[우아한형제들**기술 블로그**](http://woowabros.github.io/)

**생각하라, 객체지향처럼**

Jul 7, 2016 • 김승영http://www.gravatar.com/avatar/d112d756dca4c2b083fce8104962a3d2?s=50

2년차 쪼랩이가 객체지향을 처음으로 접하고 공부를 하면서 나름대로 정리해보았습니다.

**시작 인사**

안녕하세요. B2B시스템개발팀 김승영입니다.

먼저, 이 글은 **지난 02월에 작성되어 사내 위키에 공유되었던 글**임을 미리 알려드립니다. 참고로 현재는 Biz상품개발팀에서 지내고 있습니다.ㅋㅋ

이번 글은 < [객체지향의 사실과 오해](http://book.naver.com/bookdb/book_detail.nhn?bid=9145968) >라는 책의 ‘07장. 함께 모으기’ 중 커피 전문점 도메인 설계 및 구현 예제에 대한 내용을, 마치 제가 설명 하듯이 적어보았습니다.

사실 책을 보고 제가 다시 설명하는 방식으로 적었지만, 책의 설명을 그대로 옮긴 부분도 많습니다. (잘 설명되어 있는 책을 굳이.. 왜 그랬을까..ㅋㅋ) 그래도 제가 설명하다보면 더 공부도 되지 않을까해서 어설프게나마 끄적끄적 적어봤습니다.ㅎㅎ

저는 다른 객체지향 책을 읽어보지는 않아서 잘 모르지만, < 객체지향의 사실과 오해 >를 읽으셨다면 메시지, **책임**, **자율**, **협력** 등 용어가 좀 더 익숙하실 것이라고 생각합니다.

앞으로 많은 경험을 하면서 더 많이 공부하고 더 실력을 쌓아서, 나중에 이 글을 다시 읽을 때는 부끄러워서 얼굴이 빨개질 수 있도록 하려고 합니다. 그 때는 더 고급지고 우아하게 멋진 개발 관련 글을 적어볼께요. 그러니까 ‘아, 객체지향을 이해하려고 애쓰고 있구나..’ 라고 너그럽게 봐주십쇼.ㅋㅋㅋ

내용상 오류나 좀 더 추가적으로 설명해주실 좋은 의견이 있으시면, 개인적으로 알려주시거나 댓글에 남겨주시면 정말정말 감사할 것 같습니다!!

**커피 전문점을 객체지향적으로 바라보고 설계하고 구현하자**

커피 전문점 도메인 예제를 통해 도메인 모델에서 시작해서 최종 코드까지의 구현 과정을 간략하게나마 설명할 것입니다.

**커피 전문점 도메인**

* **도메인**이란? 사용자들이 관심을 가지고 있는 특정 분야나 주제를 말하며 소프트웨어는 도메인에 존재하는 문제를 해결하기 위해 개발된다.

**커피 주문**

다음 예제의 목적은 커피 전문점에서 커피를 주문하는 과정을 객체들의 협력 관계로 구현하는 것입니다.

**예제 설명 - 커피 제조하기**

이번에 다룰 도메인은 동네 어디서나 볼 수 있는 아담한 커피 전문점이다. 커피 전문점에서는 아메리카노, 카푸치노, 카라멜 마키아또, 에스프레소의 네 가지 커피를 판매하고 있다. 판매하는 커피도 몇 종류 되지 않고 내부도 그리 넓지 않지만 하늘색 페인트로 칠해진 벽과 코를 간지럽히는 고소한 커피 향이 어우러져 전체적으로 아늑하고 편안한 분위기를 풍기고 있었다. 조촐한 가게 내부를 둘러보니 하얀 색의 테이블들이 커피 향으로 가득찬 공간을 채우듯 옹기종기 놓여져 있었으며 그 위에는 커피의 이름과 가격이 적힌 작은 메뉴판이 비치돼 있었다. 손님이 테이블에 앉아 메뉴판을 잠시 훑어본 후 커피를 주문한다. 이제 주문받은 커피를 제조하는 것은 바리스타의 몫이다. 커피 전문점은 작고, 단순하고, 고려해야 할 사항도 거의 없는 아주 간단한 도메인이다. 우리의 최종 목표는 손님이 커피를 주문하는 사건을 컴퓨터 안에 재구성하는 것이다. 물론 객체를 이용해서 말이다.

**커피 전문점이라는 세상**

**1. 객체들로 구성된 작은 세상으로 바라보자**

객체지향 패러다임에서 가장 중요한 것은? 당연히 **객체**이죠. (클래스가 아니라.. 라는 얘기를 최근에 주위 분들께 많이 들었습니다.ㅋㅋ) 그렇기 때문에 먼저, 커피 전문점을 구성하는 요소들(손님, 바리스타 등)을 객체들로 보고 커피 전문점은 그 객체들로 이루어진 작은 세상으로 생각해보겠습니다. 손님이 커피를 주문하는 이 예제를 한 줄로 요약해보면 다음과 같습니다.

**→ 손님이 메뉴판에서 4가지 메뉴 항목들 중 하나를 선택해서 바리스타에게 선택한 메뉴(커피)를 주문하고, 바리스타는 커피를 제조해서 손님에게 건네줍니다.**

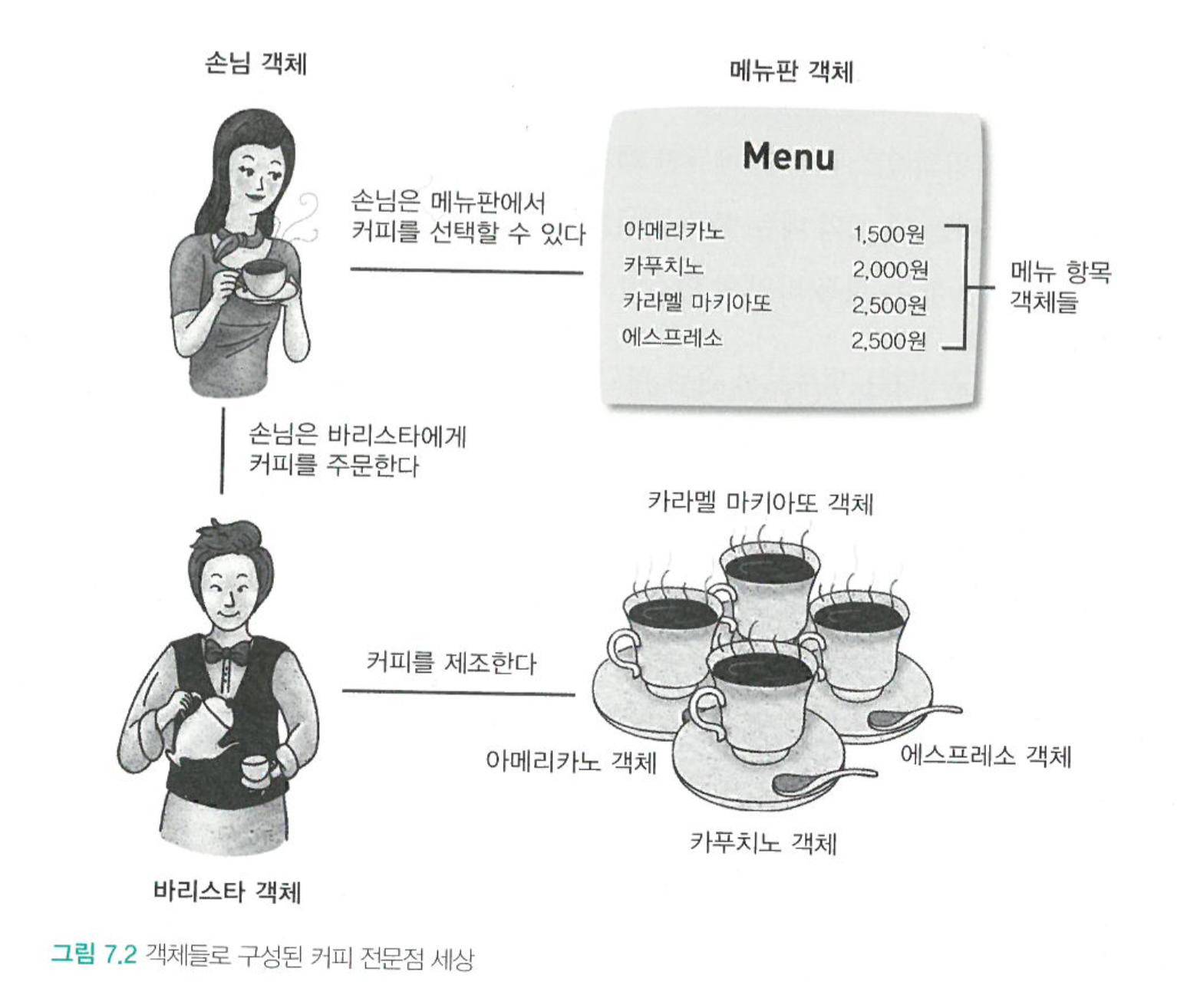
여기서 손님, 메뉴판, 메뉴 항목들(4가지), 바리스타, 커피(4가지)가 각각 하나의 객체가 될 수 있습니다. (저는 처음에 혼자 생각해볼 때 4가지의 메뉴 항목들까지 객체로 생각하지는 못 했습니다. 근데 사실 아직도 어색하긴 합니다..ㅋㅋ)

모델링에 대해서는 아직 잘 모르지만, 순서대로 보면 손님 객체와 바리스타 객체를 먼저 생각할 것 같습니다. 그리고 손님 객체가 주문을 하기 위해 메뉴가 필요할 것이고, 그럼 메뉴 객체들이 있는 메뉴판 객체가 있어야겠죠. 또한 바리스타 객체가 손님 객체의 주문에 따라 만들어서 제공해야 할 커피 객체들이 필요할 것입니다.

즉, 객체지향의 관점에서 커피 전문점이라는 도메인은 손님 객체, 메뉴 항목 객체, 메뉴판 객체, 바리스타 객체, 커피 객체로 구성된 작은 세상인 것입니다. 커피를 주문하는 과정을 객체지향 관점에서 다시 설명해보면, 아래처럼 말할 수 있을 것 같습니다.

**→ 손님 객체는 메뉴판 객체 안에 적힌 메뉴 항목 객체들 중에서 하나를 선택해서 바리스타 객체에게 전달(주문)하는 것입니다. 그리고 바리스타 객체는 주문을 받은 메뉴에 해당하는 커피 객체를 제조하는 것이죠.**

이와 같이 객체지향의 관점에서 객체들로 이루어진 커피 전문점 작은 세상을 그림으로 정리하면 아래와 같이 표현된다고 합니다. 글로만 보면서 머릿속으로 그리다가 직접 그림으로 보니까 눈에 확 들어오더라고요.

[그림 7.2 객체들로 구성된 커피 전문점 세상]

**2. 객체들 간의 관계**

지금까지는 어떤 객체가 존재하는지 보았습니다. 어렵지 않지만 저처럼 기억력이 안좋은 분들은 위의 그림 7.2를 다시 한번 보시면 좋을 것 같습니다!

다음으로는 각 객체들 사이의 관계에 대해서 살펴보겠습니다. 사실 그림 7.2에서도 각 객체들 사이의 관계는 쉽게 알 수 있죠.

손님은 메뉴판에서 커피를 선택할 수 있습니다. 이런 관계가 있는 것이죠. 손님과 바리스타, 바리스타와 커피도 마찬가지에요. 메뉴판과 커피와는 직접적으로 관계가 없는 것도 알 수 있습니다.

**3. 객체들의 분류**

이제는 동적인 객체들을 정적인 타입으로 간단하게 추상화해보겠습니다. 이를 위해 객체들을 분류할 건데, 분류를 위해서는 타입(type)을 사용합니다. 4가지 커피 객체들을 모두 같은 ‘커피 타입’으로 분류할 수 있는 것입니다.

* **타입(type)**은 특정 인터페이스를 나타낼 때 사용하는 이름입니다.

동일한 상태와 동일한 행동을 가질 수 있는 객체는 같은 타입의 인스턴스로 분류할 수 있습니다.

**클래스**와 **타입** 사이의 차이는 꼭 이해해 두어야 합니다.

**객체의 클래스**는 그 객체가 어떻게 구현되느냐를 정의합니다. 클래스는 객체의 내부 상태와 그 객체의 연산에 대한 구현 방법을 정의합니다.클래스는 구현을 정의한다.

반면, **객체의 타입**은 그 객체의 인터페이스, 즉 그 객체가 응답할 수 있는 요청의 집합을 정의합니다. 하나의 객체가 여러 타입을 가질 수 있고 서로 다른 클래스의 객체들이 동일한 타입을 가질 수 있습니다. 즉, 객체의 구현은 다를지라도 인터페이스는 같을 수 있다는 의미입니다.

— GoF의 디자인 패턴, p.46

각 객체들은 아래와 같이 모델링 할 수 있습니다.

* 손님 객체는 ‘손님 타입’의 인스턴스
* 바리스타 객체는 ‘바리스타 타입’의 인스턴스
* 4가지 커피 모두 ‘커피 타입’의 인스턴스
* 메뉴판 객체는 ‘메뉴판 타입’의 인스턴스
* 4가지 메뉴 항목 객체들 모두 동일한 ‘메뉴 항목 타입’의 인스턴스
* 메뉴판 객체는 4개의 메뉴 항목 객체를 포함

**4. 타입 간의 관계**

각 객체들을 타입별로 분류 했으니 이제는 그 타입들 간의 관계를 살펴보겠습니다. 어떤 객체들이 존재하는지 보고 객체들 사이의 관계를 알아본 것처럼 말이죠.

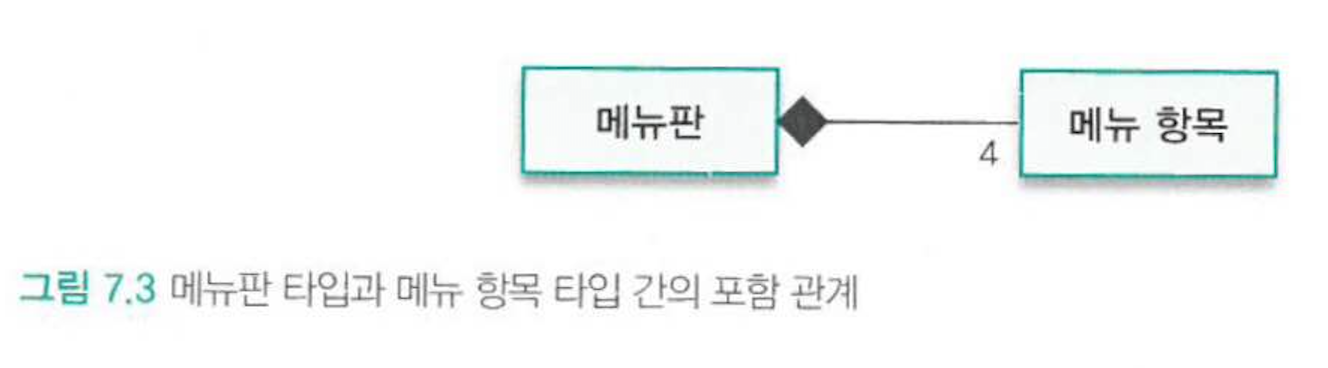
이어서 나올 내용에서 관계의 종류(합성관계, 연관관계)가 중요한 것처럼 나오는데, 사실 중요하지는 않다고 합니다. 단지, 설명을 위해서 관계들을 구분하는 것입니다.

오히려, 도메인 모델을 작성하는 단계에서는 다음 두 가지에 초점을 맞추는 것만으로도 충분하다고 합니다.

1. 어떤 타입이 도메인을 구성하느냐!
2. 타입들 사이에 어떤 관계가 존재하는지 파악함으로써 도메인을 이해하는 것!

**4-1. 메뉴판 타입과 메뉴 항목 타입 간의 관계**

메뉴 항목 객체가 메뉴판 객체에 포함되어 있으므로, 메뉴판 타입과 메뉴 항목 타입은 **합성(composition)** 관계로 단순화할 수 있습니다.  
아래 그림 7.3이 둘의 관계를 나타냅니다.

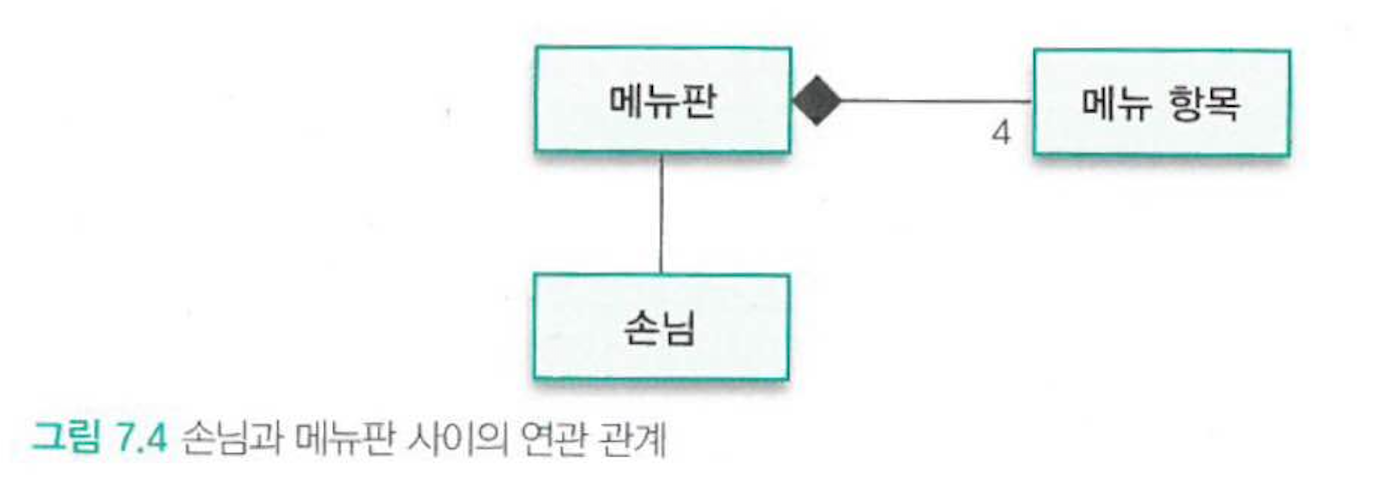
[그림 7.3 메뉴판 타입과 메뉴 항목 타입 간의 포함 관계]

검은 마름모는 **포함(containment)** 관계 또는 **합성(composition)** 관계를 나타내고, 숫자 4는 메뉴판에 포함되는 메뉴 항목의 갯수를 의미합니다.

**4-2. 손님과 메뉴판 사이의 관계**

손님 타입은 주문을 하려면 메뉴판 타입을 알아야 하지만, 그렇다고 메뉴판 타입은 손님의 일부가 아니므로 이 관계는 합성 관계는 아닙니다.

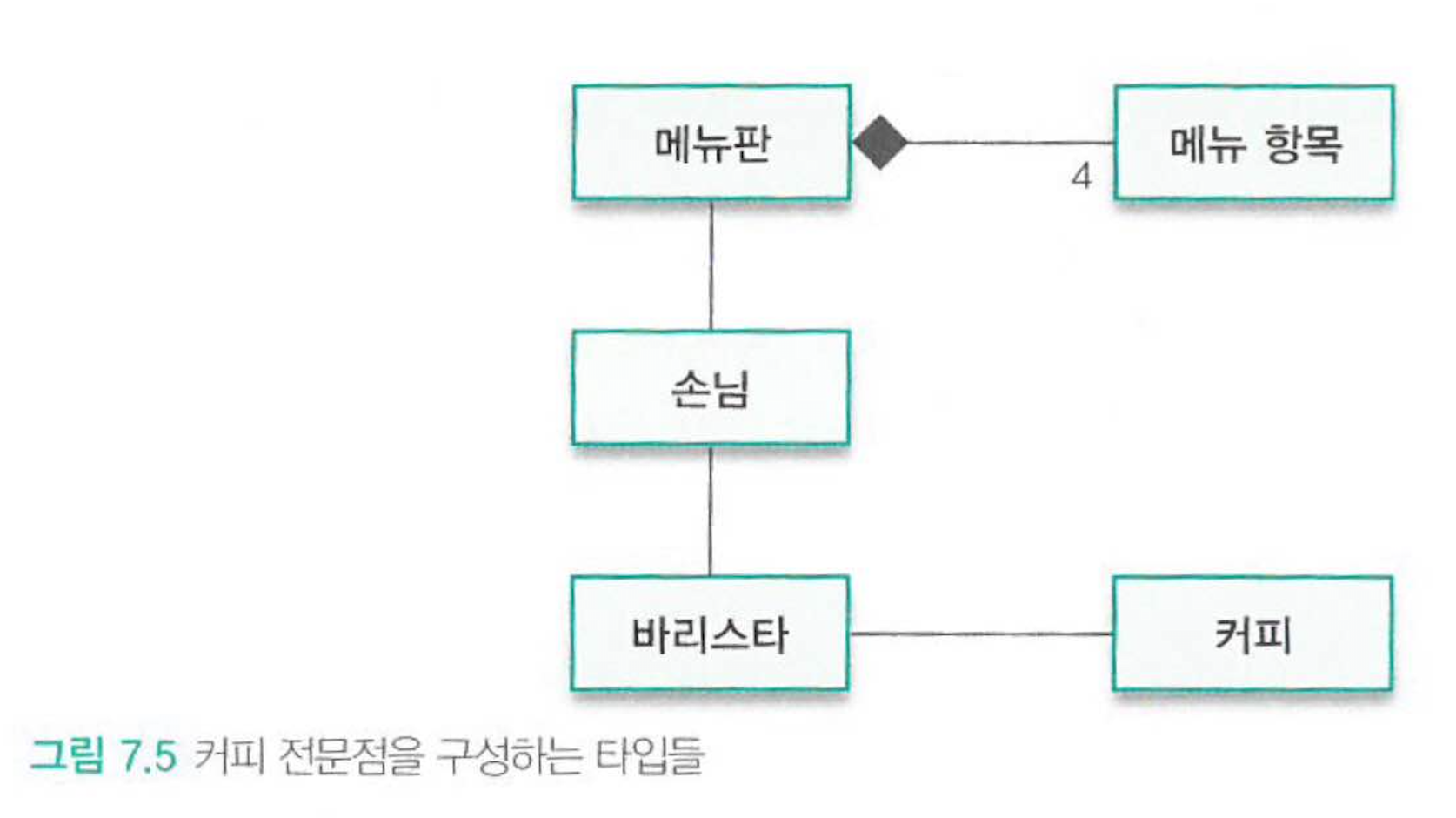
이 경우는 단순한 선으로 연결하고, **연관(association)** 관계라고 합니다.

[그림 7.4 손님과 메뉴판 사이의 연관 관계]

위와 마찬가지로 손님 타입과 바리스타 타입의 관계나 바리스타 타입과 커피 타입의 관계도 동일한 연관 관계입니다.

이렇게 해서 그림 7.5와 같이 커피 전문점 도메인을 구성하는 타입들의 종류와 관계를 표현하게 되었습니다.

이처럼 소프트웨어가 대상으로 하는 영역인 **도메인**을 단순화해서 표현한 모델을 **도메인 모델**이라고 합니다.

[그림 7.5 커피 전문점을 구성하는 타입들]

**5. 객체지향 설계**

지금까지 커피 전문점이라는 도메인을 단순화해서 이해해봤습니다. **객체지향의 세계는 협력하는 자율적인 객체들의 공동체**라는 얘기가 이 책의 초반부터 계~속 나옵니다.

다음 단계는 각 객체들의 협력을 설계하는 것입니다. 즉, 적절한 객체에게 적절한 책임을 할당하는 것입니다.

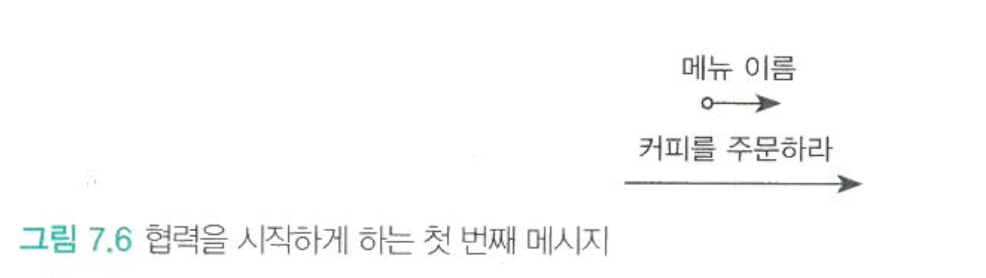
**설계하고 구현하기**

**커피를 주문하기 위한 협력 찾기**

객체지향 설계의 첫 번째 목표는 훌륭한 객체를 설계하는 것이 아니라 **훌륭한 협력을 설계하는 것** 입니다!  
훌륭한 객체는 훌륭한 협력을 설계할 때만 얻을 수 있습니다.

설계 과정에 대해 요약하면 다음과 같습니다.  
**협력을 설계할 때는 객체보다는 메시지를 먼저 선택하고 그 후에 메시지를 수신하기에 적절한 객체를 선택해야 합니다.**  
즉, 메시지가 객체를 선택하게 하는 것이죠. 그 후 메시지를 수신할 객체는 메시지를 처리할 책임을 맡게 되고 객체가 수신하는 메시지는 객체가 외부에 제공하는 공용 인터페이스에 포함됩니다.

이제 실제로 커피를 주문하는 협력을 설계해보겠습니다. 첫 번째 메시지는 ‘커피를 주문하라’일 것입니다.

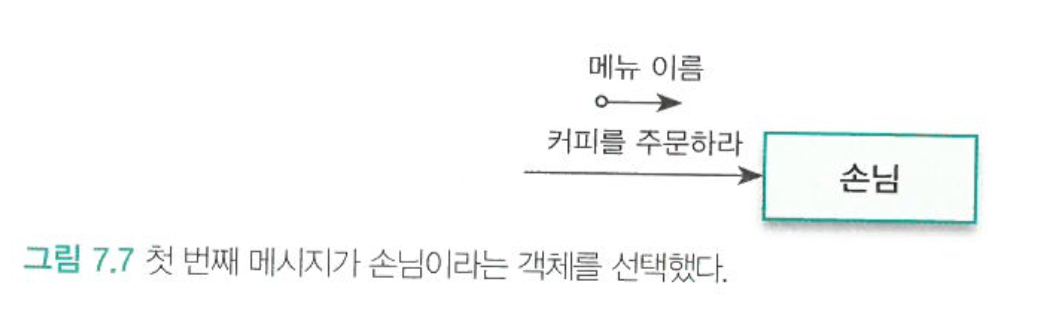
[그림 7.6 협력을 시작하게 하는 첫 번째 메시지]

메시지 위에 붙은 화살표는 메시지에 담아 전달될 부가 정보인 인자를 의미합니다. 나중에 ‘메뉴를 주문하라(아메리카노)’와 같이 인자를 포함하는 형식으로 구현될 것입니다.

메시지를 찾았으니 (설계 과정 요약에서 설명했듯이) 이제 메시지를 수신하기에 적절한 객체를 선택해야 합니다.

어떤 객체를 선택해야 할까요? 어떤 객체가 커피를 주문할 책임을 져야 할까요? -- 손님 주문하세요

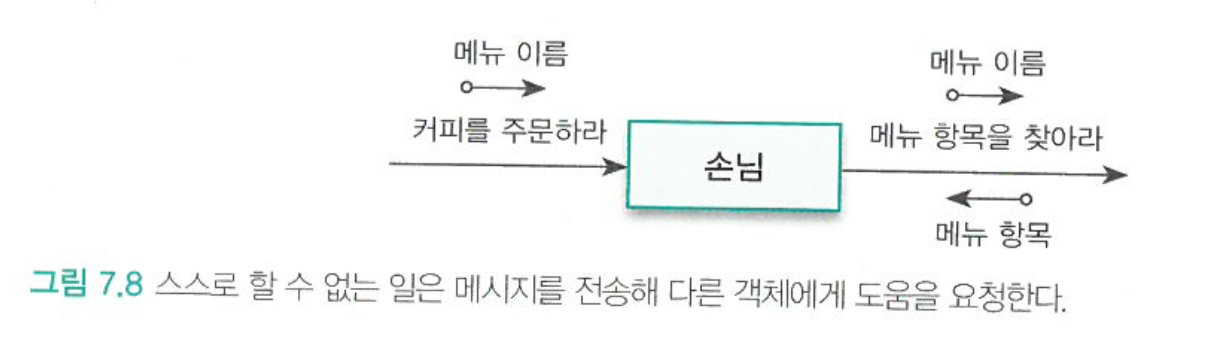
당연히 손님이겠죠? ㅎㅎ 따라서 메시지를 처리할 객체는 손님 타입의 인스턴스입니다. 이제 손님 객체는 커피를 주문할 책임을 할당받았습니다.

[그림 7.7 첫 번째 메시지가 손님이라는 객체를 선택했다.]

그런데 손님이 ‘커피를 주문하라’는 메시지를 받자마자 스스로 할 수 있는 것이 없습니다. 메뉴 항목에 대해서 모르기 때문에 스스로 바로 주문할 수 없는 상황인 것입니다. 스스로 할 수 없는 일이 있다면 다른 객체에게 이를 요청해야 합니다. 이 요청이 손님 객체에서 외부로 전송되는 메시지를 정의하고요.

손님은 자신이 선택한 메뉴 항목을 누군가가 제공해 줄 것을 요청합니다. (누구인지는 아직 모릅니다.)

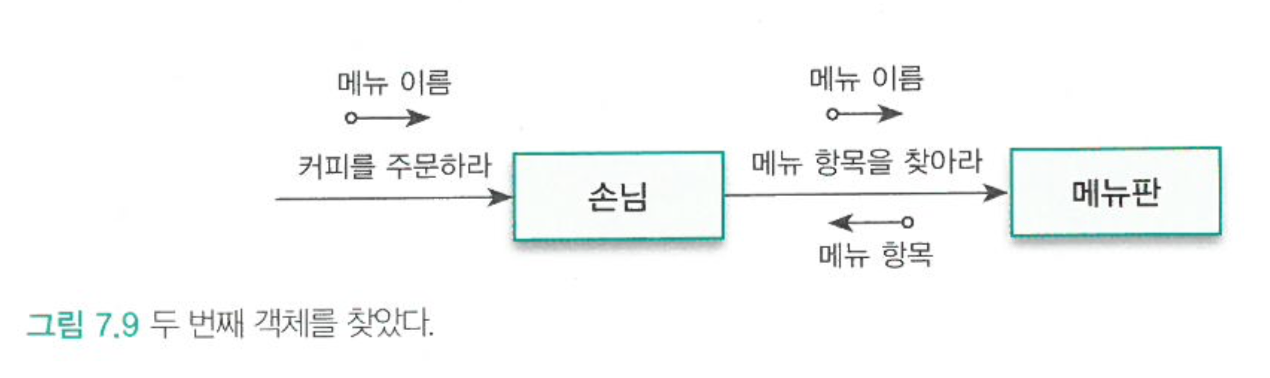
여기서 ‘메뉴 항목을 찾아라’ 라는 새로운 메시지가 등장하는 것입니다.

[그림 7.8 스스로 할 수 없는 일은 메시지를 전송해 다른 객체에게 도움을 요청한다.]

화살표 위에 있는 ‘메뉴 이름’이라는 인자를 포함해 함께 전송하고, 화살표 아래에 붙은 ‘메뉴 항목’은 손님에게 응답해야하는 것을 의미합니다.

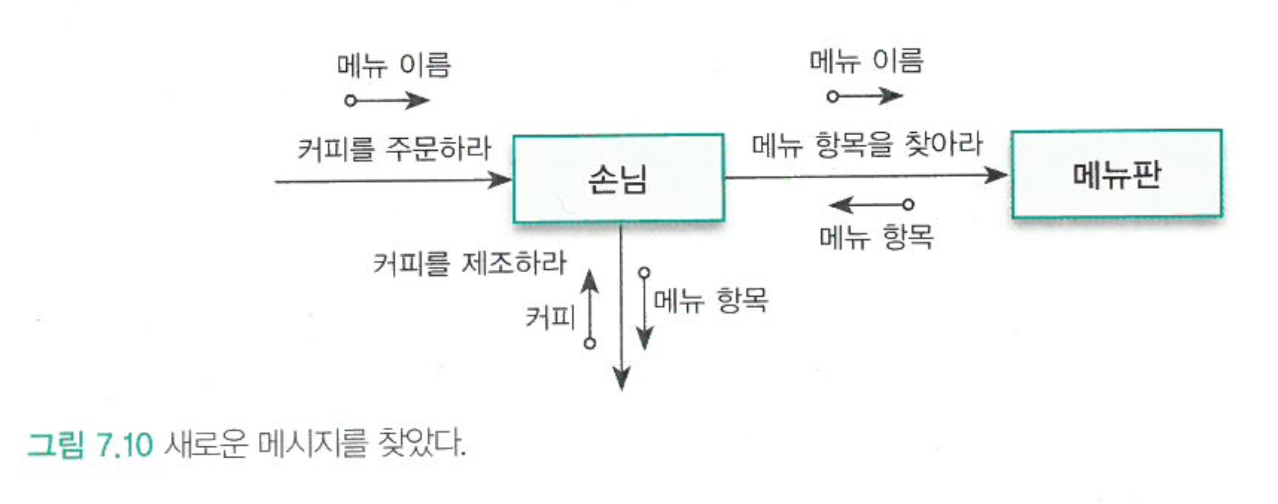
즉, ‘메뉴 항목을 찾아라’ 라는 메시지를 수신한 객체는 ‘메뉴 이름에 대응되는 ‘메뉴 항목’을 반환해야 하는 것입니다. 그럼 새로운 메시지인 ‘메뉴 항목을 찾아라’를 수신해서 메뉴 항목을 찾을 책임을 어떤 객체에게 할당하는 것이 적절할까요?

메뉴판 객체가 메뉴 항목 객체를 포함하고 있으므로 가장 적절해 보입니다.

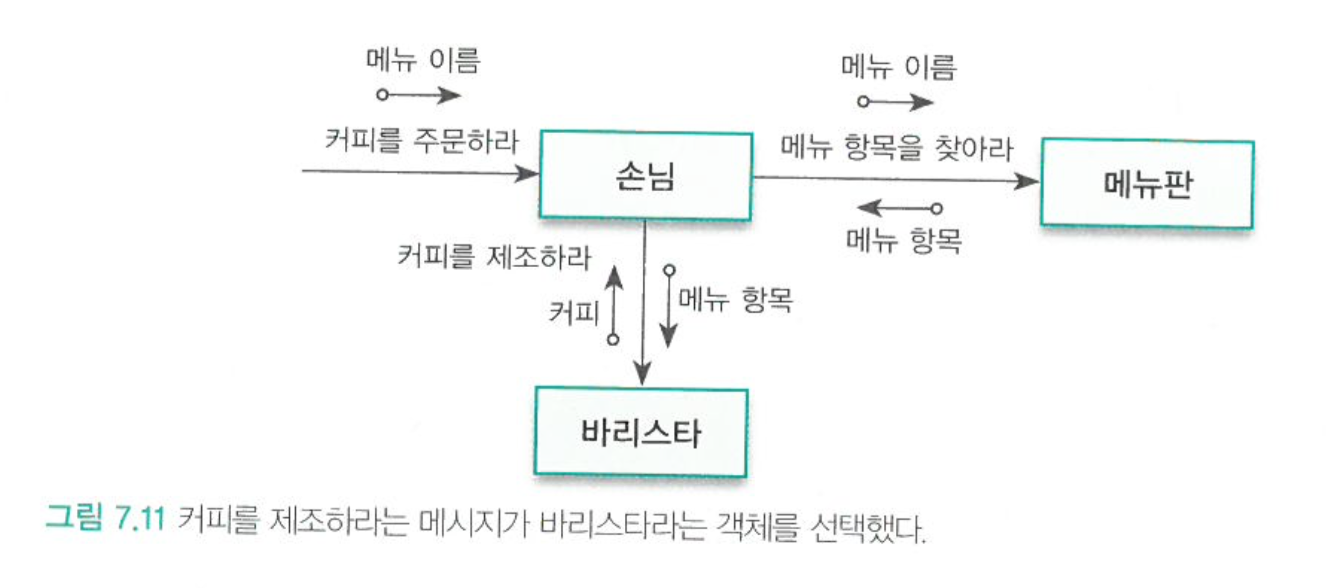
[그림 7.9 두 번째 객체를 찾았다.]

이제 손님은 자신이 주문한 커피에 대한 메뉴 항목을 얻었으니 이제 메뉴 항목에 맞는 커피를 제조해달라고 요청할 수 있습니다.

새로운 요청은 새로운 메시지가 필요하다는 신호이므로, 메시지를 먼저 정의합니다. 손님은 ‘커피를 제조하라’는 메시지의 인자로 ‘메뉴 항목’을 전달하고 반환값으로 제조된 커피를 받아야 합니다.

[그림 7.10 새로운 메시지를 찾았다.]

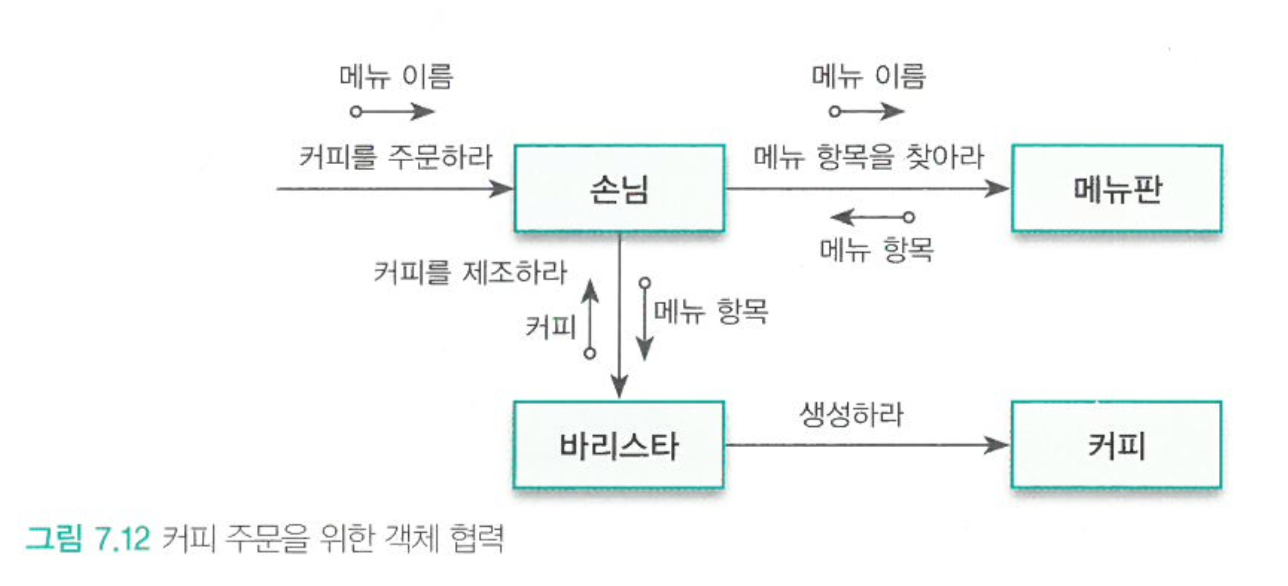
그럼 커피는 누가 제조해야 할까요? 당연히 바리스타겠죠!?

[그림 7.11 커피를 제조하라는 메시지가 바리스타라는 객체를 선택했다.]

위에서 보면 알 수 있듯이 ‘커피를 제조하라(메뉴 항목)’는 메시지를 먼저 정의하고, 그 메시지로 객체를 선택했다는 것을 잊지 말아야 합니다. 지금까지 계속해서 메시지를 먼저 정의하고 그 메시지를 수신할 객체를 선택해왔습니다.

바리스타는 아메리카노를 만드는 데 필요한 정보와 기술을 함께 구비하고 있는 전문가입니다. 아메리카노를 만들기 위한 지식은 바리스타의 상태로, 기술은 바리스타의 행동으로 간주할 수 있습니다. 이런 관점에서 바리스타는 스스로의 판단과 지식에 따라 행동하는 자율적인 존재라고 할 수 있습니다.

커피 주문을 위한 협력은 이제 바리스타가 새로운 커피를 만드는 것으로 끝납니다.

[그림 7.12 커피 주문을 위한 객체 협력] – 잘보면 함수 찾는 과정과 흡사하다

이로써 협력에 필요한 객체의 종류와 책임, 주고받아야 하는 메시지에 대한 대략적인 윤곽이 잡혔습니다. 이제 남은 일은 각 객체의 인터페이스를 구현 가능할 정도로 메시지들을 상세하게 정제하는 것입니다.

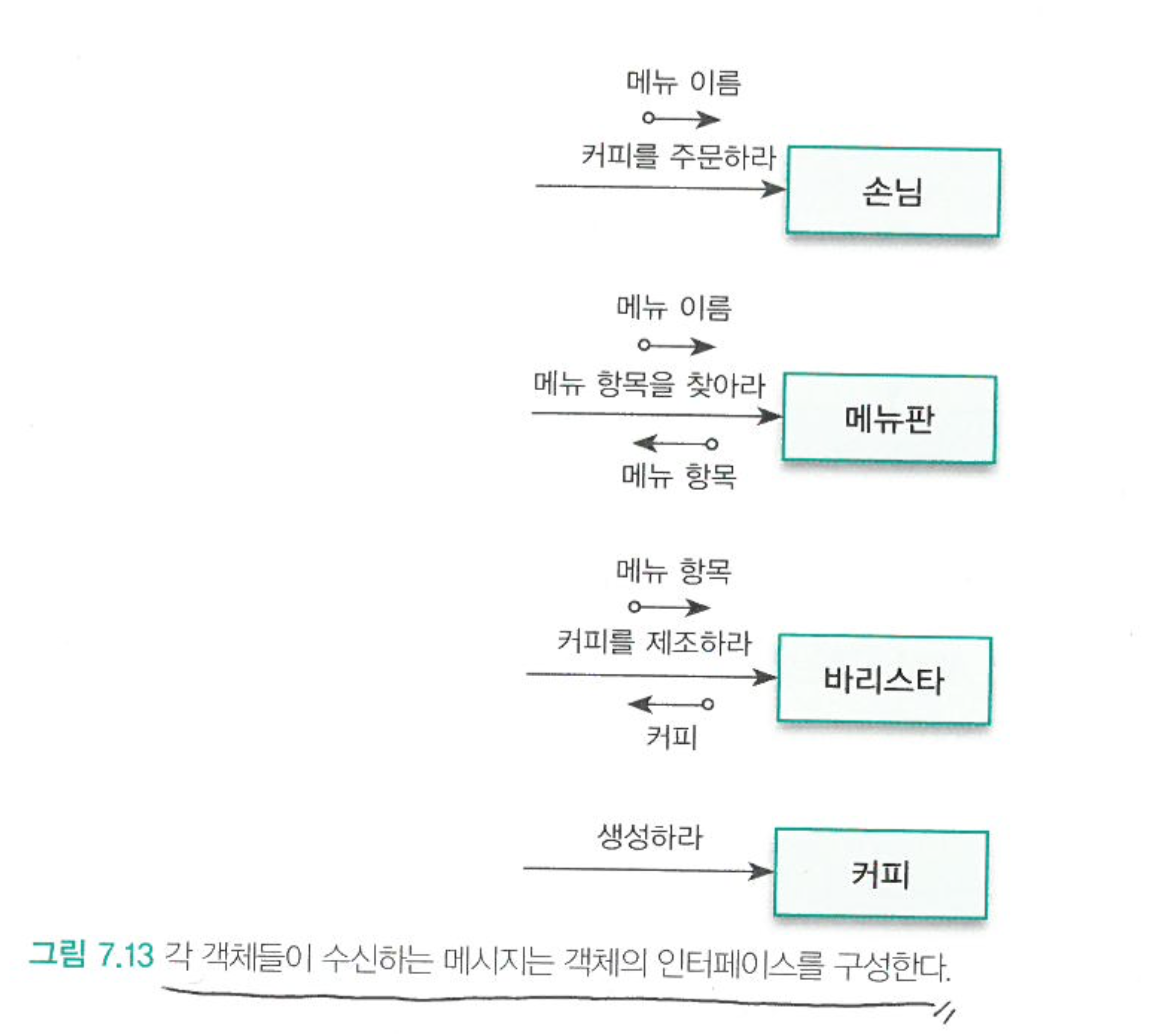
**인터페이스 정리하기**

지금까지 우리가 얻어낸 것은 객체들의 인터페이스입니다. **객체가 수신한 메시지가 객체의 인터페이스를 결정한다**는 사실을 기억해야 합니다!

메시지가 객체를 선택했고, 선택된 객체는 메시지를 자신의 인터페이스로 받아들입니다. 각 객체를 협력이라는 문맥에서 떼어내어 수신 가능한 메시지만 추려내면 객체의 인터페이스가 됩니다. 객체가 어떤 메시지를 수신할 수 있다는 것은 그 객체의 인터페이스 안에 메시지에 해당하는 오퍼레이션이 존재한다는 것을 의미합니다.

각 객체별로 설명하면 다음과 같습니다.

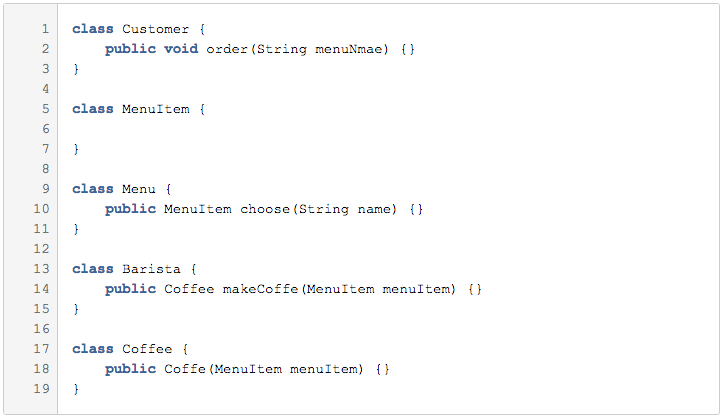
* 손님 객체의 인터페이스 안에는 ‘제품을 주문하라’ 라는 오퍼레이션이..
* 메뉴판 객체의 인터페이스 안에는 ‘메뉴 항목을 찾아라’ 라는 오퍼레이션이..
* 바리스타 객체의 인터페이스 안에는 ‘커피를 제조하라’ 라는 오퍼레이션이..
* 커피 객체는 ‘생성하라’ 라는 오퍼레이션을..

[그림 7.13 각 객체들이 수신하는 메시지는 객체의 인터페이스를 구성한다.]

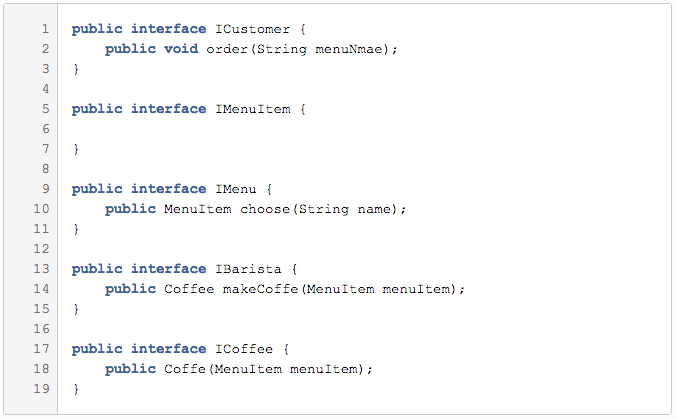
실제로 소프트웨어의 구현은 객체들을 포괄하는 타입을 정의한 후 식별된 오퍼레이션을 타입의 인터페이스에 추가해야 합니다.

객체의 타입을 구현하는 일반적인 방법은 클래스를 이용하는 것입니다. 협력을 통해 식별된 타입의 오퍼레이션은 외부에서 접근 가능한 공용 인터페이스의 일부라는 사실을 기억하라! (책에서 계속 이야기 하는 내용입니다.)

따라서 인터페이스에 포함된 오퍼레이션 역시 외부에서 접근 가능하도록 공용(public)으로 선언돼 있어야 합니다. 클래스의 인터페이스는 자바의 문법을 이용해 표기합니다.



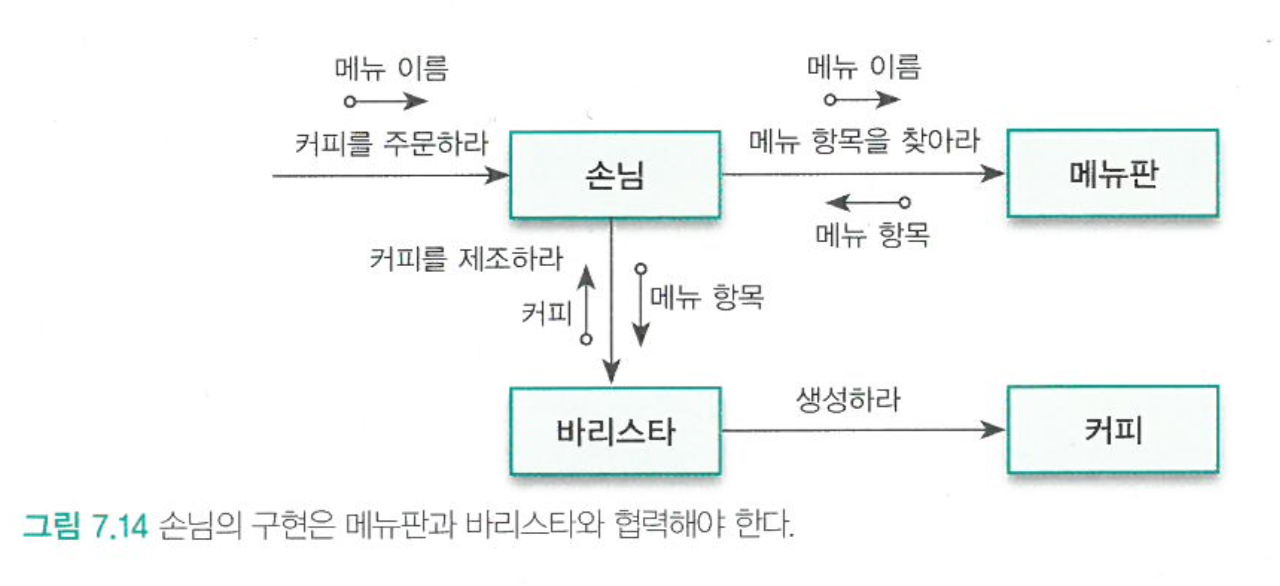
앞으로 나오는 설명에 추가적으로 interface를 정의해보겠습니다.



**구현하기**

클래스의 인터페이스를 식별했으므로 이제 오퍼레이션을 수행하는 방법을 메서드로 구현해보겠습니다.

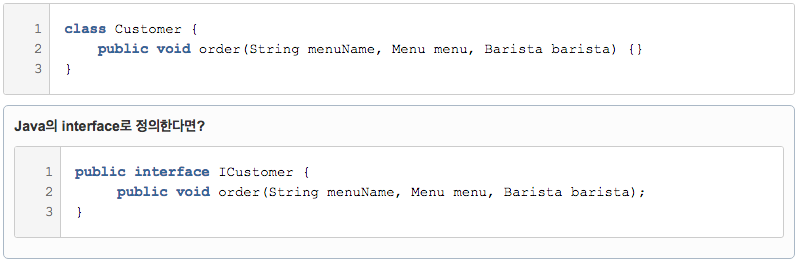
먼저 **Customer**의 협력을 살펴보겠습니다. **Customer**는 **Menu**에게 menuName에 해당하는 **MenuItem**을 찾아달라고 요청해야 합니다. 그리고 **MenuItem**을 받아 이를 **Barista**에게 전달해서 원하는 커피를 제조하도록 요청해야 합니다.

[그림 7.14 손님의 구현은 메뉴판과 바리스타와 협력해야 한다.]

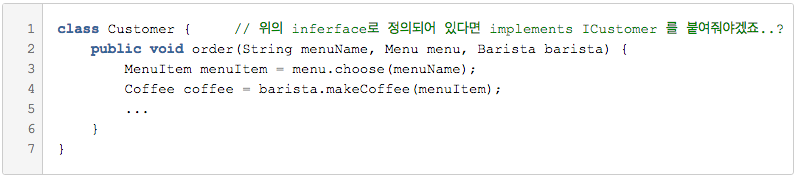
**Customer**가 **Menu** 객체와 **Barista** 객체에 접근하기 위해서, 즉 메시지를 전송하기 위해서는 어떠한 방법으로든 자신과 협력하는 **Menu** 객체와 **Barista** 객체에 대한 참조를 알고 있어야 합니다.

객체 참조를 얻는 다양한 방법이 있지만 (Spring에서의 DI가 그 중 하나이겠죠?) 여기서는 Customer의 order() 메서드의 인자로 Menu와 Barista 객체를 전달받는 방법으로 참조 문제를 해결하기로 합니다.

그럼.. **Customer**의 인터페이스를 변경해야겠죠?



이제 order() 메서드의 구현을 채워보겠습니다!



**참고**

구현하지 않고 머릿속으로만 구상한 설계는 코드로 구현하는 단계에서 대부분 변경된다. 설계 작업은 구현을 위한 스케치를 작성하는 단계지 구현 그 자체일 수는 없다.

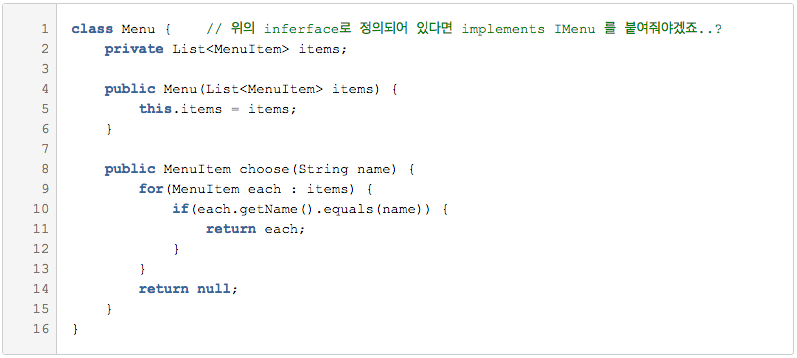
중요한 것은 설계가 아니라 코드다. 따라서 협력을 구상하는 단계에 너무 오랜 시간을 쏟지 말고 최대한 빨리 코드를 구현해서 설계에 이상이 없는지, 설계가 구현 가능한지를 판단해야 한다.

코드를 통한 피드백 없이는 깔끔한 설계를 얻을 수 없다.

**Menu**는 **menuName**에 해당하는 **MenuItem**을 찾아야 하는 책임이 있습니다.

도메인 모델을 설계할 때처럼 간단히 **Menu**가 내부적으로 **MenuItem**을 포함하게 하겠습니다.

**Menu**의 choose() 메서드는 **MenuItem**의 목록을 하나씩 검사해가면서 이름이 동일한 **MenuItem**을 찾아 반환합니다..



**참고**

MenuItem의 목록을 Menu의 속성으로 포함시킨 결정 역시 클래스를 구현하는 도중에 내려졌다는 사실에 주목하라. 객체의 속성은 객체의 내부 구현에 속하기 때문에 캡슐화되어야 한다.

객체의 속성이 캡슐화된다는 이야기는 인터페이스에는 객체의 내부 속성에 대한 어떤 힌트도 제공되어서는 안 된다는 것을 의미한다.

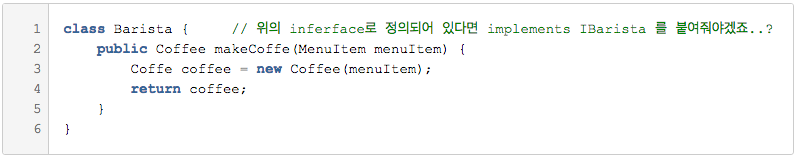
이를 위한 가장 훌륭한 방법은 인터페이스를 정하는 단계에서는 객체가 어떤 속성을 가지는지, 또 그 속성이 어떤 자료 구조로 구현됐는지를 고려하지 않는 것이다.

객체에게 책임을 할당하고 인터페이스를 결정할 때는 가급적 객체 내부의 구현에 대한 어떤 가정도 하지 말아야 한다.

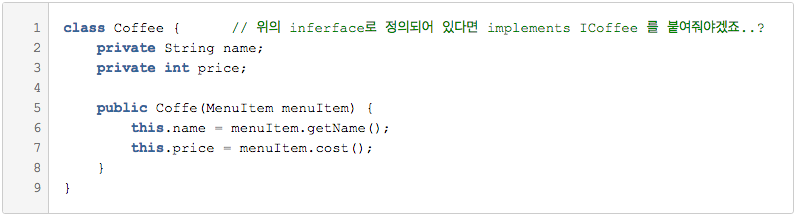
객체가 어떤 책임을 수행해야 하는지를 결정한 후에야 책임을 수행하는 데 필요한 객체의 속성을 결정하라.

이것이 객체의 구현 세부 사항을 객체의 공용 인터페이스에 노출시키지 않고 인터페이스와 구현을 깔끔하게 분리할 수 있는 기본적인 방법이다.

**Menu**는 menuName에 해당하는 **MenuItem**을 찾아야 하는 책임이 있습니다. 도메인 모델을 설계할 때처럼 간단히 **Menu**가 내부적으로 **MenuItem**을 포함하게 하겠습니다.



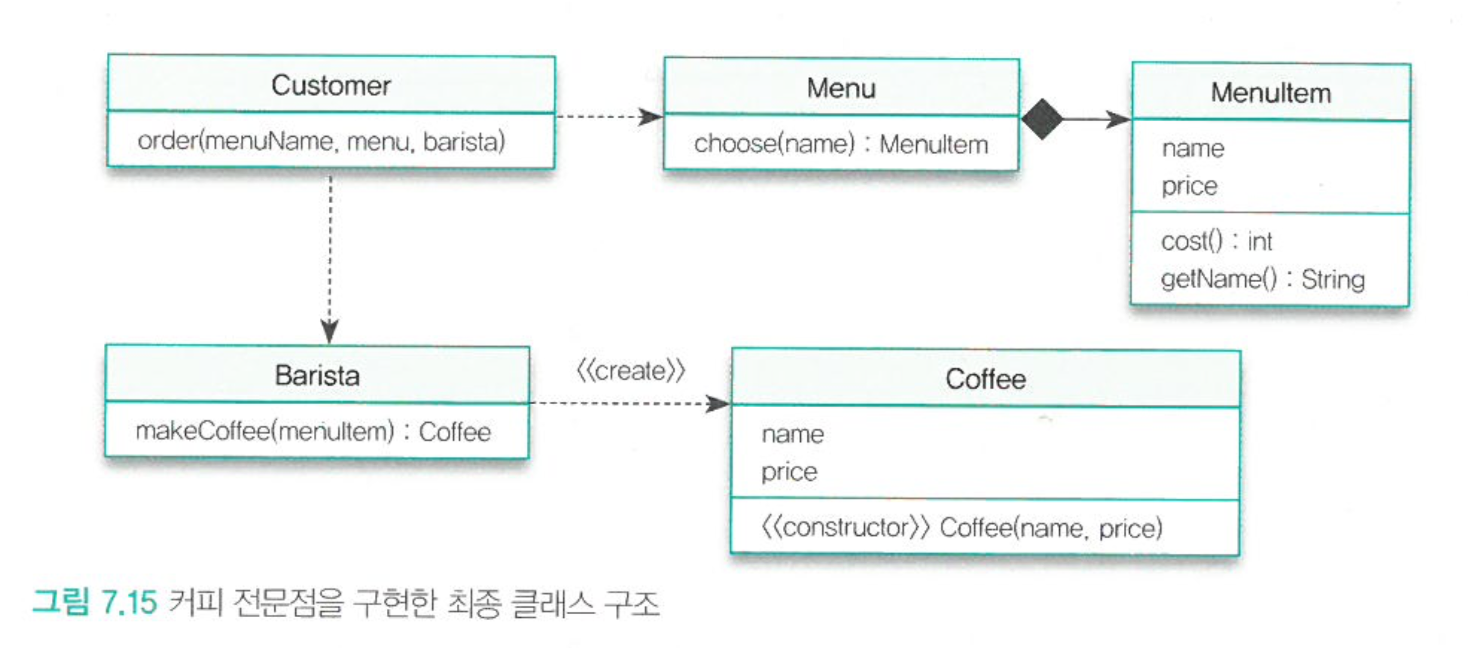
**Coffee**는 자기 자신을 생성하기 위한 생성자를 제공합니다.  
**Coffee**는 커피 이름과 가격을 속성으로 가지고 생성자 안에서 **MenuItem**에 요청을 보내 커피 이름과 가격을 얻은 후 **Coffee**의 속성에 저장합니다.



**MenuItem**은 getName() 과 cost() 메시지에 응답할 수 있도록 메서드를 구현해야 합니다.



아래 그림 7.15는 커피 전문점 코드를 클래스 다이어그램을 나타낸 것입니다. 몇 가지 사소한 관계는 생략되어 있지만 중요한 측면은 모두 포함하고 있습니다.

[그림 7.15 커피 전문점을 구현한 최종 클래스 구조.]정확하게 예측하는 것은 불가능에 가깝다.

설계를 간단히 끝내고 최대한 빨리 구현에 돌입하라. 머릿 속에 객체의 협력 구조가 번뜩인다면 그대로 코드를 구현하기 시작하라.

설계가 제대로 그려지지 않는다면 고민하지 말고 실제로 코드를 작성해가면서 협력의 전체적인 밑그림을 그려보라.

테스트-주도 설계로 코드를 구현하는 사람들이 하는 작업이 바로 이것이다. 그들은 테스트 코드를 작성해 가면서 협력을 설계한다.

와우!! 드디어 구현까지 끝났습니다!!

오류도 많고 이상한 부분도 많겠지만, “객체지향적인 설계와 구현이 무엇인지 이해하기 시작했을 것이다.” 라고 책에는 쓰여있는데… 네.. 그렇죠, 이제 시작이죠.ㅎㅎ

중요하면서 기억해야 할 개념들을 정리해보겠습니다.

* **메시지가 객체를 결정한다는 말의 의미**

→ 어떤 요청(메시지)을 처리하기에 적절한 객체를 선택 했었죠.

* **책임을 따라 설계가 이뤄지는 과정**

→ 기억나시죠? 위에서 계속 메시지를 정한 후에 그 메시지를 수신하고 처리할 책임이 있는 객체를 정했어요.

* **인터페이스와 구현의 분리** → 인터페이스는 인터페이스 대로 정의하고나서 그 후에 구현을 했습니다.

**마무리 인사**

먼저 긴 글 읽어주셔서 감사합니다! ㅎㅎ

마치 제가 저자가 된듯이 설명을 하니까 읽기만 했을 때보다 더 공부가 되는 것 같았습니다.  
물론 설계와 구현을 글과 말로 끝낼 수는 없겠죠?! 실제로 직접 설계도 해보고 구현도 해보아야 그때부터 저의 진짜 지식이 되고 실력이 될테니까요.  
그래서 요즘 우리 B2B시스템개발팀에서 진행하고 있는 스터디와 실습, 그리고 3~4월부터 시작할 신규시스템 개발을 하는 경험을 통해 더 많이 배워서 더 실력을 갈고 닦아보려고 합니다!!

일도 하고 공부도 하면서 준비할 것도 많은 요즘, 누구보다도 저 자신부터 스스로 더더더 화이팅 하려고 합니다.  
모두 화이팅 하시고, 주변에 저 같은 쪼랩이들도 많이 가르쳐주시면서 가치 있고 멋진 2016년 보낼 수 있었으면 좋겠습니다!!

그럼.. 전 이만 물러나보도록 하겠습니다~ㅎㅎ  
긴 글 읽어주셔서 다시 한 번 감사합니다!

끗.