REPORT

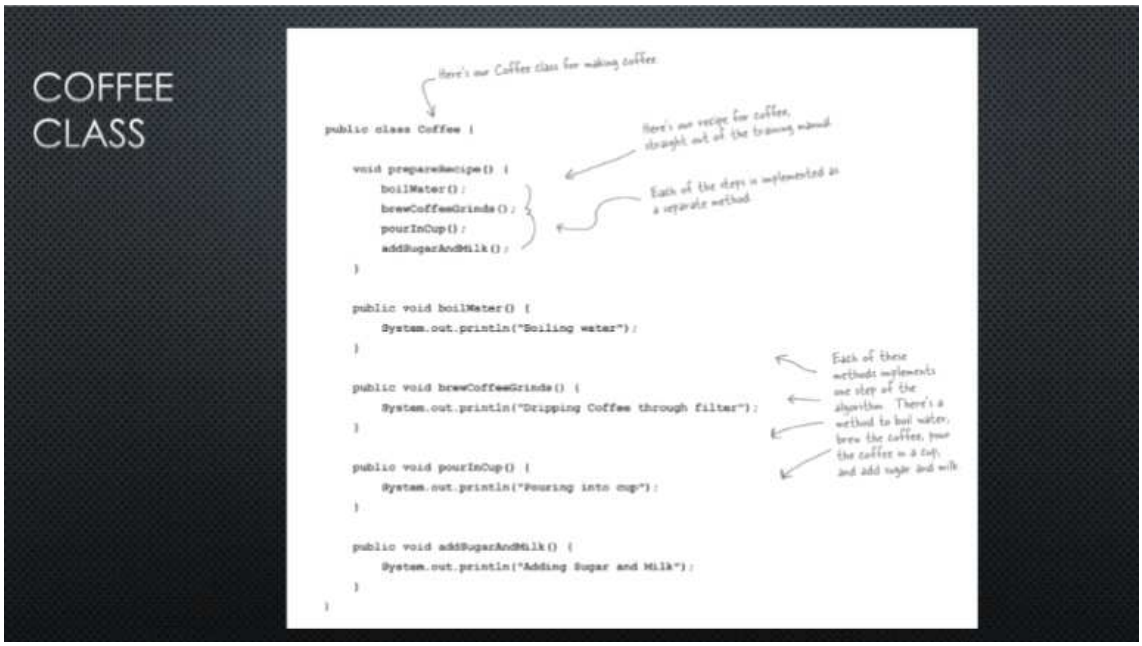
과 목 명 : 디자인패턴
담담교수 : 박제호 교수님
소　　속 : 소프트웨어학과
학　　번 : 32151671
이　　름 : 박민혁

단국대학교
Dankook University

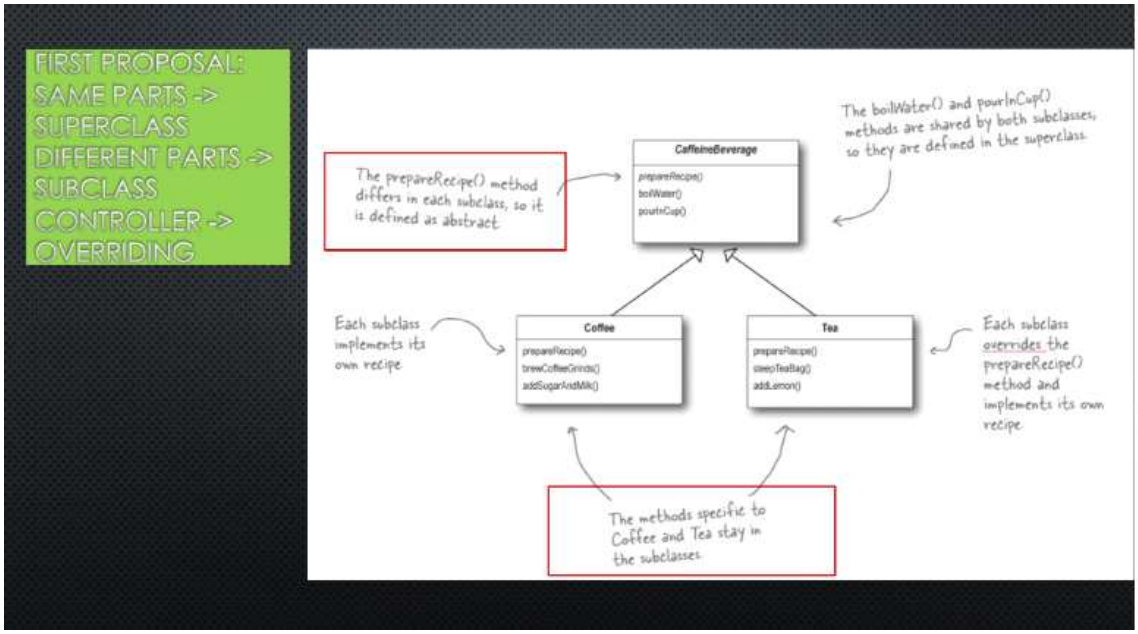
TEMPLATE PATTERN

스타버즈 커피 만드는법
(1) 물을 끓인다.
(2) 끓는 물에 커피를 우려낸다.
(3) 커피를 컵에 따른다.
(4) 설탕과 우유를 추가한다.

스타버즈 홍차 만드는법
(1) 물을 끓인다.
(2) 끓는 물에 차를 우려낸다.
(3) 차를 컵에 따른다.
(4) 레몬을 추가한다.



각 단계는 별도의 메소드로 구현
(1) 물을 끓이는 메소드
(2) 컵에 따르는 메소드
(3) 커피를 컵에 따르는 메소드
(4) 설탕하고 우유를 집어넣는 메소드



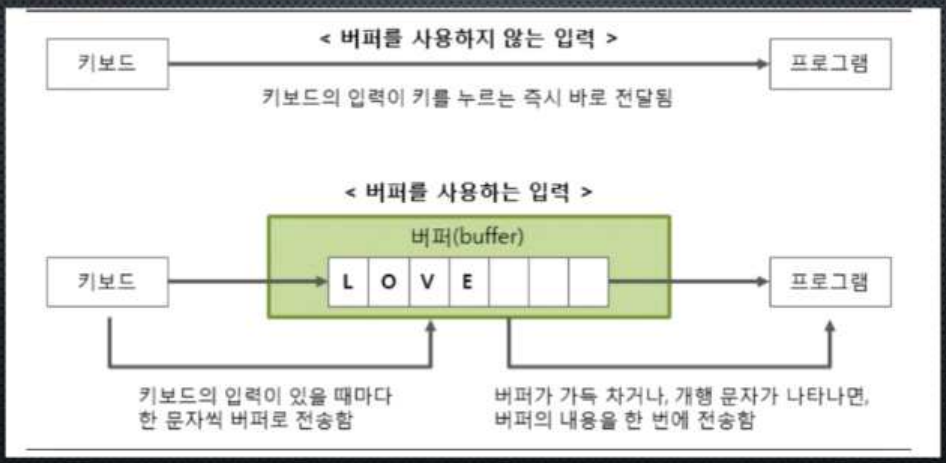
prepareRecipe() 메소드는 서브 클래스마다 다르기 때문에 추상메소드로 선언
hoilWater()하고 pourlnCup() 메소드는 두 클래스에서 겹치기 때문에 수퍼클래스에서 정의


```
public abstract class CaffeineBeverageWithHook {

  void prepareRecipe() {
    boilWater();
    brew();
    pourInCup();
    if (customerWantsCondiments()) {
      addCondiments();
    }
  }

  abstract void brew();

  abstract void addCondiments();

  void boilWater() {
    System.out.println("Boiling water");
  }

  void pourInCup() {
    System.out.println("Pouring into cup");
  }

  boolean customerWantsCondiments() {
    return true;
  }
}
```

tea와 coffe를 만들 때 똑같은 prepareRecipe() 메소드를 사용. 서브클래스에서 이 메소드를 오버라이드 해서 아무렇게나 음료를 만들지 못하도록 final로 선언.
coffee와 tea에서 이 두 메소드를 서로 다른 방식으로 처리하기 때문에 추상메소드로 선언

BUFFEREDREADER/ BUFFEREDWRITER

- 버퍼를 이용하는 입출력 메소드



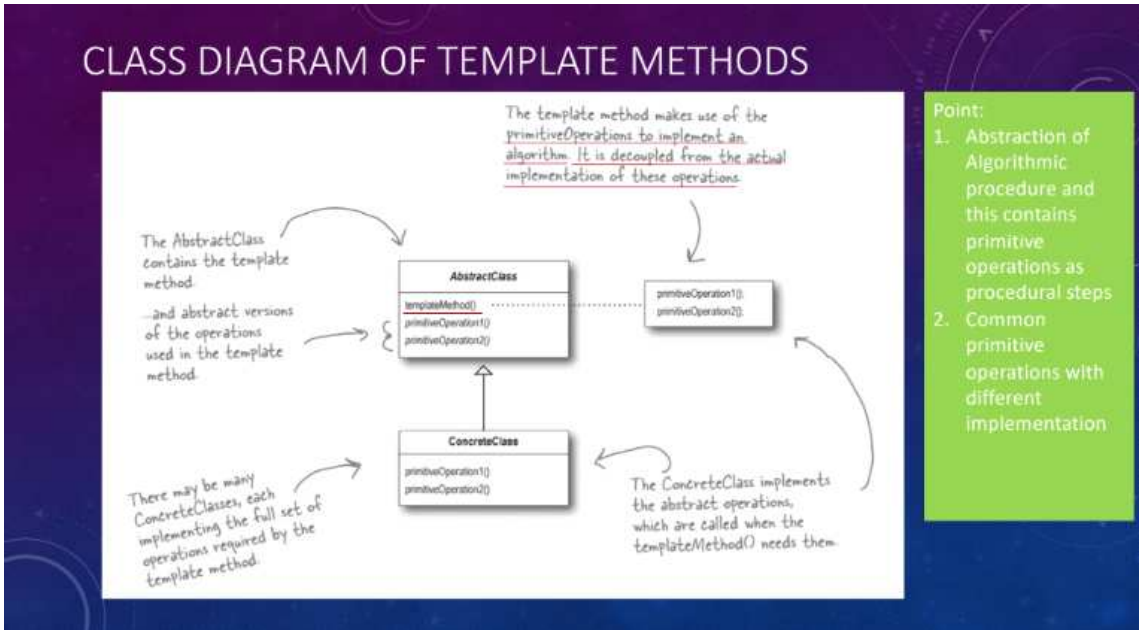
Ref: https://jhnyang.tistory.com/92

BUFFEREDREADER

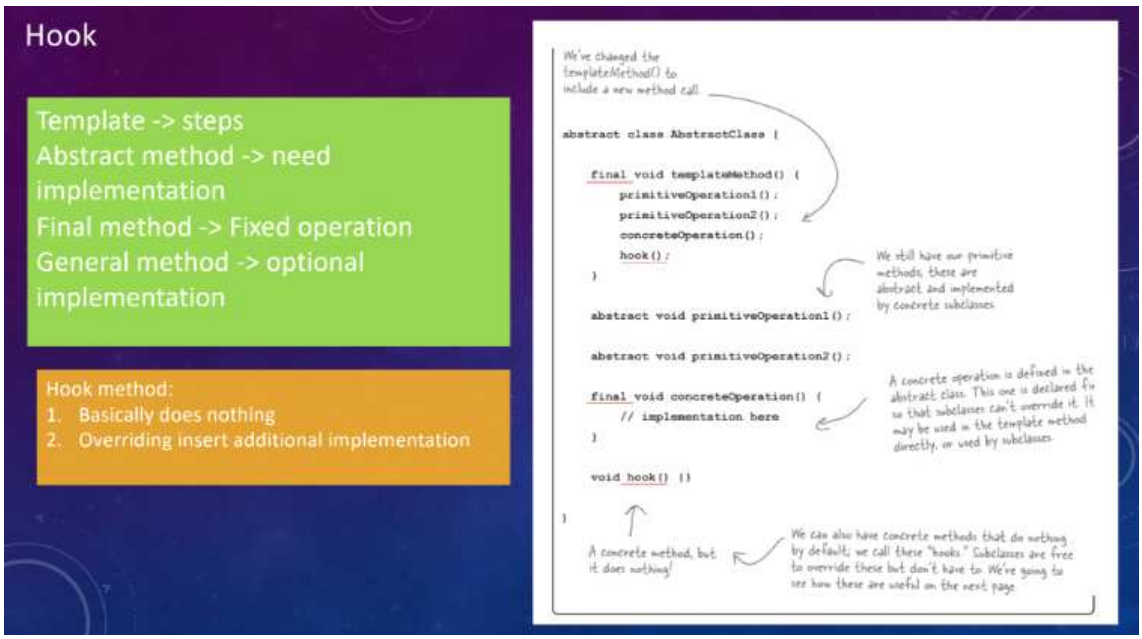
- SCANNER
  - 스페이스와 엔터를 입력값의 경계로 처리
  - 별도의 가공 필요하지 않음
- BUFFEREDREADER
  - 엔터만을 경계로 처리하고 전체를 STRING으로 처리 -> 스페이스도 입력에 포함
  - 속도는 SCANNER보다 빠름 -> 버퍼링 효과
  - 스페이스를 경계로 한 값을 읽을 때는 형변환과 TOKEN 처리 필요

```
BufferedReader bf = new BufferedReader(new InputStreamReader(System.in)); //선
String s = bf.readLine(); //String
int i = Integer.parseInt(bf.readLine()); //Int

StringTokenizer st = new StringTokenizer(s); //StringTokenizer인자값에 입력 문자열
int a = Integer.parseInt(st.nextToken()); //첫번째 호출
int b = Integer.parseInt(st.nextToken()); //두번째 호출

String array[] = s.split(" "); //공백마다 데이터 끊어서 배열에 넣음
```

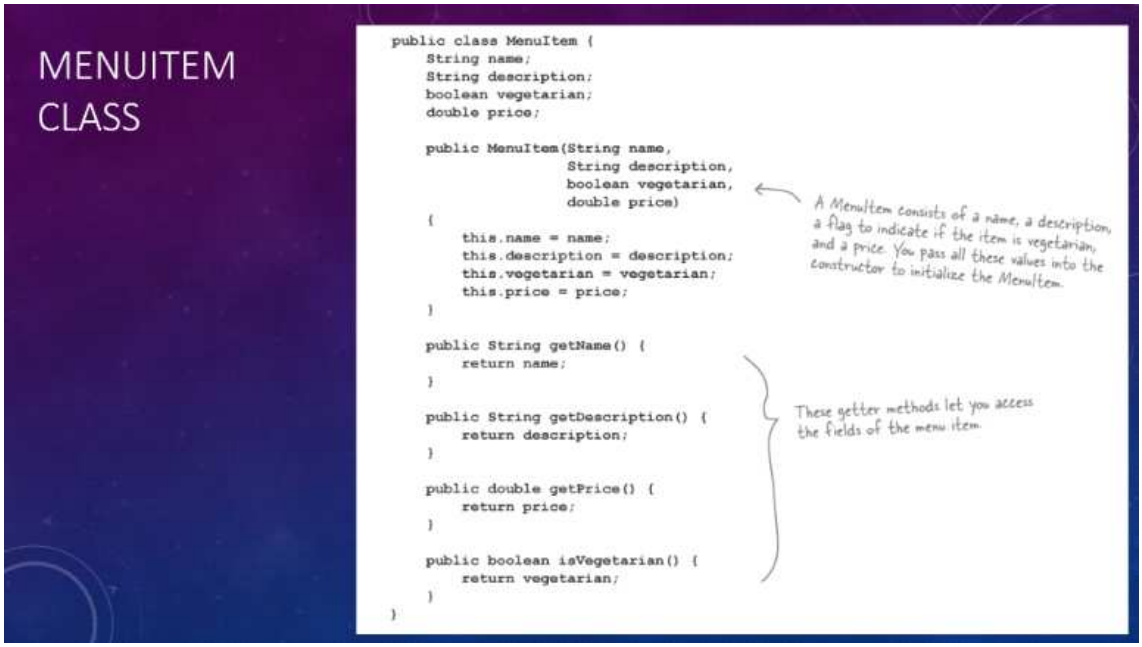
Ref: https://coding-factory.tistory.com/251



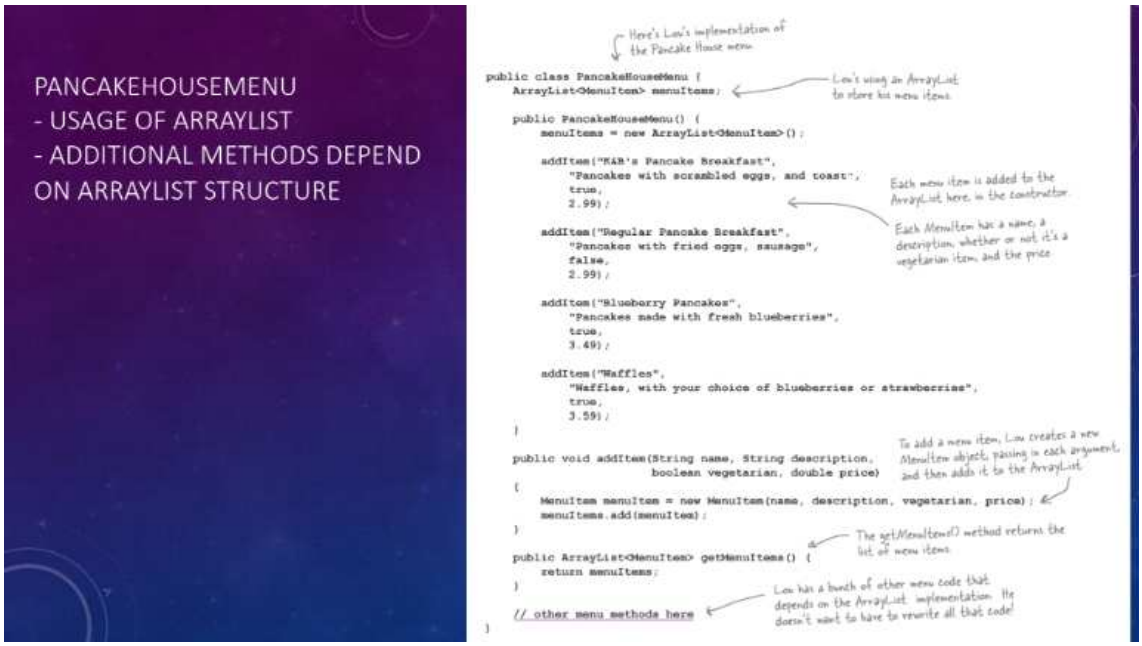
템플릿 메소드에서는 알고리즘을 구현할 때 primitiveOperational 1과 2를 활용. 알고리즘 자체는 이 단계들의 구체적인 구현으로부터 분리되어 있음.
templatemethod()에서는 이런 메소드들을 호출해서 작업처리.



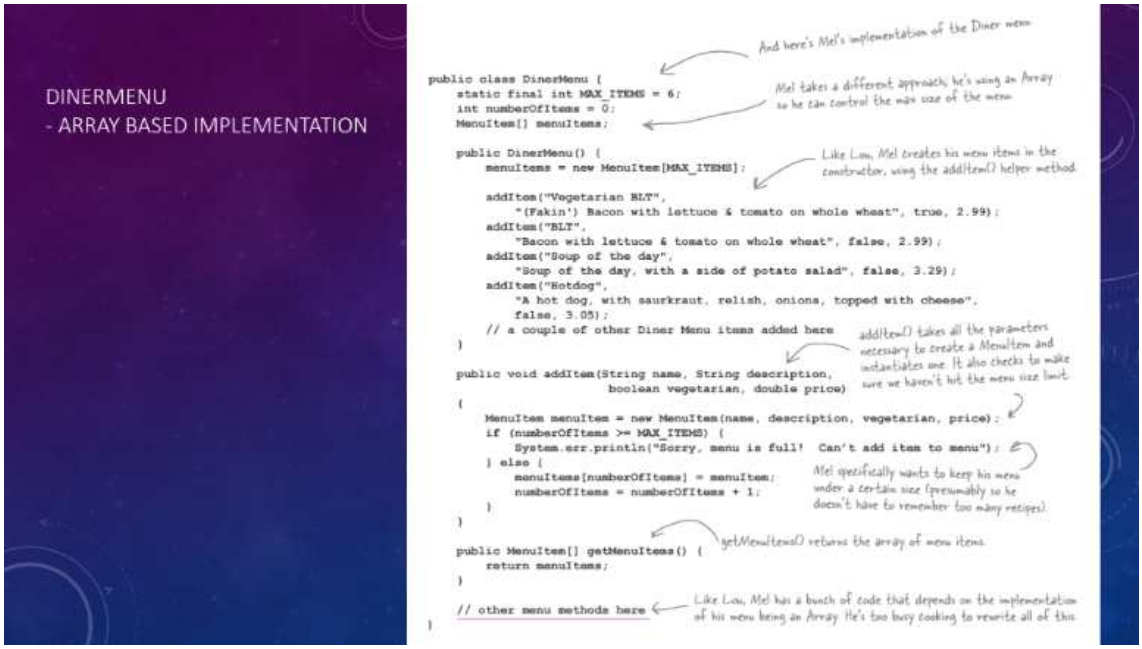
hook() 구상 메소드이긴 한데 아무것도 하지 않음. (body x)



menuitem은 이름, 설명, 채식주의 메뉴 여부, 가격으로 구성, 초기화할 때는 생성자에 이 값들을 모두 매개변수로 전달
게터 메소드를 이용하여 메뉴 항목의 각 필드에 접근 가능



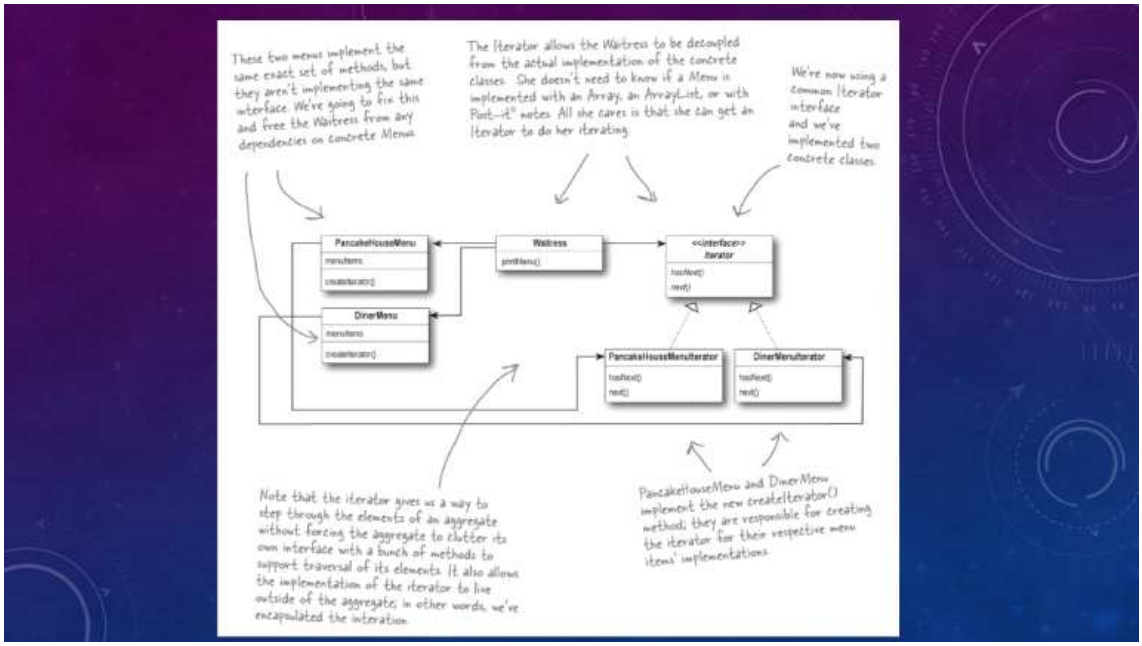
arraylist에 메뉴 항목 저장
각 메뉴 항목은 생성자에서 arraylist에 추가
각 menuitem에는 이름, 설명, 채식주의자용 메뉴여부 가격이 있음
메뉴 항목을 추가하고 싶으면 필요한 인자를 전달해서 menuitem 객체를 새로 만들고 그 객체를 추가



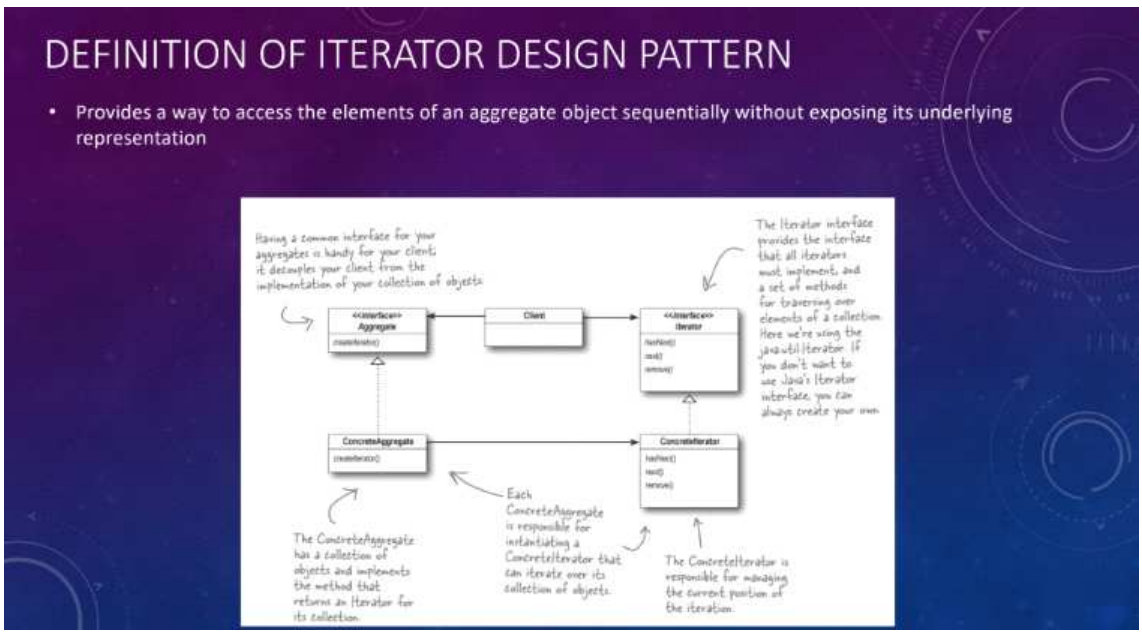
배열을 사용하고 있기 때문에 메뉴의 크기를 정해 놓았고, 객체를 캐스팅할 필요 없이 바로 꺼내서 사용 가능
additem()에서는 menuitem을 생성하기 위해 필요한 모든 매개변수를 받아서 인스턴스를 만듦 그리고 최대 메뉴 항목 개수를 넘기지 않는지도 확인

printmenu()
(1) 각 메뉴에 들어있는 모든 항목을 출력하려면 pancakehousemenu와 dinermenu의 getmenuitem() 메소드를 호출해서 메뉴 항목을 가져와야 한다. 이 두메소드의 리턴 형식이 다르다는 점에 주의

(2) pancakehousemenu에 있는 항목을 출력하기 위해서 breakfastitems arraylist에 들어있는 모든 항목들에 대해서 순환문을 돌림. 그리고 dinermenu에 들어있는 항목을 출력할 때는 배열에 대해서 순환문을 돌림.



두 메뉴에서는 똑같은 메소드들을 제공하지만 아직 같은 인터페이스를 구현x, 이 문제점을 해결 해서 waitress의 구상 메뉴 클래스에 대한 의존성 제거
Iterator 덕분에 waitress 클래스가 실제로 구현된 구상 클래스로부터 분리 가능. menu가 배열로 구현되어 있는지, arraylist로 구현되었는지, 포스트잇 메모지로 구성되어 있는지 등에 대해서는 전혀 신경 쓸 필요가 없다.
반복자를 이용하면, 컬렉션 입장에서는 그 안에 들어있는 모든 항목에 접근할 수 있게 하기 위해서 여러 메소드를 외부에 노출시키지 않으면서도, 컬렉션에 들어있는 모든 객체들에 접근할 수 있음. 그리고 반복자를 구현한 코드를 컬렉션 밖으로 끄집어낼 수 있다는 장점도 있음. 반복 작업을 캡슐화.



Iterator 인터페이스에서는 모든 반복자에서 구현해야 하는 인터페이스를 제공하며 컬렉션에

들어있는 원소들에 돌아가면서 접근할 수 있게 해 주는 메소드들을 제공함.
ConcreteAggregate에는 객체 컬렉션이 들어 있으며, 그 안에 들어있는 컬렉션에 대한 Iterator를 리턴 하는 메소드를 구현