1. 위 예제 프로젝의DinerMenuIterator를 아래와 같이 DinerMenu의 내부 클래스로 정의해 보세요.

**public** **class** DinerMenu **implements** Menu {

**static** **final** **int** ***MAX\_ITEMS*** = 6;

**int** numberOfItems = 0;

MenuItem[] menuItems;

**class** DinerMenuIterator **implements** Iterator<MenuItem> {

**int** position;

**public** MenuItem next() {

MenuItem menuItem = menuItems[position];

position = position + 1;

**return** menuItem;

}

**public** **boolean** hasNext() {

**if** (position >= menuItems.length || menuItems[position] == **null**) {

**return** **false**;

} **else** {

**return** **true**;

}

}

}

**public** DinerMenu() {

menuItems = **new** MenuItem[***MAX\_ITEMS***];

addItem("Vegetarian BLT",

"(Fakin') Bacon with lettuce & tomato on whole wheat", **true**, 2.99);

addItem("BLT",

"Bacon with lettuce & tomato on whole wheat", **false**, 2.99);

addItem("Soup of the day",

"Soup of the day, with a side of potato salad", **false**, 3.29);

addItem("Hotdog",

"A hot dog, with saurkraut, relish, onions, topped with cheese",

**false**, 3.05);

addItem("Steamed Veggies and Brown Rice",

"Steamed vegetables over brown rice", **true**, 3.99);

addItem("Pasta",

"Spaghetti with Marinara Sauce, and a slice of sourdough bread",

**true**, 3.89);

}

**public** **void** addItem(String name, String description,

**boolean** vegetarian, **double** price)

{

MenuItem menuItem = **new** MenuItem(name, description, vegetarian, price);

**if** (numberOfItems >= ***MAX\_ITEMS***) {

System.***err***.println("Sorry, menu is full! Can't add item to menu");

} **else** {

menuItems[numberOfItems] = menuItem;

numberOfItems = numberOfItems + 1;

}

}

**public** MenuItem[] getMenuItems() {

**return** menuItems;

}

**public** Iterator<MenuItem> createIterator() {

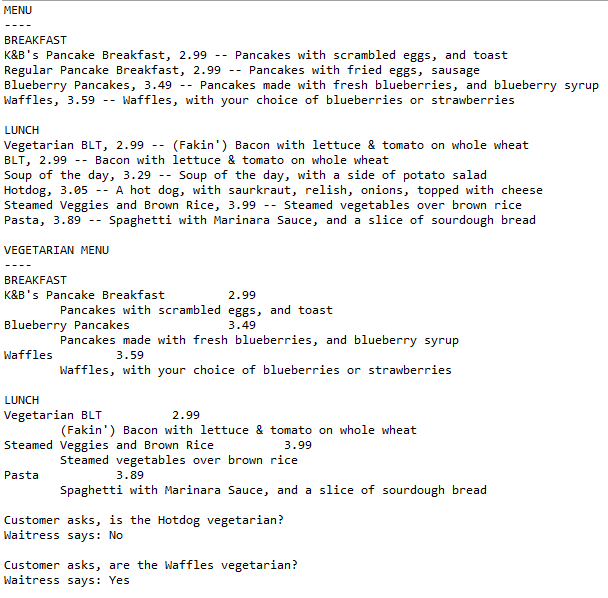
**return** **new** DinerMenuIterator();

//return new AlternatingDinerMenuIterator(menuItems);

}

// other menu methods here

}



1. 1번에서 수정한 설계를 수정 이전의 설계와 비교 분석해 보세요. 즉, 메뉴항목 컬렉션에 대한 반복작업을 별도의 클래스로 분리시킨 설계 (수정전 설계)와 메뉴항목 컬렉션에 대한 반복작업을 메뉴항목 컬렉션을 정의한 클래스에 포함시키는 설계 (수정 후 설계)와의 차이점에 대한 분석입니다.

데이터를 접근하는 방식에 차이가 있다. 수정 전 설계는 Iterator 클래스가 DinerMenu객체에 접근하기 위해 존재하고, DinerMenu의 인스턴스를 가지고 있으며 해당 인스턴스를 통해 DinerMenu의 데이터에 접근한다.

수정 후 설계는 Iterator가 직접 데이터에 접근할 수 있도록 하였다.

분리된 설계보다 비교적 간단하다는 장점이 있다.

1. 컴포지트 패턴은 단일 책임 원칙을 위반하고 있습니다. 어떠한 점이 단일책임 원칙을 위반하고 있다고 판단되나요. 그리고 어떠한 문제가 발생할 수 있는지 구체적인 예를 들어서 설명하세요.

하나의 상위클래스가 하나 이상의 서브클래스(MenuItem, Menu)를 책임지고 있는 구조로 하나의 서브클래스를 책임지는 원칙을 위반하고 있다. 복합객체와 개별객체를 동시에 다룰 수 있다는 장점이 있지만 설계과정에서 안정성이 떨어진다. 컴포지트 패턴에 정의된 메소드 전부를 서브 클래스에서 사용하지 않기 때문에 사용하지 않는 메소드(ex MenuItem의 add, remove, getChlid method)에 대해 예외처리를 하게 된다. 하지만 사용하지 않아야 하는 메소드를 사용하게 되면 프로그램이 비정상 종료되는 문제가 발생한다.