StarUML使用说明-指导手册

# 综述

UML（Unified Modeling Language，统一建模语言）是一个支持模型化和软件系统开发的图形化语言，为软件开发的所有阶段提供模型化和可视化支持，包括由需求分析到规格，到构造和配置。常见的几种UML图有

用例图：从用户角度描述系统功能，并指各功能的操作者。

类图：描述系统中类的静态结构。

顺序图：对象之间的动态合作关系，强调对象发送消息的顺序，同时显示对象之间的交互。

协作图：描述对象之间的协助关系。

状态图：是描述状态到状态控制流，常用于动态特性建模。

活动图：描述了业务实现用例的工作流程等。

StarUML是一种生成类图和其他类型的统一建模语言(UML)图表的工具。这是一个用Java语言描述的创建类图的简明手册

下载

[下载](http://staruml.io/download)StarUML-v2.8.1.msi（本文环境），可以支持mdj和uml格式

破解:

## 破解

https://blog.csdn.net/qq\_23370223/article/details/78069728

​1.使用Editplus或者Notepad++等特殊的文本编辑器打开 安装位置下/www/license/node/LicenseManagerDomain.js文件

２.修改validate方法，在如下指定的位置上添加指定的代码

function validate(PK, name, product, licenseKey) {

var pk, decrypted;

**//添加代码开始**

**return {**

**name: "hancher",**

**product: "StarUML",**

**licenseType: "vip",**

**quantity: "www.hancher.com",**

**licenseKey: "hello,hancher!"**

**}**

**//添加代码结束**

try {

pk = new NodeRSA(PK);

decrypted = pk.decrypt(licenseKey, 'utf8');

} catch (err) {

return false;

}

var terms = decrypted.trim().split("\n");

if (terms[0] === name && terms[1] === product) {

return {

name: name,

product: product,

licenseType: terms[2],

quantity: terms[3],

licenseKey: licenseKey

};

} else {

return false;

}

}

在如上位置，插入如上代码。

3.验证打开StarUML，打开菜单Help->Enter License，输入上面指定的name和license信息，分别为hancher和hello,hancher!，成功！

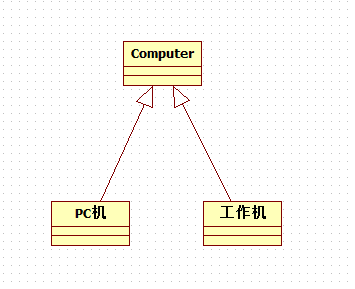
然后就有注册成功的提示信息了，help->about startuml，可以看到​提示信息已注册

## 概念

### 类与类间的关系

#### 泛化（Generalization）

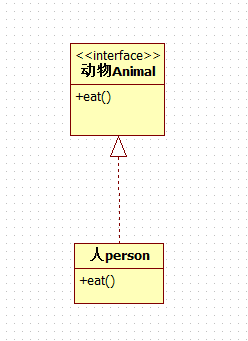
泛化（也有人直接称为继承）是类与类间或者接口与接口间最常见的的关系指的是一个类（子类、子接口）继承另一个类（父类、父接口）的属性与方法，并且可以扩展它自己的新属性与方法。二者是is-a的关系，通俗地讲就是“这个东西是那个东西的一种”，再比如PC机是计算机，工作站也是计算机。PC机和工作站是两种不同类型的计算机，但都继承了计算机的共同特性。在Java中用“extend”来表示此关系。在类图中使用带空心三角箭头的实线表示，箭头从子类、子接口指向父类、父接口。



泛化is-a（空心三角箭头的实线）

#### 实现（Realization）

指的是一个类实现接口的功能。在类图中使用带空心三角箭头的虚线表示，箭头从实现类指向接口。例如动物都有eat吃的这个行为，人是动物的一个具体实例，实现具体的eat动作

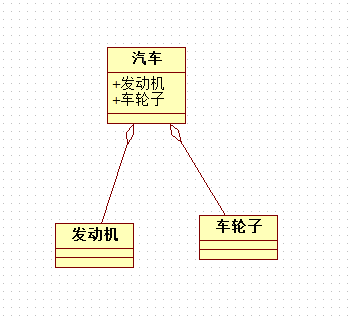


实现（空心三角箭头的虚线）

两个相对独立的对象，当一个对象的实例与另外一个对象的特定实例存在固定关系时，这两个对象之间就存在关联关系

#### 聚合（Aggregation）

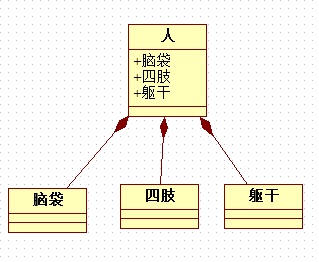
指的是整体和部分的弱关系，是has-a的关系，此时整体与部分之间是可分离的，即没有了整体，局部也可单独存在。比如公司和员工的关系，公司包含员工，但如果公司倒闭，员工依然可以换公司。处于聚合关系的两个类生命周期不同步，即他们可以具有各自的生命周期——当A创建的时候，B不一定创建；当A消亡时，B不一定消亡。在类图使用空心菱形的实线表示，菱形从局部指向整体。



聚合has-a（空心菱形的实线）

#### 组合（Composite）

指的是整体和部分的强关系，是一种contains-a的关系，部分不能脱离整体存在。整体的生命周期结束也就意味着部分的生命周期结束，比如“国破家亡”。在类图使用实心菱形的实线表示，菱形从局部指向整体。处。

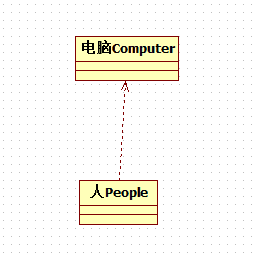


组合contains-a（实心菱形的实线）

#### 依赖（Dependency）

类A要完成某个功能必须引用类B，但这种使用关系是具有偶然性的、临时性的、非常弱的，而B类的变化会影响到A，则A与B存在依赖关系，依赖关系是弱的关联关系。比如本来人与电脑没有关系的，但由于偶然的机会，人需要用电脑写程序，这时候人就依赖于电脑。在java中表现为局域变量、方法的形参，或者对静态方法的调用。在类图使用带箭头的虚线表示，箭头从使用类指向被依赖的类。

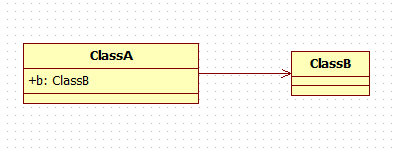
依赖（箭头的虚线）



依赖（箭头的虚线）

#### 6、关联（Association）

关联是指一个类用到了另外的一个类，但不依赖。一般是长期性的，而且双方的关系一般是平等的。关联可以是自身关联、单向关联、双向关联。在java中一般使用成员变量来实现。在类图使用带箭头的实线表示，箭头指向被拥有者。



关联（箭头的实线）

关联和依赖的区别——

从类的属性是否增加的角度看：

发生依赖关系的两个类都不会增加属性。发生关联关系的两个类，其中的一个类成为另一个类的属性，而属性是一种更为紧密的耦合，更为长久的持有关系。

从关系的生命周期来看：

依赖关系是仅当类的方法被调用时而产生，伴随着方法的结束而结束了。关联关系是当类实例化的时候即产生，当类销毁的时候，关系结束。相比依赖讲，关联关系的生存期更长。

代码演示聚合、组合间的区别

就结合上面的几个例子来讲，比如人拥有脑袋，人死了，脑袋也废了，这是个组合关系。人可以有一台电脑，但二者不是共生关系，这个是聚合关系。

public class Computer {

}

public class Brain {

}

public class Person {

private Computer computer;

public Person(Computer computer){

this.computer = computer; //聚合

}

private Brain brain;

public Person(){

this.brain=new Brain(); //组合

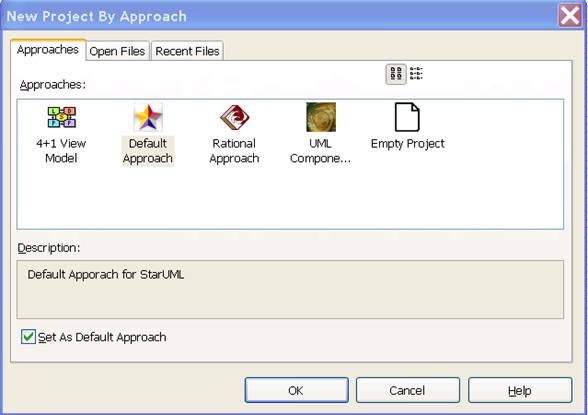
}

}

非商业转载请注明出处。明出处。

# 基本流程

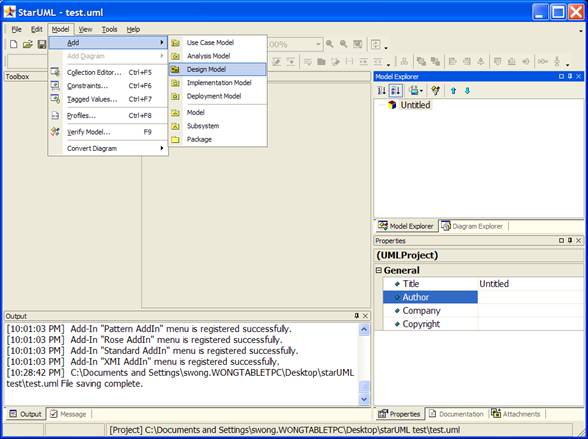
New Project By Approach的对话框会弹出。选择“Empty Project”并且按下"确定"。这里建议你不要勾选"设置为默认的做法" 复选框。



## 模块管理

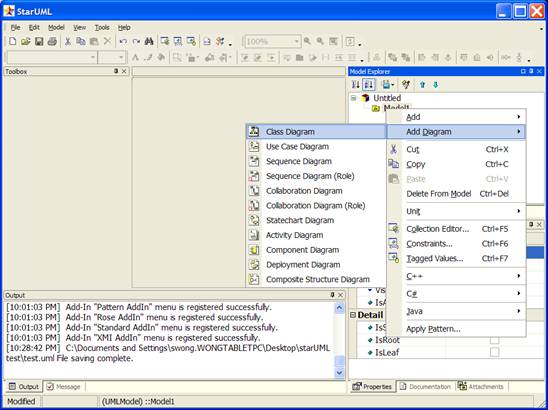
选择模块：在右边的“Model Explorer”框中选定“Untitled”模块。

添加模块：通过“Model”主菜单，或右击选定的模型，可以" Add/Design Model "



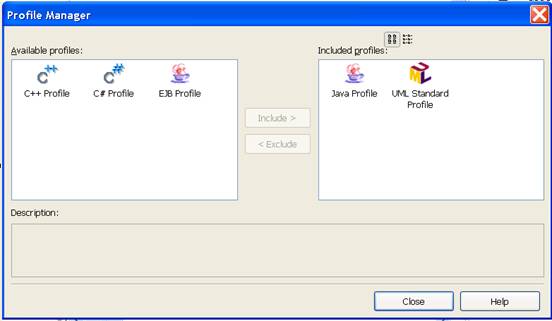
## 类图管理

通过“Model”主菜单，或右击选定模型，可以“Add Diagram/Class Diagram”：



## 设置profile

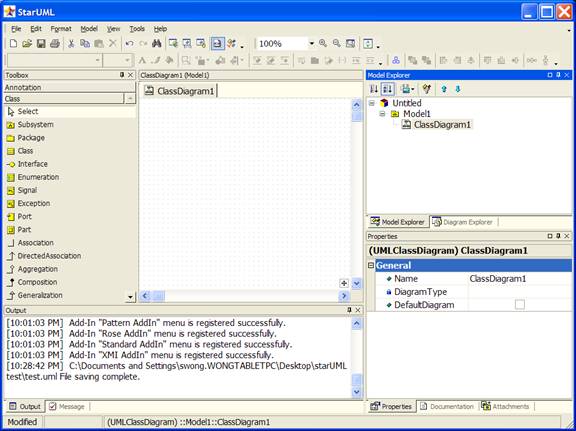
通过“Model/Profile...”菜单去设置工程所需的profile。这决定了工程所使用的规则和约定。一定要包含"JAVA Porfile"这一项目。



## 保存工程

立即就保存工程，这样在出现问题的时候，您就不会丢失信息。

从“File ”菜单，选择“Save” ，并选择一个地方以保存工程。你的StarUML项目现在应该看起来的是这样的：



## .创造图表

现在，开始真正创造图表

### 类管理

从默认就在屏幕的左边的“Toolbox”选择“类”图标，然后左键单击diagram窗口的某处。这样就使用通用名字创造了一个新的类。双击，将类改名为Circle。

#### 添加属性

右击图中的目标，在弹出菜单中选择“Add”中的“Attribute”(被标示为绿色)，为其添加一个属性(或者域)，填入期望的名字“\_radius”。

l 具体的数据类型，在属性面板（右下侧的窗口） ，由双打字，在"类型"时段。 在窗体右下边的Properties面板中，找到“Type”输入框，输入double作为\_radius属性的类型。

l 类的内部数据（域/属性）都是私有的，因为他们是严格由类内部使用的。所以，在Properties面板中将\_radius设置为“私有”。

继续进行设计

重复同样的过程，添加所谓的名字叫做Rectangle的类和double型的私有成员\_width和\_height。

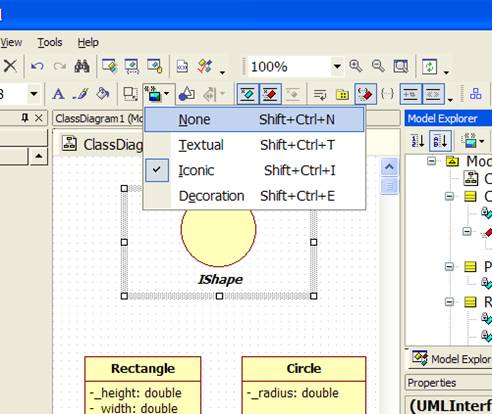
#### 创造 IShape interface

Sdaf

从toolbox中，选择“Interface”，并点击图表的某处。将其改名为IShape。创建以后，选中它。

            在顶部工具栏，选择 “Stereotype Display” 下拉按钮，将值改变为“None”。这将改变以往的圆形形状，使其变为成长方形。

还是在顶部工具栏，取消选中" Suppress Operations "。这将使我们能够看到接口所拥有的方法。



  向IShape 接口添加返回值为double的getArea方法。

      可以通过右击interface的图标，在弹出菜单中点击红色的“Operation”按钮，然后输入getArea。

      设定返回值类型。在“Model Explorer”中展开IShape节点，右击你刚刚创建的getArea方法，并选择“Add Parameter”。在“Properties”框中，将参数的名子变为空，将“DirectionKind”变为“RETURN”，将“Type”变为dboule。

            将IShape和getArea的IsAbstract属性框打上勾，他们在图标上的名字将变为斜体。这是UML的标准，表示这是接口或者其他纯虚实体。

#### 添加类和接口的关系

     可以通过从toolbox中选择表示“Realization”的箭头，并从Circle拖拽向IShape，使Circle实现接口IShape。重复同样的过程，为Rectangle添加实现关系。这是添加了Circle 和 Rectangle对于IShape接口的实现关系。

      如果想使连接线表现为直角的方式，右击连接线，并选择" Format/Line Style/Rectilinear"菜单。你通过这种方式，使箭头重叠在一起，可以使你的图看起来更整洁。

#### 添加类基于接口的行为

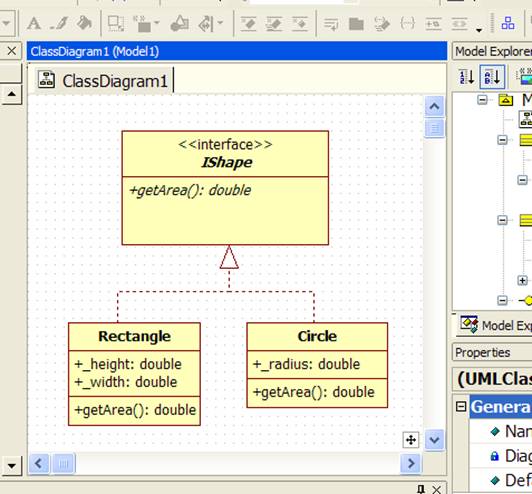
Sdfg

    由于Circle和Rectangle类都实现了IShape接口，就必须有同样的行为(方法)。

        在“Model Explorer”面板中，复制getArea法(按Ctrl-C或者右键点击并选择Copy菜单），并粘贴到Circle和Rectangle类。

        这些实现了的方法在Circle和Rectangle类中都不是抽象的，而是具体的。这是因为他们实际上是执行一些特定行为（例如，为一个圆形和长方形分别计算面积） ，所以不要勾选IsAbstract框。

你的图现在应该是这样的：



### 添加Pizza类

 向Pizza添加double型的私有域\_price。

        添加返回double类型的共有操作getPrice。

#### 为Pizza类添加IShape 的引用

        从toolbox中选择" DirectedAssociation "箭头，点击Pizza类，并向IShape拖拽 。

        选中箭头，在右边的“Properties”框上，将name一栏改为“has-a”，“End1.Aggregation” 一栏改为“AGGREGATE”(这个图示说明Pizza和shape对象是“聚合“的关系)。

        将“End2.Name”一栏改为\_shape 。这样就自动为Pizza添加一个名字为\_shape，使用IShape接口的私有域，的所谓\_shape型ishape以pizza饼。

        将“End2.Visibility”改为私有。

        为\_shape创建一个“获得者”方法，名字叫做getShape，返回IShape 。这就是创建一个行为，名字是getShape，返回IShape 。

#### 为pizza类添加构造函数

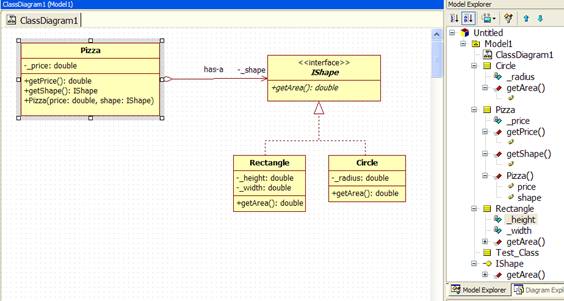
        为Pizza添加构造函数，右击，在弹出的“Add” 菜单中选择“Operation”。从这里，增加一个普通的带有dboule型price参数和IShape类型shape参数的操作

        增加一个输入参数，就像之前增加了一个返回型的输出参数一样，你指定的参数的名称，如价格和形状等，以及适当的数据类型。

        为Circle增加一个带有double型的radius参数的构造函数。

        为Rectangle增加一个带有double型width和height参数的构造函数。

你的图现在应该是这样的：



### 添加Test\_Pizza类

为了说明UML类图更多的功能，又增加了一个叫做“Test\_Pizza”的类，它用作测试目的，并使用到Pizza和IShape类。

        两个类之间的关系有多种形式。举例来说，一个类可以实例化另一个类，而不是将其作为一个成员。又或，一类的方法可能需要另一个类作为输入参数，保留一个引用仅仅是为这个方法的执行。

        通过从toolbox中选择“Dependency”箭头，从一个类拖向他所以来的类，来添加不通类之间的依赖关系。在这个例子中， Test\_Pizza 依赖于Pizza，Circle和Rectangle类，因为它实例化了它们。

        从Properties box选择name属性，或者双击图表上的“依赖线”，可以为依赖关系添加标签。特别的是，当一类实例化另一个类，我们会把依赖线叫做“instantiates”。

        你可以选中并拖动依赖线的标签，以达到更美观的效果。

        依赖关系不会影响代码生成。

## 24.保存项目

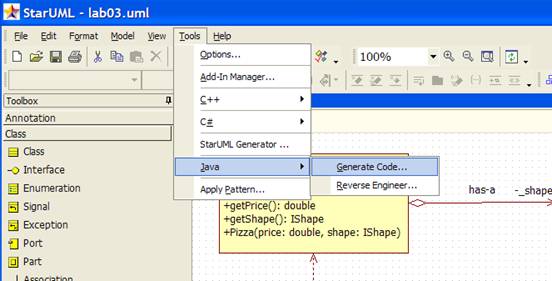
在“File”菜单中，选择“Save”。 SU的所有资料只有一个单一的项目文件，所以你目前应该只有一个文件生成。

25.导出

将图表导出为其他格式，例如图片等，是非常有用的。您可以通过选择“File”菜单的“Export Diagram” ，并且选择合适的文件类型来执行改操作。

生成Java stub代码：

l 点击主菜单的“Tools->Java”菜单，选择“Generate Code”。



从对话框中选择你的模块(这里可能Model1),点击“Next”。

l 为了使你的模块或者图标的所有类都生成stub code，选择“Select All”然后按“Next”。

l 选择一个有效的输出目录，“Next”。

l In the "Options Setup", be sure to check both "Generate the Documentation by JavaDoc" and "Generate empty JavaDoc". All other checkboxes should be unchecked. Then press "Next".在“Options Setup” ，请务必选中“Generate the Documentation by JavaDoc”，“Generate empty JavaDoc”，所有其他复选框不选中，“Next”。

l 现在StarUML将从你的图产生代码，点击“Finish”退出对话框。

l 现在，您可以编辑生成的代码，以增加应用。

# 高级使用

select ：选中某个元素

object：对象

Stimulus：对象与对象间消息

SelfStimulus：对象间自调用的消息

Combined Fragment：结合片段，比如循环片段等，分支片段

Interaction Operand： 交互操作

Frame ： 框架，主题（图最外面的边界）

# REF

2010年11月08日：[StarUML使用说明-指导手册](https://blog.csdn.net/monkey_d_meng/article/details/5995610)

# TASK

https://www.jianshu.com/p/c237396280ea