Chapter 7

Object-Oriented Design

Chapter 7 Topics

- **Responsibility Driven Design（责任驱动设计）**
  - Basis for Design
  - Reasons to Use RDD
  - CRC Cards
- **A Case Study in RDD**
- **Software Components**
  - Behavior（行为） and State（状态）
  - Instances（实例） and Classes（类）
  - Coupling（耦合） and Cohesion（内聚）
  - Interface（接口） and Implementation（实现）

Basis for Design

What aspects of a problem are known first?

- Data Structures （数据结构）
- Functions （函数）
- A Formal Specification（形式化说明）
- Behavior（行为）

A design technique based on behavior can be applied from the very beginning of a problem, **whereas techniques based on more structural properties necessarily** require more preliminary analysis.

Responsibility Driven Design

It has the following properties:

- Can deal with ambiguous and incomplete specifications.
- Naturally flows from Analysis to Solution.
- Easily integrates with various aspects of software development.

Characterization by Behavior

The goal in using Responsibility Driven Design will be to first characterize the application by *behavior*.

- **First capture the behavior of the entire application.**
- **Refine this into behavioral descriptions of subsystems.**
- **Refine behavior descriptions into code.**

Software Components

A software ***component***（组件） is simply an abstract design **entity**（实体） with which we can associate responsibilities for different tasks.

- A component must have a small well defined set of responsibilities
- A component should interact with other components to the minimal extent possible

CRC Cards
（组件、责任、协作者卡片）

Components are most easily described using CRC cards. A CRC card records the name, responsibilities, and collaborators（协作者） of a component.

Component Name	**Collaborators**
Description of the responsibilities assigned to this component	List of other components

Working Through Scenarios

We can uncover the desired behavior through **scenarios（场景）**.

- Pretend we had already a working application. Walk through the various uses of the system.
- Establish the ``look and feel'' of the system.
- Make sure we have uncovered all the intended uses.
- Develop descriptive documentation.
- Create the high level software design.

The What/Who Cycle

As we walk through scenarios, we go through cycles of identifying a what, followed by a who.

- What action needs to be performed at this moment,
- Who is the component charged with performing the action

Chapter 7 Topics

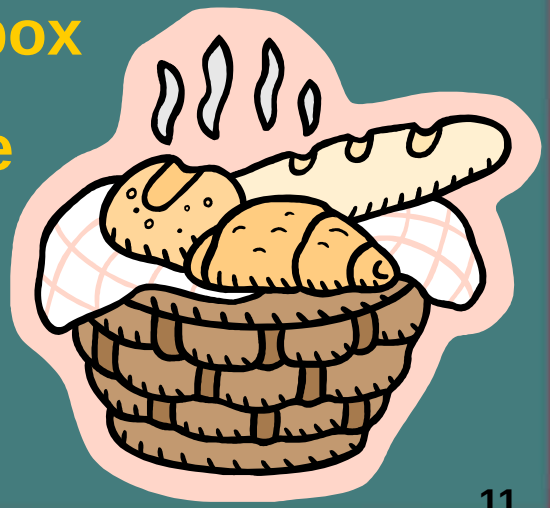
- Responsibility Driven Design
  - Basis for Design
  - Reasons to Use RDD
  - CRC Cards
- A Case Study in RDD
- Software Components
  - Behavior and State
  - Instances and Classes
  - Coupling and Cohesion
  - Interface and Implementation

An Example, the IIKH

Imagine you are the chief software architect（首席软件架构师） in a major computing firm.

Your job is to develop the software that will implement the IIKH.

Briefly, the *Interactive Intelligent Kitchen Helper*（交互式智能厨房助手, IIKH） will replace the box of index cards of recipes（食谱） in the average kitchen.



Abilities of the IIKH

Here are some of the things a user can do with the IIKH:

- Browse（浏览） a database of recipes
- Add a new recipe to the database
- Edit or annotate（注释） an existing recipe
- Plan a meal consisting of several courses
- Scale a recipe for some number of users
- Plan a longer period, say a week
- Generate a grocery（食品，杂货） list that includes all the items in all the menus（菜单） for a period

The first component, The Greeter

The first component your team defines is *the Greeter*, which offers the user the choice of several different actions.

- Casually browse the database of recipes.
- Add a new recipe.
- Edit or annotate a recipe.
- Review a plan for several meals.
- Create a plan of meals.

CRC Card for the Greeter

Greeter	**Collaborators**
Display informative initial message	Recipe Database Manager
Offer user choice of options	Plan Manager
Pass control to either	
Recipe Database Manager	
Plan Manager for processing	

The Recipe Database Manager Component

Ignoring the planning of meals for the moment, your team selects to next explore the recipe database component.

- Must Allow the user to ***browse*** the database.
- Must permit the user to ***edit*** ***or*** ***annotate*** an existing recipe.
- Must permit the user to ***add*** a new recipe.

The Recipe Component

We make the recipe itself into an active data structure. It maintains information, but also performs tasks.

- Maintains the list of ingredients（原料） and transformation algorithm.
- Must know how to edit these data values.
- Must know how to interactively display itself on the output device.
- Must know how to print itself.
- We will add other actions later (ability to scale itself, produce integrate ingredients into a grocery list, and so on).

The Plan Manager Component

Returning to the greeter, we start a different scenario. This leads to the description of the *Plan Manager*.

- Permits the user to select a sequence of dates for planning.
- Permits the user to edit an existing plan.
- Associates with *Date* object.

The Date Component

The *Date* component holds a sequence of meals for an individual date.

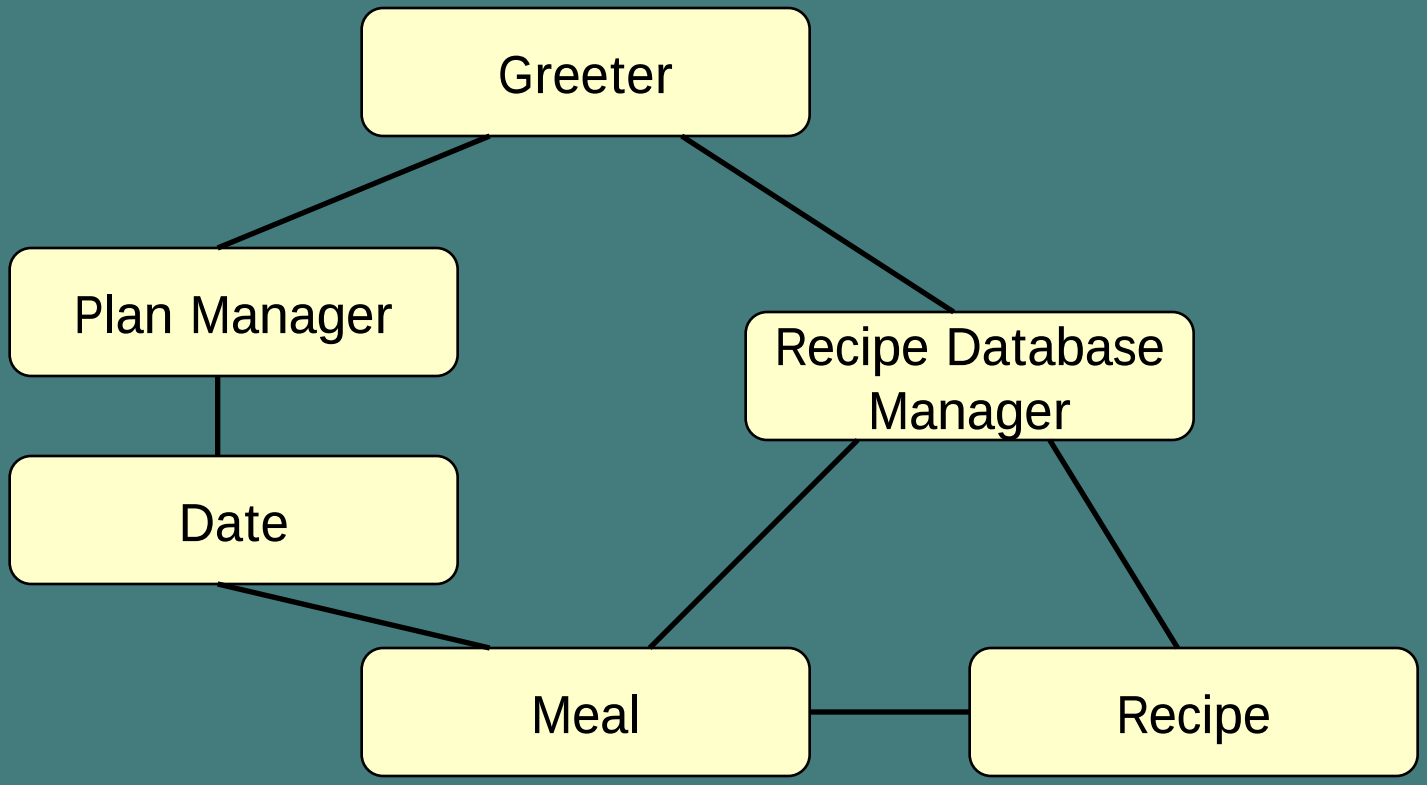
- User can edit specific meals.
- User can annotate information about dates (''Bob's Birthday'', ``Christmas Dinner'', and so on).
- Can print out grocery list for entire set of meals.

The Meal Component

The *Meal* component holds information about a single meal.

- Allows user to interact with the recipe database to select individual recipes for meals.
- User sets number of people to be present at meal, recipes are automatically scaled.
- Can produce grocery list for entire meal, by combining grocery lists from individual scaled recipes.

The Six Components



Chapter 7 Topics

- Responsibility Driven Design
  - Basis for Design
  - Reasons to Use RDD
  - CRC Cards
- A Case Study in RDD
- Software Components
  - Behavior and State
  - Instances and Classes
  - Coupling and Cohesion
  - Interface and Implementation

Characteristics of Components

- Behavior and State
- Instances and Classes
- Coupling and Cohesion
- Interface and Implementation

Behavior and State

- The *behavior* of a component is the set of actions a component can perform. The complete set of behavior for a component is sometimes called the *protocol*.
- The *state* of a component represents all the information (data values) held within a component.

Instances and Classes

There are likely many *instances* of a *class*, but they will all *behave* in the same way.

Example:

recipe class

Recipe Behavior:

Edit

Display

recipe instance

strawberry shortcake

vegetable soup

Coupling and Cohesion

- *Cohesion* is the degree to which the responsibilities of a single component form a meaningful unit.
- *Coupling* is the degree to which the ability to fulfill a certain responsibility depends upon the actions of another component.

Interface and Implementation

- The *interface* describes what a software component can perform.
- The *implementation* describes how a component goes about completing a task.