Day11. Objective-C_4

전미정

Q1. Objective-C Format Specifier 조사

A1. OS-X는 32비트와 64비트 시스템에 상관없이 동일한 서비스를 제공하기 위해 NSInteger, NSUInteger, CGFloat, and CFIndex 등의 데이터 형식을 사용한다. 하지만 이런 데이터 형식은 하드웨어 사양에 따라 다르게 정의되어 출력시 문제가 있다. 32비트 환경에서 NSInteger와 NSUInteger는 각각 int와 unsigned int로 정의되고, 64비트 환경에서는 각각 long과 unsigned long으로 정의된다. 시스템에 따라 달라지는 출력 형식자 변환을 피하기 위해 아래와 같은 형식자를 사용한다.

Type	Format Specifier
NSInteger	%ld or %lx
NSUInteger	%lu or %lx
CGFloat	%f or %g
CFIndex	%ld or %lx
Pointer	%p or %zx

```
// In the precompiled header file:
#if __LP64__
#define NSI "ld"
#define NSU "lu"
#else
#define NSI "d"
#define NSU "u"
#endif
```

**p.s. : 변수타입을 나타내는 문구의 NS는 NextStep에서 따온 것이다.

Q2. 절차지향과 객체지향의 차이점

A2. 절차지향 프로그래밍이란? 구조적 프로그래밍이라고도 불리는 절차지향 프로그래밍은 단어 그대로 명령어의 순차적인 처리가 중요한 프로그래밍 기법으로, 대표적인 절차지향 언어에는 C언어가 있다. 컴퓨터의 발전 초반에 탄생한 절차지향 언어는 컴퓨터의 작업 처리 방식과 유사하며, 사용자가 컴퓨터에게 수행할 명령을 하나하나 순서대로(Step by step)으로 입력해줘야 하는 것을 의미한다. 이것을 컴퓨터 입장에서 보면 명령어를 시작점부터 차례대로 읽어내려가다 함수를 만나면 함수로 들어가 작업을 수행하는 방식을 의미한다. 이러한 방식에서는 함수의 순서와, 알고리즘 방식이 매우 중요한 역할을 한다.

객체지향 프로그래밍의 탄생 배경은? 절차지향 방식은 초기 컴퓨터 구동방식에 적합해 소프트웨어 개발에 널리 사용되었다. 하지만 하드웨어가 급속한 속도로 발전하고, 컴퓨터의 보급과 활용이 다양해지면서 소프트웨어가 그 속도를 따라가지 못하는 문제가 발생하였다(소프트웨어의 위기). 이러한 상황에서 절차지향 프로그래밍은 대규모 소프트웨어에서 유지 보수등의 한계가 드러났고, 코드가 재사용되는 경우가 드물어 과다한 개발 기간과 인력을 소모하게 되었다. 이러한 문제점을 해결하기 위해 새로운 개발 기법이 등장하였는데, 그 중에서 가장 탁월한 방법으로 인정받은 것이 바로 객체지향 프로그래밍 방식이다.

객체지향 프로그래밍이란? 컴퓨터의 입장에 서서 컴퓨터의 논리 흐름에 따라 프로그램을 작성하는 절차지향 프로그래밍 방식을 벗어나 인간의 사고 방식에 가까운 방식을 채택한 것이 바로 객체지향 프로그래밍이다. 인간지향적인 객체지향

프로그래밍은 문제의 핵심인 데이터에 속성을 정의하고, 절차를 결합해 하나의 덩어리로 묶어서 생각하는 것으로, 객체들 간에 서로 메세지를 주고받으며 상호작용을 할 수 있다. 이러한 방식은 여러가지 작은 부품으로 컴퓨터를 조립하는 것으로 비유할 수 있으며, 각각 부품인 객체를 독립적으로 수정, 보완 할 수 있어 유지보수가 용이하다.

객체지향 프로그래밍과 절차지향 프로그램의 장단점

	객체지향 프로그래밍	절차지향 프로그래밍
장점	코드의 재활용 가능 유지보수 및 업그레이드 용이 대형 프로젝트에 적합 분석 및 설계 용이 캡슐화, 다형성, 상속	실행처리 속도가 빠름 컴퓨터의 작업 처리 구조와 비슷함
단점	처리속도가 다소 느림 프로그램 용량이 커짐 설계에 많은 시간 소요	유지보수가 어려움 순서가 바뀌면 결과값 도출이 어려움 프로그램 분석이 어려움 대형 프로젝트에 부적합

Q3. 클래스 vs 객체 vs 인스턴스 차이점

A3. 클래스란? 클래스란 문제해결을 하기 위해 비슷한 기능을 지닌 변수와 메소드의 집합이라고 할 수 있으며, 소프트웨어적인 설계도 또는 프로토타입이라 불린다. 프로그램에서 실제로 역할을 하는 객체의 상태정보와 행동에 대해 추상화 시켜 좋은 것이며 자동차로 예를 들면 실제 여러가지 자동차들(아반떼, 그랜저, 버스, 체어맨)이 공통적으로 지니는 속성을 일반화 시켜 놓은 설계도로 비교할 수 있다.

인스턴스란? 객체의 정보를 가지고있는 클래스가 실제 사용 될 수 있는 객체가 되기 위해서는 실제화라는 과정이 필요하다. 즉 클래스에 대한 변수를 선언하는 과정이 필요한데 이것이 바로 인스턴스화이다. 이는 클래스를 특정 메모리에 담고 그 메모리를 불러들일 수 있는 메모리 주소값을 변수에 저장함으로써 클래스를 이용 할 수 있게 하는 것으로, 인스턴스는 실제 메모리상에 할당된 것을 의미한다.

객체란? 객체란 클래스를 통해 실제화된 것을 의미한다. 자동차를 예를 들면 자동차의 설계도(클래스)를 통해 만들어진 실제 자동차가 객체라고 할 수 있다. 객체는 자신 고유의 속성을 가지며, 클래스에서 정의한 행위(메서드)를 수행 할 수 있다. 하나의 클래스를 바탕으로 서로 다른 행동을 하는 여러개의 객체 생성이 가능하며, 이 점이 객체지향 프로그래밍에서의 코드 재활용을 가능케한다. 만일 객체를 작은 프로그램이라고 생각하면 여러개의 객체를 레고 블럭처럼 쌓아서 대규모 프로그램을 만드는 일이 가능한데 이것 역시 객체지향 프로그래밍의 장점이다.

```
String str;
str = new String("Hello world");
System.out.println(str);
```

왼쪽의 코드(비록 자바 코드지만;)를 통해 객체와 인스턴스를 구별해보자.

먼저 첫 줄은 String이라는 클래스를 사용해 str이라는 객체(변수)를 선언한 것이다. 아직 str에는 문자열이 할당되어 있지 않은 상태이다. 두번째 라인을 보면 new 키워드를 사용해 str변수에 데이터가 생성되었다. 이렇게 객체를 실제 메모리에 할당해 클래스와 연결하는 과정을 인스턴스화라고 한다. 객체 str에 "Hello World"라는 문자열이 메모리에 할당 된 것을 인스턴스라고 한다.