**컴퓨터 발전과정.**

1930년대, 암호 디버깅 인코딩 용도(군사)

1940년대, 탄도미사일 거리 계산, 폭발 계산 등등(군사)

1950년대, 전 과학분야에서 컴퓨터를 사용하기 시작

1970년대, 컴퓨터가 보급되기 시작, 인터넷 등장, C언어 출현

1980년대, 컴퓨터가 폭발적으로 공급, 그래픽 기반 소프트웨어등장,

1990년대, WWW 웹의 등장, IT버블

2000년대, 완전히 새로운 검색엔진 구글 등장, 집단지성(wiki) 등장, 페이스북 등장,

2010년대, 빅데이터, 인공지능, 블록체인

2020년대는?

컴퓨터의 사용범위가 계속해서 증가. 프로그램을 만드는 프로그래밍언어에게 유연성이 강조되게 된다. 객체지향은 원래 절차지향 등장이전에도 있던 프로그래밍 방식. 하지만 메모리를 많이 잡아먹어서 (메소드 하나를 쓰기 위해 클래스를 메모리에 올려야 한다.) 절차지향방식에 비해 인기가 없었다.

1980년대 이후 컴퓨터 성능이 발전하고, 또 다양한 분야에서 소프트웨어를 활용하기 시작하면서 유연성이 좋은 객체지향 프로그래밍을 선호하기 시작

어떻게 하면 유연한 프로그래밍 언어를 만들 수 있을까?

인간의 언어를 연구하기 시작한다.

인간의 언어는 유연하다. 세상의 어떤 복잡한 것도 다 글로 담아낸다. 어떻게? 추상화를 사용한다.

추상화를 하는 기준은 맥락이다. 맥락에 필요한 부분만 남겨두고 다른 부분인 제외하더라. 중요한 것은 맥락이 전달되는 것이니까, 맥락에 필요한 부분만 빼고 나머지는 다 버린다.

유연성을 달성하는 방식이 바로 추상화이다. 오늘의 수업주제는 추상화이다.

추상화 : 공통적인 것은 드러내고, 다른 것은 감춘다.

**KookBabDreeper 예시**

클래스, 인스턴스 까지 설명.

**객체지향 프로그래밍 방식**

우리가 글로 세계를 묘사할 때 대상을 적는 것이 아니라 사건을 적는다. 지금 이 풍경을 글로 나타낸다면, 강사는 수업을 하고 학생들은 수업을 듣는다 라고 적지 강사 키가 얼마이고, 머리카락이 몇 이고 이런 것에 집중하지 않는다. 그런 세세한 것들은, 즉 사건의 맥락에 맞지 않는 대상의 속성들은 추상화 시켜버린다.

그럼 그 사건 이라는 것이 뭔지를 살펴봤더니, 사건이라는 것은 객체와 객체의 상호작용이더라 하고 정리가 되었다. 왜 이렇게 정리되었는가 하니까, 사람이 세상을 바라보는 관점이 이렇다는 겁니다. 굉장히 철학적인 부분이니까 이 부분은 너무 깊게 가지는 말고, 우리는 개발자스럽게, 이전의 프로그래밍과 객체프로그래밍을 비교하면서 알아보자.

**객체지향프로그래밍 설명 -> MVC패턴으로 구현**

뭐가 더 현실에 가까운 설계방식 입니까.

물론 여러분들이 혼자 살면, 순차적 프로그래밍 방식으로 사시겠죠. 혼자 다 해야 하니까, 하지만 세상은 다 같이 살죠? 그렇기 때문에 세상에서 발생하는 일들의 양상은 객체지향에 훨씬 가까운 겁니다. 그리고 현실의 양상에 가까운 방식으로 설계하니까, 당연히 현실의 각종 사건들을 훨씬 유연하게 프로그램으로 만들 수 있는 것입니다.

**PPT 4p**

현실세계는 객체로 이루어져 있다.

이 문장을 보겠습니다.

자바가 객체지향 프로그래밍이라는 말은 자바가 객체로 이루어져 있다는 뜻 입니다. 그런데 자바가 개발될 당시에는 절차지향프로그래밍의 영향력이 하도 큰 시기여서, 어쩔 수 없이 절차지향적인 부분들을 받아들이게 됩니다. 그게 바로, 여러분들이 아는 기본형 변수 입니다.

기본형 변수가 뭡니까. new연산자로 생성하지 않는 변수죠.

자바에서 객체는 new연산자로 생성하는 모든 것 입니다. 나중에 객체배열 때 다루겠지만, 배열 new로 생성하시죠? 객체입니다. String! new로 생성하시죠? 객체입니다. 지금은 이쯤에서 넘어가고 나중에 상속 때 조금 더 자세히 알아보겠습니다.

**클래스의 등장배경**

변수 -> 배열 -> 구조체 -> 클래스

절차지향에서는 데이터와 함수를 구분해서 사용.

데이터가 있고, 그 데이터를 조작하는 함수가 따로 있었음.

객체지향에서는 데이터와 함수를 한꺼번에 묶어서 사용

**추상화**

실제로 추상화가 이루어질 때는 우선 메서드가 존재한다.

요구사항 -> 요구사항 분석(기능과 유스케이스 다이어그램) -> 이후 속성 추출

여러분들이 만들고자 하는 기능에 필요한 속성을 추출하면 된다.

ex) 의자 예시

클래스를 만들 때 속성을 뽑기 위해서 반드시 추상화를 해야 한다는 것은 위험한 생각. 필요하다면 클래스를 만들 수도 있다.

**국민 클래스 코딩**

**객체지향프로그래밍에서의 객체**

**String foodName;**

**int foodPrice;**

**int kookBabPrice;**

**뭐? { foodName }? { foodName }??**

**그 돈 주고 { foodName } 먹을 바에야 뜨끈한 국밥 { foodPrice / kookBabPrice }그릇,  
든든~히 먹고 말지**

이렇게 만들어진 것이 클래스이다. 공통점은 남겨두고 차이점은 감춰서 하나의 틀로 만들었다.

여러분이 작성한 클래스(즉 코드)는 프로그램이 실행되는 순간 static영역에 저장된다.

new 생성자(매개변수)를 하는 순간 여러분이 작성한 클래스 중에서, 생성자명과 클래스명이 일같은 클래스를 찾는다. 그 뒤, 초기화 블록에 초기화 값이 있다면, 혹은 생성자에 매개변수와 초기화 코드가 있다면 초기화를 진행하고 heap영역에 올린다. 이게 인스턴스화이다.

즉 우리가 추상화를 통해서 틀로 만들었던 아래의 문장이

**뭐? { foodName }? { foodName }??**

**그 돈 주고 { foodName } 먹을 바에야 뜨끈한 국밥 { foodPrice / kookBabPrice }그릇,  
든든~히 먹고 말지**

인스턴스화를 하면 구체적인 아래의 문장이 된다.

**뭐? 치킨? 치킨?? 그 돈 주고 치킨 먹을 바에야 뜨끈한 국밥 4그릇 든든~히 먹고 말지**

**인스턴스 초기화**

초기화 블록과 생성자 |  
두 개 모두 인스턴스화를 할 때 실행된다.  
만약 클래스 내부에서 정해준 값으로 초기화를 진행할 것이라면, 초기화블록이나 매개변수가 없는 기본 생성자를 사용하면 된다.  
클래스 외부에서 값을 받아서 초기화를 할 것이라면, 외부와 연결(매개)해주는 매개변수가 있는 생성자를 사용해서 초기화를 하면 된다.

국밥드리퍼 클래스는 초기화를 해주지 않으면, dreep()메서드를 호출 할 때 오류가 발생함으로 반드시 초기화를 해줘야 하는데, 사용자가 입력한 음식과 가격을 계산해 드립 치는 것을 목적으로 만들었음으로, 매개변수가 있는 생성자를 통해 초기화하도록 해주는 것이 좋다.

**매개변수**

매개는 둘을 이어주는 것이다.  
디버그로 소스코드 읽는 순서 확인  
메소드 안의 코드를 복사해서 호출한 메서드쪽으로 붙여 넣기.  
오류가 안 나도록 수정.  
메서드의 매개변수라는 것이, 오류가 값을 다시 담아주는 역할을 하고 있었다.  
참조변수일 경우 얕은 복사가 일어나게 된다.

**반환형**

메서드 실행이 끝나고 반환되는 값이 있을 때, 어떤 자료형으로 값을 반환할 것인지 알려주는 것. 메서드가 반환해준 값을 받아서 사용하기 위해서는 반환되는 타입을 알고 있어야 알맞은 변수를 선언해서 사용할 수 있다.

**예약어**

static : 프로그램이 실행될 때 static영역에 올라가는 함수. 인스턴스화는 여러 번 될 수 있지만, 프로그램은 한 번 실행된다. 한번 올라간 변수나 메서드를 모든 인스턴스들이 공유하는 형식이다. 인스턴스 뿐만 아니라 쓰래드들도 같이 공유한다.

**접근제한자**

접근제한자는 객체의 자율성을 결정하는 아주 중요한 키워드. 객체지향설계에서 절대 빼놓을 수 없는 부분이다.

여러분들로 생각을 해보자.  
여러분들의 개인적인 활동들 중에는, 남이 시켜서 하는 게 아닌 것들이 있다. 여러분들의 취미활동을 남이 시켜서 한다면 이상하지 않냐. 넌 취미활동으로 등산해! 이거 좀 이상하다. 아니면 남이 시킨 일을 하더라도, 그 일을 진행하는 과정에서는 여러분 스타일로 여러분들만의 방식으로 처리할 수도 있을 거다.

아니면 가까운 사람이라면 그 일을 부탁할 수도 있겠지만 완전히 타인인 사람은 여러분에게 부탁 할 수 없는 일들도 있다. ‘집에 올 때 두부 한 모 사와라’ 이런 부탁은 여러분들 가족이나 정말 친한, 사랑하는 사람이라면 여러분께 시킬 수 있겠지만 제가 여러분께 ‘내일 수업 올 때 두부 한 모 사오세요.’ 이렇게 말하면 그건 진짜 이상하지 않냐.

마지막으로 여러분의 위치에 따라서 누구나 여러분에게 시킬 수 있는 일이 있다. 제가 지금은 강사니까, 누구나 와서 모르는 것을 물어볼 수 있다.

제일 처음이 private, 두 번째가 default, protected 마지막이 public이다. 만약 모든 일을, 취미활동이나 자잘한 심부름까지도 모두 public으로 해야 하는 존재가 있다면 그 존재는 자율성이 1도 없는 존재일 거다. 노예다 노예.

객체지향프로그래밍설계 방법 중에서 객체는 자율성을 가진 존재라는 말이 있다.  
이것은 생각해보면 역설이다. 객체라는 것은, 객체가 가진 메서드는 어디에선가 호출되어야 작동한다. 즉 다른 누군가가, 일 해! 라고 시켜야, 호출 해줘야 메서드가 실행되게 된다. 누군가 시켜야만 일을 할 수 있는 존재가 자율성을 가진 존재일 수 가 있나? 역설처럼 들린다.

여기서 말하는 자율성이라는 것은, 일을 시켰을 때 그 일을 어떻게 할지는 객체가 정할 수 있게 해주라는 말이다. 그렇게 하기 위해서는 모든 메서드에 다른 객체가 접근할 수 있어서는 안된다. 메서드에 접근 가능하다는 것은 메서드를 호출 할 수 있다는 뜻이고 즉 메서드에 구현되어 있는 동작을 시킬 수 있다는 뜻이 된다. 접근제한자를 통해 다른 객체가 접근 가능한 메서드와 접근 불가능한 메서드를 구분 지어줘야 한다.

그래서 public메서드와 private메서드로 나누어서 캡슐화를 진행했다.  
일단 private로 캡슐화를 진행하지 않았을 때 그림이다.  
만약 이 그림처럼 모든 메서드가 public이고 그래서 메서드를 호출하는 쪽에서 모든 메서드를 호출하는 방식이 된다면 첫 번째로 역할을 수행하는 대에 있어서 자율적이지 못하게 된다. 역할을 하기 위해 어떤 기능들을 수행할지를 모두 조장객체가 선택하기 때문이다.

두 번째로 자료조사 클래스를 수정하기가 힘들다. 어떤 메서드를 건드려도 조장클래스가 영향을 받기 때문이다. 코드를 수정하는 부분에 있어서 자율적이 못하게 된다. 더불어 유지보수도 엄청 힘들어진다. 코드를 수정한 영향이 어디까지 퍼질지 예측이 안되기 때문이다.

세 번째 시스템의 복잡도가 올라가게 된다. 최대한 단순하게 시스템을 설계해도, 복잡한 현실을 그대로 담아낸다는 것이 굉장히 힘든 일인데, 처음부터 이렇게 복잡하게 작성하면 금방 한계에 부딪히게 된다.

마지막으로 이렇게 설계하면 다른 개발자가 가져다 쓰기가 힘들다. 여러분들이 프로젝트만 해도, 다른 학생이 짠 클래스나 메서드를 호출 할 일이 많다. 그런데 여러분들이, 포트폴리오용 프로젝트보다 규모가 훨씬 큰 회사 프로젝트를 진행하면서 객체를 이런 식으로 만들어 놓으면, 아마 여러분이 만든 클래스를 사용하던 여러분들의 선배가 불러서 물어보지 않겠냐. 도대체 왜 이렇게 설계했니. 왜 이렇게 객체를 자율성 없게 만들어서, 내가 뭐 하나 하려고 할 때마다 A to Z로 다 시켜야 되게 만들었니. 나 바쁜데. 그냥 자료준비 메서드를 호출하면 내부적으로 자료조사 기능을 하게끔 자율적으로 좀 스마트하게 코드를 짜주면 안되었니? 하고 물어 볼 거다.

그래서 캡슐화를 진행한다. 다른 객체가 요청을 보내는 public 메서드에는 기능을 구현하는 코드를 넣지 않는다. 자료조사 객체가 가지고 있는 기능 중에 어떤 기능을 활용해서 요청 받은 일을 처리할지 결정하는 코드를 작성 해둔다.

기능을 구현하는 코드는 모두 private으로 선언해 외부에서 호출할 수 없게끔 작성한다. 객체를 부르는 대상은 자료조사라는 임무를 어떤 로직으로 수행하는지 알 수 없다. 자료조사객체가 알아서 처리해버리기 때문이다.

이렇게 하면 단 한번의 메시지, 요청만 발생하기 때문에 딱 봐도 시스템의 복잡도가 확 내려가게 된다. 더 복잡한 현실의 모습을 프로그래밍할 수 있게 된다.

여러분들이 나중에 스프링을 하시게 되면, 여러분들이 url로 사이트에 접근했을 때 디스페처서블릿이라는 친구가 어떤 컨트롤러로 보낼지를 결정하게 된다. 그리고 그 컨트롤러는 다시 어떤 기능, 즉 모델클래스를 호출할 지 결정하게 된다. 이 그림에서 확장될 뿐 크게 다르지 않다. 어차피 원리는 객체지향적설계에 있기 때문이다.

한가지 더 중요한 것은, 자료조사 클래스가 기능구현 메서드를 감춰버렸기 때문에 외부에서 보았을 때 자료조사 클래스가 이전보다 덜 구체적이게 보이게 된다. 또한 이 클래스를 다룰 때도 덜 구체적으로 다루게 된다. 다시 말해 추상적으로 보이게 되고, 추상적으로 다루게 된다. 추상화라는 것은 마치 객체처럼, 정의가 딱 떨어지는 것이 아니다.

오버로딩도 마찬가지이다. 하나의 이름으로 다양한 메소드를 만든다는 것. 메서드이름이 정확하게 한 기능을 가리켜야 하는 게 아니라 그 이름이 사람에게 전해주는, 맥락에만 맞으면 내부 기능구현은 어떻게 되어도 상관없다, 추상적이어도 괜찮다는 마인드가 객체지향에 있기 때문에 오버로딩을 지원해주는 것이다.

한 가지 분명한 것은 추상화 할 수록 시스템은 유연해진다는 것. 객체지향의 목표인 유연성의 확보에 더 다가갈 수 있게 된다는 것이다. 그래서 객체지향프로그래밍에서는 다양한 방식으로 추상화를 달성하려고 한다. 상속이 그렇고, 다형성이 그렇다.

이번 주에 객체가 무엇인지, 추상화와 캡슐화가 무엇인지 배웠습니다. 그리고 그것들이 코드로 어떻게 이루어지는 지도 같이 클래스를 만들면서 진행하였습니다. 코드보다 중요한 것이 개념입니다. 코드는 여러분이 추구하는 가치에 따라 계속해서 변하기 때문입니다. 여러분이 유연성에 중점을 둘 지, 부작용의 최소화에 중점을 둘 지, 생산성에 중점을 둘 지에 따라 코드 작성방법은 계속 바뀌게 될 겁니다.

바뀌지 않는 것은, 첫 시간과 오늘 말씀 드린 내용들입니다. 잘 기억해두시고, 앞으로 코드를 작성하시는 대에도 많은 도움이 되면 좋겠습니다. 내일은 객체배열이어서 사실상 오늘이 객체지향파트1 마지막 날이어서 말이 좀 길었습니다. 객체지향이라는 게 뭔지 사실 저도 잘 모릅니다. 많은 개발자들이 개발을 손에서 때는 그 날까지 고민하는 게 객체지향입니다. 그리고 그렇게 고민하는 개발자가 개발도 잘 하구요. 이번 4일을 통해서 여러분들이 객체지향이라는 게 뭔지 감이라도 잡으면 그것으로 충분할 것 같습니다.

**추상클래스**

추상클래스는 추상적인 클래스다.

우리 객체를 추상화해서 클래스를 만들 때를 생각해보자. 현실의 객체에서 추상화를 통해서 공통적인 속성을 뽑아낸다. (사람 클래스 예시: 이름, 성별) 그렇게 뽑아낸 속성에 여러분들이 어떤 값을 넣느냐에 따라 다른 객체가 만들어진다.

추상클래스는 클래스를 추상화 한 것이다.

추상클래스에는 아직 구현이 덜 된 추상메서드가 존재한다. 이 추상메서드라는 것은 추상클래스를 상속받는 모든 클래스들이 공통적으로 가지고 있지만, 내용 즉 값을 채워지지 않은 미완성 메서드이다. 우리가 객체를 추상화한 클래스의 속성에 값을 집어 넣으면 그때 객체가 되는 것처럼 클래스를 추상화한 추상클래스의 추상메서드에 구현부를 작성해주면 그때 클래스가 된다.

추상클래스는 아직 클래스가 아니기 때문에 객체를 생성할 수 없다. 즉 new 연산자를 활용할 수 없다. 하지만 상속은 가능한 만큼 다형성은 가능하다. 즉 추상클래스의 타입으로 추상클래스의 자식클래스들을 다룰 수 있다.

다형성 실습파일 -> 처음에 우리가 다형성을 배울 때 메서드오버라이딩을 활용해야겠다는 생각을 미처 하지 못해서, 부모타입으로 자식객체에 각각 있는 printchild1, printchild2 메서드를 사용하기 위해 instanceof 연산자를 활용해서 조건문으로 다뤄야 했다. 만약 누가 부모클래스에 있는 어떤 메서드를 오버라이딩 하라고 가이드를 해줬으면, 우리가 그렇게 할 필요가 없었을 것이다. 그럴 때 추상메서드로 만들어주면 나중에 이 parent클래스를 상속받을 사람들에게 가이드를 해줄 수 있게 된다.

실제로 추상클래스를 활용할 때는 보통 라이브러리를 만들 때 많이 쓴다. 여러분들이 전 세계에서 가장 빨리 정보를 습득할 수 있는 라이브러리를 만들었다고 생각해보자. 정보라는 것은 매우 중요하니까 데이터가 암호화해서 사용자에게 제공한다. 만약 사용자가 이 데이터를 보고 싶다면 자신 나라의 언어로 번역해야만 한다. 물론 200여개의 언어로 다 번역해주는 메서드를 만들 수도 있겠지만, 그건 또 너무 번거롭고, 혹은 사용자가 언어로 번역 하는 게 아니라 다시 자신들 만의 암호로 데이터를 인코딩하고 싶을 수도 있을 것이다. 이럴 때 여러분들은 암호화 해서 정보를 보내주는 것 까지만 클래스를 완성시켜주고, 그 데이터를 어떻게 가공할 지를 결정하는 메서드는 추상메서드로 만들면 된다. 그러면 사람들이 여러분들의 라이브러리를 가져가서 자기들 입맛에 맞게끔 바꿔서 사용할 거다.

자바 11 api -> abstract list

**인터페이스**

인터페이스란 뭐냐. 일단 인터페이스 라는 게 두 가지 정도로 사용이 된다.

사람의 클래스를 상속받는 KH강사 클래스와 KH학생 클래스가 있다. 그리고 이 강사 클래스와 학생 클래스는 모두 자바를 할 줄 안다. 자식 클래스가 모두 자바 라는 기능이 있으니까 부모클래스에 자바라는 메서드를 넣으면 좋겠...는데 사람이 자바를 할 줄 안다? 이거는 좀 이상하다. 그래서 프로그래밍언어라는 인터페이스를 만든 다음에 자바라는 메서드를 넣어뒀다. 그 다음에, 이 인터페이스를 implements 하는 거다.

클래스는 명사다. 사람, 주문, 선물, 상품

인터페이스는 형용사에 가깝다. 자바를 할 줄 아는, 반복 가능한, 비교할 수 있는, 인터페이스와 클래스를 동시에 상속받고 있다면 ~하는 객체라고 해석하면 얼추 맞다. 그래서 인터페이스이름은 able로 끝나는 애들이 많다.

자 다사 자바 11 api를 보자

인터페이스는 추상메서드로만 이루어진 추상클래스의 변형체이다.

추상메서드만으로 이루어져있다는 이야기는 메서드구현부는 하나도 작성이 안되어 있다는 뜻이다. 그럼 이것을 도대체 왜 쓰느냐, 어제 실습할 때를 생각해보자.

일단 메서드 선언부가 작성했다. 그 다음에 manager클래스와 menu클래스 중에서 menu클래스를 먼저 작성했다. manager 클래스가 작성되지 않았음에도, 완성되었다고 가정하고 실습을 진행할 수 있었다. 왜냐면 메서드 선언부는 이미 완성되어 있었으니까.

이렇게 되면 분업이 가능해진다. 어제는 여러분들이 혼자 작업했지만, 만약 두 명이 같이 과제를 수행했다고 생각을 해보자. 그랬다면 여러분들이 menu를 만드는 동안 같이 일하는 동료가 manager클래스를 만들고 있었을 것이다.

인터페이스가 바로 이런 역할을 해준다. 일단 인터페이스만 빨리 완성시킨 다음에, 그 인터페이스를 impleaments 받아서 여러 명의 개발자가 분업으로 작업을 진행하게 된다.

우리가 실제로 사용할 때는, 유지보수 쪽에서, 이미 완성된 시스템에 기능을 추가해야 할 경우에는 코드를 재활용할 수 있는 상속이 적당할 거다. 여러분들이 많은 사람들이 사용할 라이브러리를 만든다면, 여러분들의 라이브러리를 활용할 사람들이 자기 입맛에 맞게 사용할 수 있도록 추상클래스를 활용할 수 있을 거다. 마지막으로 프로젝트를 시작하는 단계라면, 인터페이스를 통해서 빠르게 메서드 선언부를 그려놓고, 분업으로 작업을 진행할 수 있을 거다.