Java

객체

클래스& 객체

정의

- 클래스(class)
  - 클래스는 객체를 정의 한다
  - 객체의 모든 속성과 기능이 정의 되어 있다
    a. 서로 관련된 변수들을 정의하고 이들에 대한 작업을 순행하는 함수들을 함께 정의한다.

- 객체(Object)
  - 객체(Object)는 다수의 속성과 다수의 기능을 갖는다.
  - 클래스에 정의된 내용대로 메모리에 생성된 것

- 인스턴스(Instance)
  - 클래스로부터 만들어진 객체

클래스& 객체

정의

- 클래스(class)의 기본 구성
  - 접근 지정자 + 멤버 변수 + 멤버 메소드

- 모든 코드는 class안에 포함되어야 한다
  - 클래스 밖에서 쓰일 수 있는 것은 package, import, 또 다른 클래스

클래스& 객체

패키지(package)

- 패키지(package)
  - 서로 관계가 있는 클래스들의 묶음
  - 물리적으로 클래스 파일을 포함하는 하나의 디렉토리이다
  - 사용자 정의 클래스
  - 라이브러리 클래스

- 하나의 소스파일에는 첫 번째 문장으로 단 한 번의 패키지 선언 만을 허용한다.

- 모든 클래스는 반드시 하나의 패키지에 속해야 한다
  - 패키지에 속한 클래스는 해당 디렉토리에 존재하는 class 파일이어야 한다
  - 동일한 이름의 클래스가 서로 충돌하는 것을 피할 수 있다

클래스& 객체

패키지(package)

- 패키지 선언
  - Package 패키지 명;
- 사용자 정의 패키지를 만들려면 클래스의 첫 부분에 패키지 구문을 포함 시킨다
  - Package userpackage;
- 최상위 패키지의 이름은 java로 시작 할 수 없다
  - Java패키지는 표준 API들의 모음을 구성해 놓은 패키지
- 대분류와 소분류 패키지를 구분할 때는 '.'을 사용한다
  - Java.lang.String
  - 클래스의 이름은 패키지 명을 포함한다
- 예약어를 패키지 이름으로 사용할 수 없다
- 패키지를 선언하지 않으면 기본적으로 제공하는 '이름없는 패키지'에 포함된다

클래스& 객체

import

- 다른 패키지의 클래스를 사용하려면 패키지 명이 포함된 클래스 이름을 사용해야 한다.

- 소스 파일에 사용된 클래스의 패키지에 대한 정보를 컴파일러에게 제공
  - 다른 패키지에 있는 외부 클래스를 사용하고 싶은 경우 import키워드를 사용하여 해당 클래스의 전체 경로를 지정 해 줘야 한다.
  - Import문으로 클래스의 패키지를 명시해 주면 소스코드에 사용되는 클래스 이름에서 패키지 명을 생략할 수 있다
  - 같은 패키지 내의 클래스들은 import문을 지정하지 않고 생략할 수 있다

클래스& 객체

import

- Import문 선언
  - package문 다음에, 그리고 클래스 선언 문 이전에 선언해야 한다
  - 한 소스파일에 여러 번 선언 할 수 있다.
  - import 패키지명.클래스명;
  - import 패키지명.*;

- 클래스 이름 대신 *를 사용하는 것이 하위 패키지에 적용되지 않는다

클래스& 객체

객체 구성 요소

- 클래스로부터 객체를 생성하면 클래스에 정의된 속성과 기능을 가진 객체가 만들어진다.
  - 객체는 다수의 속성과 다수의 기능의 집합

- 속성(property)
  - 멤버 변수(member variable), 특성(attribute), 필드(field), 상태(state)

- 기능 (function)
  - 메서드(method), 행위(behavior), 함수(function)

클래스& 객체

객체 구성 요소

- 클래스와 객체

객체의 선언	객체의 생성	객체의 선언과 생성
클래스명 객체참조변수 (변수이름);	객체참조변수 (변수이름) = new 클래스명();	클래스명 객체참조변수 (변수이름) = new 클래스명();

레퍼런스 변수,
클래스객체

클래스& 객체

인스턴스

- 클래스로부터 만들어지는 각각의 객체를 그 클래스의 인스턴스(instance)라고 한다.
  - 인스턴스는 참조변수를 통해서만 다룰 수 있다
  - 참조 변수의 타입은 인스턴스의 타입과 일치해야 한다.

- 자신을 참조하고있는 참조 변수가 하나도 없는 인스턴스는 가비지 컬렉터(Garbage Collector)에 의해서 자동적으로 메모리에서 제거 된다.

- 인스턴스 생성
  - 클래스명 변수명; //클래스의 객체를 참조하기 위한 참조변수를 선언한다
  - 변수명 = new 클래스명(); //클래스 객체를 생성 후, 객체의 주소를 참조변수에 저장

  - Info data //Info클래스 타입의 참조변수 data를 선언
  - data = new Info(); //Info인스턴스를 생성한 후, 생성된 Info인스턴스의 주소를 data에 저장

변수와 메서드

변수와 메서드

- 변수가 선언된 위치에 따라 변수의 종류가 결정 된다.
  - 인스턴스 변수
  - 클래스 변수
  - 지역 변수

- 인스턴스 멤버
  - 클래스 영역에 선언된 멤버들

- 클래스 변수
  - 멤버들 중 static이 키워드가 붙어 있는 것은 인스턴스 변수

- 지역 변수
  - 인스턴스 변수를 제외한 나머지 변수들
  - 메서드 내에서 변수의 선언문이 실행되었을 때 생성된다

변수와 메서드

인스턴스 멤버

- 인스턴스 멤버
  - 인스턴스가 생성 되었을 때 생성 된다
  - 클래스 영역에서 선언
  - 인스턴스마다 독립적인 저장 공간을 갖는다
  - 인스턴스 멤버를 참조 또는 호출하기 위해서 인스턴스를 생성해야 한다.

변수와 메서드

정적 멤버(static member)

- 모든 객체를 통틀어서 하나만 생성되고 모든 객체가 이것을 공유하게 된다.
  - 인스턴스를 생성하지 않고도 사용 할 수 있다.

- 정적 변수(클래스 변수)
  - 모든 객체에 공통인 변수이다.
  - 하나의 클래스에 하나만 존재한다
  - 클래스 영역에서 선언

- 정적 메소드
  - static 수식자를 메소드 앞에 붙이며 클래스 이름을 통하여 호출되어야 한다.
  - 모든 인스턴스에 공통적으로 사용해야 하는 것에 static를 붙인다.

변수와 메서드

지역 멤버

- 지역 멤버
  - 변수가 선언 되었을 때 생성 된다
  - 클래스 영역 이 외의 영역
    a. 메서드
    b. 생성자
    c. 초기화 블록 내부

변수와 메서드

메서드

- 메소드
  - 어떤 작업을 수행하기 위한 명령문의 집합
  - 입력을 받아서 처리를 하고 결과를 반환한다
  - 하나의 메서드는 한 가지 기능만 수행하도록 작성

변수와 메서드

메서드

- 작성 방법

```
리턴 타입 메서드이름(타입 변수명,타입 변수명,...){
    //메서드 호출시 수행될 코드
}

int add(int a, int b){
    int result = a + b;

    return result;    //return데이터 타입은 자동형변환이 가능한 byte, short, char 중 하나
}

void power(){
    power = !power;
}
```

변수와 메서드

메서드 호출

- 메서드 호출
  - 참조변수.메서드이름();
  - 참조변수.메서드이름(값1, 값2,…);

- 같은 클래스의 메서드끼리는 참조 변수를 사용하지 않고 호출 가능

```
void addArray(int a, int b, int[] result) {
        result[0] = a+b;
        result[1] = a+b + 10;

        result[2] = add(a,b) + result[1] ;
}
```

변수와 메서드

메서드 호출

- static메서드는 같은 클래스 내의 인스턴스 메서드를 호출할 수 없다

```
int add(int a, int b) {
        return a+b;
}

static void addArray(int a, int b, int[] result) {
        result[0] = a+b;
        result[1] = a+b + 10;
        result[2] = add(a,b) + result[1] ; //호출할 수 없다
}
```

변수와 메서드

접근 제어자

- 캡슐화(encapsulation)
  - 외부로 부터 데이터를 보호하기 위해 접근을 제한한다

- 정보 은닉(information hiding)
  - 객체를 캡슐로 싸서 객체의 내부를 보호하는 하는 것이다.
  - 객체의 실제 구현 내용을 외부에 감춘다
  - 외부에서 접근할 필요가 없는 멤버들을 private로 지정하여 접근을 차단한다

- 다형성

변수와 메서드

접근 제어자(access modifier)

- 외부에서 멤버 또는 클래스에 접근하는것을 제한 한다
  - private : 같은 클래스 내에서만 접근 가능
  - default : 같은 패키지 내에서만 접근 가능
  - protected : 같은 패키지, 그리고 다른 패키지의 자손 클래스에서 접근 가능
  - public : 접근 제한이 없다

- 사용 가능한 접근 제어자
  - 클래스 : public, default
  - 메서드/멤버 변수 : public, protected, default, private
  - 지역 변수 : 사용하지 않는다.

변수와 메서드

접근 제어자(access modifier)

- setter
  - 멤버 변수의 값을 세팅하는 메소드
  - 매개변수명은 일반적으로 멤버변수명과 동일하게 적는다
  - 상황에 따라 줄임말을 적거나 앞에 '_'를 붙이기도 한다

- 형식

```
void set멤버변수명(매개변수) {
        코드정의;
}
```

변수와 메서드

접근 제어자(access modifier)

- getter
  - 멤버 변수의 값을 반환하는 메소드

- 형식

```
반환자료형 get멤버변수명() {
        코드정의
        return 멤버변수명;
}
```

변수와 메서드

기본형 & 참조형 매개 변수

- 메서드를 호출할 때 매개 변수로 지정한 값을 복사해서 전달한다

- 메소드로 기본형 변수가 전달되는 경우
  - 기본형 값이 복사 된다.
  - 변수의 값을 읽기만 할 수 있다.

- 메소드로 객체가 전달되는 경우
  - 객체가 복사되어 전달되는 것이 아니고 참조 변수의 값이 복사되어서 전달된다.
  - 변수의 값을 읽고 변경 할 수 있다
  - 인수와 매개변수 모두 동일한 객체를 가리킨다

- 메소드로 배열이 전달되는 경우
  - 배열도 객체이기 때문에 배열을 전달하는 것은 배열 참조 변수를 복사하는 것이다.

변수와 메서드

메소드 오버로딩(method overloading)

- 메소드 오버로딩(method overloading)
  - 한 클래스 내에 같은 이름의 메서드를 여러 개 정의할 수 있다
  - 메서드의 타입을 구분하기 위해서 메서드 명과 매개변수의 정보를 사용한다

- 오버로딩 성립 조건
  - 메서드 이름이 동일해야 한다
  - 매개변수의 개수 또는 타입이 달라야 한다
  - 리턴 타입은 함수 타입 구분 정보에 포함되지 않는다.

변수와 메서드

메소드 오버로딩(method overloading)

- 잘못된 메소드 오버로딩 예

```
int add(int a, int b) {
        return a+b;
}

int add(int x, int y){
        return x+y;
}
```

변수와 메서드

메소드 오버로딩(method overloading)

- 잘못된 메소드 오버로딩 예

```
int add(int a, int b) {
        return a+b;
}

long add(int a, int b){
        return (long)(a+b);
}
```

변수와 메서드

메소드 오버로딩(method overloading)

- 올바른 메소드 오버로딩 예

```
int add(int a, long b) {
        return a+b;
}

long add(long a, int b){
        return a+b;
}
```

변수와 메서드

메소드 오버로딩(method overloading)

- 올바른 메소드 오버로딩 예

```
int add(int a, int b) {
        return a+b;
}

long add(long a, long b){
        return a+b;
}

int add(int [] a){
        int result = 0;

        for(int i=0; i<a.length; i++){
                result += a[i];
        }
        return result;
}
```