객체지향 시작하기

객체 지향 입문 용어

객체와 인스턴스

객체와 인스턴스

객체

3. 지급재료목록		자격종목	제빵기능사		
일련번호	재료명	규격	단위	수량	비고
1	밀가루	강력분	g	850	1인용
2	밀가루	중력분	g	360	1인용
3	설탕	정백당	g	121	1인용
4	소금	정제염	g	22	1인용
5	이스트	생이스트	g	60	1인용
6	제빵개량제	제빵용	g	25	1인용
7	마가린	제과제빵용	g	110	1인용
8	탈지분유	제과제빵용	g	36	1인용
9	달걀	60g(껍질포함)	개	3	1인용
10	식용유	대두유	ml	50	1인용
11	얼음	식용	g	200	1인용(겨울철제외)
12	위생지	식품용(8절지)	장	10	1인용
13	제품상자	제품포장용	개	1	5인 공용

인스턴스

객체와 인스턴스

코드상에서 부가설명

객체지향 용어

책임, 의존, 위임

Responsibility(책임)

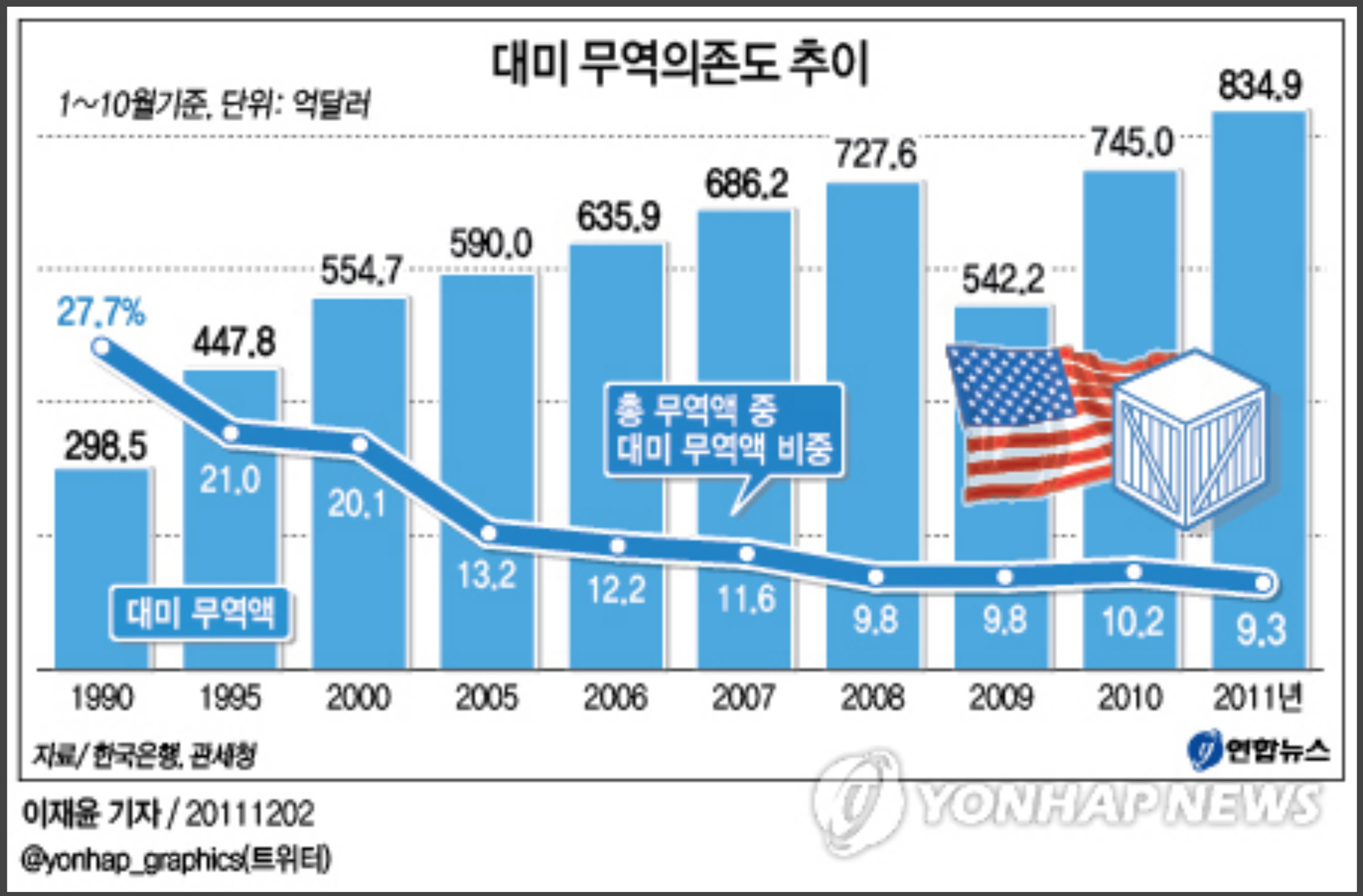
책임은 클래스의 목적

객체지향 용어

Dependency(의존)

의존은 클래스관의 관계

객체지향 용어



이재윤 기자 / 20111202

@yonhap_graphics(트위터)

객체지향 용어

만약 내일 트럼프가 밀수출을 철회하면?

객체지향 용어

코드 상에서 이해하기

가장 중요한 위임

위임은 책임을 전가하는 것

그러나 아직 위임을 배우기엔

이릅니다. 한 시간뒤에 보죠

객체지향 용어

객체 지향

본격적으로 시작하기

상속은 특징과 기능을 물려받는 것

나의 조상은

호모사피엔스

호모사피엔스의 특징

사냥을함

빗살무늬토기를 만들수있음

고기를 불에 잘익혀먹음

상속과 다형성

내일부터 호모사피엔스로

살수 있을까?

상속과 다형성

주옥 같을 뿐 할 수 있다

안녕.. 카톡,롤, 옵치

즐거웠어

뱀의 특징

탈피를 한다

온도조절이가능하다

내일부터 뱀으로

살수 있을까?

아 이건

사람불러야겠는데;

상속과 다형성

코드 상에서 보기

상속은 특징과 기능을 물려받는 것

그럼과 동시에

특징과 기능을 받았음으로

해당 객체로 '간주 될 수 있다'.

상속을 받지 않았다면

특징과 기능을 받지 못했기

‘간주 될 수 없다’

다형성

같은 identifer를 가져도

상황에 따라 다르게 실행됨

상속과 다형성

아까 코드 한번 더보기

다형성

같은 행위를 해도

상황(문맥)에 따라 다르게 실행됨

그럼

상황,문맥(Context)

이 뭐냐?

애내가 상황

동일한 행위

```
YoungJoon youngJoon = new YoungJoon();
HomoSapience homoYoungjoon = new YoungJoon();
```

상속과 다형성

파인트

100g

하프갤런

400g

아이스크림 많아봐야

‘그릇’ 까지 밖에 못담는다

상속과 다형성

파인트

100g

하프갤런

400g

상속과 다형성

호모사피엔스

사냥, 빗살무늬토기,
고기구워먹기

영준(현대인류)

사냥,빗살무늬토기,
고기구워먹기, 카톡하기,
코딩하기

상속과 다형성

사냥,빗살무늬토기,
고기구워먹기, 카톡하기,
코딩하기

캭아앜! 퉤!

사냥,빗살무늬토기,
고기구워먹기

코딩하기
카톡하기

결국 그릇에 맞는

특성과 기능만 가져가고

나머지는 쓰레기통으로 보냄

그릇
문맥(상황)
간주

이 세개만 기억나도(이해x)

성공한겁니다

(계속하면서 자세히 봅시다)

개발용어

캐스팅

캐스팅이란, 해당 객체를

다른 객체로 '간주' 하는 행위

캐스팅이란, 해당 객체를

다른 객체로 '간주' 하는 행위

```
YoungJoon youngJoon = new YoungJoon();
HomoSapience homoYoungjoon = new YoungJoon();
```

YoungJoon 타입이 HomoSpience 타입으로 간주됨

캐스팅의 분류

상위객체로 넘어가기 (업캐스팅)
하위객체로 넘어가기 (다운캐스팅)
캐스팅을 시스템에 맡김(암묵적 캐스팅)
캐스팅을 직접 컨트롤함(명시적 캐스팅)

상속과 다형성

코드에서 직접 확인하기

업캐스팅(UpCasting)

메모리 낭비를 줄임

명확한 기능 제공(*)

다운캐스팅(DownCasting)

하위 객체의 기능이 필요한 경우

즉 기능을 늘리기 위해함

명시적 캐스팅

(Homo)youngjoon 처럼

바꿀 타입앞에 (변환할타입)을

써줌으로써 명시적으로 캐스팅함을 표시

암묵적 캐스팅

가장 중요함.

시스템이 알아서 변환 하도록 함.

모든 사회악의 근원(고슬링..당신은 대체..)

개발이 어려운 이유

드디어 위임!

그전에 보고갈 용어

시그니쳐

함수/메서드/생성자 에 있어서

해당함수의 설계를 담당하는 부분

[Identifier(name) + parameter + return]

개발용어

시그니쳐?

시그니처 : 이 기능이 뭐하는 놈인지 간략하게 설명함

```
boolean sayMyName(String name){
    if(name != "") {
        System.out.println("맼섬노잊!");
        System.out.println("새마넴!");
        System.out.println("꺄아아아아앜" + name + "!!!!");
        return true;
    }else{
        return false;
    }
}
```

이 부분은 로직, 돌아가는 동작

개발용어

```
boolean sayMyName(String name);
```

indentifer : sayMyName
parameter : String
return : boolean

개발용어

```
boolean sayMyName(String name);
```

그냥 중괄호를 다지우면 됩니다.
참 쉽죠?

개발용어

컴파일 타임과 런타임

(제대로하려면 토나오니까
그냥 우리한테 필요한 부분만봅시다)

개발용어

컴파일 타임 = 시그니쳐만 검사
런타임 = 로직을 포함해서 실행시킴

개발용어

시그니쳐가 잘못되면 실행도안됨 (컴파일타임)

로직이 잘못하면 실행은 되는데 실행하다 꺼지거나, 작동하다 꺼짐 (런타임)

그럼 시그니쳐만 적으면

컴파일타임은 넘기겠구나!

거기에 로직을 나중에 구현해주면

실행까지 되겠구나!

개발용어

그래서 나온게

인.터.페.이.스

(Swift = 프.로.토.콜)

개발자 평균수명 감축의 근원;;

개발용어

인터페이스의 핵심

시그니쳐만 적고

로직은 딴대서 구현해라

이것이 바로 위임(로직을 짬때림)!

+ 추상화 (시그니쳐만으로 프로그래밍)

코드가 유연해지고 의존성이 낮아짐

개발용어

그럼 안드로이드, iOS가

어떻게 만들어졌는지 해볼까요

Button 클래스 직접만들어보기

개발용어

인터페이스의 두 가지 구현 방식

1. 인터페이스를 상속받아서 this로 넘기기

2. 익명객체 활용하기

개발용어

코드에서 봅시다

개발용어

권장사항

안드로이드 -> 익명객체(OOP)

iOS -> 상속 리턴(POP)

개발용어

이제

Android와 iOS에서

여러분이 지금껏 작성했지만

몰랐던 것을 실습해봅시다

객체지향 설계원칙

SOLID

객체지향설계원칙

객체지향 설계원칙이 왜 필요한데?

코드를 유연하게 만드는법

클라이언트(사장님,외주원청)에 대한 고찰

그들은 개발에 대해 얼마나 알고 있을까?

알면 지들이 했겠지..

왜 시키겠어요..

햐.. 이거아닌데?

고로 수정이 오지게 많다

->

코드가 유연하지 못하면 수명이 단축

객체지향설계원칙

단일 책임의 원칙(Single Responsibilty)

한 클래스에는 하나의 책임만 넣는다

객체지향설계원칙

단일 책임의 원칙(Single Responsibilty)

한 클래스에는 하나의 책임만 넣는다

객체지향설계원칙

단일 책임의 원칙 지키는법

하나에 클래스에 짱박지말고

클래스를 더 만들어서 스플리팅하자

객체지향설계원칙

개방 폐쇄의 원칙(Open-Closed)

기능 확장에는 열려있고

기능 수정에는 닫혀있어야한다.

개방 폐쇄의 원칙 지키는법

직접 데이터를 넘기지 않고

데이터를 넘기는 기능을 만든다.

객체지향설계원칙

리스코프 치환의 원칙(Liskov Subsititution)

상속관계에 있어서, 부모 클래스의 인스턴스가 사용된 자리에 자식 클래스를 넣어도 작동해야함

리스코프 치환의 원칙 지키기

지키기 어렵다.

Android / iOS 는 순수 코딩이아닌

프레임워크단이기때문에..

객체지향설계원칙

인터페이스 분할 원칙(Interface Segregation)

사용하지 않는 인터페이스가 없어야 된다.

인터페이스 분할 원칙 지키기

안쓰는 시그니쳐 지우고

가능한 인터페이스를 더만들자

객체지향설계원칙

의존역전의 원칙(Dependency Inversion)

고수준 모듈이 저수준 모듈의 구현에 의존해선 안된다

저수준 모듈이 고수준 모듈의 추상타입을 의존해야한다

객체지향설계원칙

모바일 프로그래밍 버전

ViewController / Activity / Fragments 가

다른 라이브러리 / 객체를 참조해야한다

객체지향설계원칙

의존역전의 원칙 지키기

아키텍쳐 패턴 지키기

iOS - MVC
Web - MVC
Android - MVP
ReactiveX - MVVM

객체지향설계원칙

의존역전의 원칙 지키기

의존성 주입 라이브러리(Dependency Injection)

안드로이드 (Dagger2)

iOS (Typoon, Cleanse, Swinject)

객체지향설계원칙

안드로이드는 세가지

단일책임, 개방폐쇄, 인터페이스 분할

iOS는 두가지

단일책임, 개방폐쇄 (POP는 조금 다름!)

끄