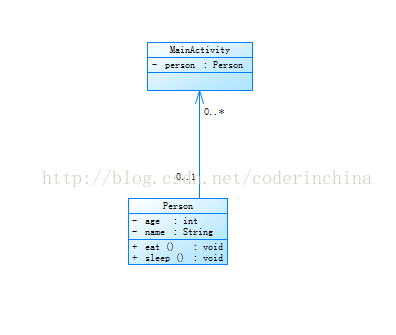
package com.power;import android.support.v7.app.AppCompatActivity;import android.os.Bundle;public class MainActivity extends AppCompatActivity {

private Person person; @Override protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState); setContentView(R.layout.*activity\_main*); person = new Man(); }

}

上面的activity中声明了一个Person类型的成员变量,这就是关联,其实也是一种依赖,类图如下：



它也可以自关联,比如在android中我们也经常写这种代码：

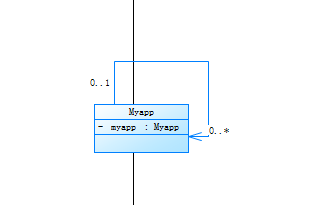
package com.power;import android.app.Application;*/\*\* \* Created by admin on 2017/2/6. \*/*public class MyApp extends Application {

private static MyApp *myApp*; @Override public void onCreate() {

super.onCreate(); *myApp* = this; }

}

它就是一种自关联的关系



聚合（Aggregation）

是整体与部分的关系, 且部分可以离开整体而单独存在,如车和轮胎是整体和部分的关系,轮胎离开车仍然可以存在

聚合关系是关联关系的一种，是强的关联关系；关联和聚合在语法上无法区分，必须考察具体的逻辑关系

比如一个学校有高中部和初中部,这些学生加起来都是属于这个学校的,那么从聚合的概念上讲整体就是这个学校,部分就是学生了,且部分可以离开整体而单独存在,这意思是说学生可以转校去别的学校,但是这个学校还是存在的,不会因为一个学生转校了找个学校就没了,

箭头指向:带空心菱形的实心线，菱形指向整体

代码体现：

package com.power;import java.util.List;*/\*\* \* Created by admin on 2017/2/6. \*/*public class School {

private List<Student> students;}

学生：

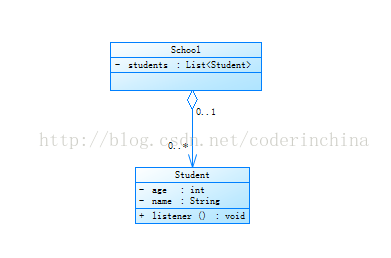
package com.power;*/\*\* \* Created by admin on 2017/2/6. \*/*public class Student {

public int age; public String name; public void listener(){

}

}

类图如下：



组合(Composition)

是整体与部分的关系, 但部分不能离开整体而单独存在. 如公司和部门是整体和部分的关系, 没有公司就不存在部门

组合关系是关联关系的一种，是比聚合关系还要强的关系，它要求普通的聚合关系中代表整体的对象负责代表部分的对象的生命周期

代码体现:成员变量

箭头指向:带实心菱形的实线,菱形指向整体

比如一个大厦肯定是由很多种材料构成的,如果没有这些材料,这大厦就不存在了,意思是这个大厦生命周期over了,

package com.power;*/\*\* \* Created by admin on 2017/2/6. \* 材料 \*/*public class Material {

private int reinforced;//钢筋 在这就用int类型表示了意思下}

House.java

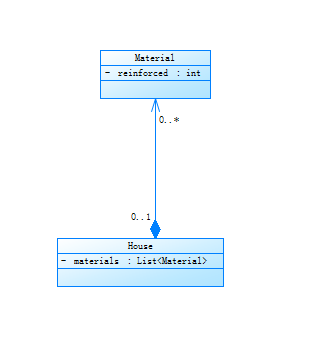
package com.power;import java.util.ArrayList;import java.util.List;*/\*\* \* Created by admin on 2017/2/6. \* 大厦 \*/*public class House {

private List<Material> materials; public House(){

materials = new ArrayList<>(); }

}

类图：



依赖(Dependency)

是一种使用的关系,  即一个类的实现需要另一个类的协助,所以要尽量不使用双向的互相依赖

代码表现:局部变量、方法的参数或者对静态方法的调用

箭头指向：带箭头的虚线，指向被使用者

package com.power;*/\*\* \* Created by admin on 2017/2/6. \*/*public class Student {

public int age; public String name; public void listener(){

}

public void notice(){

System.*out*.print("高三学生初八开始上课"); }

}

package com.power;import java.util.List;*/\*\* \* Created by admin on 2017/2/6. \*/*public class School {

public void notiy(List<Student> students){

if(students!=null&&!students.isEmpty()){

for(Student stu:students){

stu.notice();//学校通知所有学生去上课 }

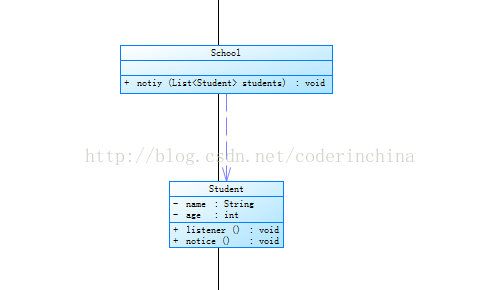
}

}

}

这个School类中的notity()方法功能就是通知所有高三的学生什么时候开始上课,我们看到notiy()方法中的形参是一个集合,这种就是依赖关系,即一个类的实现需要另一个类的协助,也就是说School这个类要实现notiy()方法必须得依赖Student这个类,不然没办法实现!

类图：



各种关系的强弱顺序：

泛化 >实现>组合> 聚合> 关联> 依赖

最后把这些关系在java代码中的体现总结下：

1) 依赖关系：关系对象出现在局部变量或者方法的参数里，或者关系类的静态方法被调用   
(2) 关联关系：关系对象出现在实例变量中   
(3) 聚合关系:关系对象出现在实例变量中   
(4) 组合关系：关系对象出现在实例变量中   
(5) 泛化关系: extends   
(6) 实现： implements