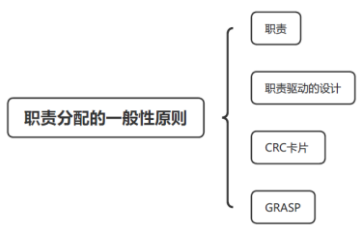
OOA

纲要

- OOA概述
- 标示实体—领域模型
- **职责分配的一般性原则**
- 分析类和用例实现
- 架构分析



职责（1/2）

- UML把职责视为classfiers的一种合约和obligations
- 类或子系统提供的服务或服务集
- 职责体现了元素的目标和权力

- 根据职责、角色和协作进行软件对象和组件的设计是一种广受欢迎的方法。

职责（2/2）

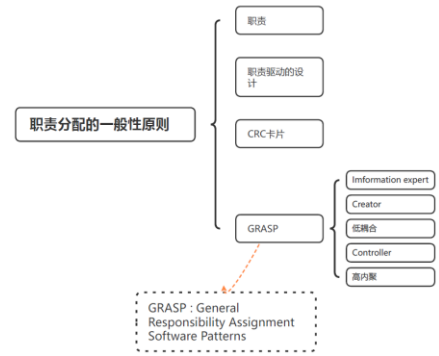
- Do responsibility:
  - ✓ doing something itself, such as creating an object or doing a calculation
  - ✓ initiating action in other objects
  - ✓ controlling and coordinating activities in other objects
- Knowing responsibility:
  - ✓ knowing about private encapsulated data
  - ✓ knowing about related objects
  - ✓ knowing about things it can derive or calculate
- **实例**
  - ✓ "a Sale is responsible for creating SalesLineItems" (a doing),
  - ✓ "a Sale is responsible for knowing its total" (a knowing).

职责驱动的设计

- RDD把面向对象分析和设计视为互相协作的，各司其职的对象的Community
  - ✓ **定义**：职责驱动设计是一种将重点放在系统中的各个对象及其职责上的设计策略。它主张从系统行为的角度出发，而非仅从数据模型的角度来进行设计。
  - ✓ **核心理念**：在职责驱动设计中，对象不仅包含数据，更重要的是它们的行为，即它们需要做什么。通过为对象分配具体的职责，可以促使各个对象之间形成协同的关系来完成任务。
- 过程
  - ✓ **识别对象**
  - ✓ **分配职责**：根据系统需求，为每个对象分配具体的职责。这些职责描述了对象需要执行的任务或操作。分配职责时，需要考虑对象的属性和行为，以及它们与其他对象之间的关系。
  - ✓ **定义接口**：为了对象之间的交互，需要定义清晰的接口。这些接口描述了对象之间可以发送和接收的消息或操作。通过接口，对象可以协同工作以完成任务。
  - ✓ **实现对象**

CRC卡片

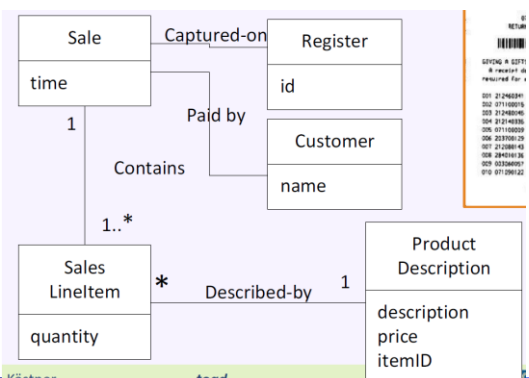
- **CRC** 代表Class-Responsibility-Collaborator.
- CRC方法是一种确定对象职责的有用方法，由Kent Beck和Ward Cunningham提出

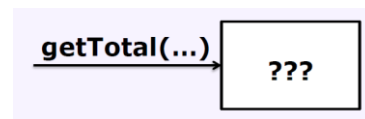


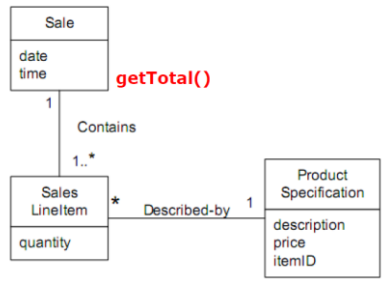
- ***Fred: "Where do you think we should place the responsibility for creating a SalesLineItem? I think a Factory."***
- ***Wilma: "By Creator, I think Sale will be suitable."***

Imformation Experts

- 问题：分配对象职责的基本原则是什么？
- 方案：如果类具有完成某职责所需要的信息，则把该职责分配给此类。
- 分析类和设计类，而不是概念类

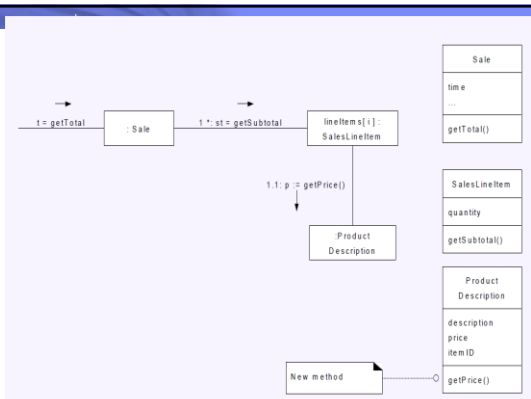






- 为了完成GetTotal职责，还需要进行相关职责的分配：

Design Class	Responsibility
Sale	knows sale total
SalesLineItem	knows line item subtotal
ProductSpecification	knows product price



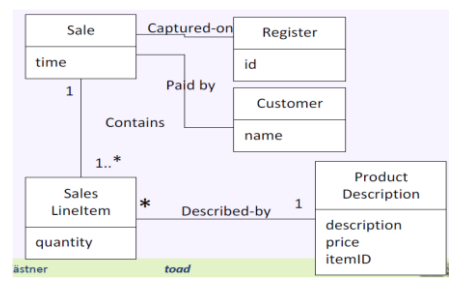
Information expert：设计目标和策略

- 信息隐藏，显式而精致的接口
- 低耦合
- 也许与内聚性冲突
  - 谁负责把sale存进数据库?
  - 把该职责分配给Sale将会导致数据库逻辑分布在各种类中
- 权衡：重用、变更和理解等多种因素

Creator

- 问题：who creates a A?
- 方案：假如以下关系之一成立，则确定B create 的职责
  - B "contains" or compositely aggregates A.
  - B records A.
  - B closely uses A.
  - B has the initializing data for A.

- 谁负责创建SalesLinItem对象?



Creator

- 问题：who creates a A?
- 方案：假如以下关系之一成立，则确定B create 的职责
  - **B "contains" or compositely aggregates A.**
  - **B records A.**
  - **B closely uses A.**
  - **B has the initializing data for A.**
- **关键思想：Creator需要引用和频繁使用被创建的对象。**

- 谁负责创建SalesLinItem对象？

Creator:设计目标和策略

- 提升低耦合性和可重用性
  - ✓ 需要引用的类负责创建对象
  - ✓ 创建对象自身应避免依赖于另外的类
- 信息隐藏
  - ✓ 对象创建细节被隐藏，可以局部替换
- 设计模式中具有更多的模式，处理更复杂的情况。

低耦合

- 问题：如何减少“变化”导致的冲击？

---封装和抽象

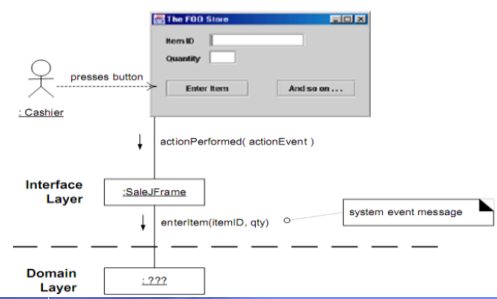
- **实例**

Class Automobile:
  start()
  stop()
  changeTires()
  drive()
  wash()
  checkOil()
  getOil()

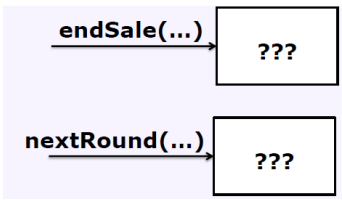
这样分配职责违反了单一职责原理，违反了低耦合，高内聚的原则。

Controller

- 什么对象首先接收信息，协调系统操作？



- 什么对象首先接收信息，协调系统操作?
  - ✓ 点击按钮的用户
  - ✓ 网络回复
  - ✓ 数据库连接断开



- 方案：把“控制”的职责分配给：
  - ✓ 总的系统、装置、子系统或者用例控制器
- Register, POSSystem
- ProcessSaleSession, ProcessSaleHandler

Controller

- 控制器代理和协调别的对象，控制活动
- 当控制事务过多时，可以进行分层（Façade模式）
- 可重用性、适应变更和易理解

Controller的典型职责

- **处理用户输入**：表单数据，HTTP请求和API调用
- **MVC中的控制器类**
  - ✓ 根据用户输入，控制器类会调用相应的模型类（Model）来执行业务逻辑和数据操作。模型类通常负责访问和操作数据库。
  - ✓ 在处理完用户请求并获取必要的数据后，控制器类会选择一个适当的视图（View）来展示数据。视图通常是一个模板，负责生成HTML页面或其他格式的响应。
  - ✓ 控制器类将处理结果和数据传递给视图类，以便视图能够正确地渲染这些内容。
- **处理用户会话和状态管理**：控制器类可能会处理用户会话信息，如登录状态、用户权限等，并根据这些信息来决定如何响应用户请求。
- **路由请求**：在一些框架中，控制器类还负责定义路由规则，将特定的URL请求映射到相应的处理方法上。
- **执行验证和错误处理**：控制器类会执行必要的输入验证，确保用户输入的数据是有效和安全的。

高内聚

- 问题：如何使得对象可理解，可管理，支持低耦合?
- **高内聚**：模块或组件内部的元素彼此紧密相关，共同实现了单一的责任或功能。这样的设计使得模块易于理解和维护，因为每个模块都专注于一个明确的目标。
- **低内聚**：模块或组件内部的元素之间关联程度较低，没有明确的联系或共享的逻辑。这样的设计增加了代码的复杂性和维护难度，修改一个功能可能会影响到其他不相关的功能。