UML 基础教程

这次大家重点看类图和活动图即可

Trinea

目录

目录

1.1 前言

本资料对UML1.5各种模型图的构成和功能进行说明，通过本资料的学习达到可以读懂UML模型图的目的。本资料不涉及模型图作成的要点等相关知识。

1.2 UML概述

1.2.1 UML简介

UML (Unified Modeling Language)为面向对象软件设计提供统一的、标准的、可视化的建模语言。适用于描述以用例为驱动，以体系结构为中心的软件设计的全过程。

UML的定义包括UML语义和UML表示法两个部分。

(1) UML语义：UML对语义的描述使开发者能在语义上取得一致认识，消除了因人而异的表达方法所造成的影响。

(2) UML表示法：UML表示法定义UML符号的表示法，为开发者或开发工具使用这些图形符号和文本语法为系统建模提供了标准。

1.2.2 UML模型图的构成

事物(Things)：UML模型中最基本的构成元素，是具有代表性的成分的抽象

关系(Relationships)：关系把事物紧密联系在一起

图(Diagrams)：图是事物和关系的可视化表示

1.3 UML事物

UML包含4种事物：构件事物 行为事物 分组事物 注释事物

1.3.1 构件事物：UML模型的静态部分，描述概念或物理元素

它包括以下几种：

类：具有相同属性相同操作 相同关系相同语义的对象的描述

接口：描述元素的外部可见行为，即服务集合的定义说明

协作：描述了一组事物间的相互作用的集合

用例：代表一个系统或系统的一部分行为，是一组动作序列的集合

构件：系统中物理存在，可替换的部件

节点：运行时存在的物理元素

另外，参与者、信号应用、文档库、页表等都是上述基本事物的变体

1.3.2 行为事物：UML模型图的动态部分，描述跨越空间和时间的行为

交互：实现某功能的一组构件事物之间的消息的集合，涉及消息、动作序列、链接

状态机：描述事物或交互在生命周期内响应事件所经历的状态序列

1.3.3 分组事物：UML模型图的组织部分，描述事物的组织结构

包：把元素组织成组的机制

1.3.4 注释事物：UML模型的解释部分，用来对模型中的元素进行说明，解释

注解：对元素进行约束或解释的简单符号

1.4 UML关系

1.4.1依赖

依赖(dependency)是两个事物之间的语义关系，其中一个事物(独立事物)发生变化，会影响到另一个事物(依赖事物)的语义

1.4.2关联

关联(association)是一种结构关系，它指明一个事物的对象与另一个事物的对象间的联系

1.4.3泛化

泛化(generalization)是一种特殊/一般的关系。也可以看作是常说的继承关系

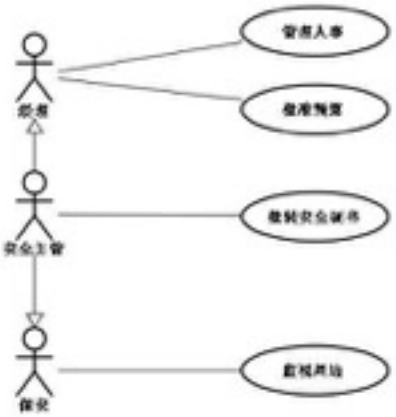
1.4.4实现

实现(realization)是类元之间的语义关系，其中的一个类元指定了由另一个类元保证执行的契约

1.5 各UML图及特征

1.5.1 用例图(Use Case Diagram)

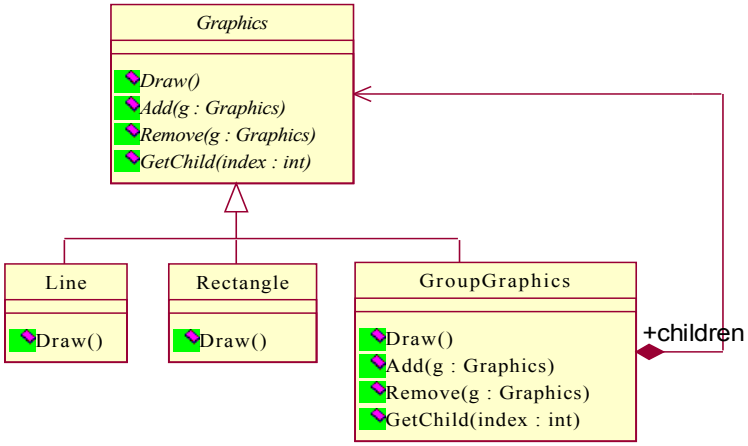
※ 用例图是从用户角度描述系统功能，是用户所能观察到的系统功能的模型图，用例是系统中的一个功能单元



1.5.2 类图(Class Diagram)

※ 类图描述系统中类的静态结构。不仅定义系统中的类，表示类之间的联系如关联、依赖、聚合等，也包括类的内部结构(类的属性和操作)

※ 类图是以类为中心来组织的，类图中的其他元素或属于某个类或与类相关联

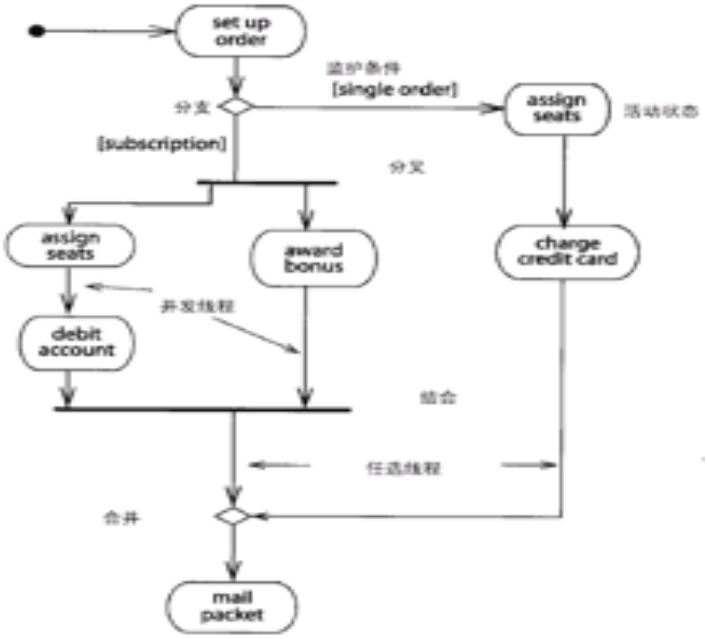


1.5 各UML图及特征

1.5.7 活动图(Activity Diagram)

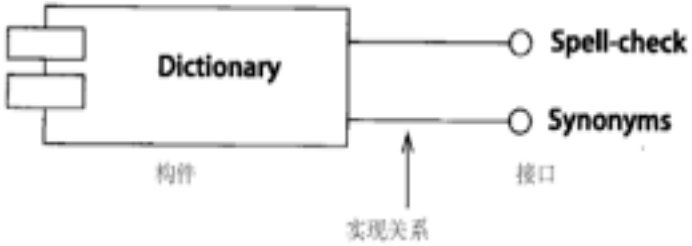
※ 活动图是状态图的一个变体，用来描述执行算法的工作流程中涉及的活动

※ 活动图描述了一组顺序的或并发的活动



1.5.8 构件图(Component Diagram)

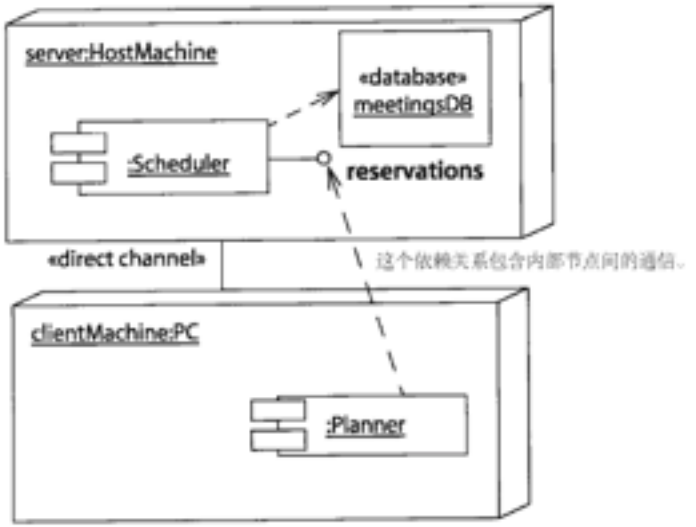
※ 构件图为系统的构件建模型—构件即构造应用的软件单元—还包括各构件之间的依赖关系，以便通过这些依赖关系来估计对系统构件的修改给系统可能带来的影响



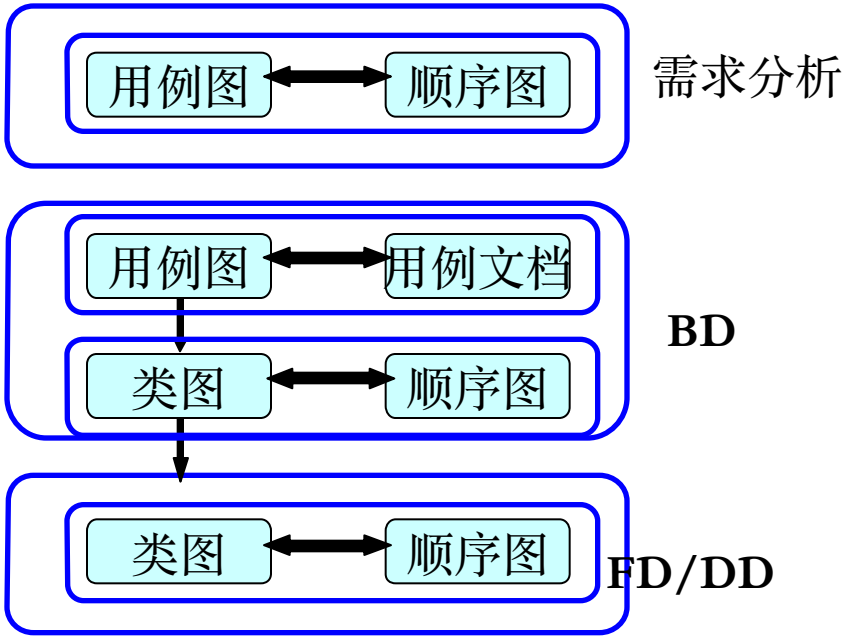
1.5 各UML图及特征

1.5.9 部署图(Deployment Diagram)

部署视图描述位于节点实例上的运行构件实例的安排。节点是一组运行资源，如计算机、设备或存储器。这个视图允许评估分配结果和资源分配



1.6 各UML图的关系



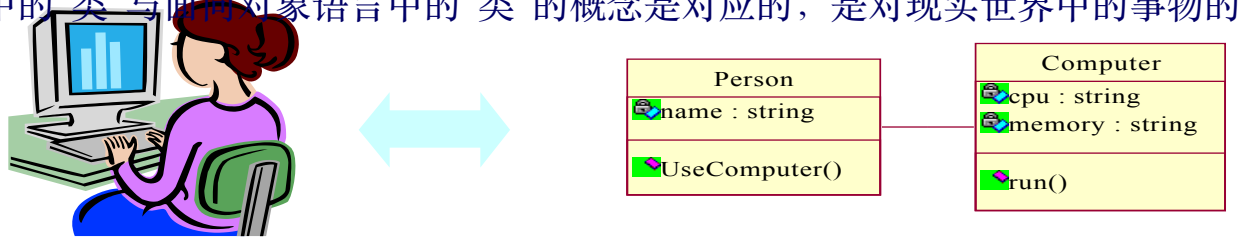
主要图之间的关系

3. 类图

3.1 类图概要

※ 类图以反映类的结构(属性、操作)以及类之间的关系为主要目的，描述了软件系统的结构，是一种静态建模方法

※ 类图中的“类”与面向对象语言中的“类”的概念是对应的，是对现实世界中的事物的抽象



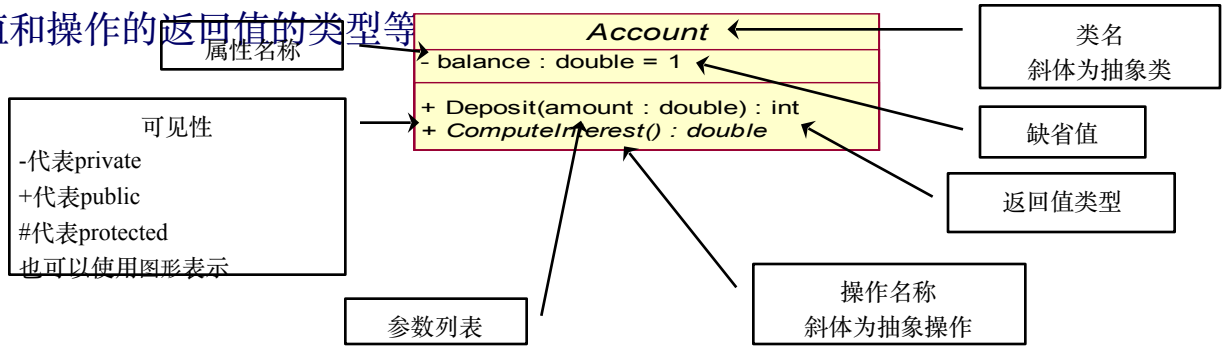
3.2 类图中的事物及解释

3.2.1 类

※ 从上到下分为三部分，分别是类名、属性和操作。类名是必须有的

※ 类如果有属性，则每一个属性都必须有一个名字，另外还可以有其它的描述信息，如可见性、数据类型、缺省值等

※ 类如果有操作，则每一个操作也都有一个名字，其它可选的信息包括可见性、参数的名字、参数类型、参数缺省值和操作的返回值的类型等



3.2 类图中的事物及解释

3.2.2 接口

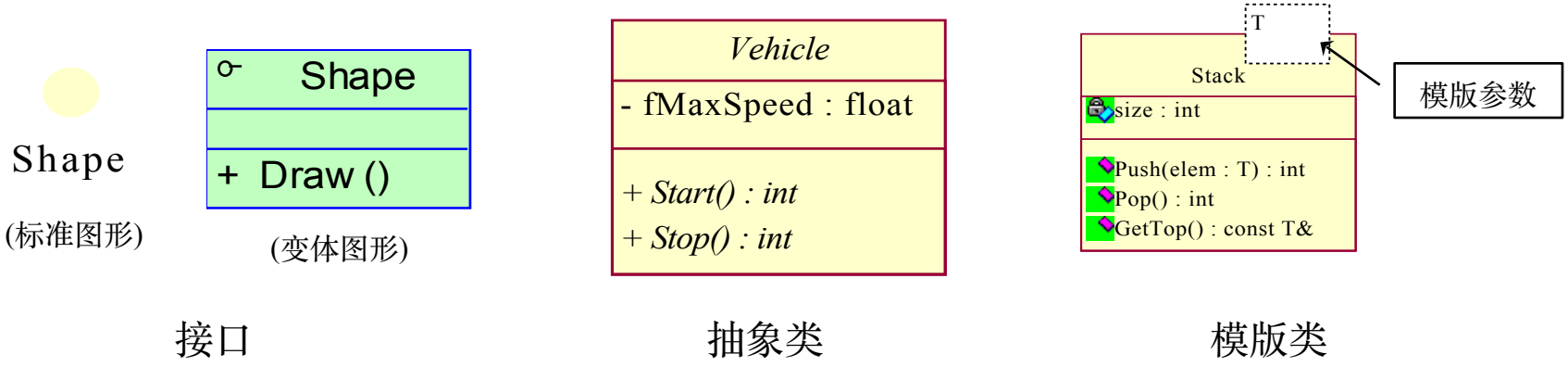
※ 一组操作的集合，只有操作的声明而没有实现

3.2.3 抽象类

※ 不能被实例化的类，一般至少包含一个抽象操作

3.2.4 模版类

※ 一种参数化的类，在编译时把模版参数绑定到不同的数据类型，从而产生不同的类

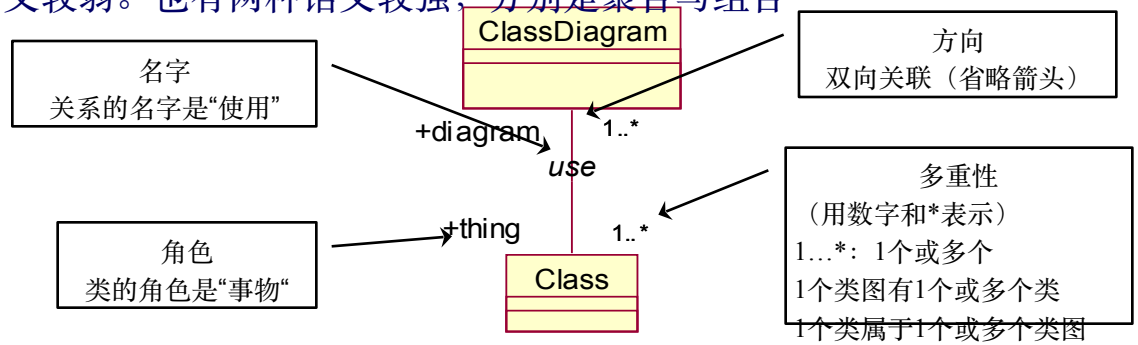


3.3 类图中的关系及解释

3.3.1 关联关系

※ 描述了类的结构之间的关系。具有方向、名字、角色和多重性等信息。一般的关联关系语义较弱。也有两种语义较强，分别是聚合与组合

UML表示法



实例

聚合关系

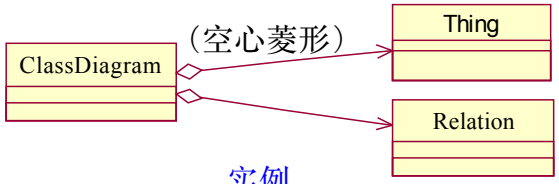
➢ 特殊关联关系，指明一个聚集（整体）和组成部分之间的关系

UML表示法

组合关系

➢ 语义更强的聚合，部分和整体具有相同的生命周期

UML表示法



实例

类图包含有事物和关系，类图不存在了，事物和关系还可用于其它的类图

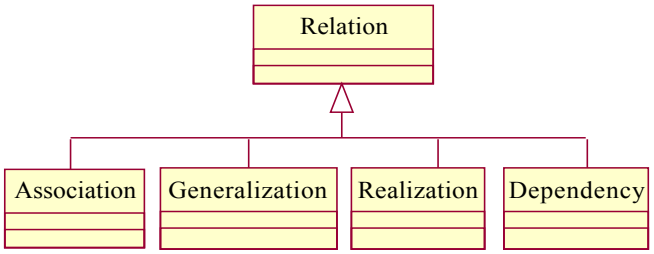
实例

类与关联关系之间有组合关系，类不存在了，则相应的关联关系也不存在

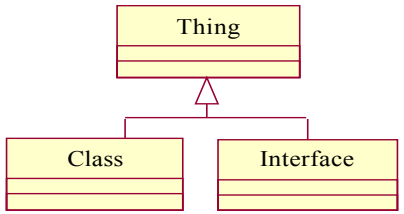
3.3.2 泛化关系

※ 在面向对象中一般称为继承关系，存在于父类与子类、父接口与子接口之间

UML表示法



关联、泛化、实现、依赖都是一种关系

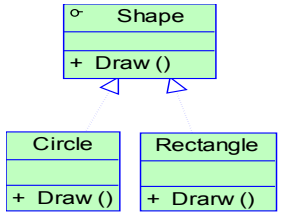


类、接口都是一种事物

3.3.3 实现关系

※ 对应于类和接口之间的关系

UML表示法

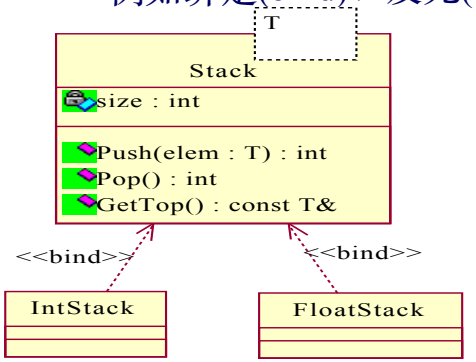


类Circle、Rectangle实现了接口Shape的操作

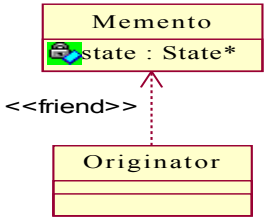
3.3.4 依赖关系

※ 描述了一个类的变化对依赖于它的类产生影响的情况。有多种表现形式，例如绑定(bind)、友元(friend)等

UML表示法



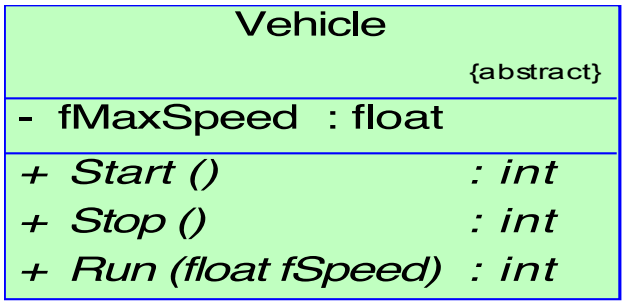
模板类Stack<T>定义了栈相关的操作；IntStack将参数T与实际类型int绑定，使得所有操作都针对int类型的数据



类Memento和类Originator建立了友元依赖关系，以便Originator使用Memento的私有变量state

3.4 类图与代码的映射

3.4.1 类的映射



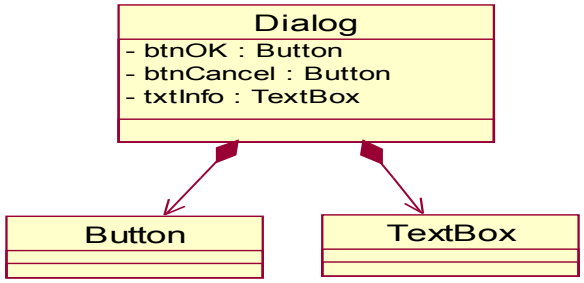
C++代码

```
class Vehicle
{
public:
   virtual int Start() = 0;
   virtual int Stop() = 0;
   virtual int Run(float
fSpeed) = 0;
private:
   float fMaxSpeed;
};
```

Java代码

```
public abstract class Vehicle
{
  public abstract int Start();
  public abstract int Stop();
  public abstract int Run(float
fSpeed);

  private float fMaxSpeed;
}
```

3.4.2 关联关系的映射

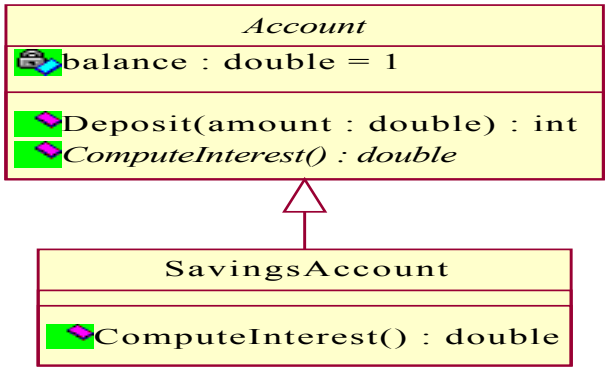


组合关系，代码表现为Dialog的属性有Button和TextBox的对象

C++代码

```
class Dialog
{
private:
  Button btnOK;
  Button btnCancel;
  TextBox txtInfo;
};
class Button
{};
class TextBox
{};
```

3.4.3 泛化关系的映射



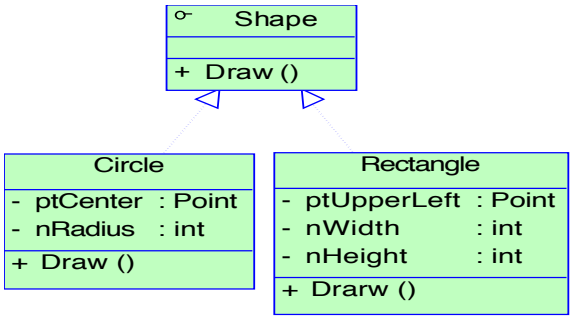
C++代码

```
class SavingsAccount : public Account
{ };
```

Java代码

```
public class SavingsAccount extends Account
{ }
```

3.4.4 实现关系的映射

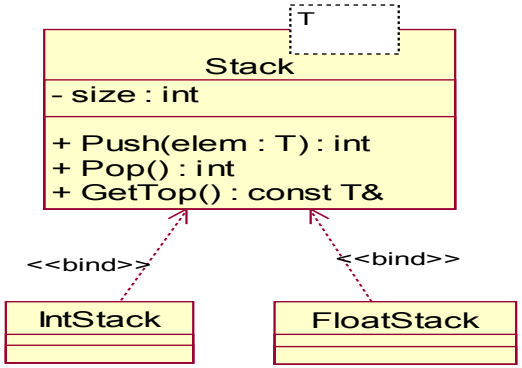


在C++语言里面，使用抽象类代替接口，
使用泛化关系代替实现关系
在Java语言里面，有相应的关键字
interface、implements

C++代码
```
class Shape
{
public:
  virtual void Draw() = 0;
};

class Circle : public Shape
{
public:
 void Draw();
private;
  Point ptCenter;
  int nRadius;
};
```

Java代码
```
public interface Shape
{
public abstract void Draw();
}

public class Circle implements Shape
{
public void Draw();

private Point ptCenter;
private int nRadius;
}
```

3.4.5 依赖关系的映射



绑定依赖

C++代码
```
template<typename T>
class Stack
{
private:
  int size;
public:
  int Push(T elem);
  int Pop();
  const T& GetTop();
};

typedef Stack<float> FloatStack;
```

C++代码(编译器生成)
```
class FloatStack
{
private:
  int size;
public:
  int Push(float elem);
  int Pop();
  const float& GetTop();
};
```

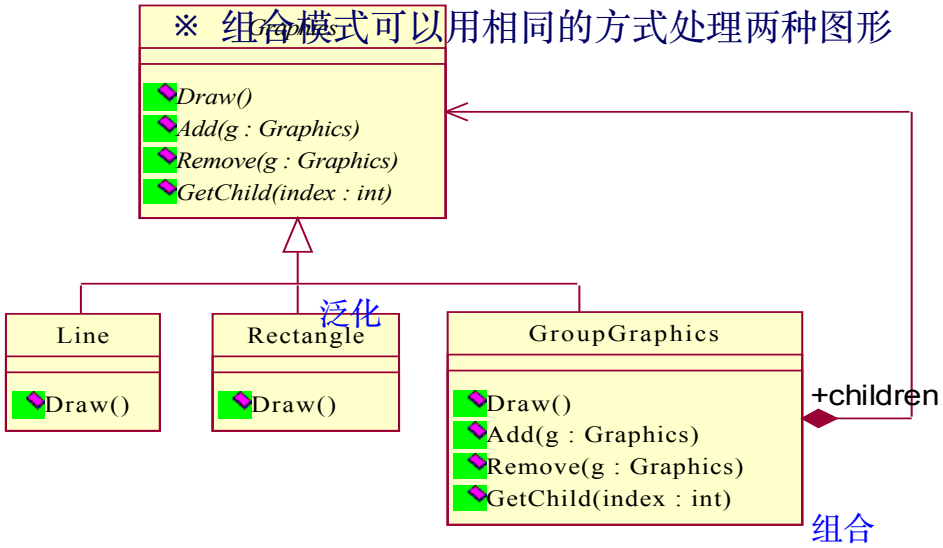
3.5 类图例子

3.5.1 图形编辑器

※ 图形编辑器一般都具有一些基本图形，如直线、矩形等，用户可以直接使用基本图形画图，也可以把基本图形组合在一起创建复杂图形

※ 如果区别对待基本图形和组合图形，会使代码变得复杂，而且多数情况下用户认为二者是一样的

※ 组合模式可以用相同的方式处理两种图形



组合模式

Graphics: 基本图形和组合图形的父类，声明了所有图形共同的操作，如Draw；也声明了专用于组合图形管理子图形的操作，如Add、Remove

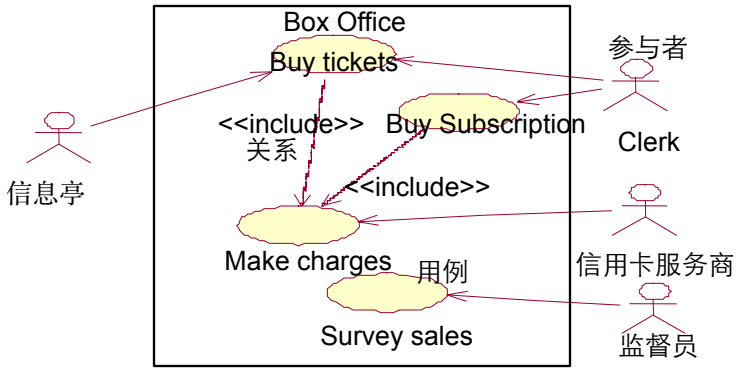
Line、Rectangle: 基本图形类

GroupGraphics: 组合图形类，与父类有组合关系，从而可以组合所有图形对象(基本图形和组合图形)

3.5.2 演出售票系统

在用例驱动的开发过程中，通过分析各个用例及参与者得到类图。分析用例图的过程中需要根据面向对象的原则设计类和关系，根据用例的细节设计类的属性和操作

用例图



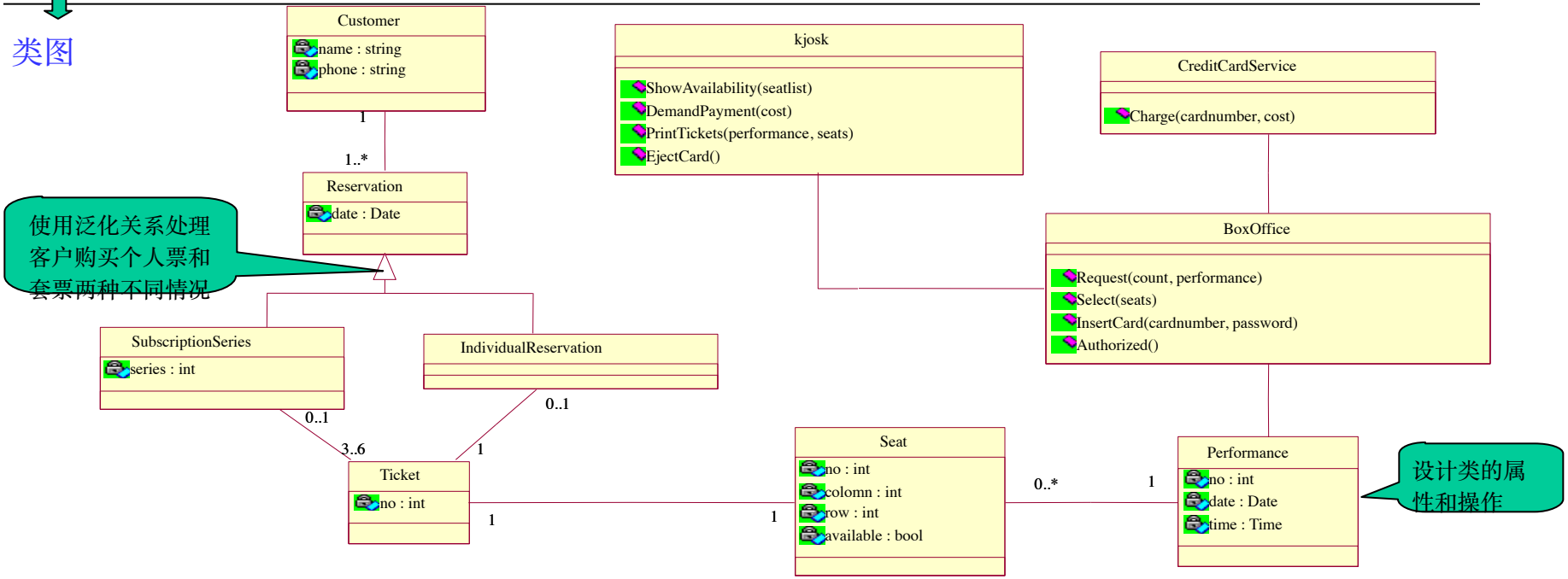
在这里只考虑以下三个用例：

Buy tickets: 买个人票

Buy Subscription: 买套票

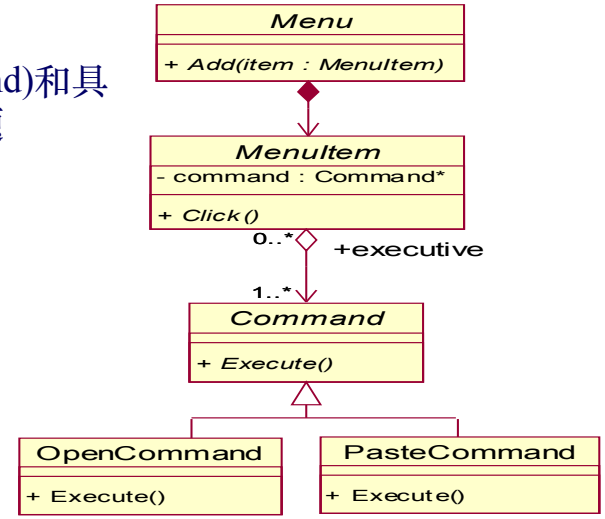
Make charges: 信用卡付款

类图



3.6 习题

※ 右图描述了菜单(Menu)、菜单项(MenuItem)、抽象命令类(Command)和具体命令类(OpenCommand， PasteCommand)之间的关系，完成1-4题



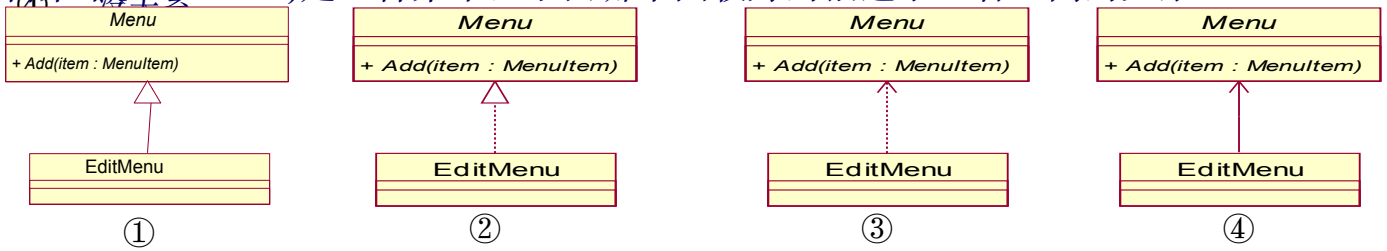
(1)哪两个类之间存在组合关系

① Menu、MenuItem
② MenuItem、Command
③ Command、OpenCommand
④ Command、PasteCommand

(2)OpenCommand和PasteCommand是什么关系

① 组合
② 泛化
③ 聚合
④ 没关系

(3)编辑菜单(EditMenu)是一种菜单，下面哪个图较好的描述了二者之间的关系



(4)下面哪份代码(C++)最接近于图中对MenuItem的描述

①
```
class MenuItem
{
private:
  virtual void Click() =0;
public:
  Command* command;
};
```

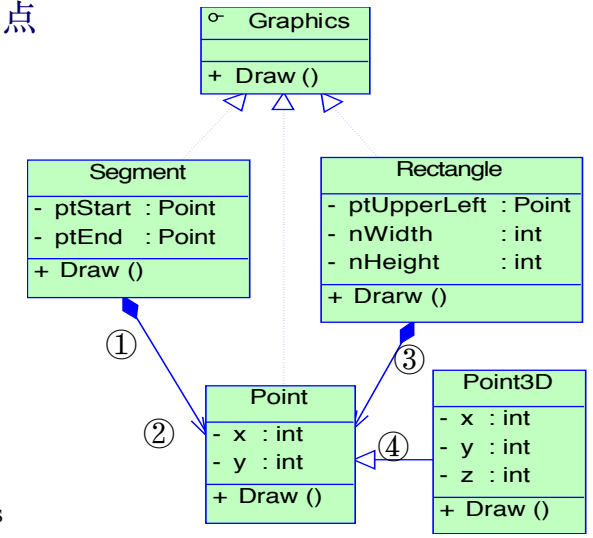
②
```
class MenuItem
{
public:
  virtual void Click() = 0;
private:
  Command* command;
};
```

③
```
class MenuItem
{
private:
  virtual void Click() = 0;
  void undo();
public:
  Command* command;
};
```

④
```
class menuitem
{
public:
  virtual void Click() = 0;
private:
  Command* command;
};
```

3. 类图

※ 右图描述了图形接口(Graphics)、线段(Segment)、矩形(Rectangle)、点(Point)和三维点(Point3D)之间的关系， 完成5-7题



(5)下面哪个关系没有在图中出现

①关联　②泛化　③实现　④依赖

(6)下面对图中①②③④四处的多重性的描述哪个不正确

① 0...*　②1　③0...*　④1

(7)下面哪份代码(Java)最接近于图中对Segment的描述

```
public class Segment implements Graphics
{
privatc void Draw();
public Point ptStart;
public Point ptEnd;
}
```

①

```
public class Segment extends Graphics
{
public void Draw();
private Point ptStart;
private Point ptEnd;
}
```

②

```
public class Segment implements Graphics
{
private Point ptStart;
private Point ptEnd;
public void Draw();
}
```

③

```
public class segment implements graphics
{
public void Draw();
private Point ptStart;
private Point ptEnd;
}
```

④

7. 活动图

7.1 活动图概要

※ 描述系统的动态行为。

※ 包含活动状态(ActionState)，活动状态是指业务用例的一个执行步骤或一个操作，不是普通对象的状态。

※ 活动图适合描述在没有外部事件触发的情况下的系统内部的逻辑执行过程；否则，状态图更容易描述。

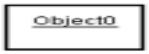
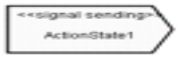
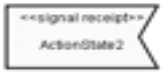
※ 类似于传统意义上的流程图。

※ 活动图主要用于：

业务建模时，用于详述业务用例，描述一项业务的执行过程；

设计时，描述操作的流程。

7.2活动图事物

活动 (ActionState)	动作的执行	
起点 (InitialState)	活动图的开始	
终点(FinalState)	活动图的终点	
对象流(ObjectFlowState)	活动之间的交换的信息	
发送信号(signalSending)	活动过程中发送事件，触发另一活动流程	
接收信号(SignalReceipt)	活动过程中接收事件，接收到信号的活动流程开始执行	
泳道(SwimLane)	活动的负责者	

7.3 活动图关系

迁移(transition)	活动的完成与新活动的开始	
分支(junction point)	根据条件，控制执行方向	
分叉(fork)	以下的活动可并发执行	
结合(join)	以上的并发活动再此结合	

7.4 活动图实例

1. 一般的活动图

本活动图描述一个处理订单的用例执行过

(1)执行setup order

(2)根据order的类型是执行不同的分支：

single order：执行assign seat、charge credit card

subscription：同时执行assignseats、debit account或 **award bonus**

single order与subscription两步可同时进行

(3) 最后mail packet。

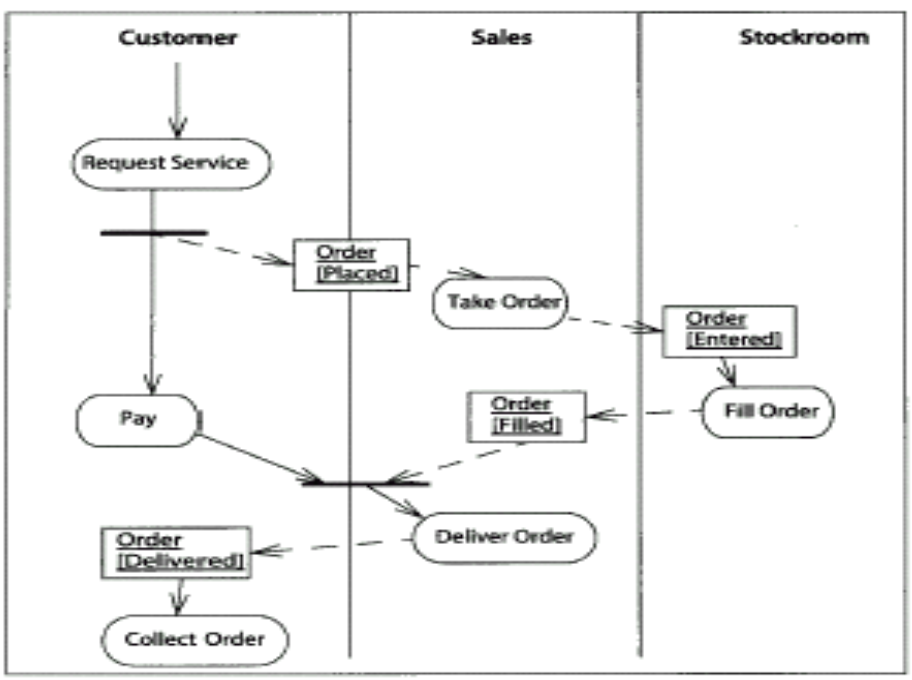
2. 带泳道的活动图

本例为一个按活动职责(带泳道)组织的处理订单用例的活动图(模型中的活动按职责组织)。活动被按职责分配到用线分开的不同区域(泳道):

Customer
Sales
Stockroom

(1)顾客要求服务，Sales负责接收定单，并提交到Stockroom

(2) Stockroom处理定单，与此同时，Customer付款，并由Sales处Deliverorder至Customer。



附录 各个阶段用到*UML*模型图

	需求分析	BD	FD	DD
用例图	◎	◎	-	-
类图	-	○	◎	◎
顺序图	-	○	◎	◎
活动图	○	○	○	○
对象图	-	△	△	△
协作图	-	△	△	△
状态图	-	△	○	○
构件图	-	-	○	○
部署图	○	○	△	△

◎：最适用
○：适用
△：可能适用
- ：不适用

附录 UML全部图的关系