面向对象设计模式　迭代器与组合

曹东刚
caodg@pku.edu.cn

北京大学信息学院研究生课程 - 面向对象的分析与设计
http://sei.pku.edu.cn/~caodg/course/oo

问题的提出

有很多种方法可以把对象堆起来形成一个集合 (collection)，如数组、堆栈、列表或散列表等。

问题：如何遍历集合中的对象，而无需关心集合的具体实现？

答案：迭代器模式 (Iterator Pattern)

例：处理分别由数组和 ArrayList 实现的遗产系统

需求：某遗产系统有两个构件，一个处理早餐订餐业务，一个处理午餐订餐业务，分别用 ArrayList 和数组记录菜单项。现在要实现一个统一的订餐系统，要求能够按照顾客的要求打印出定制的菜单，应如何设计？

难点：程序员不希望改变两个构件的现有存储方式，因为改动的代价太大

现有系统：相同的 MenuItem

早餐和午餐订餐系统的菜单项是相同的

```
1  public class MenuItem {
2    String name;
3    String description;
4    boolean vegetarian;
5    double price;
6    pubic MenuItem(String name, String description,
7      boolean vegetarian, double price){ //init
8    }
9    public String getName() { return name; }
10   public String getDescription() { return description; }
11   public double getPrice() { return price; }
12   public boolean isVegetarian() { return vegetarian; }
13 }
```

两个构件各自不同的遍历方法

早餐项遍历需要使用 ArrayList 的 size 和 get 方法

```
1 ArrayList breakfastItems = breakfMenu.getMenuItems() ;
2 for (int i = 0; i < breakfastItems.size(); i++) {
3   MenuItem menuItem = (MenuItem)breakfastItems.get(i);
4 }
```

午餐项遍历需要使用数组的 length 字段和下标操作

```
1 MenuItem[] lunchItems = dinerMenu.getMenuItems() ;
2 for (int i = 0; i < lunchItems.length; i++) {
3   MenuItem menuItem = lunchItems[i];
4 }
```

两个构件不同的实现方式带来的问题

- 统一订餐系统必须针对两个构件的具体实现编程，而不是针对接口
- 统一订餐系统需要知道每个菜单如何表达内部的菜单项集合细节，违反了封装原则

如何解决：

1. 让各个菜单实现一个相同的遍历内部元素的接口
2. 将遍历（系统中变化的部分）封装

创建迭代器进行遍历

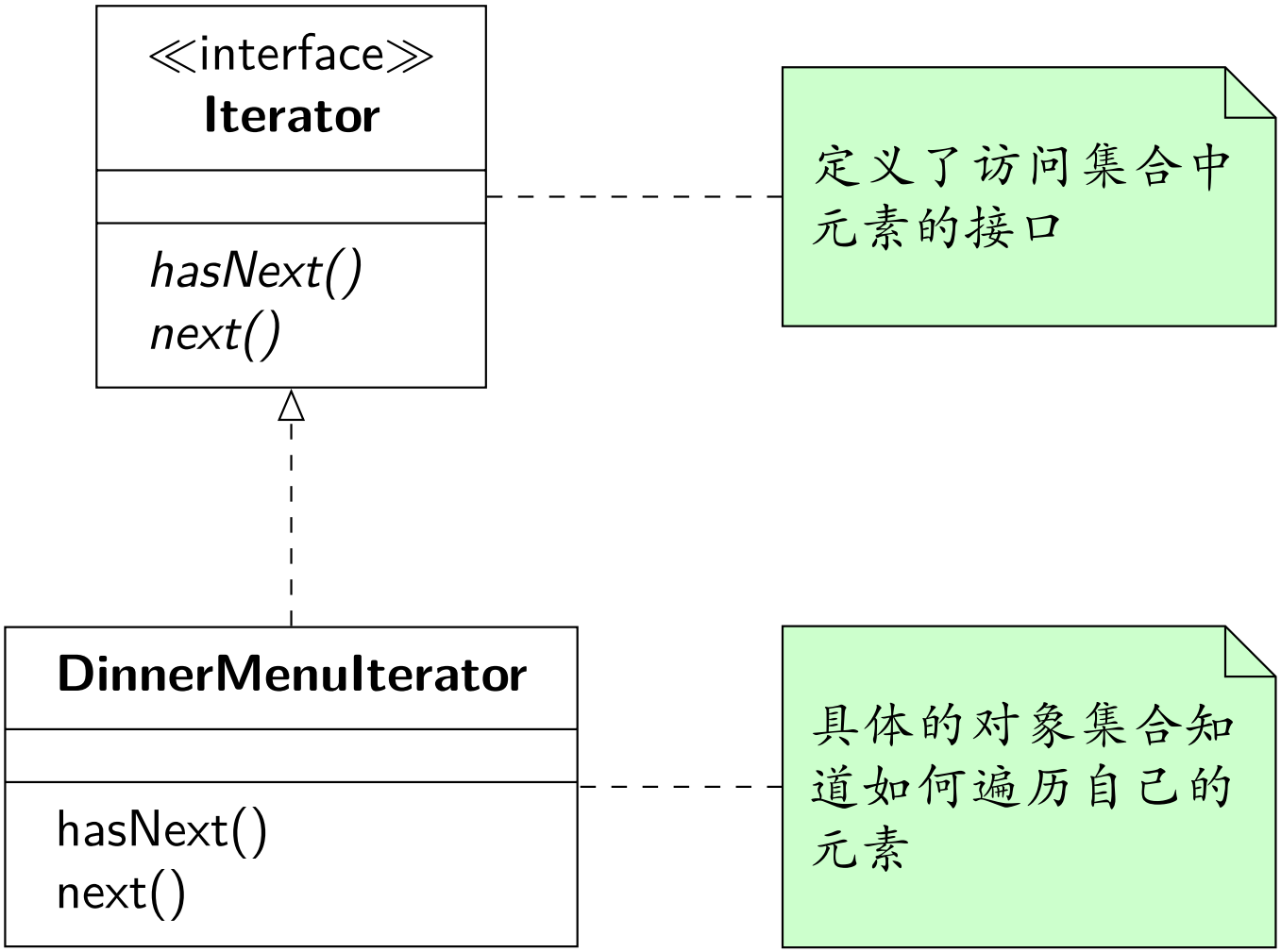
遍历 ArrayList

```
1  Iterator iterator = breakfastMenu.createIterator();
2  while (iterator.hasNext()) {
3    MenuItem menuItem = (MenuItem)iterator.next();
4  }
```

遍历数组

```
1  Iterator iterator = lunchMenu.createIterator();
2  while (iterator.hasNext()) {
3    MenuItem menuItem = (MenuItem)iterator.next();
4  }
```

认识迭代器模式 (Iterator Pattern)



在餐厅菜单中加入迭代器

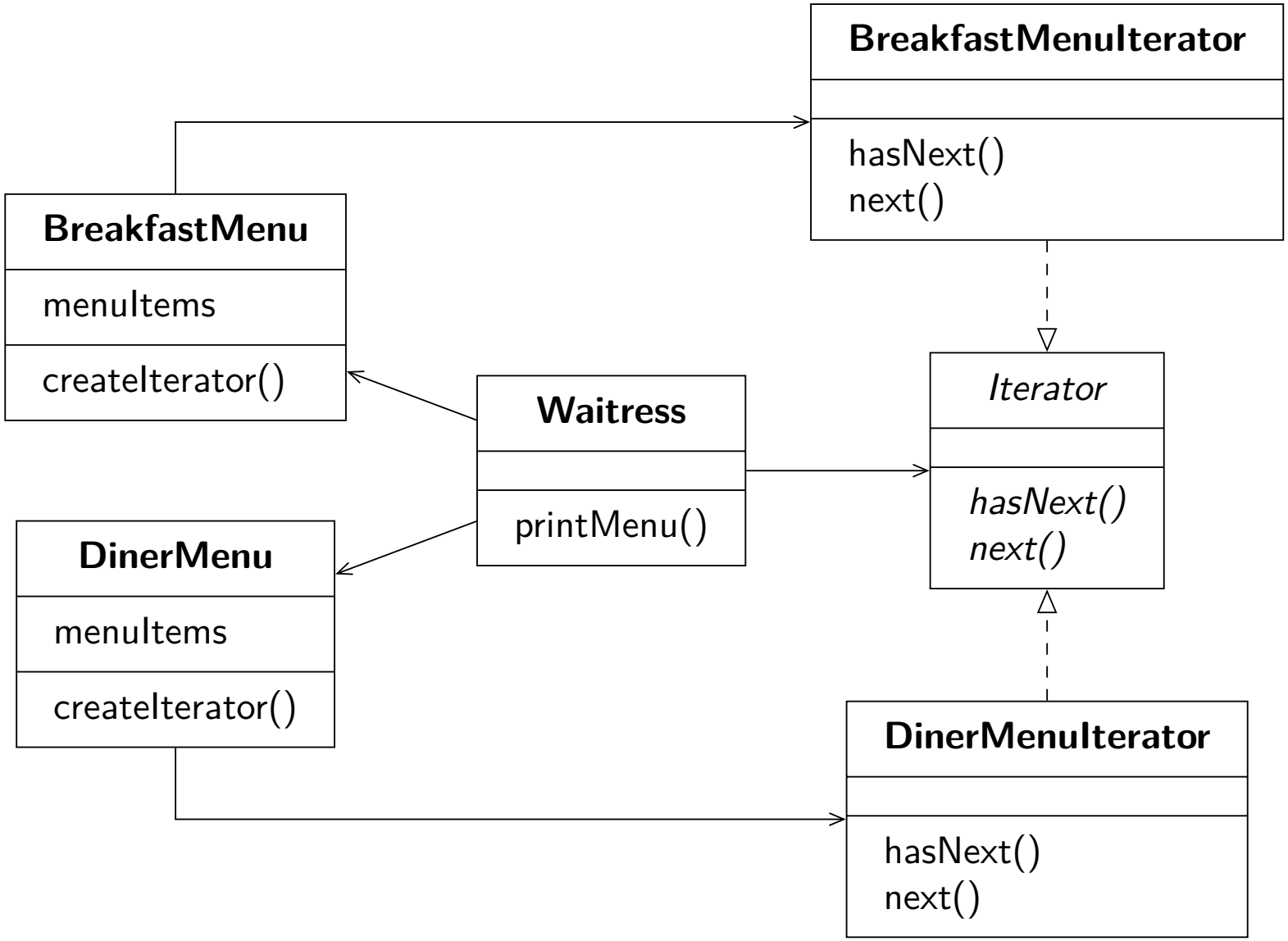
晚餐菜单迭代器：遍历数组

```
1  public class DinerMenuIterator implements Iterator {
2    MenuItem[] items;
3    int position = 0;
4    public Object next() {
5      MenuItem menuItem = items[position];
6      position = position + 1;
7      return menuItem;
8    }
9    public boolean hasNext() {
10     if (position >= items.length || items[position] == null)
11       return false;
12     else
13       return true;
14   }
```

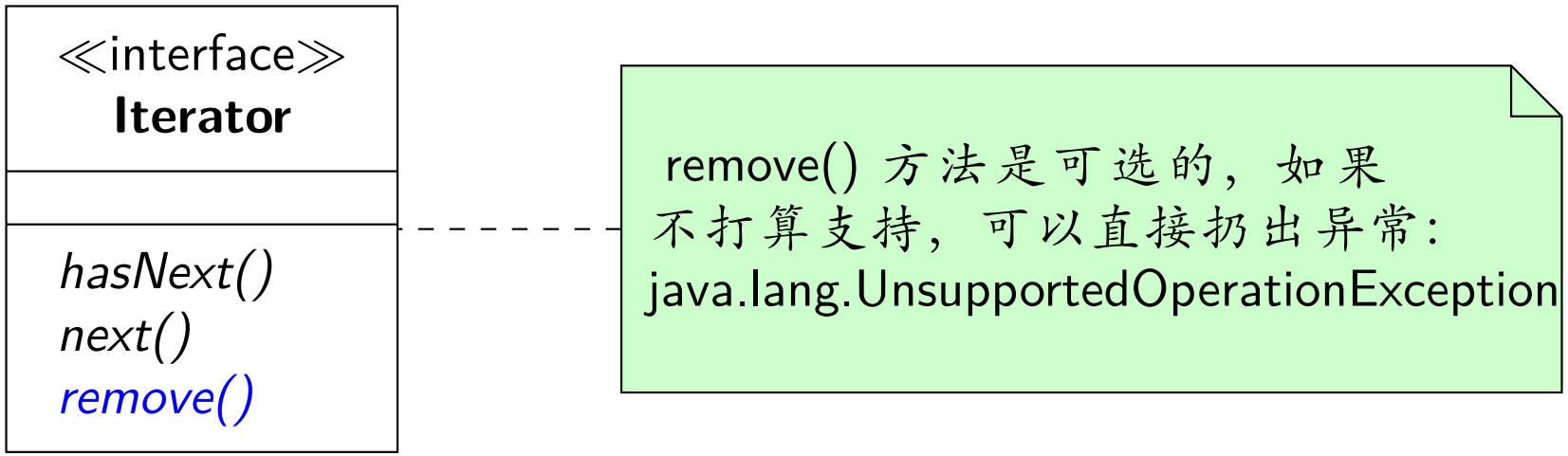
改写晚餐菜单

```
1  public class DinerMenu implements Menu {
2    static final int MAX_ITEMS = 6;
3    int numberOfItems = 0;
4    MenuItem[] menuItems;
5    public MenuItem[] getMenuItems() {
6      return menuItems;
7    }
8    public Iterator createIterator() {
9      return new DinerMenuIterator(menuItems);
10   }
11   // other methods here
12 }
```

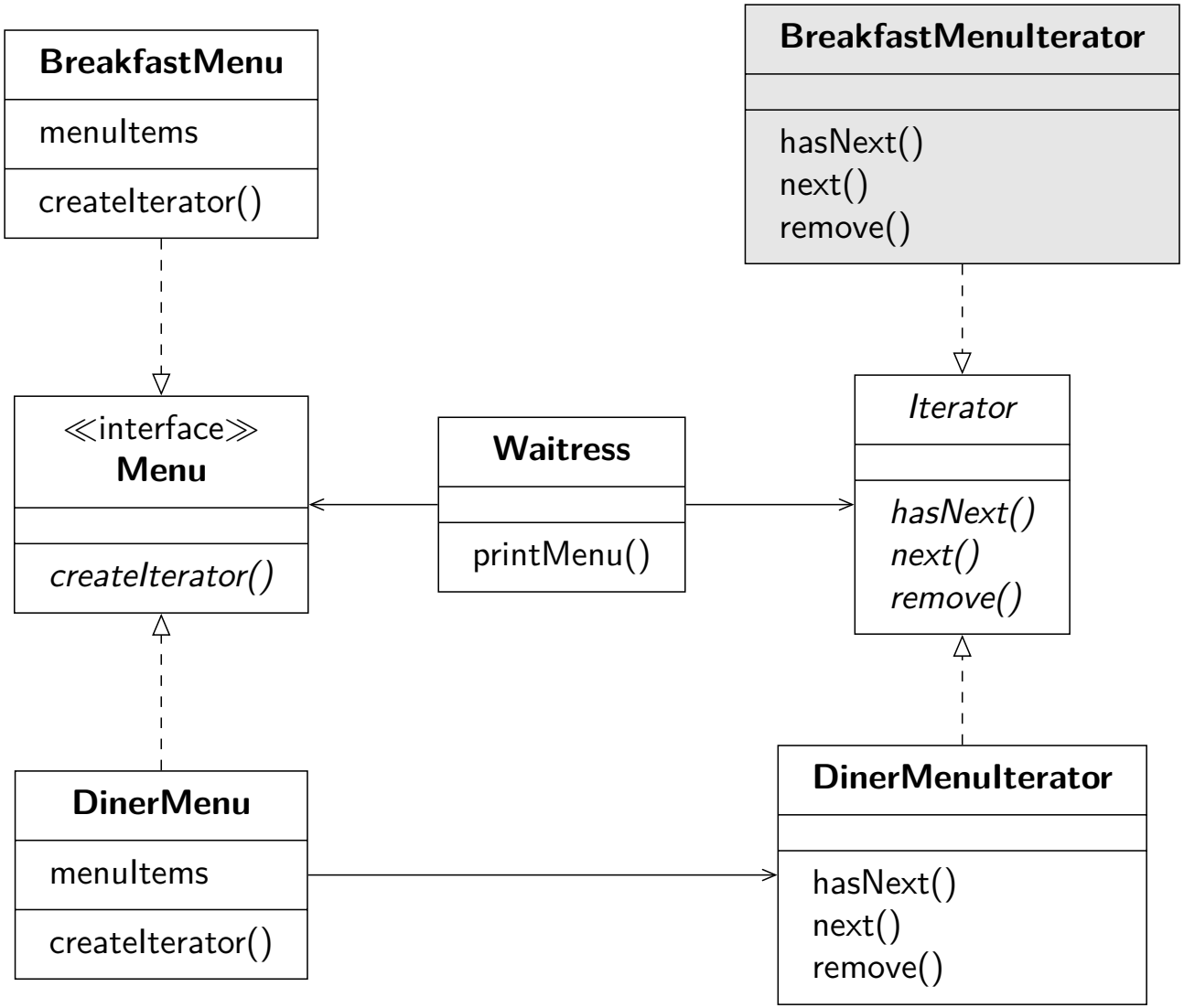
设计的类图


BreakfastMenuIterator
hasNext()
next()
BreakfastMenu
menuItems
createIterator()
Waitress
printMenu()
Iterator
hasNext()
next()
DinerMenu
menuItems
createIterator()
DinerMenuIterator
hasNext()
next()

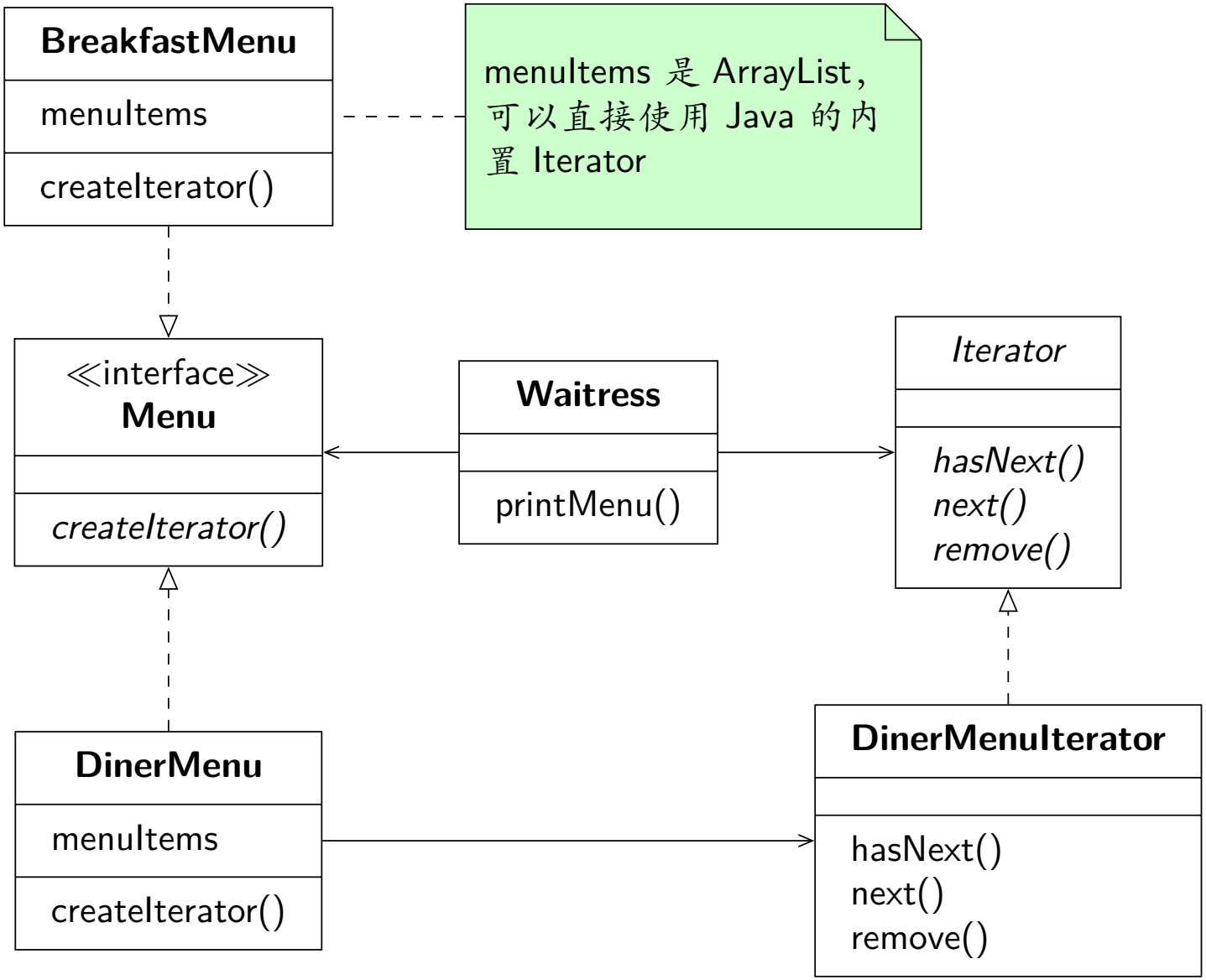
Java 中的迭代器 java.util.Iterator



改进的设计：使用 JDK 的 Iterator



改进的设计：使用 JDK 的 Iterator



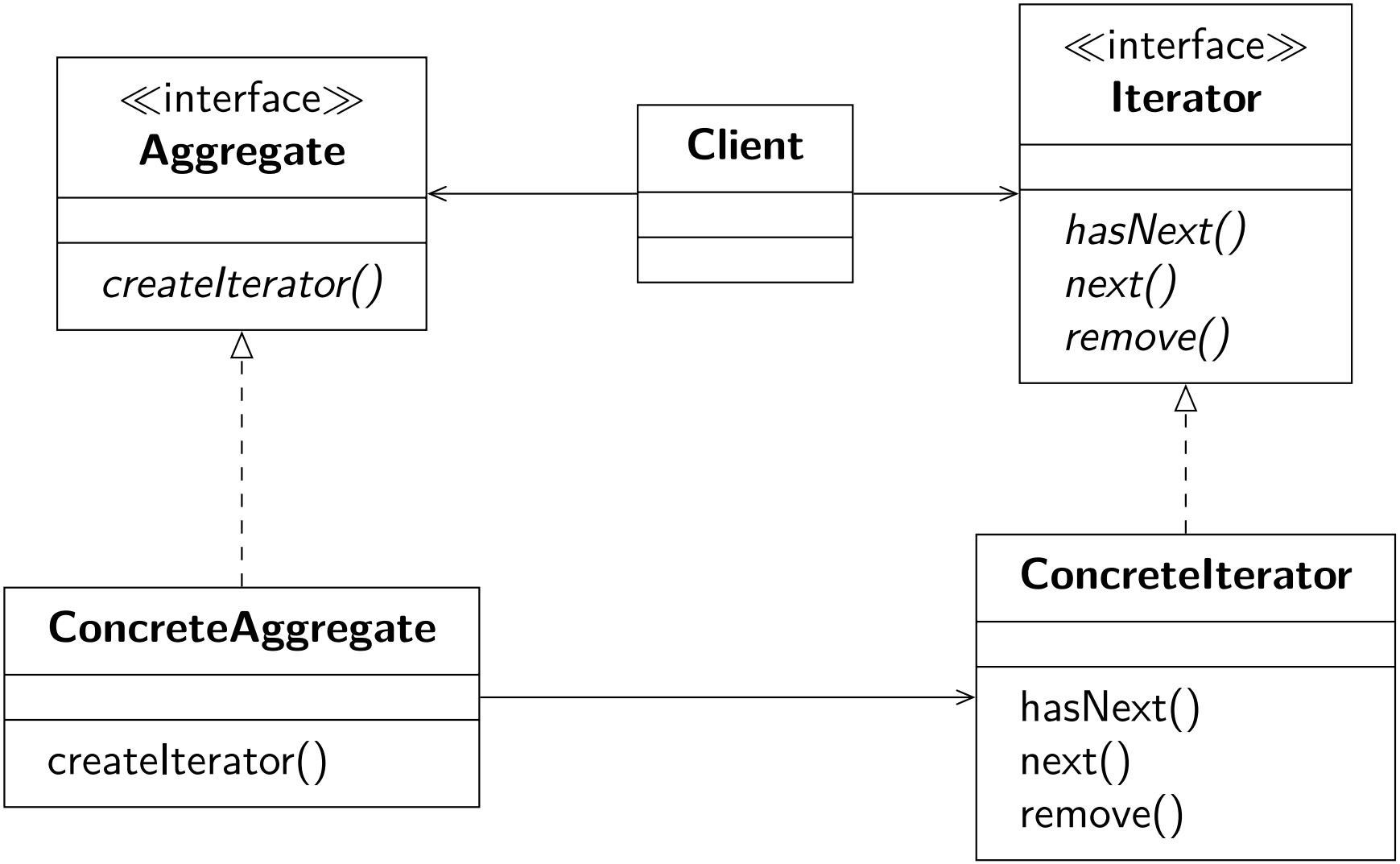
定义迭代器模式

迭代器模式 (Iterator Pattern)

迭代器模式提供了一种方法，能够顺序访问一个集合对象中的各个元素，而又不暴露其内部实现细节

将遍历集合中的元素这件任务交给 Iterator，而不是交给集合，简化了集合类的设计，并使得各自的责任明确

定义迭代器模式


≪interface≫
Aggregate
createIterator()
Client
≪interface≫
Iterator
hasNext()
next()
remove()
ConcreteAggregate
createIterator()
ConcreteIterator
hasNext()
next()
remove()

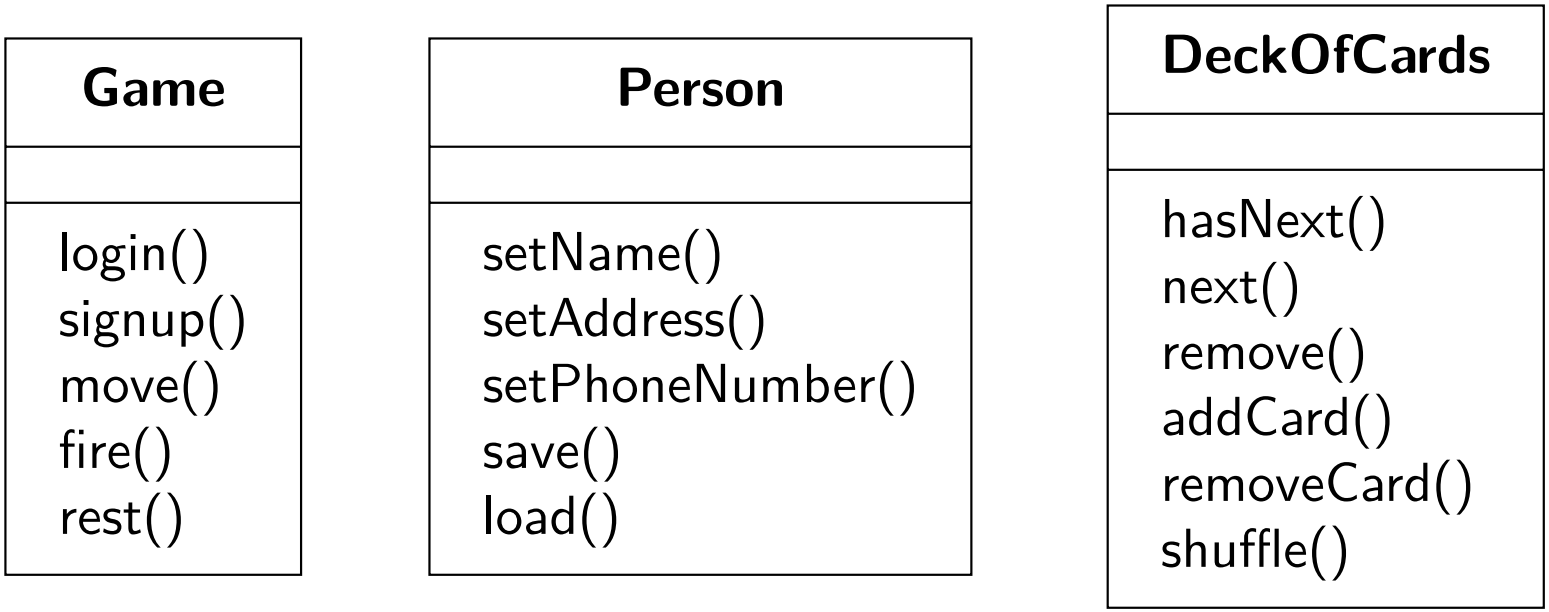
设计原则：单一责任 (Single Responsibility)

单一责任原则

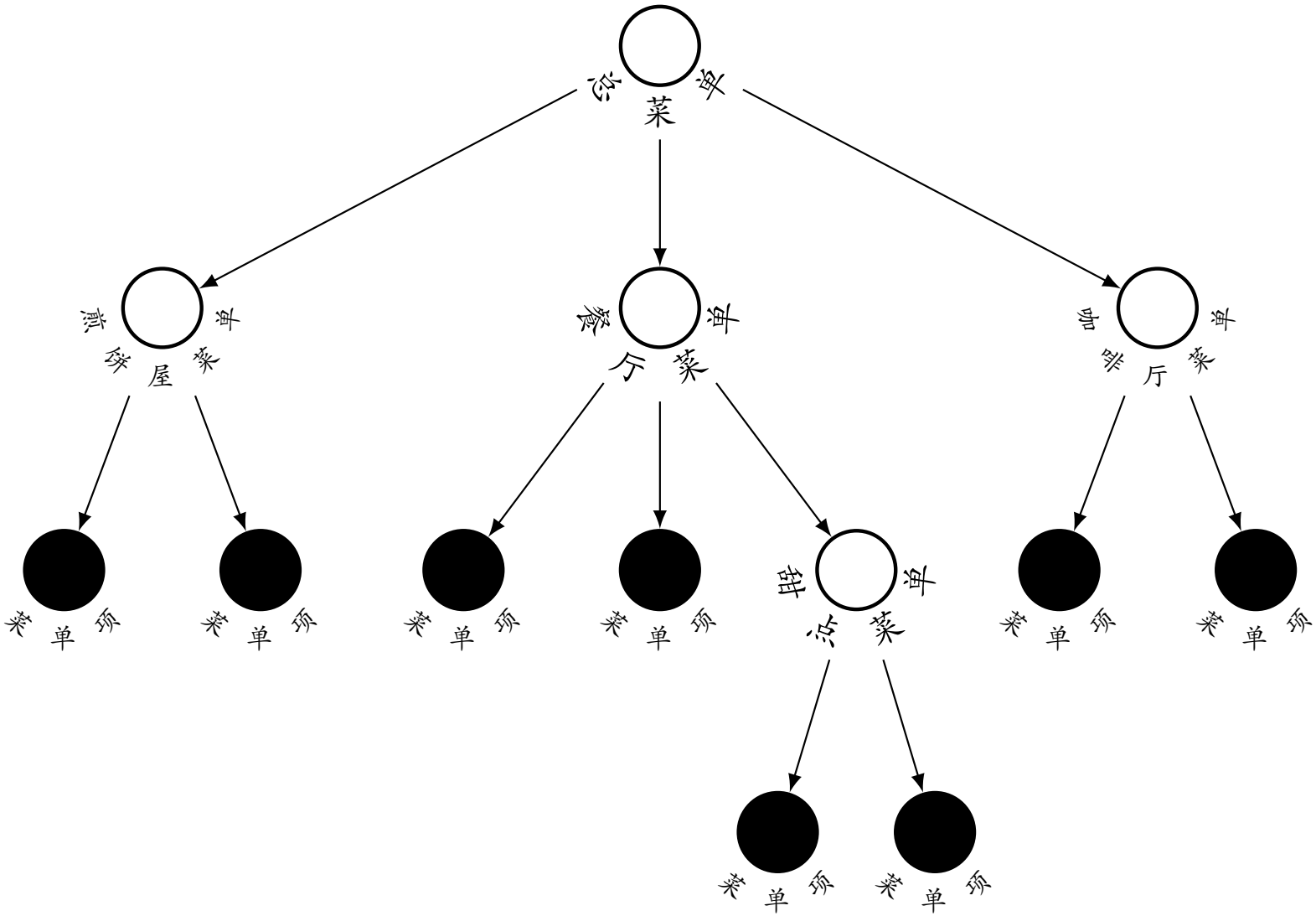
一个类应该只有一个引起变化的原因

- 管理集合里的元素和遍历所有元素是两件事
- 类所承担的每个责任都是潜在的代码变动源头。多个责任意味着多个变动之处
- 每个类只应当承担单一责任

分析类的责任


Game
login()
signup()
move()
fire()
rest()
Person
setName()
setAddress()
setPhoneNumber()
save()
load()
DeckOfCards
hasNext()
next()
remove()
addCard()
removeCard()
shuffle()

新需求: 支持菜单中的菜单



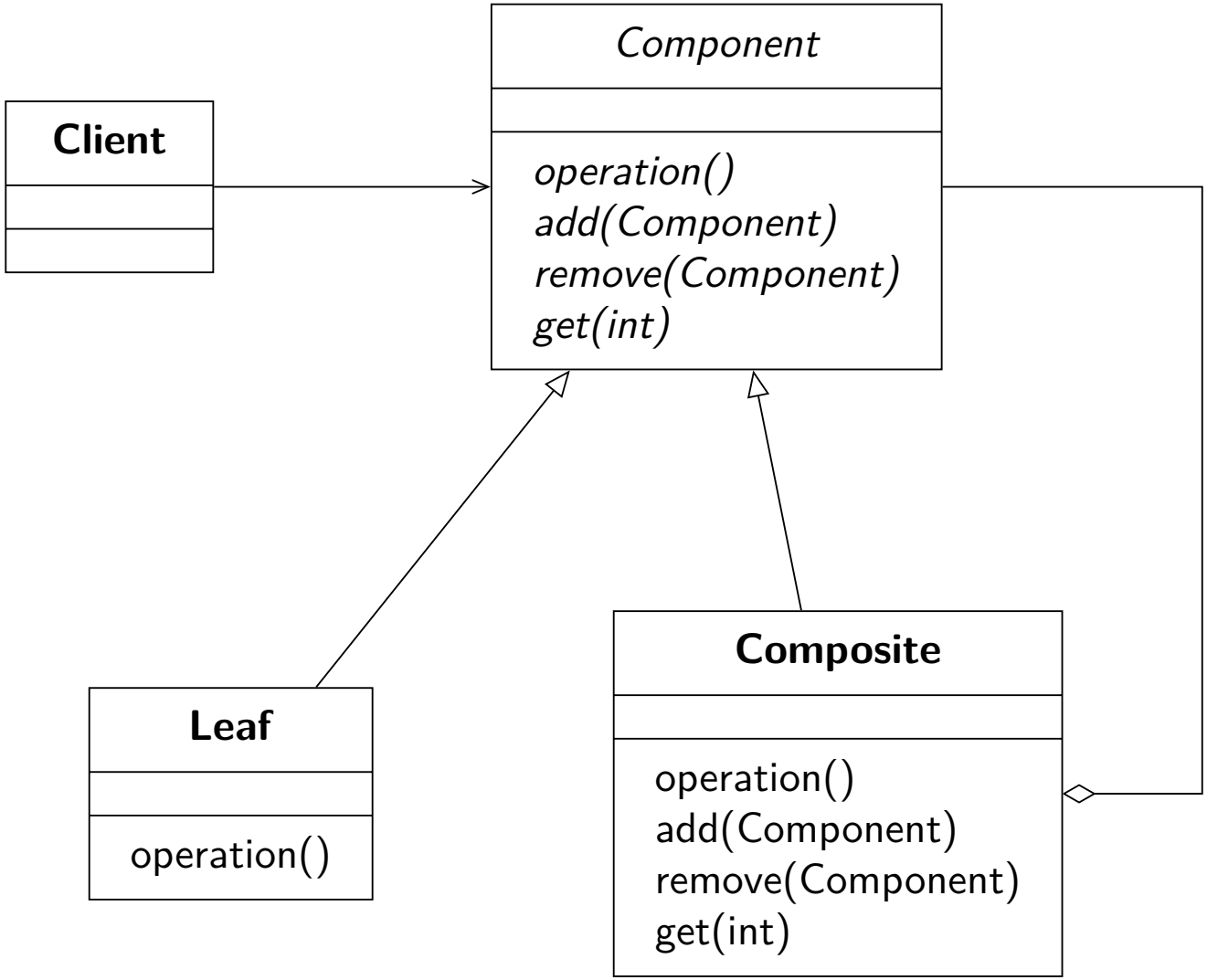
定义组合模式

组合模式 (Composite Pattern)

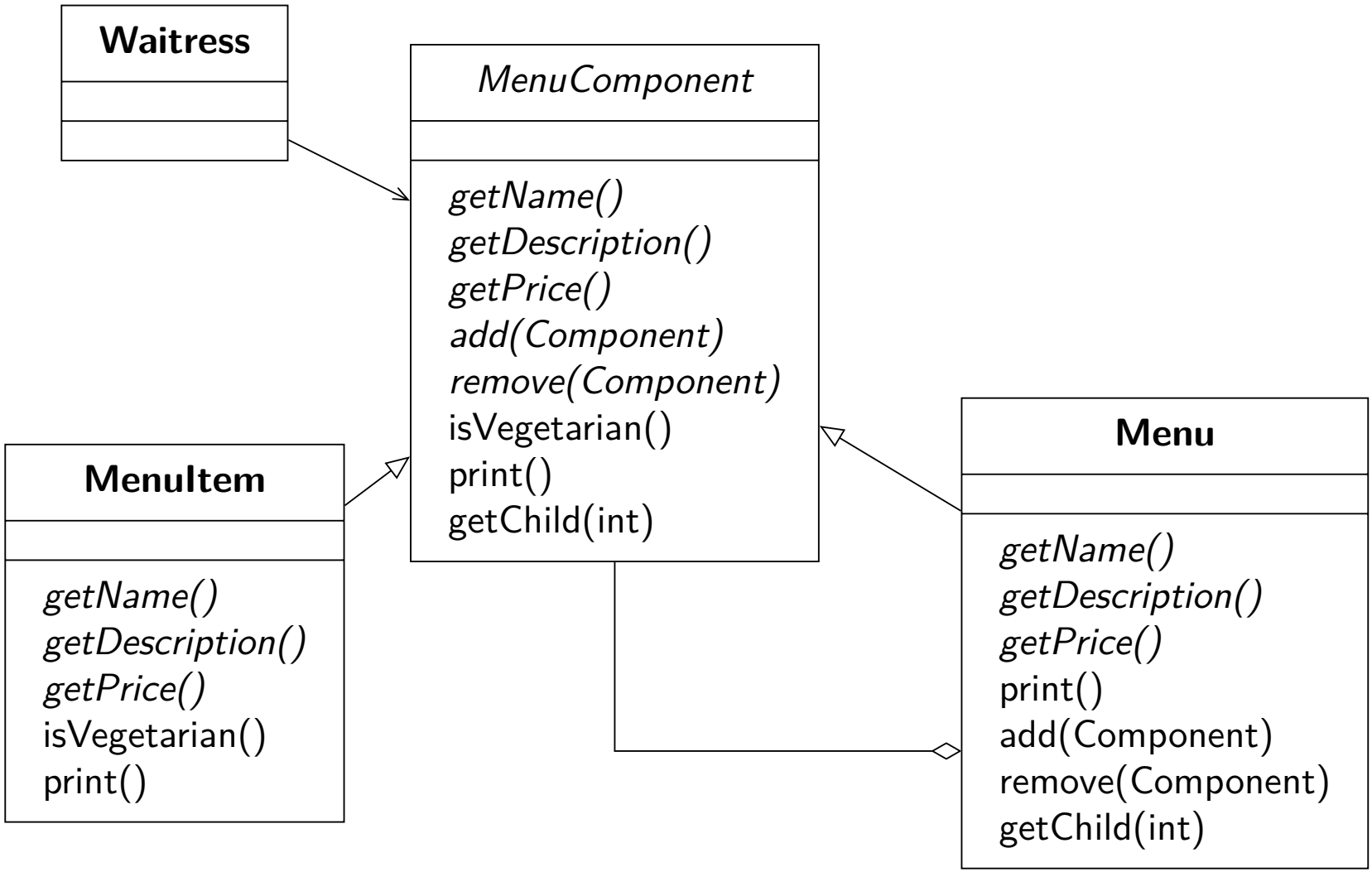
组合模式允许讲对象组合成树型结构来表现“整体—部分”层次结构，支持客户以一致的方式处理个别对象与组合对象

使用组合结构，可以把相同的操作应用在组合对象和个别对象上，即在大多数情况下，可以忽略组合对象和个别对象之间的差别

定义组合模式


Component
Client
operation()
add(Component)
remove(Component)
get(int)
Leaf
operation()
Composite
operation()
add(Component)
remove(Component)
get(int)

利用组合模式设计菜单


Waitress
MenuComponent
getName()
getDescription()
getPrice()
add(Component)
remove(Component)
isVegetarian()
print()
getChild(int)
MenuItem
getName()
getDescription()
getPrice()
isVegetarian()
print()
Menu
getName()
getDescription()
getPrice()
print()
add(Component)
remove(Component)
getChild(int)

实现组合菜单

留作一个练习

将组合模式和迭代器结合

留作一个练习

关于迭代器和组合模式的小结

- 迭代器使得无需了解集合的内部结构即可遍历集合的元素
- 迭代器将对集合的遍历进行了封装
- 迭代器将集合遍历的任务从集合分离
- 迭代器提供了一个遍历集合的公共接口，允许客户利用多态机制编写代码使用集合的元素
- 程序员应当尽量让类只承担唯一的责任
- 组合模式提供一种结构，可同时包容个别对象和组合对象
- 组合模式允许以一致的方式处理个别对象和组合对象

关于设计原则的小结

1. 封装变化
2. 多用聚合、少用继承
3. 针对接口编程，不针对实现编程
4. 尽最大可能将要交互的对象设计为松耦合的
5. 对扩展开放，对修改封闭
6. 依赖抽象，不要依赖具体类
7. 只和朋友交谈
8. 别找我，我会找你
9. 类应该只有一个引起改变的理由