第1次作业: 类图

1.讨论下列4类事物, 每类有哪些共同的属性(或描述值)和行为(或操作),在每个类上添加更多的事物,对a.b.c.d.的4组事物分别设计4个类图.

a. 显微镜/望远镜/眼镜/瞄准镜/

b. 自行车/卡车/飞机/摩托车/马

c. 帐篷/宿舍/工棚/摩天大楼

d. 手机/服务器/台式机/便携计算机

### ****a. 显微镜/望远镜/眼镜/瞄准镜****

#### ****共同属性****

1. **放大倍数** (ZoomLevel)
2. **材质** (Material)
3. **视场** (FieldOfView)
4. **重量** (Weight)
5. **品牌** (Brand)

#### ****共同行为****

1. **调整焦距** (adjustFocus)
2. **观察物体** (observe)
3. **改变放大倍数** (changeZoom)

### 

### ****b. 自行车/卡车/飞机/摩托车/马****

#### ****共同属性****

1. **速度** (Speed)
2. **承载能力** (LoadCapacity)
3. **能源类型** (EnergyType)
4. **用途** (Usage)
5. **颜色** (Color)

#### ****共同行为****

1. **启动** (start)
2. **停止** (stop)
3. **运输** (transport)
4. **加速** (accelerate)

### 

### ****c. 帐篷/宿舍/工棚/摩天大楼****

#### ****共同属性****

1. **面积** (Area)
2. **材质** (Material)
3. **容纳人数** (Capacity)
4. **地址** (Location)
5. **建筑用途** (Usage)

#### ****共同行为****

1. **提供居住空间** (provideShelter)
2. **分配空间** (allocateSpace)
3. **维护** (maintain)

### 

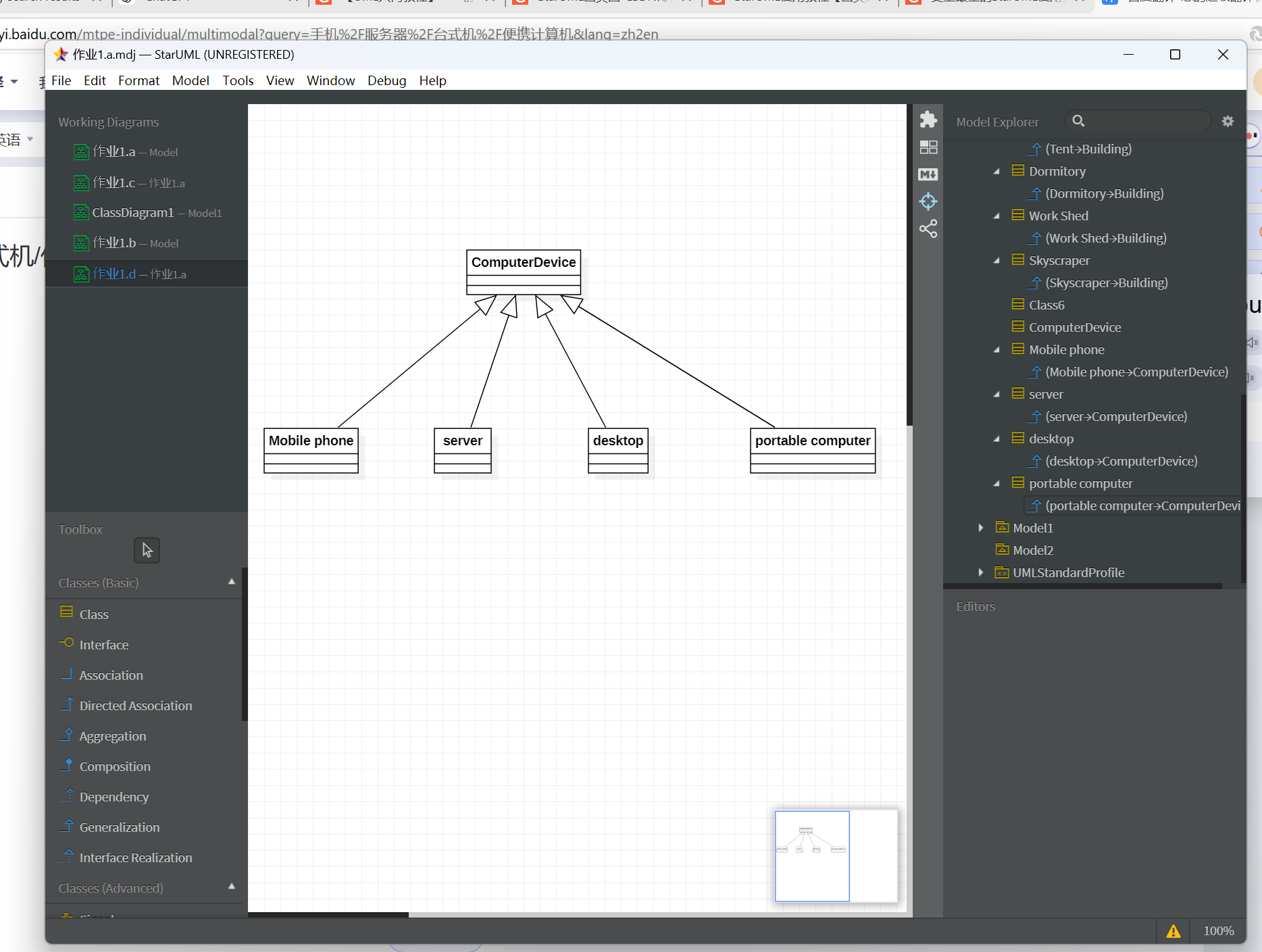
### ****d. 手机/服务器/台式机/便携计算机****

#### ****共同属性****

1. **处理器** (Processor)
2. **内存** (RAM)
3. **存储空间** (Storage)
4. **操作系统** (OperatingSystem)
5. **电池容量** (BatteryCapacity, 可选)

#### ****共同行为****

1. **运行程序** (runApplication)
2. **连接网络** (connectToNetwork)
3. **存储数据** (storeData)
4. **进行通信** (communicate)



2. 当几个独立系统要确定同一对象时,就会出现问题.例如,机动车辆管理部门/保险公司/银行/警察可能都需要识别某台机动车.讨论使用以下识别方法的好处和或不足.对车辆对象设计一个类模型.

a. 根据车辆的拥有者识别;

b. 根据厂商/模型编号/年份等属性确定;

c. 使用由厂商分配给汽车的车辆识别号(VIN);

d. 使用有关管理部门内部生成的ID.

### ****讨论不同识别方法的优劣****

#### ****a. 根据车辆的拥有者识别****

**优点**：

* 能直观地将车辆与其所有者关联。
* 管理方便，对于个人车辆尤其适用。

**不足**：

* 拥有者可能会更换，导致车辆信息失效。
* 不能唯一标识车辆（多人可拥有同款车辆，或者一人有多辆车）。

#### ****b. 根据厂商/模型编号/年份等属性确定****

**优点**：

* 可以体现车辆的基本特性，如车型、年份等。
* 易于查询车辆类别相关数据。

**不足**：

* 不能唯一标识车辆，不同车辆可能共享相同的厂商、型号和年份。
* 难以处理同一车型的多辆车。

#### ****c. 使用由厂商分配的车辆识别号（VIN）****

**优点**：

* 全球唯一，能够唯一标识车辆。
* 广泛应用于车辆管理中，便于跨系统交互。

**不足**：

* 需要依赖厂商的正确分配和维护。
* 用户通常不容易记住 VIN，查询可能不直观。

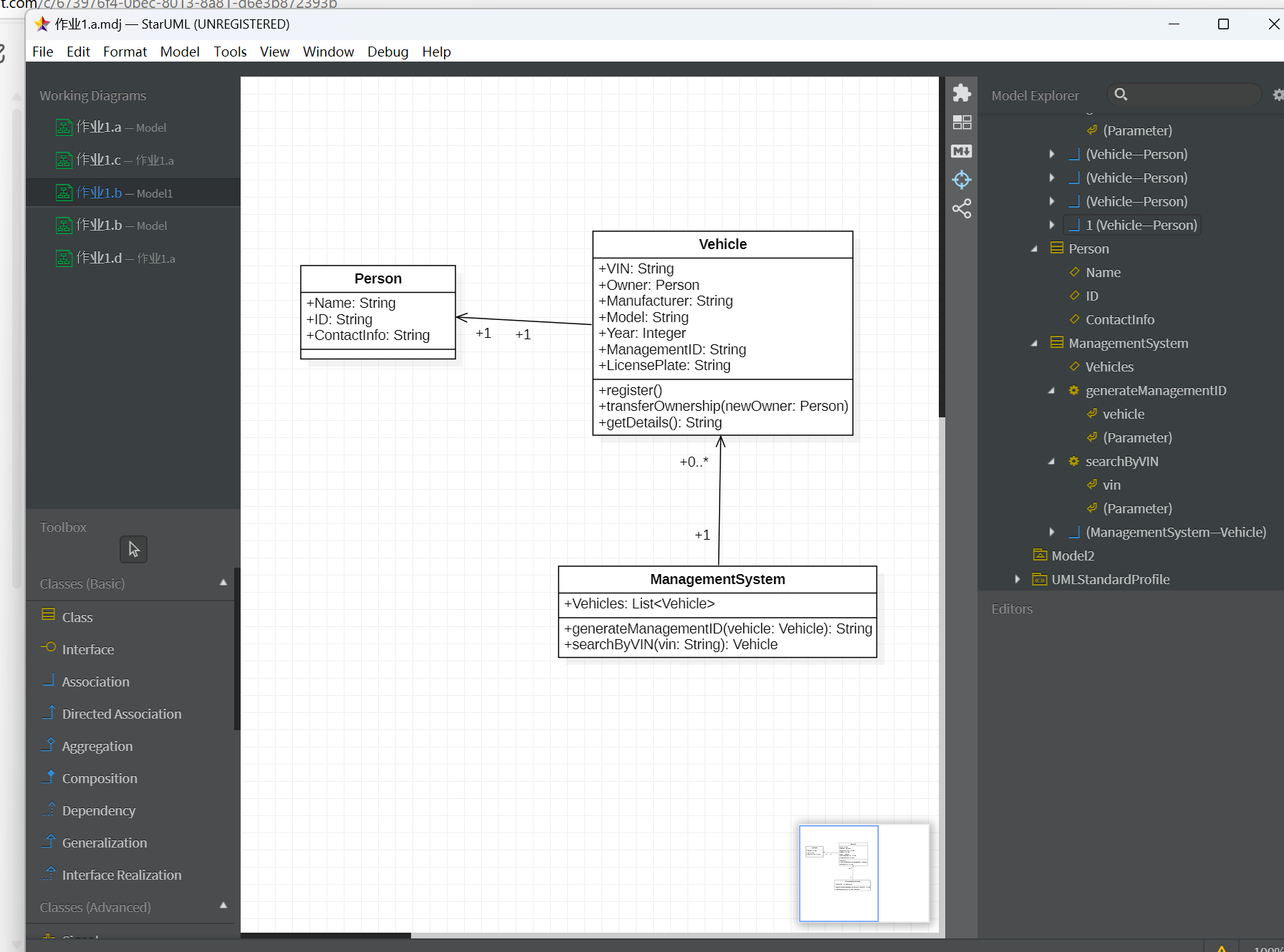
#### ****d. 使用有关管理部门内部生成的 ID****

**优点**：

* 能适应本地管理需求，生成规则灵活。
* 可结合其他信息，如地区、登记时间等。

**不足**：

* 不具备全球唯一性，跨系统交互时可能冲突。
* 管理部门需要额外维护生成规则。



3 所有的对象都有标识, 都可以辨别. 考察一张商品的购买发票, 如图所示, 其中对买方和买方分别采用了哪些标识, 这些标识如何设计的?例如:购买方名称:"华中科技大学"字符串,购买方名称全称.对发票中的全部数据项设计类和类图.



### ****发票中的标识分析****

从提供的发票图片中，可以看到以下几个重要标识：

#### ****买方（购买方）的标识****

* **名称**: 购买方名称 （如 "华中科技大学"），用来标识购买方的主体。
* **纳税人识别号**: 唯一标识一个购买方的税务身份。
* **地址及电话**: 进一步验证买方信息并提供联络方式。
* **开户行及账号**: 标识买方的银行账户信息，用于交易结算。

#### ****卖方（销售方）的标识****

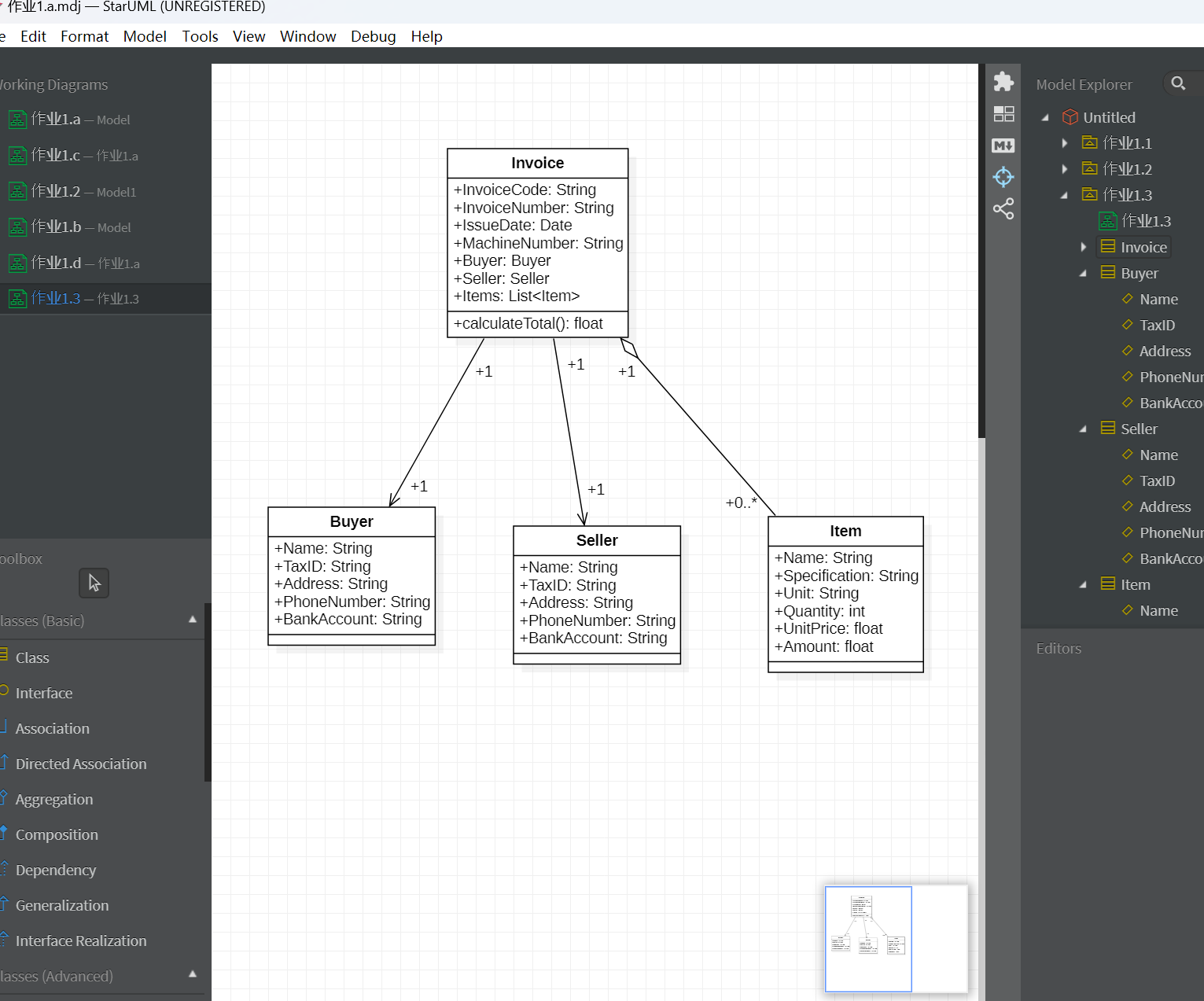
* **名称**: 销售方名称，如四川创维电器有限公司。
* **纳税人识别号**: 唯一标识卖方的税务身份。
* **地址及电话**: 用于核实卖方身份和联系卖方。
* **开户行及账号**: 识别卖方银行账户，用于交易和资金流转。

#### ****发票的标识****

* **发票代码**: 唯一标识发票模板。
* **发票号码**: 唯一标识一张具体的发票。
* **机器编号**: 标识发票的打印机器。
* **开票日期**: 用于记录和验证交易时间。

#### ****商品（服务）的标识****

* **商品（服务）名称**: 描述购买的商品或服务。
* **规格型号**: 标识商品的具体属性。
* **单位和数量**: 表示商品数量及计量单位。
* **单价与金额**: 标识交易价格。



4 绘制一个类模型来描述无向图.无向图由一组顶点和边组成.边连接顶点对.包括顶点的名称和位置,边的名称/宽度和颜色.类似的, 绘制一个类模型来描述有向图, 有向图的边是有方向的.

### ****1. Graph（图）类****

#### ****属性****

* Vertices: List<Vertex>  
  存储图中所有的顶点，使用列表存储顶点对象。
* Edges: List<Edge>  
  存储图中所有的边，使用列表存储边对象。

#### ****操作****

* addVertex(vertex: Vertex)  
  向图中添加一个顶点。
* addEdge(edge: Edge)  
  向图中添加一条边。
* removeVertex(vertex: Vertex)  
  从图中移除一个顶点，同时移除与该顶点相关的边。
* removeEdge(edge: Edge)  
  从图中移除一条边。

### ****2. Vertex（顶点）类****

#### ****属性****

* Name: String  
  顶点的名称，用于唯一标识顶点。
* Position: (float, float)  
  顶点的位置坐标，通常是二维坐标（x, y）。

#### ****操作****

（无操作，顶点通常只存储数据。）

### ****3. Edge（边）类****

#### ****属性****

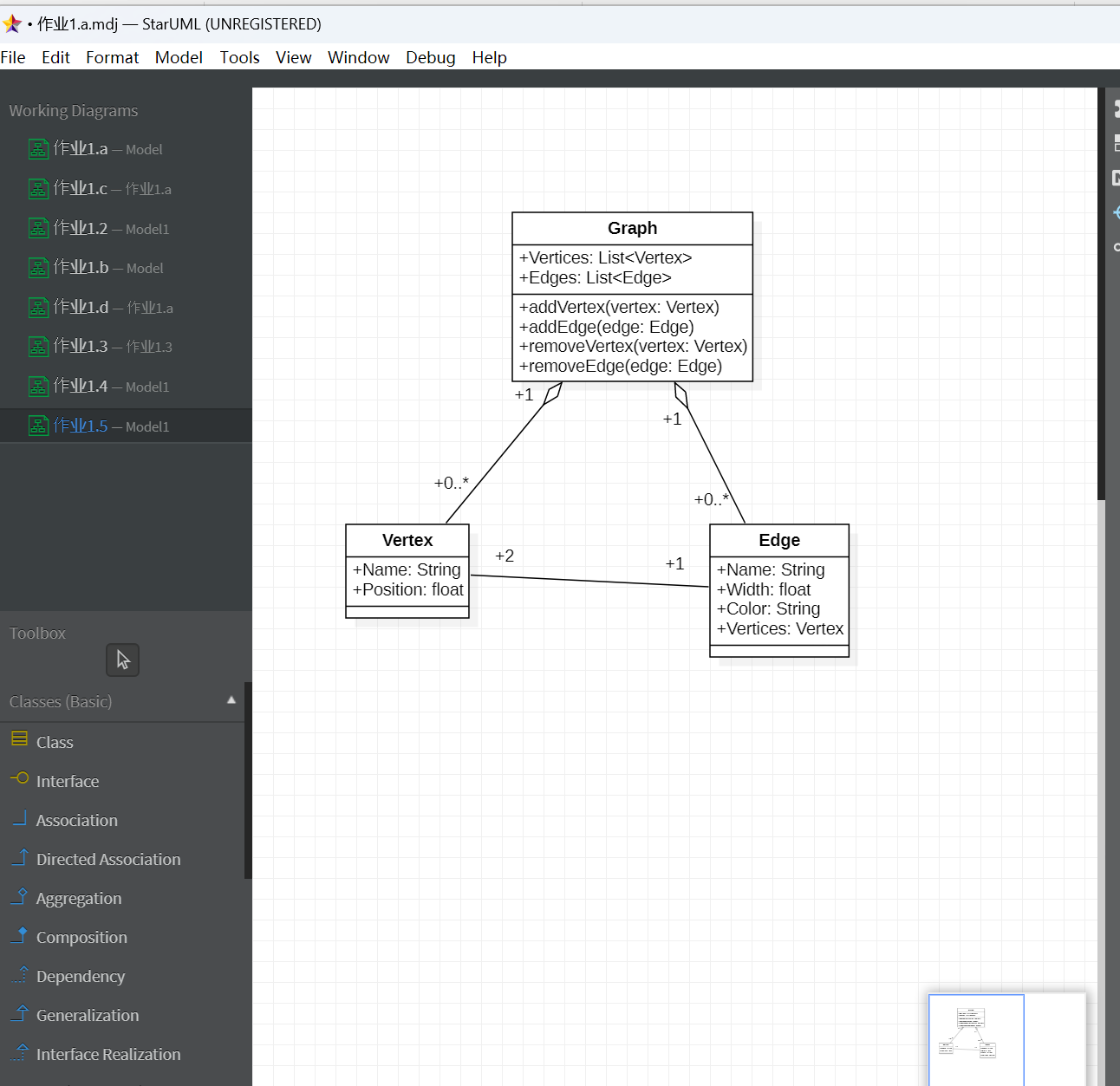
* Name: String  
  边的名称，用于唯一标识边。
* Width: float  
  边的宽度，表示边的粗细。
* Color: String  
  边的颜色，用于区分不同类型的边或表示某种状态。
* Vertices: (Vertex, Vertex)  
  边连接的两个顶点。

#### ****操作****

（无操作，边通常只存储数据。）

### ****类关系****

* Graph **类** 是核心类，负责管理顶点和边。
* Vertex **和** Edge 是 Graph 的组成部分，分别表示图的顶点和边。
* Edge 与 Vertex 通过属性 Vertices 建立关联。



5 为哲学家就餐问题绘制一个类图.圆桌周围有5位哲学家, 5把餐叉. 每位哲学家可以使用两把餐叉, 一边一把. 每把餐叉可以被两位哲学家使用. 每把餐叉要么放在桌上, 要么正在被哲学家使用. 哲学家必须用两把餐叉才能吃饭.

### ****1. Philosopher（哲学家）类****

#### ****属性****

* Name: String  
  哲学家的名字，用于唯一标识哲学家。
* State: String  
  哲学家的状态，可以是以下之一：

**Thinking**：哲学家正在思考。

**Eating**：哲学家正在进餐。

**Waiting**：哲学家正在等待餐叉。

#### ****操作****

* think()  
  表示哲学家进入思考状态。
* eat()  
  表示哲学家进入进餐状态。
* pickUpForks(left: Fork, right: Fork)  
  哲学家尝试拾起左侧和右侧的餐叉，如果成功，则改变状态为 Eating。
* putDownForks(left: Fork, right: Fork)  
  哲学家放下两把餐叉，并改变状态为 Thinking。

### ****2. Fork（餐叉）类****

#### ****属性****

* ID: int  
  餐叉的编号，用于唯一标识餐叉。
* State: String  
  餐叉的状态，可以是以下之一：
  + **OnTable**：餐叉在桌子上，未被使用。
  + **InUse**：餐叉正在被哲学家使用。
* UsedBy: Philosopher  
  当前使用该餐叉的哲学家对象。

#### ****操作****

* pickUp(philosopher: Philosopher)  
  餐叉被某个哲学家拾起，将状态变为 InUse，并设置 UsedBy 为哲学家。
* putDown()  
  餐叉被放回桌子，将状态变为 OnTable，并清空 UsedBy。

### ****3. DiningTable（餐桌）类****

#### ****属性****

* Philosophers: List<Philosopher>  
  餐桌上的所有哲学家对象的列表。
* Forks: List<Fork>  
  餐桌上的所有餐叉对象的列表。

#### ****操作****

* startDining()  
  开始哲学家就餐活动，哲学家可以尝试拾起餐叉并吃饭。
* stopDining()  
  结束哲学家就餐活动，哲学家停止使用餐叉。

### ****类之间的关系:****

**·DiningTable 和 Philosopher:**

**关系类型**: 聚合关系  
餐桌包含哲学家的列表，哲学家是餐桌的一部分，但可以独立存在。

**·DiningTable 和 Fork：**

**关系类型**: 聚合关系  
餐桌包含餐叉的列表，餐叉是餐桌的一部分，但可以独立存在。

**·Philosopher 和 Fork：**

**关系类型**: 使用关系  
哲学家使用餐叉，但不拥有餐叉。哲学家需要两把餐叉才能进餐。

