**< 전체 구조 스케치 >**

Java (Standard Edition) – 1. 기본 프로그래밍

2. OOP(Object Oriented Programming : 상속, 다형성, 캡슐화, 추상클래스, 인터페이스...)

3. API

Database(Oracle, MySQL, MS SQL Server > RDBMS(Relational Database Management System))

--------------------------------------------------------------------------------------------------------------------------J2EE

JDBC : Java로 DB를 연결시키는 API의 일종

Servlet/JSP (container 역할) J2EE(JE, Java 2 Enterprise Edition) : spec set

WAS(tomcat, web logics, Jeus) Java Enterprise Dev. –

EJB (Bean = object, container 역할) 기업을 대상으로 하는 어플리케이션 개발(Enterprise가 붙음)

문제가 많음 특징1. Large Scale Business

Spring (container) – EJB를 대체함 2. Web 기반

객체를 그대로 넣음 3. C/S (Network)

4. DB

5. 분산환경, 자원관리, 컴포넌트

* Container를 통해 Large scale과 컴포넌트 해결!

Java로 Web programming을 하는 것

Client(browser) Server(Servlet/JSP)

HTML/CSS

JavaScript(language)

: JQuery

< 객체지향 – Review>

- 객체(“데이터+메서드”를 가짐)들의 집합 = 프로그램

- 객체는 속성+기능의 집합, 이 때 속성과 기능이 멤버 : 속성은 변수, 기능은 메서드로 정의

- 객체가 인스턴스보다 포괄적이고 넓은 의미로 분석, 설계 단계에서도 사용

<> 코딩 상에서 new 키워드로 만들어내는 것은 주로 인스턴스라고 함

1. 캡슐화

- 코드의 은닉, 숨기기 : 외부 접근 차단, 접근 제한자(private…) 사용 > 외부에서는 메서드로 접근할 수 있게 함

2. 상속

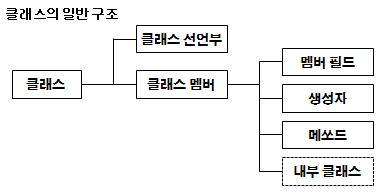
- 코드의 재사용

3. 다형성

3.1 오버로딩(overloading) – 쌓는 것

3.2 오버라이딩(overriding) – 덮는 것

**\* Class**



Class Car{ 클래스 헤더

String name; 멤버 필드

Int speed;

Public void setName() { 멤버 메서드

}

}

**\* 멤버변수**

- 클래스변수, 인스턴스 변수는 클래스 영역에 선언 <> 지역변수는 메서드 영역에 선언

- 지역변수는 수행 후 메모리에서 삭제

- 클래스 변수(static 변수, 공유변수)는 클래스가 메모리에 올라갈 때 선언 > 접근 시 ‘ .ClassName’

- 인스턴스 변수는 인스턴스 생성 시 선언, 지역변수는 변수 선언문 수행 시 선언

\* 생성자 : 인스턴스 초기화(인스턴스 변수에 적절한 값을 저장) 메서드

- 반드시 new 뒤에다가 붙여서 사용 가능

Employee e = new Employee() e는 객체를 참조하는 참조변수(reference 변수)

class Employee{ 생성자

}

- 모든 클래스에는 반드시 하나 이상의 생성자가 필요

- 주소가 아닌 참조값이 반환됨 > 값을 기준으로 찾아감

- 기본 생성자(default)

클래스에 생성자가 없으면 컴파일러가 기본 생성자를 추가함

생성자가 하나라도 있으면 기본 생성자 추가 안 함(따라서 코딩 시 직접 해야 함)

\* 메서드(객체의 행동을 정의하는 procedure, function)

- 메서드 호출은 “dot(.)” 연산자 사용 (‘ .MathodName’)

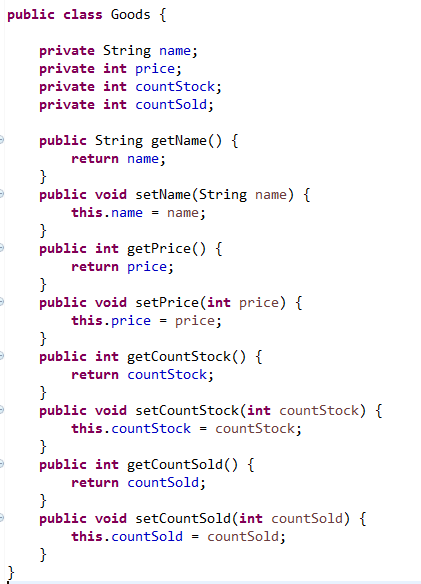
1. public void method1() {.....} object.method1()

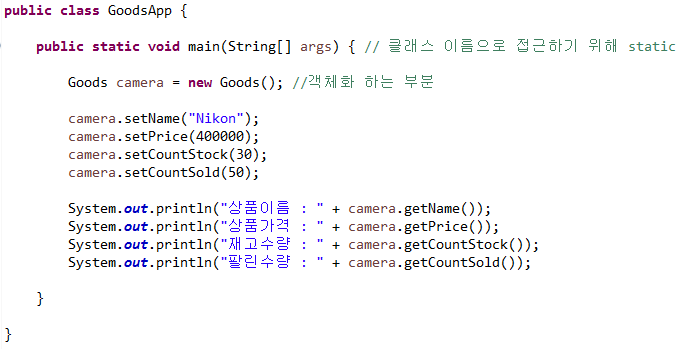
2. public int method(Boolean b) {….} object.method2(true)

3. public int method3(int x, int y, Car t) {.....} object.method3(3,4,car)

- public return\_type methodName (parameters) { … } 의 형식으로 사용

**실습예제 1**





**실습예제 2**

**public** **class** Songs {

**private** String title;

**private** String artist;

**private** String album;

**private** String composer;

**private** **int** year;

**private** **int** track;

**public** String getTitle() {

**return** title;

}

**public** **void** setTitle(String title) {

**this**.title = title;

}

**public** String getArtist() {

**return** artist;

}

**public** **void** setArtist(String artist) {

**this**.artist = artist;

}

**public** String getAlbum() {

**return** album;

}

**public** **void** setAlbum(String album) {

**this**.album = album;

}

**public** String getComposer() {

**return** composer;

}

**public** **void** setComposer(String composer) {

**this**.composer = composer;

}

**public** **int** getYear() {

**return** year;

}

**public** **void** setYear(**int** year) {

**this**.year = year;

}

**public** **int** getTrack() {

**return** track;

}

**public** **void** setTrack(**int** track) {

**this**.track = track;

}

**public** **void** show(){

System.***out***.println(artist+" "+title+"("+album+","+year+","+track+

"번 track,"+composer+" 작곡)");

}

}

**public** **class** SongsMain {

**public** **static** **void** main(String[] args) {

Songs iu = **new** Songs();

iu.setTitle("좋은날");

iu.setArtist("아이유");

iu.setAlbum("Real");

iu.setComposer("이민수");

iu.setTrack(3);

iu.setYear(2010);

iu.show();

}

}

**실습예제3**

**public** **class** Point {

**private** **int** x;

**private** **int** y;

**public** **int** getX() {

**return** x;

}

**public** **void** setX(**int** x) {

**this**.x = x;

}

**public** **int** getY() {

**return** y;

}

**public** **void** setY(**int** y) {

**this**.y = y;

}

**public** **void** show(){

System.***out***.println("좌표 [x= "+x+", y= "+y+"]에 점을 그렸습니다.");

}

}

**public** **class** ShapeTest {

**public** **static** **void** main(String[] args) {

Point shape1 = **new** Point();

Point shape2 = **new** Point();

shape1.setX(2);

shape1.setY(5);

shape2.setX(10);

shape2.setY(23);

shape1.show();

shape2.show();

}

}

**\* static의 활용**

- 전역변수, 전역함수(어디에서나 계속 사용되는 변수나 함수)를 만들 때 사용

모든 클래스에서 공유함

- 객체 생성 필요 없음

- this 사용 불가

Mcqueen Name = “My Car”

speed = 0

Hudson Name = “My Car”

speed = 0

Car  *numberOfCars* numberOfCars는 변수 저장 위치가 다름

(static으로 선언하였으므로)

객체와 상관없이 항상 사용 가능

((code))

**class** Car {

String name;

**int** speed;

**static** **int** *numberOfCars*; // numberOfCars는 전역변수

**public** Car() {

name = "My Car";

speed = 0;

*numberOfCars*++;

}

**public** **void** setName(String name) {

name = name;

}

**public** **void** setSpeed(**int** s) {

speed = s;

}

**public** **int** getSpeed() {

**return** speed;

}

}

**public** **class** CarTest {

**public** **static** **void** main() {

Car mcqueen = **new** Car();

System.***out***.println(Car.*numberOfCars* + " 대의 차가 생산되었습니다.");

Car hudson = **new** Car();

Car marter = **new** Car();

System.***out***.println(Car.*numberOfCars* + " 대의 차가 생산되었습니다.");

}

}

**\* JVM 메모리 구조**

메서드 클래스 static 변수 Method Area

필드 정보 static Method

Heap

Stack

((오류 코드))

**public** **class** StaticMethod {

**int** *n*;

**static** **int** *m*;

**void** f1(**int** x) { *n* = x; }

**void** f2(**int** x) { *m* = x; }

**static** **void** s1(**int** x) { *n* = x; }

**static** **void** s2(**int** x) { *f1*(3); }

**static** **void** s3(**int** x) { *m* = x; }

**static** **void** s4(**int** x) { *s3*(3); }

}

((수정 코드))

**public** **class** StaticMethod {

**static** **int** *n*;

**static** **int** *m*;

**static** **void** f1(**int** x) { *n* = x; }

**void** f2(**int** x) { *m* = x; }

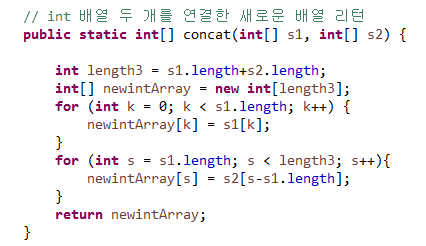
**static** **void** s1(**int** x) { *n* = x; }

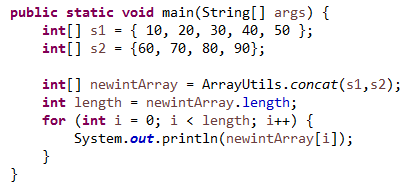
**static** **void** s2(**int** x) { *f1*(3); }

**static** **void** s3(**int** x) { *m* = x; }

**static** **void** s4(**int** x) { *s3*(3); }

}





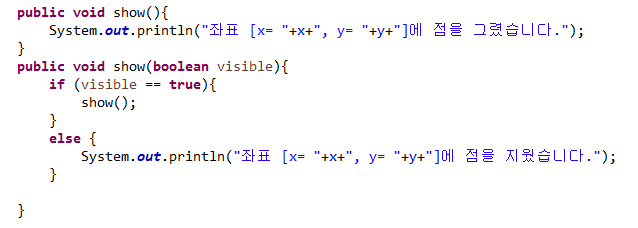
**public** Songs(String title, String artist){ this() 사용법

**this**(title, artist, "","",0,0);

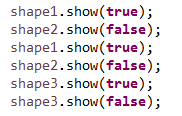
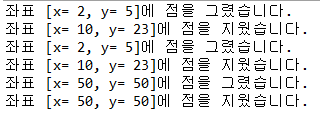
}

**\*메서드 오버로딩**

실습예제 6



((result))

**\* 상속**

- 코드 재사용성 측면에서 사용하는 것이 유리

- OO is a kind of XX : 말이 될 경우에 상속 관계

- 학생은 말하고 먹고 걷고 자고 공부하는 것이 가능

사람 학생

말하기() 공부하기()

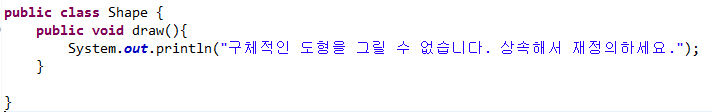
먹기()

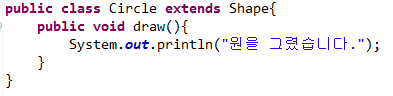
걷기()

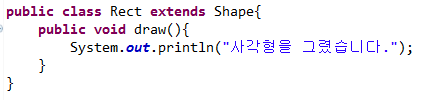
잠자기()

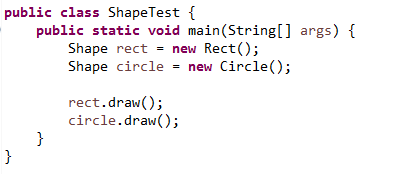
**\* 메소드 오버라이딩**

- 부모클래스의 메서드를 동일한 이름으로 재작성 > 부모클래스 메서드를 무시함









((result))



**\* 객체의 형변환 – 업캐스팅 & 다운캐스팅**

- 큰 거에서 작은 거로의 캐스팅(업캐스팅)은 가능 (ex. (int) double 가능)

- 다운캐스팅의 경우에는 명시적으로 캐스팅 해줘야 함

- 상위 레퍼런스 타입은 자식 객체를 참조(업캐스팅)할 수 있으나 반대의 경우는 명시적 캐스팅 필요

Shape s = new Rect() 가능

**\* 추상클래스**

- 추상 메서드를 포함하기 때문에 객체화 할 수 없음.

- 추상 메서드는 추상 클래스에만 존재할 수 있음

- 추상 메서드는 메서드에 해한 구현을 가지지 않음

abstract class Aclass{

public abstract void Amethod() ; 추상 메서드를 구현하지 않음

}

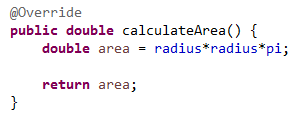
- 추상 클래스의 자식 클래스는 부모 클래스에서 구현하지 않은 추상 메서드를 반드시 구현해야 함

abstract class Bclass extends Aclass {

public abstract void Bmethod() {…..} 추상 메서드를 구현

}

- @Override 를 붙이면 override 가능성 여부를 알려줌!



**\* 인터페이스**

class Parking System {

void park(ICar car) {

car.달리다();

car.가속하다();

car.방향바꾸다();

car.멈추다();

}

}

ICar

달리다( )

가속하다( )

방향바꾸다( )

멈추다( )

SportsCar

Bus

- 클래스들 사이의 유사 특성을 부자연스러운 상속 관계를 설정하지 않고 얻어냄

Shape을 상속받는 Point에서는

Shape

String lineStyle

darw( )

calculateArea( )

Circle

lineStyle이 부자연스러움

Drawable

draw( )

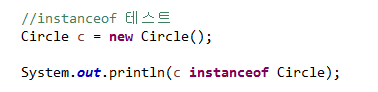
그러나 draw( )는 필요

Point

<<interface>>

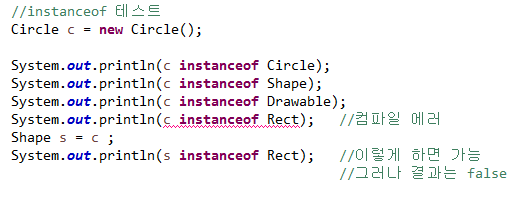
* 인터페이스를 통해 해결 가능

- instanceof

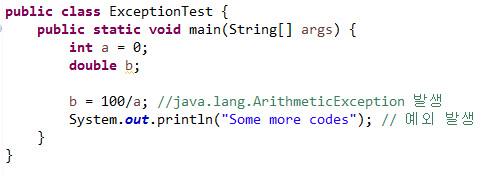


* 결과는 true 로 나옴

Instanceof를 쓰기 위해서는 반드시 상속관계이어야 함(컴파일 에러, 아래 예시)



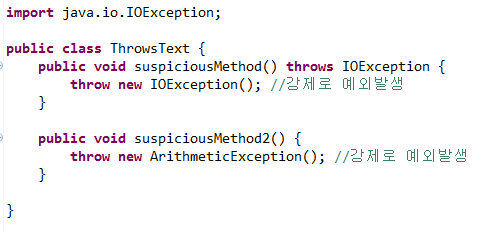
**\* 예외처리**

****

1. **Checked Exception : 컴파일 에러 발생하므로 예외처리가 필요**
2. **Unchecked Exception : 실행 에러 발생하므로 예외처리 안 해도 컴파일 가능**

**- 배열과 연관된 for 반복문, 파일과 입출력을 다루는 경우 예외처리가 유용!**

**- 위험요소가 있는 메서드의 경우에는 throws 키워드를 사용하여 예외 종류를 적어 강제로 try-catch 수행**

****