*
将学生和工人的共性描述提取出来，单独进行描述，
只要让学生和工人与单独描述的这个类有关系，就可以了。

继承：
1，提高了代码的复用性。
2，让类与类之间产生了关系。有了这个关系，才有了多态的特性。

注意：千万不要为了获取其他类的功能，简化代码而继承。
必须是类与类之间有所属关系才可以继承。所属关系 is a。

Java 语言中：java 只支持单继承，不支持多继承。

因为多继承容易带来安全隐患:当多个父类中定义了相同功能，
当功能内容不同时，子类对象不确定要运行哪一个。

```
class A
{
	void show()
	{
		System.out.println("a");
	}
}
class B
{
	void show()
	{
		System.out.println("b");
	}
}

class C extends A,B
{}

C c = new C();
c.show();
```

*
子父类出现后，类成员的特点：

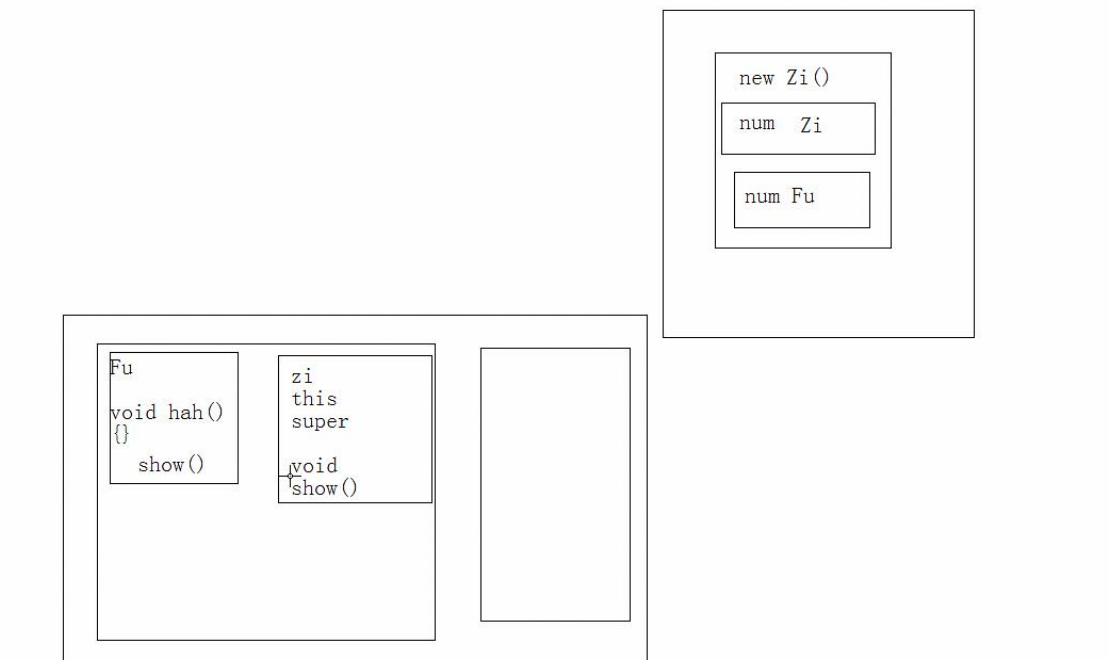
类中成员：
1，变量。

2，函数。
3，构造函数。

1,变量
如果子类中出现非私有的同名成员变量时，
子类要访问本类中的变量，用 this
子类要访问父类中的同名变量，用 super。

super 的使用和 this 的使用几乎一致。
this 代表的是本类对象的引用。
super 代表的是父类对象的引用。

*/



/*
2,子父类中的函数。

当子类出现和父类一模一样的函数时，　　这样有利于扩展

当子类对象调用该函数，会运行子类函数的内容。
如同父类的函数被覆盖一样。

这种情况是函数的另一个特性：重写(覆盖)

当子类继承父类，沿袭了父类的功能，到子类中，
但是子类虽具备该功能，但是功能的内容却和父类不一致，
这时，没有必要定义新功能，而是使用覆盖特殊，保留父类的功能定义，并重写功能内容。

覆盖：
1，子类覆盖父类，必须保证子类权限大于等于父类权限，才可以覆盖，否则编译失败。

2，静态只能覆盖静态。

记住大家：
重载：只看同名函数的参数列表。
重写：子父类方法要一模一样。
*/

/*
3，子父类中的构造函数。

在对子类对象进行初始化时，父类的构造函数也会运行，
那是因为子类的构造函数默认第一行有一条隐式的语句 super();
super():会访问父类中空参数的构造函数。而且子类中所有的构造函数默认第一行都是 super();

为什么子类一定要访问父类中的构造函数。

因为父类中的数据子类可以直接获取。所以子类对象在建立时，需要先查看父类是如何对这些数据进行初始化的。
所以子类在对象初始化时，要先访问一下父类中的构造函数。
如果要访问父类中指定的构造函数，可以通过手动定义 super 语句的方式来指定。

注意：super 语句一定定义在子类构造函数的第一行。

子类的实例化过程。

结论：
子类的所有的构造函数，默认都会访问父类中空参数的构造函数。
因为子类每一个构造函数内的第一行都有一句隐式 super();

当父类中没有空参数的构造函数时，子类必须手动通过 super 语句形式来指定要访问父类中的构造函数。

当然：子类的构造函数第一行也可以手动指定 this 语句来访问本类中的构造函数。
子类中至少会有一个构造函数会访问父类中的构造函数。
*/

/*
final：最终。作为一个修饰符，
1，可以修饰类，函数，变量。
2，被 final 修饰的类不可以被继承。为了避免被继承，被子类复写功能。
3，被 final 修饰的方法不可以被复写。
4，被 final 修饰的变量是一个常量只能赋值一次，既可以修饰成员变量，有可以修饰局部变量。
　　当在描述事物时，一些数据的出现值是固定的，那么这时为了增强阅读性，都给这些值起个名字。方便于阅读。
　　而这个值不需要改变，所以加上 final 修饰。作为常量：常量的书写规范所有字母都大写，如果由多个单词组成。
　　单词间通过_连接。
5，内部类定义在类中的局部位置上是，只能访问该局部被 final 修饰的局部变量。

*/

*

当多个类中出现相同功能，但是功能主体不同，

这是可以进行向上抽取。这时，只抽取功能定义，而不抽取功能主体。

抽象：看不懂。

抽象类的特点：

1，抽象方法一定在抽象类中。

2，抽象方法和抽象类都必须被 abstract 关键字修饰。

3，抽象类不可以用 new 创建对象。因为调用抽象方法没意义。

4，抽象类中的抽象方法要被使用，必须由子类复写起所有的抽象方法后，建立子类对象调用。

如果子类只覆盖了部分抽象方法，那么该子类还是一个抽象类。

抽象类和一般类没有太大的不同。

该如何描述事物，就如何描述事物，只不过，该事物出现了一些看不懂的东西。

这些不确定的部分，也是该事物的功能，需要明确出现。但是无法定义主体。

通过抽象方法来表示。

抽象类比一般类多个了抽象函数。就是在类中可以定义抽象方法。

抽象类不可以实例化。

特殊：抽象类中可以不定义抽象方法，这样做仅仅是不让该类建立对象。

```
/*
接口：初期理解，可以认为是一个特殊的抽象类
    当抽象类中的方法都是抽象的，那么该类可以通过接口的形式来表示。
class 用于定义类
interface  用于定义接口。

接口定义时，格式特点：
1，接口中常见定义：常量，抽象方法。
2，接口中的成员都有固定修饰符。
    常量：public static final
    方法：public abstract
记住：接口中的成员都是 public 的。

接口：是不可以创建对象的，因为有抽象方法。
需要被子类实现，子类对接口中的抽象方法全都覆盖后，子类才可以实例化。
否则子类是一个抽象类。

接口可以被类多实现，也是对多继承不支持的转换形式。java 支持多实现。
*/
interface Inter
{
    public static final int NUM = 3;
    public abstract void show();
}

interface InterA
{
    public abstract void show();
}

class Demo
{
    public void function(){}
}

class Test extends Demo implements Inter,InterA
{
    public void show(){}
}
```

多态

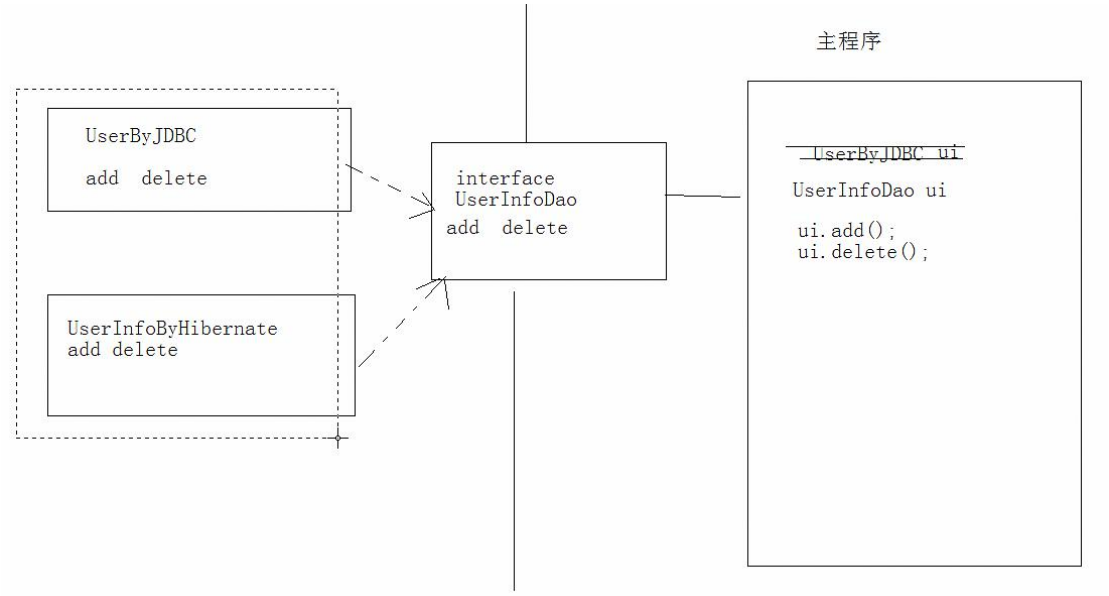
```
abstract class Animal
{
        public abstract void eat();
}
class cat extends Animal
{
        public void eat()
        {
                System.out.println("fish");
        }
        void sleep()
        {
                System.out.println("i like sleeping");
        }
}
class dog extends Animal
{
        public void eat()
        {
                System.out.println("bone");
        }
}
class extendDemo2
{
        public static void main(String[] args)
        {
                cat c= new cat();
                dog d = new dog();

                function(c);
                function(d);
                function(new cat());
        }

  public static void function (Animal a) //jia jingtai shi yinwei
zhuhanshu yao diaoyong {
                a.eat();
        }
}
```

/*
在多态中成员函数的特点：
在编译时期：参阅引用型变量所属的类中是否有调用的方法。如果有，编译通过，如果没有编译失败。
在运行时期：参阅对象所属的类中是否有调用的方法。
简单总结就是：成员函数在多态调用时，编译看左边，运行看右边。

在多态中，成员变量的特点：
无论编译和运行，都参考左边(引用型变量所属的类)。

在多态中，静态成员函数的特点：
无论编译和运行，都参考做左边。

*/
关于数据库的一些操作，如何扩展



嵌套类

/*

内部类的访问规则：

1，内部类可以直接访问外部类中的成员，包括私有。

之所以可以直接访问外部类中的成员，是因为内部类中持有了一个外部类的引用，格式 外部类名.this

2，外部类要访问内部类，必须建立内部类对象。

访问格式：

1,当内部类定义在外部类的成员位置上，而且非私有，可以在外部其他类中。

可以直接建立内部类对象。

格式

外部类名.内部类名　变量名 = 外部类对象.内部类对象;

Outer.Inner in = new Outer().new Inner();

2,当内部类在成员位置上，就可以被成员修饰符所修饰。

比如，private：将内部类在外部类中进行封装。

static:内部类就具备 static 的特性。

当内部类被 static 修饰后，只能直接访问外部类中的 static 成员。出现了访问局限。

在外部其他类中，如何直接访问 static 内部类的非静态成员呢？

new Outer.Inner().function();

在外部其他类中，如何直接访问 static 内部类的静态成员呢？

uter.Inner.function();----------（不重要）

注意：当内部类中定义了静态成员，该内部类必须是 static 的。

当外部类中的静态方法访问内部类时，内部类也必须是 static 的。

当描述事物时，事物的内部还有事物，该事物用内部类来描述。

因为内部事务在使用外部事物的内容。

*/

匿名内部类

```
new AbsDemo() //niming duixiang, zhuyi zheli meiyou fenhao
{
    void show()
    {
      System.out.println("x=="+x);
    }
}.show();
```

异常处理

/*
异常：就是程序在运行时出现不正常情况。
异常由来：问题也是现实生活中一个具体的事物，也可以通过 java 的类的形式进行描述。并封装成对象。
其实就是 java 对不正常情况进行描述后的对象体现。

对于问题的划分：两种：一种是严重的问题，一种非严重的问题。

对于严重的，java 通过 Error 类进行描述。
对于 Error 一般不编写针对性的代码对其进行处理。

对与非严重的，java 通过 Exception 类进行描述。
对于 Exception 可以使用针对性的处理方式进行处理。

无论 Error 或者 Exception 都具有一些共性内容。
比如：不正常情况的信息，引发原因等。

```
Throwable
        |--Error
        |--Exception
```

2,异常的处理
java 提供了特有的语句进行处理。

```
try
{
        需要被检测的代码;
```

```
}
catch(异常类 变量)
{
        处理异常的代码；(处理方式)
}
finally
{
        一定会执行的语句；
}

3，对捕获到的异常对象进行常见方法操作。
        String getMessage(): 获取异常信息。
*/
class demo
{
        int div (int a, int b)
        {
                return a/b;
        }
}
class exceptionDemo
{
        public static void main(String[] args)
        {
                demo d = new demo();
                try{
                        int x = d.div(4,0);
                        System.out.println(x);
                }

                catch(Exception e)
                {
                        System.out.println("there is sometihing wrong");
                        System.out.println(e.getMessage());
                        System.out.println(e.toString());
                        e.printStackTrace();
                }
                System.out.println("lalalalala");
        }
}
```

throw 应该有主函数 throw 或者 catch 兜着，否则编译不通过！

对多异常的处理。

1，声明异常时，建议声明更为具体的异常。这样处理的可以更具体。

2，对方声明几个异常，就对应有几个 catch 块。不要定义多余的 catch 块。

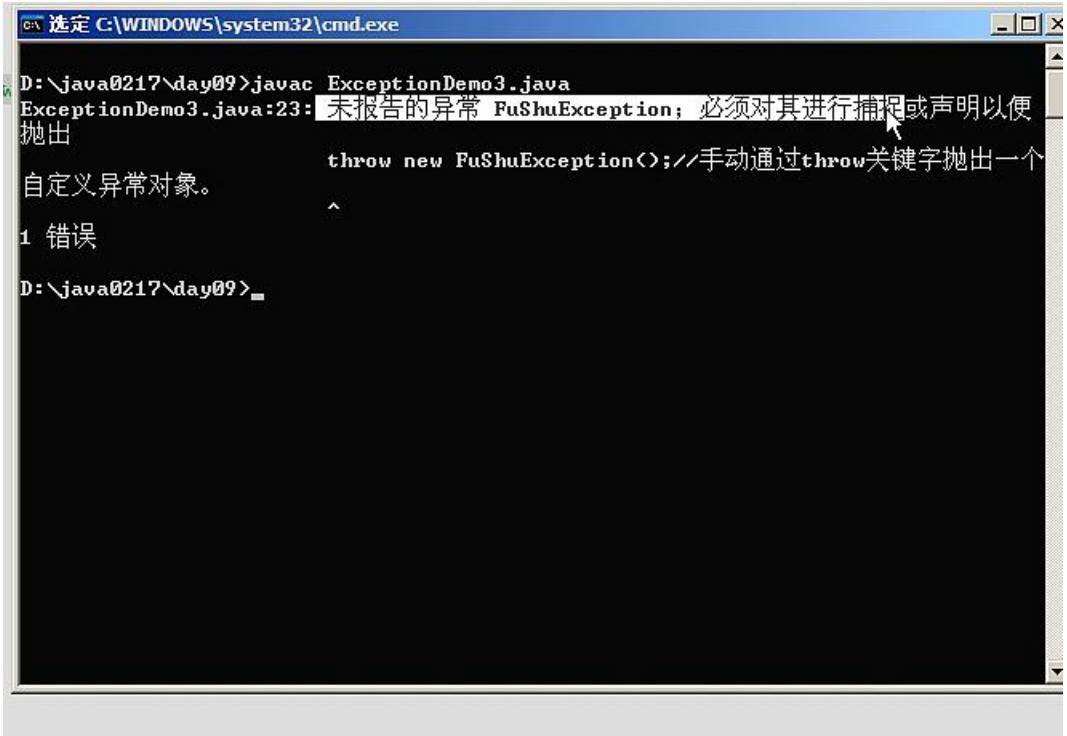
如果多个 catch 块中的异常出现继承关系，父类异常 catch 块放在最下面。

建立在进行 catch 处理时，catch 中一定要定义具体处理方式。

不要简单定义一句 e.printStackTrace()，

也不要简单的就书写一条输出语句。

自定义异常



当出现这个的时候，说明编译器已经觉得语法没有问题了，只剩下这个问题

/*
因为项目中会出现特有的问题，
而这些问题并未被 java 所描述并封装对象。
所以对于这些特有的问题可以按照 java 的对问题封装的思想。
将特有的问题。进行自定义的异常封装。

自定义异常。

需求：在本程序中，对于除数是-1，也视为是错误的是无法进行运算的。
那么就需要对这个问题进行自定义的描述。

当在函数内部出现了 throw 抛出异常对象，那么就必须要给对应的处理动作。
要么在内部 try catch 处理。
要么在函数上声明让调用者处理。

一般情况在，函数内出现异常，函数上需要声明。

发现打印的结果中只有异常的名称，却没有异常的信息。
因为自定义的异常并未定义信息。

如何定义异常信息呢？
因为父类中已经把异常信息的操作都完成了。
所以子类只要在构造时，将异常信息传递给父类通过 super 语句。
那么就可以直接通过 getMessage 方法获取自定义的异常信息。！！！！

自定义异常：
必须是自定义类继承 Exception。

继承 Exception 原因：
异常体系有一个特点：因为异常类和异常对象都被抛出。
他们都具备可抛性。这个可抛性是 Throwable 这个体系中独有特点。

只有这个体系中的类和对象才可以被 throws 和 throw 操作。

throws 和 throw 的区别
throws 使用在函数上。
throw 使用在函数内。

throws 后面跟的异常类。可以跟多个。用逗号隔开。
throw 后跟的是异常对象。
*/

Runtime exception

/*

Exceptoin 中有一个特殊的子类异常 RuntimeException 运行时异常。

如果在函数内容抛出该异常，函数上可以不用声明，编译一样通过。

如果在函数上声明了该异常。调用者可以不用进行处理。编译一样通过；

之所以不用在函数声明，是因为不需要让调用者处理。
当该异常发生，希望程序停止。因为在运行时，出现了无法继续运算的情况，希望停止程序后，
对代码进行修正。

自定义异常时：如果该异常的发生，无法在继续进行运算，
就让自定义异常继承 RuntimeException。

对于异常分两种：
1，编译时被检测的异常。

2，编译时不被检测的异常(运行时异常。RuntimeException 以及其子类)

*/

Finally

/*

finally 代码块：定义一定执行的代码。

通常用于关闭资源。

*/

分层处理（这也是对于问题的封装）

```
class NoException extends Exception
{
}

public void method()throws NoException
{

        连接数据库;
         数据操作;//throw new SQLException()
        关闭数据库;//该动作，无论数据操作是否成功，一定要关闭资源。


        try
        {

                连接数据库;

                数据操作;//throw new SQLException();
        }
        catch (SQLException e)
        {
                会对数据库进行异常处理;
                throw new NoException();

        }
        finally
        {
                关闭数据库;
        }


}
```

```
/*
异常在子父类覆盖中的体现；
1，子类在覆盖父类时，如果父类的方法抛出异常，那么子类的覆盖方法，只能抛出父类的异常或者该异常的子类。
2，如果父类方法抛出多个异常，那么子类在覆盖该方法时，只能抛出父类异常的子集。
3，如果父类或者接口的方法中没有异常抛出，那么子类在覆盖方法时，也不可以抛出异常。
        如果子类方法发生了异常。就必须要进行 try 处理。绝对不能抛。

*/
```

正常流程代码和问题流程代码的分离

Package

对类文件进行分类管理。

给类提供多层命名空间。

写在程序文件的第一行。

类名的全称的是　包名.类名。

包也是一种封装形式。

运行文件和源文件相分离

Javac -d c:.....　。。。。。。Java

```
PackageDemo.java:8: 找不到符号
符号:  类 DemoA
位置:  类 pack.PackageDemo
                DemoA d = new DemoA();
                ^
PackageDemo.java:8: 找不到符号
符号:  类 DemoA
位置:  类 pack.PackageDemo
                DemoA d = new DemoA();
                          ^
2 错误
```

错误原因：类名写错。

因为类名的全名是：包名.类名

```
PackageDemo.java:8: 软件包 packa 不存在
                packa.DemoA d = new packa.DemoA();
                    ^
PackageDemo.java:8: 软件包 packa 不存在
                packa.DemoA d = new packa.DemoA();
                                    ^
2 错误
```

错误原因：packa 包不在当前目录下
需要设置 classpath，告诉 jvm 去哪里找指定的 packa 包。

```
PackageDemo.java:8: packa.DemoA 在 packa 中不是公共的；无法从外部软件包
中对其进
行访问
                packa.DemoA d = new packa.DemoA();
                    ^
PackageDemo.java:8: packa.DemoA 在 packa 中不是公共的；无法从外部软件包
中对其进
行访问
                packa.DemoA d = new packa.DemoA();
                                    ^
2 错误
```

错误原因：有了包，范围变大，一个包中的类要被访问，必须要有足够大的权限。
所以被访问的类要被 public 修饰。

```
PackageDemo.java:9: show() 在 packa.DemoA 中不是公共的；无法从外部软件包
中对其进
行访问
                d.show();
                 ^
1 错误
```

错误原因：类公有后，被访问的成员也要公有才可以被访问。

Jar 压缩包 day10 最后一个视频还没看。。。