

chapter02

자바 객체지향 프로그래밍

1. 객체지향 기본개념
2. 객체지향 개요
3. 클래스의 정의
4. 상속과 다형성
5. 추상클래스와 인터페이스
6. 예외처리

01 객체지향 기본개념

■ 도서관대여점 프로그램



- ✓ 책
- ✓ 책장
- ✓ 고객
- ✓ 고객장부
- ✓ 금고
- ✓ 대여장부
- ✓ ...

- 기능이 많아지고 복잡해진다.
- 새로운 개발 방법이 필요해짐



✓ **객체지향 프로그래밍**
(OOP: Object Oriented Programming)

01 객체지향 기본개념

■ 미녀와 야수



- ✓ 주전자
- ✓ 촛대
- ✓ 시계
- ✓ 스푼
- ✓ ...

- 물체들을 의인화 해본다. → 각각의 특성과 능력이 있다
- 물체들은 서로 메시지를 통해 이야기한다.

■ 도서관대여점 객체지향 설계

- ✓ 책
 - 책을 의인화 해봅시다.
- ✓ 책장
- ✓ 고객
 - 책이 살아 있다면 책에게 다음과 같은 질문을 할 수 있을 것입니다.
 - 책 제목은 어떻게 되니?
 - 저자는 어떻게 되니?
 - 출판사는 어떻게 되니?
 - 가격은 어떻게 되니?
 - ...
- ✓ 고객장부
- ✓ 금고
- ✓ 대여장부
- ✓ ...

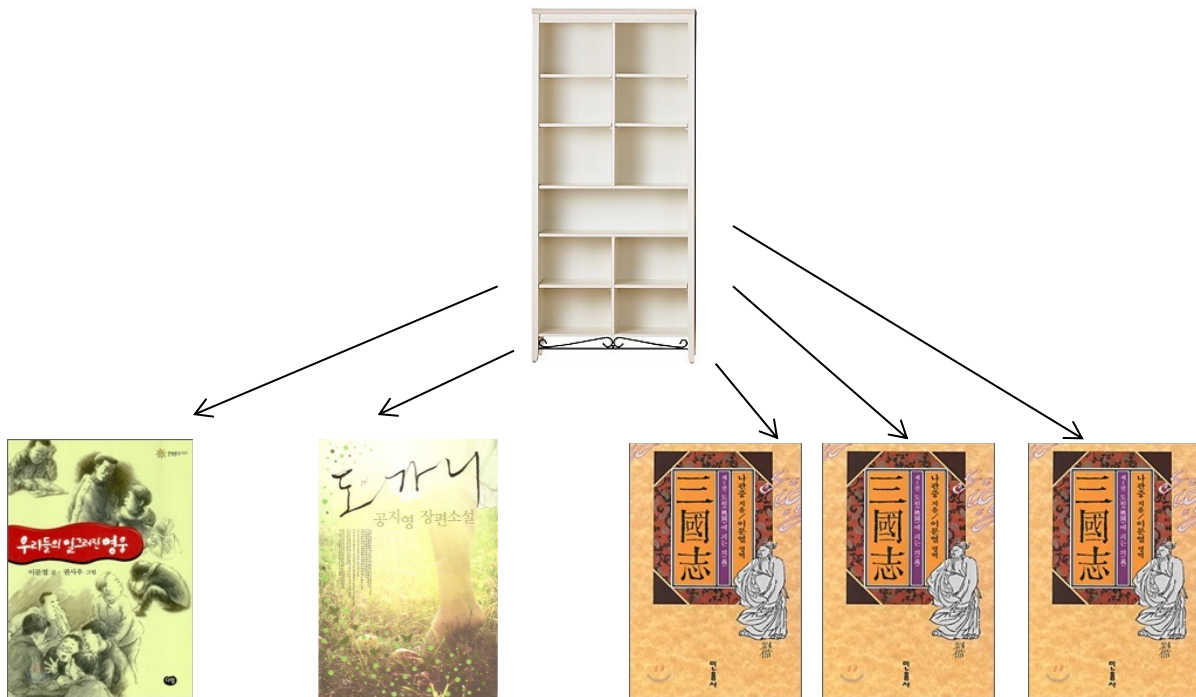
■ 도서관대여점 객체지향 설계

- ✓ 책
 - 책장도 의인화 해봅시다.
- ✓ 책장
 - 책장이 살아 있다면 책장에게 다음과 같은 질문을 할 수 있을 것입니다.
 - 가지고 있는 책은 모두 몇권이니?
 - “도가니” 라는 소설을 가지고 있니?
 - “이문열”이 쓴 책은 어떤 것들이 있니?
- ✓ 고객
 - 책을 가지고 있는 것은 책장입니다.
- ✓ 고객장부
 - 의인화 해보면 책장이 책을 관리하고 있습니다.
- ✓ 금고
- ✓ 대여장부
- ✓ ...

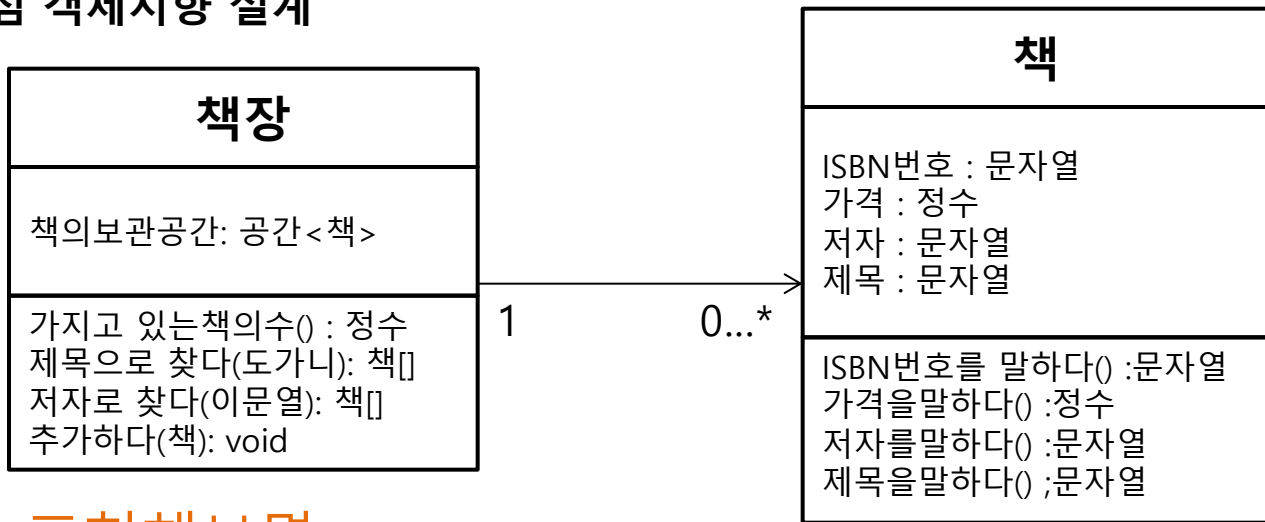
■ 도서관여점 객체지향 설계

- ✓ 책
- ✓ 책장
- ✓ 고객
- ✓ 고객장부
- ✓ 금고
- ✓ 대여장부
- ✓ ...

• 책과 책장을 도식화 해보면



■ 도서관여점 객체지향 설계



•관계를 표현해보면

- 책장은 책을 저장하는 공간을 가지고 있다. (속성)
- 책은 ISBN번호, 가격, 저자, 제목을 가지고 있다. (속성)
- 책장과 책은 자신이 가지고 있는 속성을 이용하는 기능을 가지고 있다(기능)
- 책장은 책을 가질 수 있다. (관계)

■ 도서관대여점 객체지향 설계

- ✓ 책
 - 고객과 고객장부의 관계를 그려보기
- ✓ 책장
- ✓ 고객
- ✓ 고객장부
- ✓ 금고
- ✓ 대여장부
- ✓ ...

01 객체지향 기본개념

■ 도서관대여점 객체지향 설계

• 도서관 대여점 고객 vs 신발 가게 고객

- 도서관 대여점의 고객에게 신발 사이즈를 물어본다면?
- 도서관 대여점의 고객에게 몸무게를 물어본다면?
- 신발 가게 고객에게 신발 사이즈를 물어본다면?

- 도서관대여점 고객은 이름과 연락처가 중요
- 신발가게는 신발사이즈가 중요

중요한것은 남기고, 불필요한 것을 없애는 것을 객체지향에서
"추상화"라고 말한다.

■ 도서관대여점 객체지향 설계

• 모든것이 중요하지 않다.

- 도서관 대여점의 책장은 색이 중요하지 않지만, 가구점에서는 색이 중요하다.
- 어느관점에서 보느냐에 따라서 중요한 속성도 있고 그렇지 않은 것도 있다.

- 객체지향 프로그래밍은 관점에 따라 객체가 가지는 속성과 기능을 다르게 표현하게 된다.

- 따라서 재사용이 어려울 수 있다.

■ 객체지향이 추구하는것?

•현실을 그대로 옮긴다.

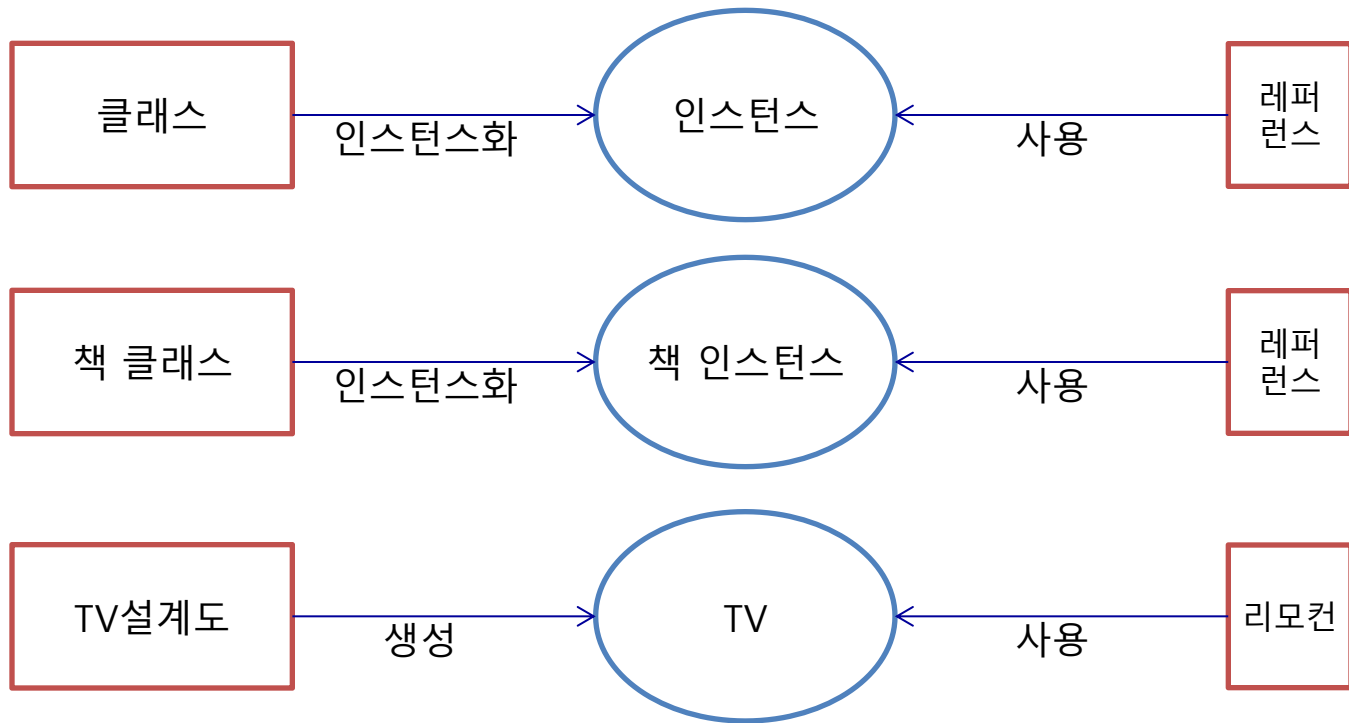
- 객체들이 가지고 있는 속성과 기능 중 필요한 것만 남기고 불필요한 것은 제거한다.
- 추상화 한다.

•재사용한다. →??

- 도서관대여점의 책장은 도서관에서도 사용될 수 있다.
- 가구점에서는 사용하기 어렵다.
- 경우에 따라 객체는 재사용 되기 어려울 수 있다.

01 객체지향 기본개념

■ 클래스, 인스턴스, 레퍼런스



01 객체지향 기본개념

■ 자바 문법으로 이해하기

책 b1 = new 책();

레퍼런스
타입

레퍼런스
변수

인스턴스
생성
키워드

생성자

인스턴스화

책 클래스

사용

b1

책 인스턴스

책

ISBN번호 : 문자열
가격 : 정수
저자 : 문자열
제목 : 문자열

ISBN번호를 말한다() : 문자열
가격을말하다() : 정수
저자를말하다() : 문자열
제목을말하다() : 문자열

01 객체지향 기본개념

■ 자바 문법으로 이해하기

```
public static void main(){
```

```
책 b1 = new 책(001);  
책 b2 = new 책(002);  
책 b3 = new 책(003);
```

```
책 c1 = new 책(111);  
책 c2 = c1;
```

```
c1 = null;
```

```
}
```

b1 → 책 인스턴스
ISBN: 001

b2 → 책 인스턴스
ISBN: 002

b3 → 책 인스턴스
ISBN: 003

c1 → 책 인스턴스
ISBN: 111

c2

책

ISBN번호 : 문자열
가격 : 정수
저자 : 문자열
제목 : 문자열

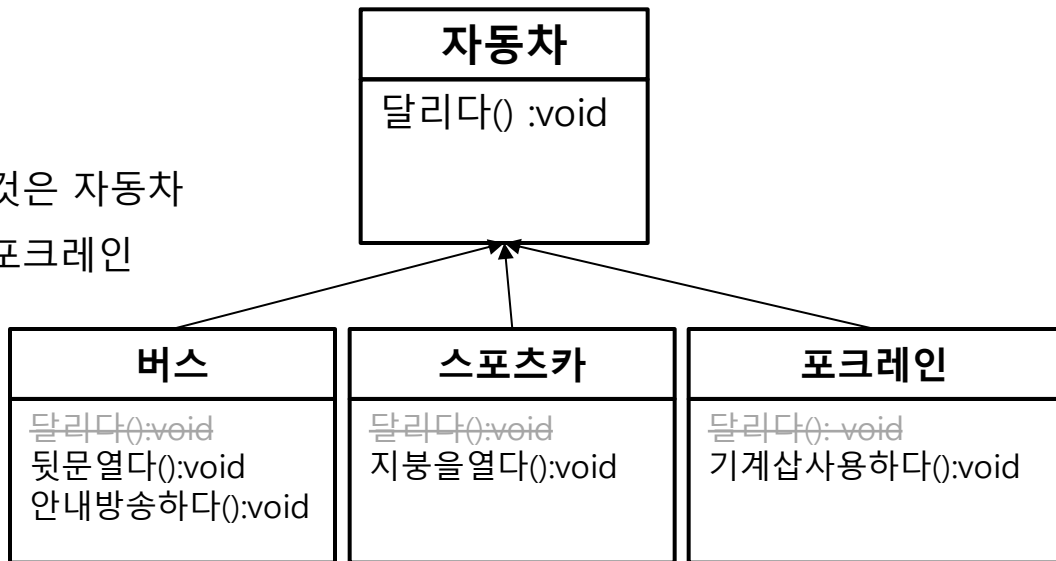
ISBN번호를 말하다() : 문자열
가격을말하다() : 정수
저자를말하다() : 문자열
제목을말하다() : 문자열

■ 자동차중 버스, 스포츠카, 포크레인을 정의해 보세요

버스	스포츠카	포크레인
달리다():void 뒷문열다():void 안내방송하다():void	달리다():void 지붕을 열다():void	달리다(): void 기계삽사용하다():void

■ 주차장 프로그램 설계

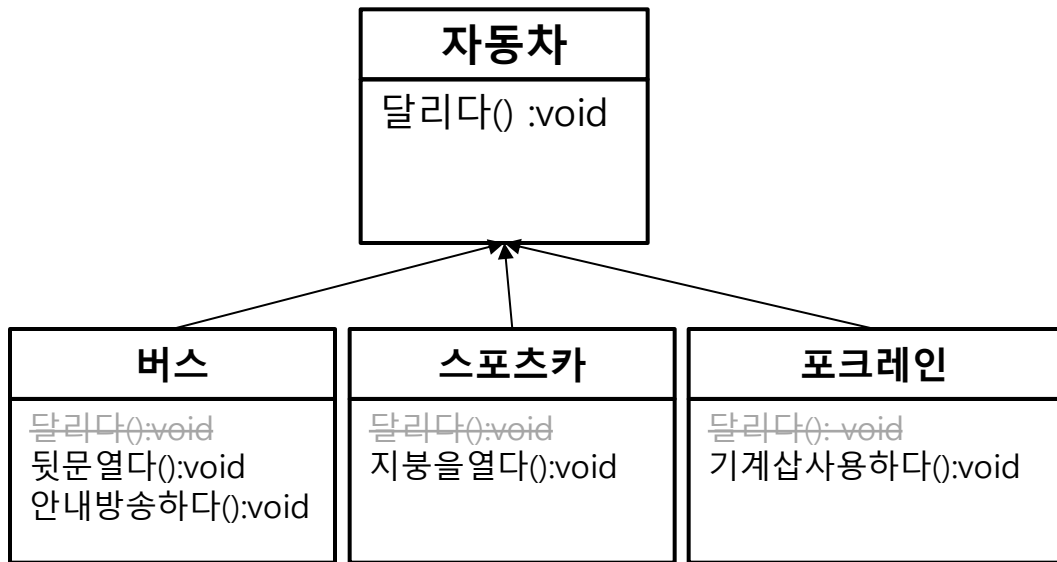
- 버스, 스포츠카 포크레인은 자식 클래스
- 자동차는 부모 클래스
- 버스, 스포츠카, 포크레인을 일반화 한 것은 자동차
- 자동차를 상속한 것은 버스, 스포츠카, 포크레인



01 객체지향 기본개념

■ 주차장 프로그램 설계

- 버스는 자동차다
- 스포츠카는 자동차다
- 포크레인은 자동차다



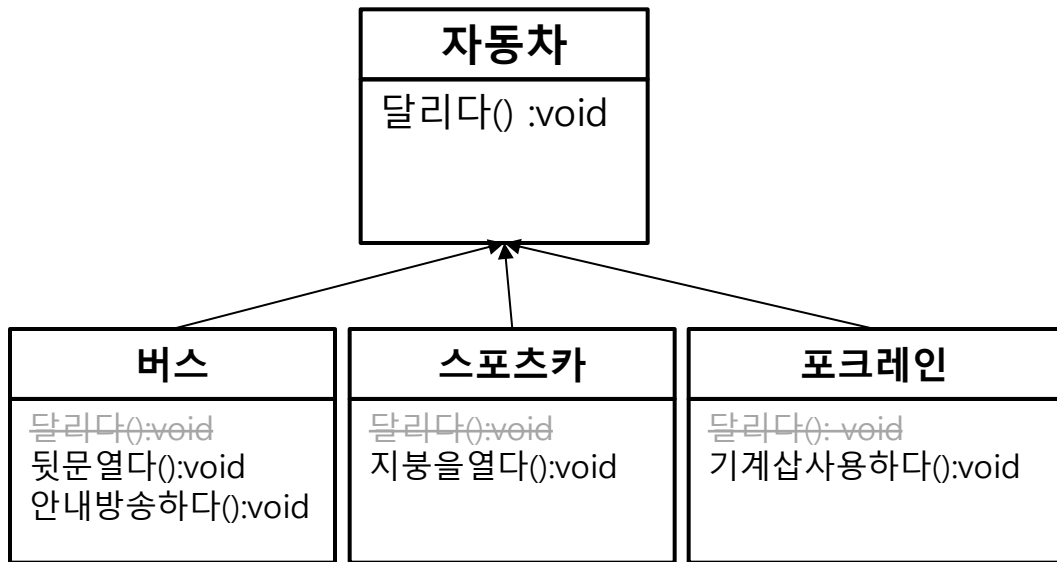
공통점이 있는 여러가지 물체들을 하나의 이름으로 부르는것을 "일반화"라고 말한다.

01 객체지향 기본개념

주차장 프로그램 설계

```
버스 bus1 = new 버스();  
bus1.달리다();  
bus1.뒷문열다();  
bus1.안내방송하다();
```

- 버스를 대신 주차해 달라고 하니 안내방송을 실행
- 스포츠카를 대신 주차해 달라고 하니 지붕을 열고 닫음
- 포크레인을 대신 주차해 달라고 하니 땅을 파고 있음



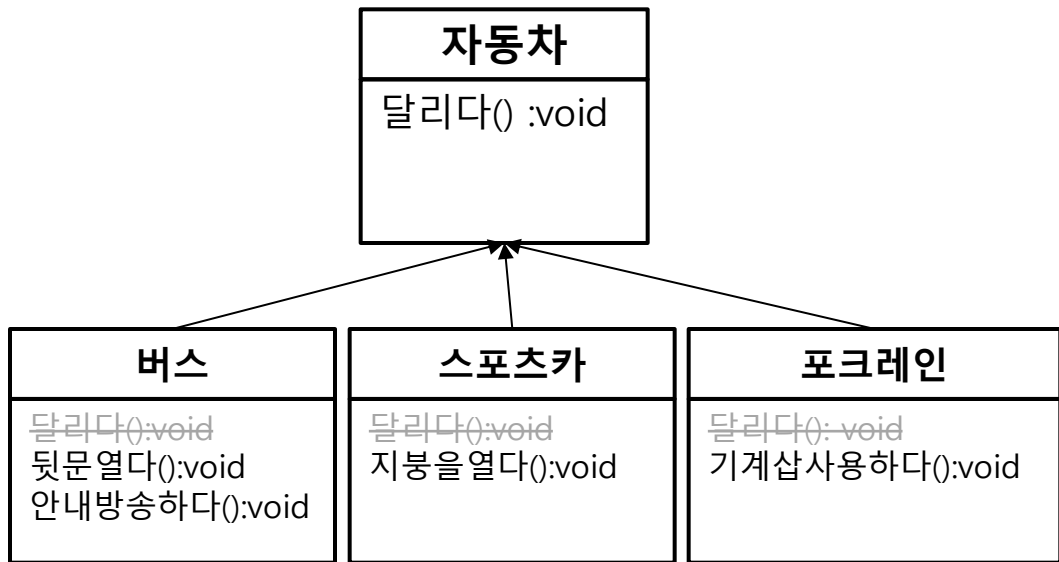
자동차의 주인은 자동차의 기본 기능만 이용하여 주차하길 바란다.

01 객체지향 기본개념

주차장 프로그램 설계

```
버스 bus1 = new 버스();  
bus1.달리다();  
bus1.뒷문열다();  
bus1.안내방송하다();
```

```
자동차 bus2 = new 버스();  
bus2.달리다();  
bus2.뒷문열다(); (X)  
bus2.안내방송하다(); (X)
```



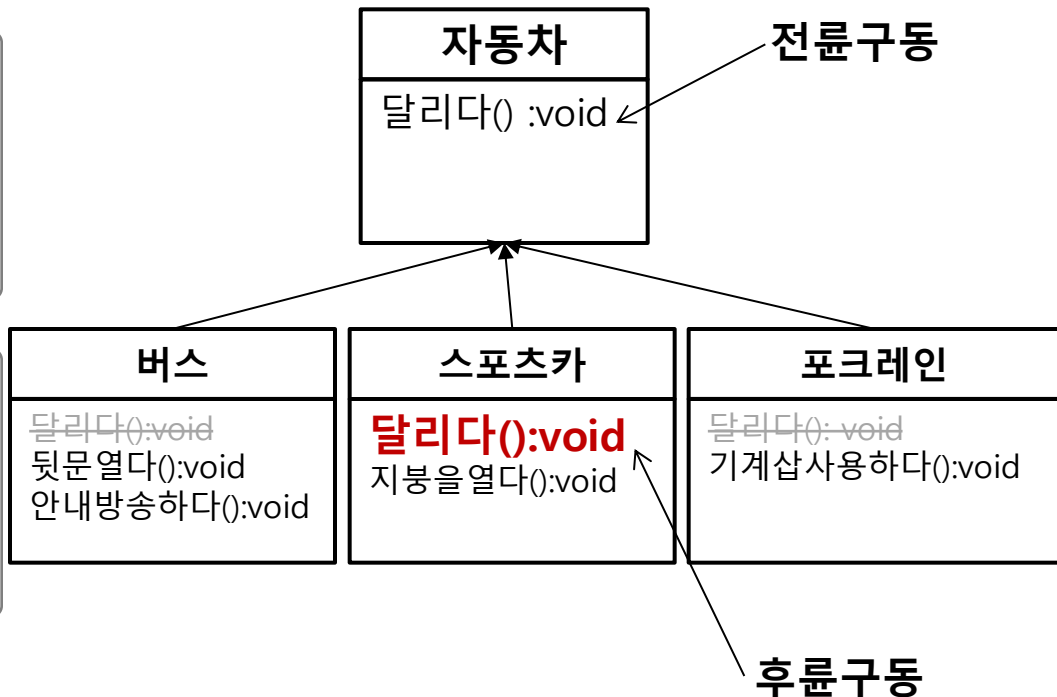
공통된 부분 이외의 기능은 사용할 수 없다.

01 객체지향 기본개념

주차장 프로그램 설계

```
자동차 spoCar = new 스포츠카();
spoCar.달리다();
spoCar.뒷문열다(); (X)
spoCar.안내방송하다(); (X)
```

```
자동차 spoCar = new 스포츠카();
spoCar.달리다(); ← 후륜구동
spoCar.뒷문열다(); (X)
spoCar.안내방송하다(); (X)
```



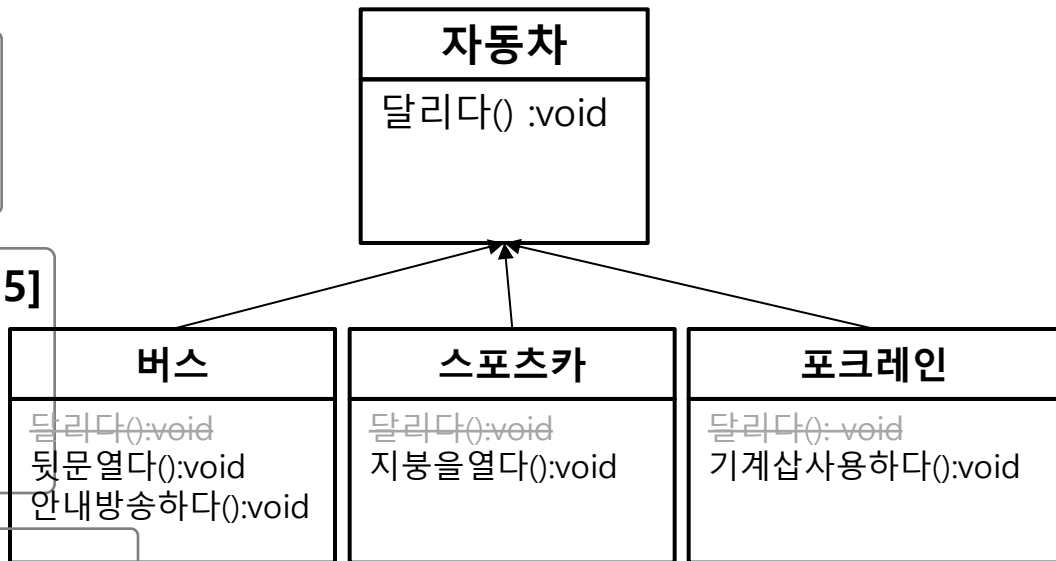
메소드 오버라이드(Override: 덮다)

주차장 프로그램 설계

```
자동차 bus1= new 버스();  
자동차 spo1= new 스포츠카();  
자동차 fork1= new 포크레인();
```

```
자동차[] carArray = new 자동차[15]  
carArray[0] = bus1;  
carArray[1] = spo1;  
carArray[2] = fork1;
```

```
버스[] busArray = new 버스[5]  
스포츠카[] spoArray = new 스포츠카[5]  
포크레인[] forkArray = new 포크레인[5]
```



chapter02

자바 객체지향 프로그래밍

1. 객체지향 기본개념
- 2. 객체지향 개요**
3. 클래스의 정의
4. 상속과 다형성
5. 추상클래스와 인터페이스
6. 예외처리

■ 객체지향 언어의 역사

- 실 세계의 모의실험(simulation)을 위해 컴퓨터를 이용, 가상세계를 구현하려는 노력으로부터 시작됨
- 1960년대 최초의 객체지향언어 Simula탄생
- 1980년대 절차방식의 프로그래밍의 한계를 객체지향방식으로 극복하려고 노력함.
(C++, Smalltalk과 같은 발전된 객체지향언어가 탄생)
- 1995년 말 Java탄생. 객체지향언어가 프로그래밍 언어의 주류가 됨.

■ 객체지향 언어의 특징

- 데이터의 변화/저장을 위한 알고리즘(함수) 중심으로 프로그램이 구성되는 절차지향 언어에 비해 객체지향 언어에서는 객체들의 집합이 프로그램이 된다.
- 객체는 데이터와 그 데이터를 처리할 수 있는 메소드를 가지게 된다.
- 소프트웨어 품질 향상과 관련하여 객체지향 기술은 소프트웨어의 부품화, 소프트웨어 재사용을 주요 목표로 한다.
- 요구에 맞는 객체를 만들기 위해 절차지향 언어에 비해 분석이나 설계에 더 비중 있게 생각한다.
- 캡슐화(encapsulation), 상속(inheritance), 다형성(polymorphism) 등의 주요 개념이 있다.
- 기존의 프로그래밍 언어와 크게 다르지 않다.
- 코드의 재 사용성이 높다.(??)
- 코드의 관리가 쉬워졌다.
- 신뢰성이 높은 프로그램 개발을 가능하게 한다.

02 객체지향 개요

■ 객체지향 언어의 정의

- 실 세계에 존재하는 것, 사물 또는 개념
- 객체는 정보를 효율적으로 관리하기 위하여 의미를 부여하고 분류하는 논리적 단위

■ 객체의 구성 요소

- 보통 객체는 속성(정보)과 기능의 집합이며, 속성과 기능을 객체의 멤버(member, 구성요소)라고 한다.
- 속성은 필드로, 기능은 메소드로 정의한다. (클래스가 정의될 때)

■ 클래스의 정의

- 클래스는 객체를 정의해 놓은 것이다
- 클래스는 객체를 생성하는 데 사용된다

클래스	객체
제품 설계도	제품
TV설계도	TV
붕어빵기계	붕어빵

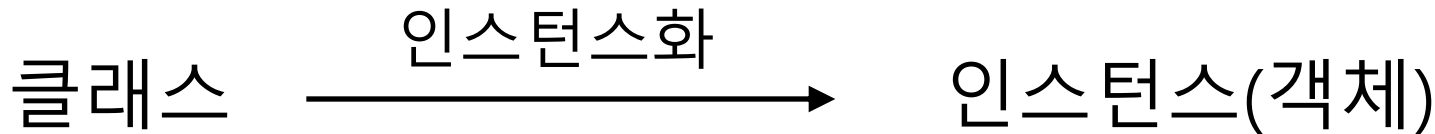
02 객체지향 개요

■ 인스턴스

- 객체(Object)는 인스턴스(Instance)의 일반적 의미
- 객체가 메모리에 할당되어 실제 사용될 때 인스턴스라고 부른다.
- 객체 ≡ 인스턴스
- 클래스로부터 인스턴스를 생성하는 것을 인스턴스화(instantiate)라고 한다.

책상은 인스턴스다.
책상은 객체다.

책상은 책상 클래스의 객체다.
책상은 책상 클래스의 인스턴스다.



■ 캡슐화(Encapsulation)

- 객체의 실제 구현된 내용을 외부에 감추는 것을 말한다.
- 객체를 사용하는 쪽(외부 객체)에서는 그 객체의 내부 구조를 알지 못하며 노출된 필드와 메소드를 통해 객체를 이용한다. 즉, 객체의 데이터가 실제 어떻게 처리되는지 자세히 알 필요가 없다.
- 객체를 작성할 때 개발자는 숨겨야 하는 필드와 메소드 그리고 공개하는 필드와 메소드를 구분하여 작성한다.
- 외부에서는 공개된 필드와 메소드만 접근이 가능 하다 (Information Hiding, 정보은닉)
- 접근 제한자를 사용해서 객체의 필드와 메소드의 사용 범위를 제한하게 된다.

■ 상속(inheritance)

- 이미 만든 객체와 비슷하지만 필드와 메소드가 약간 차이가 나는 객체를 생성해야 하는 경우
- 기존의 클래스에서 필드와 메소드를 상속(재사용) 하고 더 필요한 필드와 메소드를 추가한다.
- 상속의 개념은 코드를 간결하게 하고 코드의 재 사용성을 높이는 객체지향 개념이다.

■ 다형성(Polymorphism)

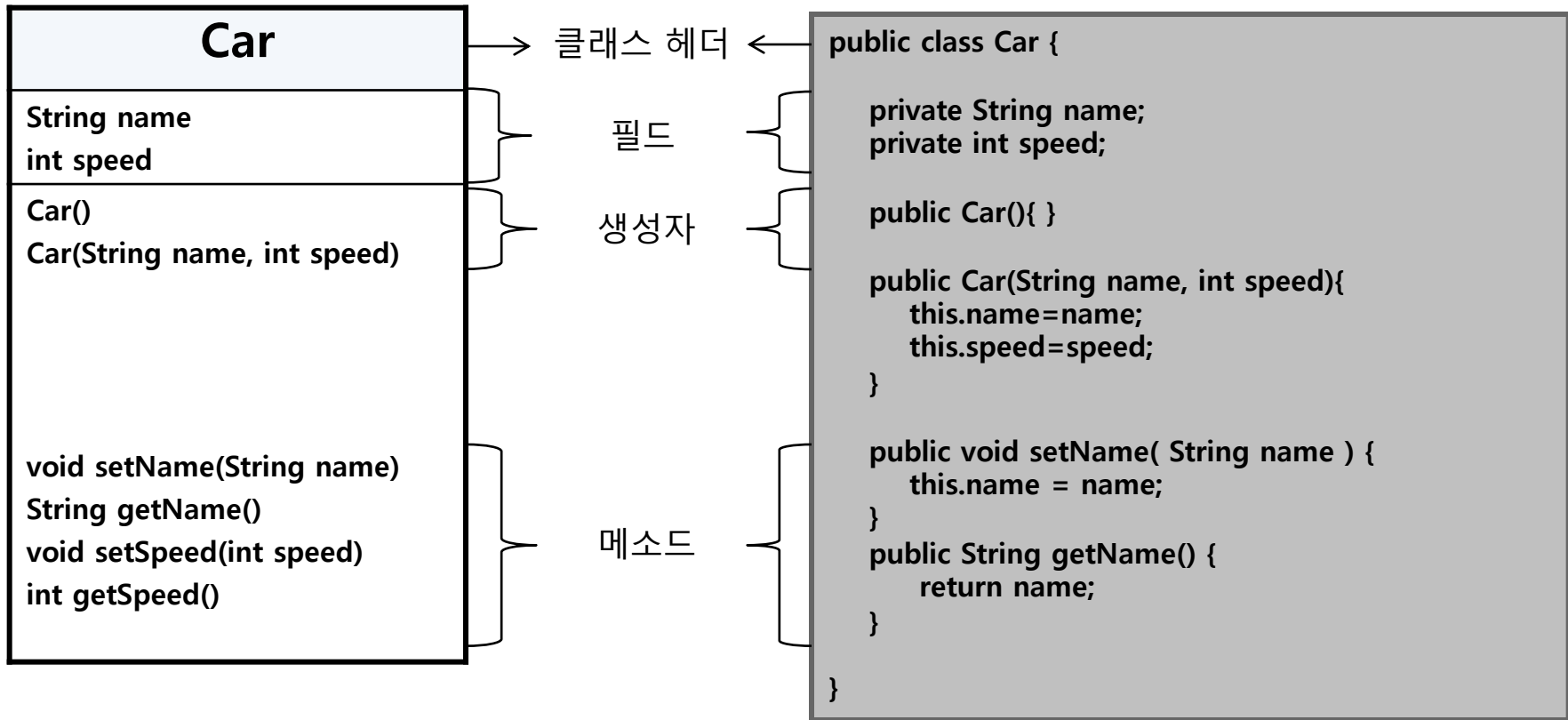
- 객체지향의 꽃
- 하나의 메소드나 클래스를 다양한 구현으로 사용 가능하게 하는 개념
- 오버로드(Overload) 와 오버라이드(Override)를 통해 다형성을 구현하게 된다.

chapter02

자바 객체지향 프로그래밍

1. 객체지향 기본개념
2. 객체지향 개요
- 3. 클래스의 정의**
4. 상속과 다형성
5. 추상클래스와 인터페이스
6. 예외처리

■ 클래스의 구조



■ 필드

- 객체의 데이터, 상태를 저장하는 변수
- 주로 기본 타입 또는 참조 타입 변수들을 정의하게 되는데 멤버 변수라 하기도 한다

[문제]

쇼핑몰에서 상품을 관리하기 위해 상품관리 프로그램을 만들려고 합니다. 프로그램을 만들기 전에 업무(비즈니스) 분석을 통해 상품 객체를 분석하고 다음과 같은 Goods클래스를 정의 하였습니다.

Goods 클래스를 정의하고 GoodsApp 클래스에서 Goods 클래스를 테스트 하세요.

- 1) Goods 객체를 하나 생성하고 이 객체에 대한 레퍼런스 변수를 camera 로 합니다.
- 2) 이 객체의 데이터인 각 각의 인스턴스 변수는 다음과 같은 값을 가지도록 합니다.
상품명 : "nikon", 가격: 400,000
- 3) 값을 세팅 한 후, 객체의 데이터를 출력해 보세요.

Goods
String name int price

[문제] GoodsApp2.java

다음의 데이터를 추가한 후, 출력해 보세요

-상품이름 : "LG그램", 가격: 900000

-상품이름 : "머그컵", 가격: 2000

Goods

String name

int price

■ 접근자

- 주로 객체에 대한 정보(필드와 메소드)의 접근을 제한하기 위해 사용한다.
- **캡슐화 정보은닉을 위한 방법이다.**
- public, protected, default, private 네 가지 종류가 있다.

지시자	클래스 내부	동일 패키지	상속받은 클래스	이외의 영역
private	●	×	×	×
default	●	●	×	×
protected	●	●	●	×
public	●	●	●	●

public > protected > default > private

[문제]

- Goods 클래스의 필드 접근자를 public으로 변경해 봅니다.
- Goods 클래스의 필드 접근자를 default보다 강한 접근 제어자인 private로 지정하여 어떤 변화가 있는 지 확인해 봅니다.

지시자	클래스 내부	동일 패키지	상속받은 클래스	이외의 영역
private	●	×	×	×
default	●	●	×	×
protected	●	●	●	×
public	●	●	●	●

공개

비공개

`public > protected > default > private`

■ 메소드

- 객체의 기능 또는 행동을 정의하는 함수
- 정의방법

```
  ①  ②  ③  ④  
public int getSum( int i, int j ) {  
    int result = i + j; ⑤  
    return result; ⑥  
}
```

1. 접근 지정자
2. 리턴 타입
3. 메소드 이름
4. 메소드 인자 (파라미터)
5. 구현코드
6. 리턴문

- 호출방법

```
  ①  ②  ③  ④  ⑤  
int sum = util.getSum(3, 2);
```

1. 자료형
2. 변수명
3. 레퍼런스변수(객체)
4. 메소드명
5. 메소드 인자 (파라미터)

■ 메소드

- 매개변수(parameter)
 - 메소드를 선언할 때 괄호 안에 표현된 Input 값을 나타내는 변수 (type1 name1, type2 name2, ...)
 - 메소드 호출에서 들어가는 구체적인 값은 인자(Argument)라고 함
- 반환타입(return type)
 - 메소드는 0개 혹은 1개의 값을 Output으로 반환할 수 있음
 - 반환 값이 없을 때: void
 - 반환 값이 있을 때: int, boolean, Goods, ...
 - 반환 되는 값은 메소드 선언에서 정의된 반환 타입과 일치해야 함
- 메소드이름
 - 자바의 식별자 규칙 대/소문자, 숫자, _ \$ 조합하여 지을 수 있고 숫자로 시작할 수 없다.
 - 관례에 따라 소문자로 작성하고 두 단어가 조합될 경우 두 번째 시작문자는 대문자로 짓는다.
 - 메소드명은 기능을 쉽게 알 수 있도록 작성하는 것이 좋은 이름이다

[문제]

아래와 같이 클래스를 정의하여 프로그램을 작성하세요

Goods 클래스를 만드세요

- 1) 필드접근자를 private로 작성해서 외부에서 접근할 수 없게 합니다.
- 2) 필드에 값을 저장할 수 있도록 set메소드를 만드세요.
- 3) 필드에 값을 읽을 수 있도록 get메소드를 만드세요.
- 4) 아래와 같이 상품의 모든 정보를 출력해 주는 showInfo()를 만드세요.

Goods	
String name	
int price	

GoodsApp 클래스 만드세요

- 1) showInfo()메소드를 이용하여 다음과 같이 출력하세요.



```
<terminated> GoodsApp [Java Application] C:\Program Files\Java\jdk1.8
상품이름 : 니콘
가격 : 400000

상품이름 : LG그램
가격 : 900000

상품이름 : 머그컵
가격 : 2000
```

■ 메소드

• Getter 와 Setter 메소드

- 일반적으로 객체의 데이터는 객체 외부에서 직접적으로 접근하는 것을 막는다.
- 객체의 외부에서 객체 내부의 데이터를 마음대로 읽고 쓸 경우 데이터의 무결성을 보장하기 힘들기 때문이다.
- 메소드를 통한 접근을 하게 되면 객체의 데이터를 변경할 경우 무결성 체크를 할 수 있다.

✓클래스를 정의할 때 필드는 private로 하여 객체 내부의 정보를 보호하고(정보은닉) 필드에 대한 Setter와 Getter를 두어 객체의 값을 변경하고 참조하는 것이 좋다.

- 외부에서 읽기만 가능하게 하기 위해선 Getter만 해당 필드에 대해서만 작성하면 된다.
- 외부에서 쓰기만 가능하게 하기 위해선 Setter만 해당 필드에 대해서만 작성하면 된다.
- Getter와 Setter가 없으면 객체 내부 전용 변수가 된다.

[문제]

아래와 같이 클래스를 정의하여 프로그램을 작성하세요

Point 클래스를 만드세요

- x, y 좌표를 나타낼 수 있는 필드 작성
- x, y 좌표에 접근할 수 있는 getter/setter 메소드 작성
- 다음 실행 결과를 참조하여 draw()메소드 작성

PointApp 클래스 만드세요

- 1) draw() 메소드를 호출하여 다음과 같이 출력하세요

Problems @ Javadoc Declaration Console

```
<terminated> PointApp [Java Application] C:\#Program Fil  
점 [x=5, y=5]을 그렸습니다.  
점 [x=10, y=23]을 그렸습니다.
```

[문제]

아래와 같이 클래스를 정의하여 프로그램을 작성하세요

Book 클래스를 만드세요

- 책제목 , 책저자를 나타낼 수 있는 필드 작성
- 각 필드에 접근할 수 있는 setter 메소드 작성
- 다음 실행 결과를 참조하여 showInfo()메소드 작성

BookApp 클래스 만드세요

1) showInfo() 메소드를 호출하여 다음과 같이 출력하세요

책[이문열:삼국지]

책[박경리:토지]

[문제]

아래와 같이 클래스를 정의하여 프로그램을 작성하세요

Song 클래스를 만드세요

Song 클래스는 다음과 같은 필드를 가지고 있습니다.

- 노래의 제목을 나타내는 title
- 가수를 나타내는 artist
- 노래가 속한 앨범 제목을 나타내는 album
- 노래의 작곡가를 나타내는 composer
- 노래가 발표된 연도를 나타내는 year
- 노래가 속한 앨범에서 트랙 번호를 나타내는 track

1) 필드의 접근을 제한하고 getter/setter 메소드를 통해 접근하세요.

2) 노래정보를 출력하는 showInfo() 메소드를 작성하세요.

SongApp 클래스를 만드세요

1) showInfo() 메소드를 호출하여 다음과 같이 출력하세요

Problems @ Javadoc Declaration Console

No consoles to display at this time.

아이유, 좋은날 (Real, 2010, 3번 track, 이민수 작곡)

BIGBANG, 거짓말 (Always, 2007, 2번 track, G-DRAGON 작곡)

버스커버스커, 벚꽃엔딩 (버스커버스커1집, 2012, 4번 track, 장범준 작곡)

■ 생성자(Constructor)

- new 연산자와 같이 사용되어 클래스로부터 객체를 생성할 때 호출되고 객체의 초기화를 담당 한다.
- 생성자를 실행 시키지 않고 클래스로부터 객체를 만들 수 없다.
- 생성자가 성공적으로 실행되면 JVM의 Heap영역에 객체가 생성되고 객체의 참조 값이 참조변수에 저장 된다.
- 몇 가지 조건을 제외하면 메소드와 같다.
- 모든 클래스에는 반드시 하나 이상의 생성자가 있어야 한다.

03 클래스의 정의

■ 생성자(Constructor)

- 생성자의 정의

- ✓ 생성자의 이름은 클래스와 같아야 한다.
- ✓ 생성자의 리턴값이 없다. (하지만 void를 쓰지 않는다)

```
접근자 클래스이름 ( 파라미터 ) {  
  
    // 인스턴스 생성시 수행할 코드  
    // 주로 인스턴스 변수의 초기화 코드  
  
}
```

```
public class Goods {  
    public Goods() {  
        // 초기화 코드  
    }  
  
    public Goods( String name, int price ) {  
        // 초기화 코드  
    }  
  
}
```

03 클래스의 정의

■ 생성자(Constructor)

- 기본생성자
 - 매개변수가 없는 생성자
 - 클래스에 생성자가 한 개도 정의되어 있지 않으면 컴파일러가 **기본생성자**를 추가한다.
 - 생성자가 한 개라도 있으면 기본생성자를 추가 하지 않는다.

```
public class Goods {  
  
    public Goods() {  
        // 초기화 코드  
    }  
  
}
```

- 여러 개의 생성자를 사용할 수 있다.



✓ **생성자 오버로딩**

[문제]

- 1) Goods 클래스에서 생성되는 모든 필드 초기화 하는 생성자를 정의합니다.
- 2) 오류가 발생하면 오류가 발생하는 원인을 생각해 보세요.
- 3) 생성자 오버로딩을 확인합니다.

[문제]

- 1) Point 클래스의 기본 생성자와 모든 필드를 초기화 할 수 있는 생성자를 작성하고 테스트 합니다.

[문제]

- 1) Song 클래스의 기본 생성자와 모든 필드를 초기화 할 수 있는 생성자를 작성하고 테스트 합니다.

■ this 키워드

- this 키워드는 메소드 호출을 받는 객체를 의미한다.
- 현재 사용중인 객체 그 자체를 의미한다.
- this() 는 클래스의 한 생성자에서 다른 생성자를 호출 할 때 사용할 수 있다.

[문제]

- 1) Goods 클래스에서 생성되는 모든 필드 초기화 하는 생성자를 정의합니다.
- 2) 이 생성자에서 다른 생성자를 호출하도록 합니다.

[문제]

- 1) Song 클래스에서 노래 제목과 가수만 입력 받아 필드를 초기화하는 생성자를 하나 더 오버로딩 합니다.
- 2) 이 생성자에서 다른 생성자(모든 필드 초기화 하는 생성자)를 호출하도록 합니다.

03 클래스의 정의


■ 메소드 오버로딩

- 하나의 클래스에 같은 이름의 메소드가 여러 개 존재할 수 있다.
- 그 메소드들은 매개변수의 타입, 개수, 그리고 순서가 다른 형태로 구별된다.
동일한 이름의 상이한 시그니처

※ 메소드 시그니처(Signature) : 메소드 인자의 타입, 개수, 순서

[문제]

1) Point 클래스에 점을 안보이게 할 수 있는 기능까지 추가된 draw() 메소드를 하나 더 추가하고 아래 실행결과가 나오도록 테스트 하세요



```
<terminated> PointApp [Java Application] C:\Program Files\Java
점 [x=5, y=5]을(를) 그렸습니다.
점 [x=10, y=23]을(를) 그렸습니다.
점 [x=5, y=5]을(를) 지웠습니다.
점 [x=10, y=23]을(를) 지웠습니다.
```

■ 연습문제

아래 TV 클래스의 main 메소드를 실행할 수 있도록, 요구 조건을 참조하여 TV 클래스를 정의 하세요.

- 1) 모든 필드는 private으로 접근 제어를 하고 getter만 작성합니다. (channel, volume, power 필드 read-only)
- 2) channel, volume, power의 초기값을 각각 7, 20, false 로 초기화 하는 생성자 작성
- 3) 기본 생성자 오버로딩
- 4) void power(boolean on) 메소드 구현
- 5) void channel(int channel) 메소드 구현 (1~255 유지)
- 6) void channel(boolean up) 메소드 오버로딩 (1~255 유지, 1씩 증감)
- 7) void volume(int volume) 메소드 구현 (0 ~ 100 유지)
- 8) void volume(boolean up) 메소드 오버로딩 (0 ~ 100 유지, 1씩 증감)
- 9) void status() 메소드 구현(TV 정보 출력)

TV

```
int channel
int volume
boolean power
```

```
power( boolean )
channel( int )
channel( boolean )
volume( int )
volume( boolean )
status();
```

```
public class TVApp {
    public static void main( String[] args ) {
        TV tv = new TV(); // 7, 20 , false

        tv.status();

        tv.power( true );
        tv.volume( 120 );
        tv.status();

        tv.volume( false );
        tv.status();

        tv.channel( 0 );
        tv.status();

        tv.channel( true );
        tv.channel( true );
        tv.channel( true );
        tv.status();

        tv.power( false );
        tv.status();
    }
}
```


■ 클래스(Static) 변수 와 인스턴스 변수, 지역변수

- 선언위치에 따른 변수의 종류

```
public class Goods{
```

```
    static int countOfGoods;    //클래스(static)변수
    private String name;        //인스턴스변수
    private int price;          //인스턴스변수
```

```
    public void setPrice(int price){
```

```
        int localVal;          //지역변수
    }
```

```
}
```

클래스영역

메소드영역

변수의 종류	선언위치	생성시기
클래스(static) 변수	클래스 영역	클래스가 메모리에 올라갈 때
인스턴스 변수		인스턴스 생성 시
지역변수	메서드 영역	변수 선언문 수행 시

- **인스턴스 변수(instance variable)**

- 각 인스턴스의 개별적인 저장공간 인스턴스 마다 다른 값 저장가능
- 인스턴스 생성 후, '참조변수.인스턴스 변수명'으로 접근
- 인스턴스를 생성할 때 생성되고, 참조변수가 없을 때 가비지 컬렉터에 의해 자동 제거됨

- **클래스(static) 변수(class variable)**

- 같은 클래스의 모든 인스턴스들이 공유하는 변수
- 인스턴스 생성없이 '클래스이름.클래스(스태틱)변수명'으로 접근
- 클래스가 로딩될 때 생성되고 프로그램이 종료될 때 소멸

- **지역 변수(local variable)**

- 메소드 내에 선언되며, 메소드의 종료와 함께 소멸
- 조건문, 반복문의 블록{ } 내에 선언된 지역변수는 블록을 벗어나면 소멸

- **인스턴스 변수 와 클래스(static) 변수**

- 인스턴스 변수는 인스턴스가 생성될 때마다 생성되므로 인스턴스 마다 각기 다른 값을 유지할 수 있지만, 클래스(static)변수는 모든 인스턴스가 하나의 저장공간을 공유하므로 항상 공통된 값을 갖는다.

■ 클래스(Static) 메소드

- 전역변수와 전역함수를 만들 때 사용
- 모든 클래스에서 공유하는 전역 변수나 전역 함수를 만들어 사용할 수 있다.
- 객체를 생성하지 않고 접근할 수 있다.
- static 메소드에서는 this 사용 불가
- static 메소드에서는 static 멤버만 접근할 수 있다.

```
MyMath.PI = 3.14  
MyMath m = new MyMath();  
m.PI
```

MyMath

- PI
- plus()
- plus()
- CircleArea()

m

m2

```
public class MyMath {  
  
    public static double PI = 3.14;  
  
    public static int plus(int a, int b) {  
        return a+b;  
    }  
  
    public static double plus(double a, double b) {  
        return a+b;  
    }  
  
    public static double CircleArea(int radius){  
        return radius*radius*PI;  
    }  
}
```

chapter02

자바 객체지향 프로그래밍

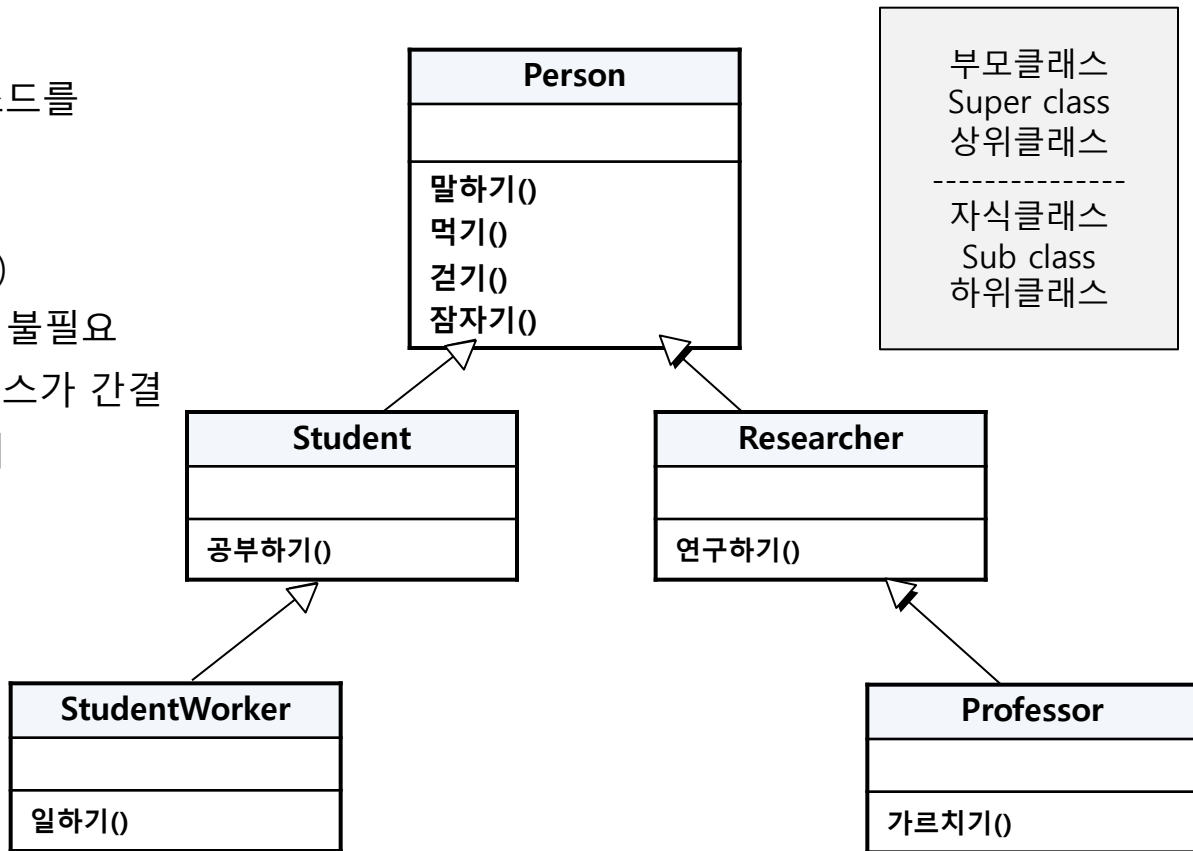
1. 객체지향 기본개념
2. 객체지향 개요
3. 클래스의 정의
- 4. 상속과 다형성**
5. 추상클래스와 인터페이스
6. 예외처리

■ 상속

부모 클래스에 정의된 필드와 메소드를
자식 클래스가 물려 받는것

•왜 상속을 하는가?(상속의 필요성)

- 클래스 사이의 멤버 중복선언 불필요
- 필드, 메소드 재사용으로 클래스가 간결
- 클래스간 계층적 분류 및 관리



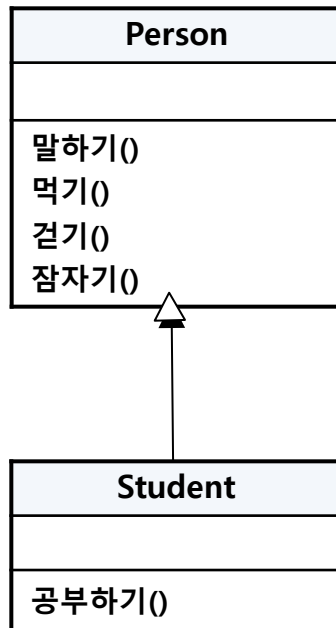
• 자바상속의 특징

- 자바에서는 다중 상속을 지원하지 않는다.
- 자바에서는 상속의 횟수에 제한을 두지 않는다.
- 자바에서 계층구조의 최상위에 있는 클래스는 java.lang.Object 이다.

• 상속선언

```
public class Person {  
    public void tel()  
    public void eat()  
    public void walk()  
    public void sleep()  
}
```

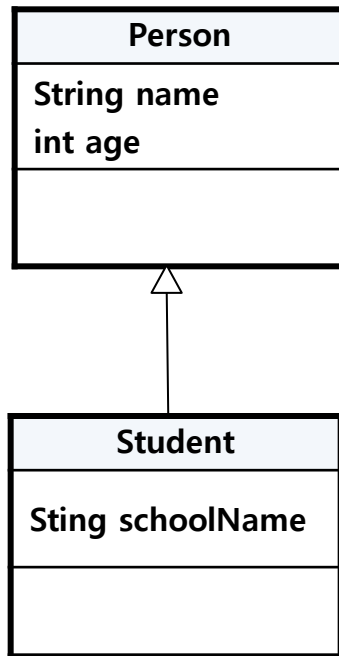
```
public class Student extends Person {  
    public void study()  
}
```



• 연습문제

[문제]

- Person 클래스를 만드세요.
✓생성자, getter/setter, showInfo()
- Person 클래스를 상속받아 Student 클래스를 만드세요.
✓생성자, getter/setter, showInfo()
- PersonApp 클래스를 통해서 인스턴스를 생성하고 showInfo()를 통해 확인하세요.
✓Person p = **new Person("정우성", 45);**
✓Student s1 = **new Student("서울고등학교");**
✓Student s2 = **new Student("이정재", 45, "한국고등학교");**
- 자식 클래스와 부모클래스의 생성자 순서를 확인하세요



• 상속과 생성자

- 자식생성자에서 특별한 지시가 없으면 부모클래스의 기본생성자가 선택된다.
- 부모클래스의 특정 생성자를 호출해야 할 경우에는 `super()`를 이용하여 명시적으로 부모클래스의 생성자를 호출된다.
- 부모의 필드나 메소드에 접근시에는 `super` 키워드를 사용한다.

• 상속과 접근제한자

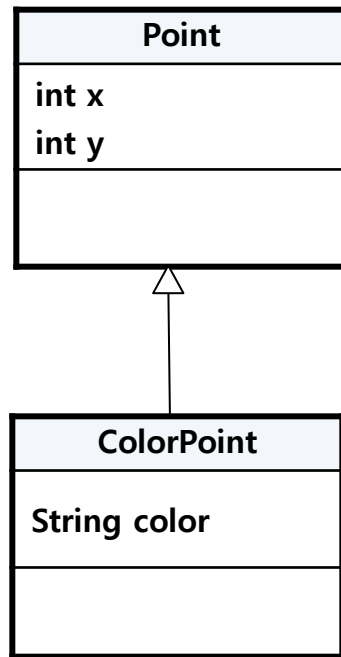
지시자	클래스 내부	동일 패키지	상속받은 클래스	이외의 영역
private	●	×	×	×
default	●	●	×	×
protected	●	●	●	×
public	●	●	●	●

`public > protected > default > private`

• 연습문제

[문제]

- Point 클래스를 만드세요.
 - ✓생성자, getter/setter, showInfo()
- Point 클래스를 상속받아 ColorPoint 클래스를 만드세요.
 - ✓생성자, getter/setter, showInfo()
- PointApp 클래스를 통해서 인스턴스를 생성하고 showInfo()를 통해 확인하세요.
 - ✓Point p = **new Point(4,4);**
 - ✓ColorPoint cp1 = **new ColorPoint("red");**
 - ✓ColorPoint cp2 = **new ColorPoint(10,10,"blue");**
- 자식 클래스와 부모클래스의 생성자 순서를 확인하세요

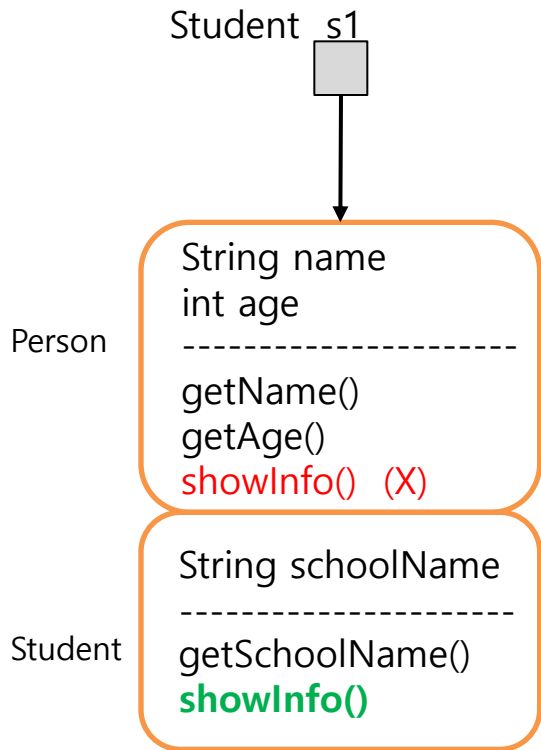


04 상속과 다형성

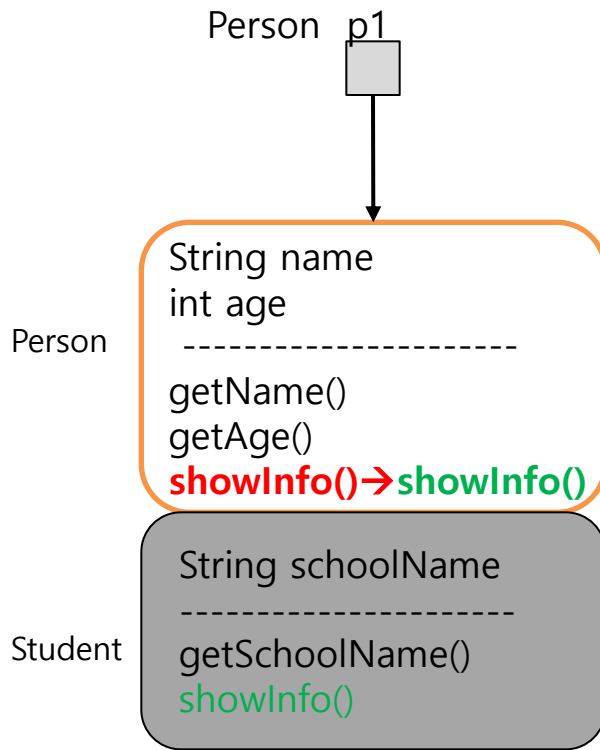
- 메소드 오버라이딩
 - OOP의 꽃
 - 부모클래스와 자식 클래스의 메소드 사이에서 발생하는 관계
 - 부모클래스의 메소드를 동일한 이름으로 재작성 (같은 이름, 같은 리턴 타입, 같은 시그니처)
 - 부모클래스 메소드 무시하기

04 상속과 다형성

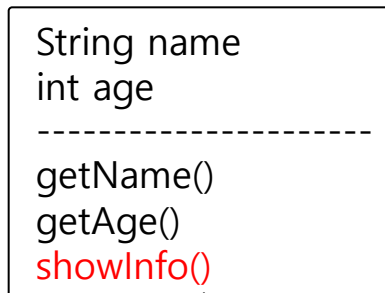
Student s1 = new Student()



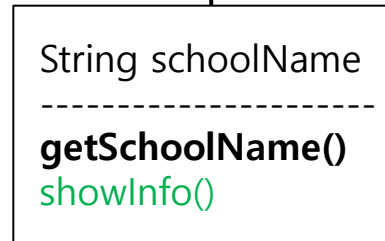
Person p1 = new Student()



Person



Student



04 상속과 다형성

- 업캐스팅(Up Casting)
 - 자식클래스가 부모클래스 타입으로 변환되는 것
 - 명시적으로 타입 변환을 하지 않아도 된다.

- 다운캐스팅(Down Casting)
 - 업캐스팅 된 것을 원래대로 되돌리는 것
 - 명시적으로 타입변환을 하여야 한다.

chapter02

자바 객체지향 프로그래밍

1. 객체지향 기본개념
2. 객체지향 개요
3. 클래스의 정의
4. 상속과 다형성
- 5. 추상클래스와 인터페이스**
6. 예외처리

05 추상클래스와 인터페이스

■ 추상클래스

- 추상화

객체들이 가지고 있는 속성과 기능 중에 중요한 것들을 남기고 필요 없는 것은 없애는 것
또는 객체들간의 공통되는 특성을 추출하는 것

- 추상클래스

- 실체 클래스의 공통적인 특성들을 추출해서 선언한 클래스
- 실체 클래스를 만들기 위한 부모 클래스로만 사용되는 클래스(객체를 직접 생성해서 사용할 수 없다)
- **확장 만을 위한 용도**로 사용된다.
- **하나 이상의 추상 메소드**를 가진다.
- **속성(필드)와 기능(메소드)를 정의** 할 수 있다.

- 추상메소드

- **구현이 불가능한 메소드**로서 선언만 한다.
- 추상 클래스를 상속하는 실체 **자식 클래스는 추상 메소드를 반드시 구현**해야 한다.
- 추상 메소드는 추상 클래스에만 존재한다.

■ 추상클래스

- 추상클래스의 선언

추상클래스를 선언할 때는 클래스 선언에 **abstract** 키워드를 붙인다.

```
public abstract class 클래스명 {  
    //필드  
    //생성자  
    //메소드  
}
```

- 추상클래스의 상속

- 추상 클래스의 상속에도 extends 키워드 사용
- 추상 클래스를 상속하는 클래스는 반드시 추상 클래스의 추상 메소드를 구현해야 함

- 추상클래스의 활용

여러 클래스들이 상당수 공통점을 가지고 있으나 부분적으로 그 처리 방식이 다를 경우 부모 클래스를 추상 클래스로 정의하여 자식 클래스들이 각각 해당 메소드를 구현

05 추상클래스와 인터페이스

• 연습문제

- Shape 클래스를 상속받은 Circle, Triangle, Rectangle 클래스를 정의 하세요.
- Shape 클래스는 추상 클래스로 추상 메소드 area()를 가짐
- Shape 클래스를 상속 받은 각 클래스들은 각각 자신만의 area() 메소드를 구현

• 연습문제

- 다음 두 클래스의 공통된 속성과 기능을 추출하여 부모 클래스 Phone을 정의한 후, Telephone(유선) 과 SmartPhone(무선) 클래스도 정의하고 테스트해 보세요.
- 전원메소드를 trunOn(boolean on) 메소드로 사용하세요

Telephone
String number;
power(Boolean on) call(String number)

Smartphone
String number;
turnOn(boolean on) call(String number) searchInternet(String url)

Phone
String number;
call(String number);

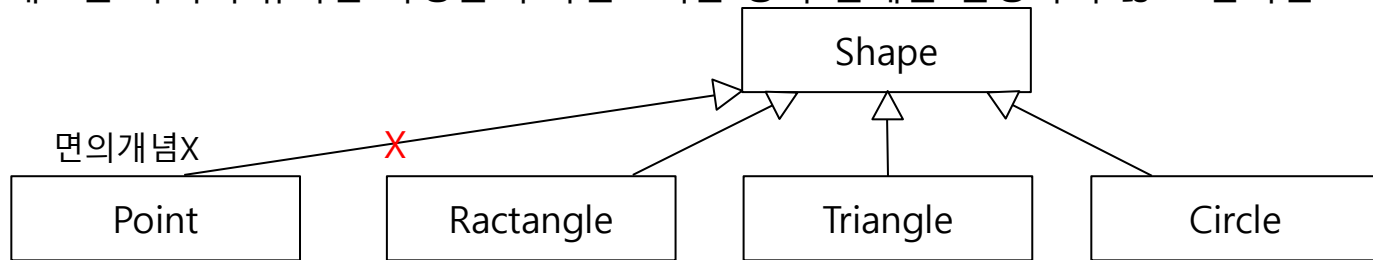
■ 인터페이스

• 개념

- 서로 관계가 없는 물체들이 상호 작용을 하기 위해서 사용하는 장치나 시스템
- 클래스 구조상의 관계와 상관 없이 클래스들에 의해 구현되어질 수 있는 규약

• 목적

- 클래스들 사이의 유사한 특성을 부자연스러운 상속 관계를 설정하지 않고 얻어냄



• 활용

- 하나 또는 그 이상의 클래스들에서 똑같이 구현되어질 법한 메소드를 선언하는 경우
- 클래스 자체를 드러내지 않고 객체의 프로그래밍 인터페이스를 제공하는 경우

- 인터페이스 선언

```
public interface 인터페이스명 {  
  
    //메소드 (선언)  
  
}
```

- 인터페이스 구현

```
public class Point implements 인터페이스명 {  
  
    //메소드 (구현)  
  
}
```

- 인터페이스는 다중 상속을 지원하지 않는 자바에서 다중 상속의 장점을 활용하기 위해 도입

```
public class Point implements Drawable, Resizable {  
  
}
```

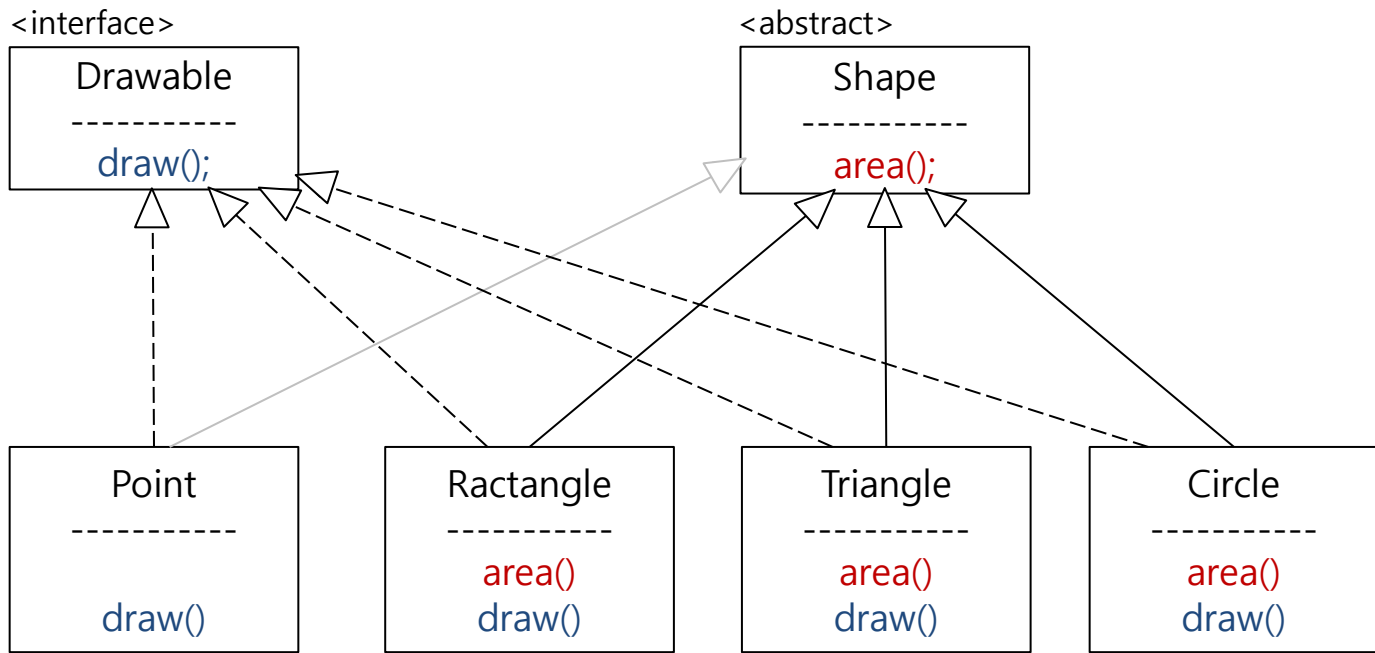
- 연습문제

Shape 예제에서 Point 클래스를 추가하고 Drawable 인터페이스를 추가해 봅니다.

```
public interface Drawable {  
    public void draw();  
}
```

- instanceof 연산자

```
Shape c = new Circle();  
  
// 객체가 Circle 클래스의 인스턴스 인가?  
System.out.println( c instanceof Circle );  
  
// 객체가 Drawable 인터페이스를 구현하였는가?  
System.out.println( c instanceof Drawable );  
  
// 객체가 Rectangle 클래스의 인스턴스 인가?  
System.out.println( c instanceof Rectangle );  
  
// 객체가 Shape 클래스의 인스턴스 인가?  
System.out.println( c instanceof Shape );
```



면의개념이 아님(shape이 될 수 없음)
그림판에서는 같이 관리 되어야 함

05 추상클래스와 인터페이스

■ 일반클래스 vs 추상클래스 vs 인터페이스

	일반클래스	추상클래스	인터페이스
메소드 (Method)	모두 완결한 메소드 *	완결한 메소드 + 추상 메소드	모두 추상 메소드
필드 (Instance Variable)	가질 수 있음	가질 수 있음	가질 수 없음
객체화 (Instantiation)	가능	불가	불가

- 완결한 메소드(concret Method)
 - 메소드의 구현을 포함한 일반적인 메소드를 concret method라 함.
 - 추상 메소드(abstract Method)의 반대개념

chapter02

자바 객체지향 프로그래밍

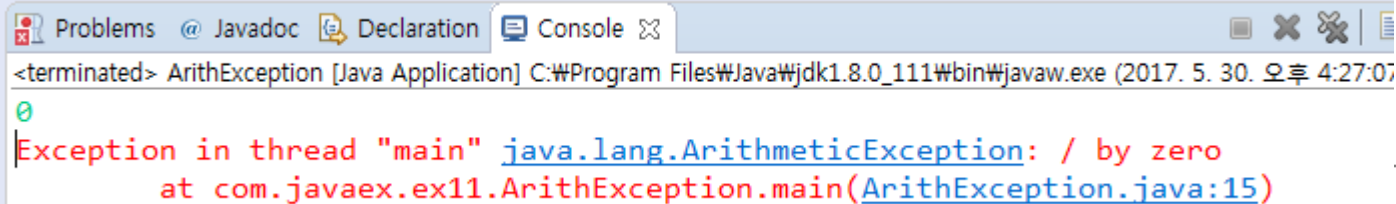
1. 객체지향 기본개념
2. 객체지향 개요
3. 클래스의 정의
4. 상속과 다형성
5. 추상클래스와 인터페이스
6. **예외처리**

■ 예외처리

- 예외(Exception)
 - 프로그램이 실행되는 동안 발생할 수 있는 비정상적인 조건
 - 번역시의 에러가 아닌 실행시의 에러를 예외라 함
- 자바에서의 예외처리
 - 예외처리를 위한 Exception 클래스 정의
 - 기본적인 예외는 자바에 미리 정의된 예외를 통해 처리 가능
 - 사용자가 필요한 예외를 직접 정의할 수 있음
 - 예상되는 예외는 미리 처리해주면 무조건적인 프로그램의 종료를 피할 수 있음
 - 예외처리의 사용은 프로그램의 신뢰성을 높여줌

• 예제

```
public class ArithException {  
  
    public static void main(String[] args) {  
  
        double result;  
        int num;  
        Scanner sc = new Scanner(System.in);  
  
        num = sc.nextInt();  
  
        result = 100/num;    // java.lang.ArithmeticException 발생  
  
        System.out.println(result); // 예외 발생으로 수행되지 않음  
  
        sc.close();  
    }  
}
```



The screenshot shows an IDE's console window with tabs for Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the output of a Java application. The first line is a status message: "<terminated> ArithException [Java Application] C:\#Program Files\Java\jdk1.8.0_111\bin\javaw.exe (2017. 5. 30. 오후 4:27:07)". The second line is an exception message: "Exception in thread "main" java.lang.ArithmeticException: / by zero at com.javaex.ex11.ArithException.main(ArithException.java:15)". The exception message is color-coded: "Exception in thread" is red, "main" is in quotes, "java.lang.ArithmeticException" is blue, "/ by zero" is red, and the stack trace "at com.javaex.ex11.ArithException.main(ArithException.java:15)" is blue.

```
<terminated> ArithException [Java Application] C:\#Program Files\Java\jdk1.8.0_111\bin\javaw.exe (2017. 5. 30. 오후 4:27:07)  
0  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
    at com.javaex.ex11.ArithException.main(ArithException.java:15)
```


- try~catch~finally 구문

0

try {

1

예외가 발생할 가능성이 있는 실행문

} catch (처리할 예외 타입 선언) {

2

예외 처리문

} finally {

3

예외 발생 여부와 상관없이 무조건 실행되는 문장 (생략가능)

}

4

- try 블록에서 예외가 발생한 경우 : **0 -> 1 -> 2 -> 3 -> 4**
- try 블록에서 예외가 발생하지 않은 경우 : **0 -> 1 -> 3 -> 4**

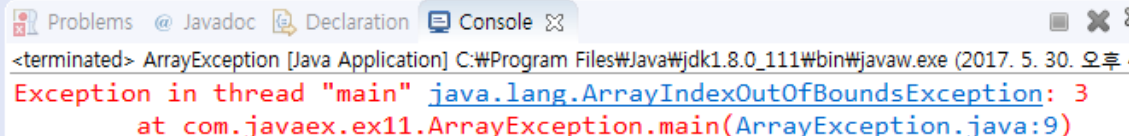
• 예외 클래스

예외 클래스	예외 발생 경우
ArithmeticException	어떤 수를 0으로 나눌 때
NullPointerException	null 객체를 참조할 때
ClassCastException	변환할 수 없는 타입으로 객체를 변환 할 때
ArrayIndexOutOfBoundsException	배열을 참조하는 인덱스가 잘못된 경우
NumberFormatException	문자열이 나타내는 숫자와 일치하지 않은 타입의 숫자로 변환한 경우
IOException	입출력 동작 실패, *인터럽트 발생할 경우

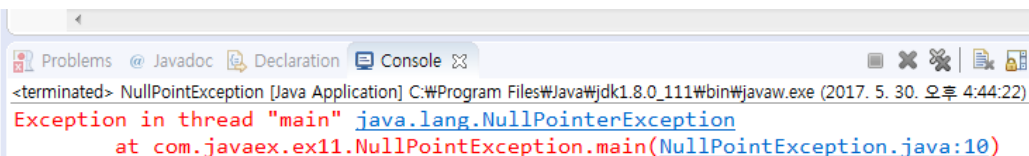
• 연습문제

```
public class ArrayException {  
  
    public static void main(String[] args) {  
  
        int[] intArray = new int[]{3,6,9};  
  
        System.out.println(intArray[3]);  
  
    }  
}
```

```
public class NullPointerException {  
  
    public static void main(String[] args) {  
  
        String str = new String("hello");  
        str = null;  
  
        System.out.println(str.toString());  
  
    }  
}
```



The screenshot shows an IDE window with tabs for Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the following text:
<terminated> ArrayException [Java Application] C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (2017. 5. 30. 오후 4:44:22)
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
at com.javaex.ex11.ArrayException.main(ArrayException.java:9)



The screenshot shows an IDE window with tabs for Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the following text:
<terminated> NullPointerException [Java Application] C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (2017. 5. 30. 오후 4:44:22)
Exception in thread "main" java.lang.NullPointerException
at com.javaex.ex11.NullPointerException.main(NullPointerException.java:10)

- 예외(Exception)의 구분
 - Checked Exception : 컴파일 할 때 확인 되는 예외로 예외처리가 필요함
 - Unchecked Exception : 실행시점에 확인되는 예외로 예외처리를 하지 않아도 컴파일 됨
- 예외처리가 유용한 경우
 - 파일을 다루는 경우
해당 파일이 존재하지 않거나 다른 프로세스에 의해 사용중인 경우 예외 발생
 - 입출력을 다루는 경우
이미 닫힌 입출력 스트림에 대해 작업하려 할 경우 예외 발생
 - 네트워크를 통한 데이터 통신
서버나 클라이언트 한 쪽에서 응답이 없는 경우
네트워크 상태가 안 좋아서 정해진 시간동안 데이터를 받지 못하는 경우

- 메소드 정의시 예외처리
 - 해당 메소드를 호출하는 메소드에서 예외를 처리하도록 명시한다.
 - throws 키워드를 사용하여 예외의 종류를 적어줌

ThrowsExepApp.java

```
public class ThrowsExepApp{  
  
    public static void main( String[] args ) {  
  
        ThrowsExep exep= new ThrowsExep();  
  
        // IO exception발생  
        exep.executeExcept();  
  
    }  
}
```

ThrowsExep.java

```
public class ThrowsExep {  
  
    public void executeExcept() throws IOException {  
  
        System.out.println( "강제예외발생" );  
  
        throw new IOException();    //강제로 예외 발생  
  
    }  
}
```