**객체지향 프로그래밍** | 최윤정 교수님 | Summary #06 이서현 2276219  
[Chapter 04] 객체지향 프로그래밍 기본개념 클래스와 객체

내용 : OOP 개념, 클래스와 객체와 구성, 생성자(객체의 생성과 참조), this 키워드, 객체의 치환/소멸/객체배열, 메소드의 인자전달(기본변수와 배열, 객체변수. 49페이지), 메소드 오버로딩, 접근지정자 4가지(public, private, default, protected(상속전용)), static/final, 싱글톤

객체 - 세상 모든 것이 객체. 실세계 객체의 특징: 객체마다 고유한 특성(state)와 행동(behavior) 가짐, 다른 객체들과 상호작용.

객체지향 프로그래밍 OOP(Object-Oriented Programming Language)

부품 객체를 먼저 만들고 하나씩 조립해 가는 프로그램 기법. / 객체: **데이터(필드)+기능(메소드)**

3R: Readability, Reusability, Reliability

OOP의 목적: 소프트웨어의 생산성 향상, 실세계에 대한 쉬운 모델링. 요구사항분석을 바탕으로 객체의 표현 및 상호작용 묘사가 용이.

객체지향 프로그램이 갖춰야 할 속성: 상속성(Inheritance) 은닉성(Information Hiding, Encapsulation) 다형성(Polymorphism)

절차지향 프로그래밍과 객체지향 프로그래밍

* 절차지향: 작업 순서 파악 후 표현, 각 작업을 함수로 작성(모듈화). 보통 순서도를 사용하여 절차를 표현하고 구현.
* 객체지향: 객체 파악 후 설계하고 객체 간의 상호작용을 파악하여 표현. 여러 다이어그램 이용해 구조와 순서를 표현. 메소드 작성 시 독립적인 기능별로 분리하며(모듈화는 항상 필요) 이때 일련의 동작에 관한 작업, 논리는 기존 절차지향적 접근을 사용.

객체지향의 특성 - (1)캡슐화 (2)상속 (3)다형성

(1)캡슐화

객체의 필드, 메소드를 하나로 묶고 실제 구현 내용을 감추는 것. (보여줄 필요 없는 부분 은닉)

외부 객체는 객체 내부 구조를 알지 못하며 객체가 노출해 제공하는 필드와 메소드만 이용이 가능하다.

* 접근 제한자 사용: public, protected, private (private 속성을 공유할 때는 getter/setter를 구현.)

필드와 메소드를 캡슐화하여 보호하는 이유: 외부의 잘못된 사용으로 객체가 손상되지 않도록. 캡슐화된 멤버들의 외부 노출 유무 결정위해.

* 클래스(class): 객체 모양을 선언한 틀. 메소드(멤버 함수)와 필드\*(멤버 변수)는 모두 클래스 내에 구현한다.  
  클래스를 만들면 클래스 모양으로 객체를 만들게 될 수 있다.  
  \* 필드(field): 클래스에 포함되어 객체의 속성을 정의하는 변수. (클래스 변수/인스턴스 변수)
* 객체: 클래스의 모양대로 생성된 실체(인스턴스 instance). 객체 내부 데이터를 보호하고 외부 접근을 제한한다.  
  객체 외부에서는 비공개 멤버(필드, 메소드)에 직접 접근 불가, 접근이 필요하면 공개된 메소드를 통해 비공개 멤버에 접근하도록 만든다. (private 적극 권장! + getter + setter 메소드를 사용한다.)

(2)상속

기존에 만들어놓은 클래스의 기능을 그대로 물려받아서 사용하는 것.

여러 객체를 설계하면서 일반화개념이 발견되었거나 새로운 .객체를 정의할 때 기존 설계도를 재사용하는 용도로 사용한다.

하위 객체는 상위에서 허용한 모든 멤버를 물려받아 사용하며 상위에서 정의되지 않은 새로운 멤버를 추가하거나 확장하고 수정한다.

하위 객체가 상위에서 물려받은 메소드를 재정의할 수 있다.(🡪오버라이딩)

* 상속 조건: 일반화가 가능해야 한다  
  ‘-이다’의 관계가 성립되고 타당할 때만 사용한다. (Is-A, Is-Kind-Of) (ex. 강아지는 동물이다.)  
  상속대상은 상위 개체에서 허용한 멤버(속성과 일반 메소드)들이다. 생성자는 상속되지 않는다. (대신 자식 클래스로 인스턴스를 생성할 때 부모 클래스의 기본 생성자를 자동으로 호출하게 된다.)

**자바의 상속 유의점**

* 자바에서 클래스 상속은 단일 상속만 허용 (extends ~ 뒤에 하나만 가능)   
  하위 클래스가 상위 클래스가 허용한 멤버(속성, 메소드)들을 물려받고 기능 확장  
  (상위 클래스=부모=슈퍼 클래스: 하위 클래스들의 일반화. /하위 클래스=자식=서브 클래스: 수퍼 클래스를 재사용하고 새로운 특성 추가)
* 기능을 명시하는 인터페이스는 다중 상속을 허용(implements ~, ~…

상속의 효과

* 여러 개의 하위 객체를 일반화시켜 참조할 수 있음
* 기존에 만들어진 상위 객체를 재사용해서 하위 객체를 빠르게 개발 (반복 코드 적음, 유지 보수 편리성, 객체의 다형성 구현)  
  상위객체 = 필드1+메소드1 | 상속된 하위객체 = 필드1+메소드1(물려받음) + 필드2+메소드2(추가 데이터 및 기능)

(3) 다형성

다형성(Polymorphism): 같은 이름의 메소드가 클래스나 객체에 따라 다르게 동작하도록 구현하는 것.

다형성 사례: 메소드 오버로딩, 메소드 오버라이딩, 인터페이스, 동적 바인딩

* 메소드 오버로딩: 한 클래스 내에 이름이 같은 메소드 여러 개 작성. 같은 이름이지만 인자의 형태, 수에 따라 optional 하게 동작하도록 만든다. “매개변수의 개수나 타입, 순서”가 서로 다르다. 리턴 타입은 오버로딩과 관련 없음.
* 메소드 오버라이딩: 부모 클래스에서 상속받은 메소드를 자식 클래스에서 재정의하여 자식 클래스마다 다르게 구현하는 것.
* 인터페이스: 메소드만 정의한 클래스로, 기능을 명시한다. interface를 만들어 연결하면 객체의 사용법(임무)을 쉽게 알 수 있다.  
  interface 타입으로 참조할 수도 있다.
* 동적 바인딩: 실행할 시점에 동작이 변경될 수 있는 것. 어떤 메소드가 실행될지 실행 중 동적으로 바인딩, 컴파일할 때 동작이 결정되는 정적 바인딩과 반대됨.

객체 생성 및 접근

(1)레퍼런스 변수 선언, (2)객체 생성(new 키워드 이용하여 객체의 생성자 호출), (3)객체의 멤버 접근: 레퍼런스변수.멤버

생성자(Constructor)

객체가 생성될 때 초기화하기 위한 목적으로 사용한다.

리턴 타입이 없다. new를 통해 객체를 생성할 때, 한 번 호출된다.

모든 클래스는 생성자가 반드시 존재한다. 생성자 이름은 클래스 이름과 반드시 동일해야 한다. 메소드 오버로딩 가능.

this 레퍼런스

this : 객체(인스턴스) 자신의 참조주소를 갖는 레퍼런스. this.멤버 형태로 멤버 사용

this 사용의 장점: 객체 내부에서 인스턴스 멤버임을 명확히 한다, 매개변수와 필드이름이 같을 때 구분이 명확하다, 다른 메소드 호출 시 객체 자신의 레퍼런스를 전달하거나 반환할 때 유용하다.

접근지정자 private protected public 디폴트(접근 지정자 생략)

public 클래스: 다른 모든 클래스에게 접근 허용

디폴트 클래스(접근지정자 생략): 같은 패키지의 클래스에만 접근 허용

public 멤버: 패키지에 관계 없이 모든 클래스에게 접근 허용

private 멤버: 동일 클래스 내에만 접근 허용, 상속 받은 서브 클래스에서 접근 불가

protected 멤버: 같은 패키지 내의 클래스에게만 접근 허용. 상속받은 서브 클래스는 다른 패키지에 있어도 접근 가능

디폴트(default) 멤버: 같은 패키지 내의 다른 클래스에게 접근 허용

접근지정자에 따른 getter/setter

필드는 일반적으로 private으로 선언 후 전용 메소드로 접근. private 속성을 공유할 때는 getter/setter를 구현한다

(왜 사용? 읽기 전용 필드가 있을 수 있음(Getter의 필요성), 외부에서 엉뚱한 값으로 변경할 수 없도록(Setter의 필요성))

* getter: private 필드의 값을 리턴하는(가져오는) 역할. 필요할 경우 필드 값 가공.  
  getFieldName()또는 isFieldName() 메소드 (필드 타입이 boolean일경우 isFieldName())
* setter: 외부에서 주어진 값을 필드 값으로 수정
* setFieldName(타입 변수) 메소드: 매개 변수 타입은 필드의 타입과 동일하도록