modifier

		class	변수	메소드	생성자
public	모든 클래스 접근	o	o	o	o
protected	상속받은 클래스에서 접근	x	o	o	o
(default) =(package) =(no modifier	같은 패키지에서 접근	o	o	o	o
private	같은 클래스에서만 접근	x	o	o	o
static	객체생성없이 사용	x	o	o	x
final	상속 오버라이딩 불가능 상수로만 사용	o	o	o	x
abstract	상속 오버라이딩 의무화	o	x	o	x

- 타입별 배열

```
int [] i = {1,2,3,4,5}
char [] c = {'a','b','c','d'}
double d[] = {1.1, 2.2, 3.3}
String s[] = {"aa", "bb", "cc"}
```

- 상속

class A{} class B{} class C{} interface I {} interface J{}	class D extends A implements I, J{} -> 가능 class D extends A,B,C implements I, J{} ->불가능 class D extends A,B implements I{} -> 불가능 class D extends C implements I{} -> 가능 클래스 단일상속, 인터페이스 다중상속 구현

형변환

class Parent{} class Chidl1 extends Parent{} class Child2 extends Parent{} class Child3 extends Parent{}	Child1 c1 = (Child1)new Child2();->컴파일오류 Child1 c1 = (Child1)new Child3();->컴파일오류 Child3 c3 = (Child3)new Child1();->컴파일오류 Child1 c1 = (Child1)new Parent();->컴파일가능.실행오류 ->상속관계가 존재하면 형변환 연산자 사용은 가능하지만 실행시 생성된 객체 타입(new Parent())을 체크하여 실행오류를 발생할 수 있다. Parent p1 = new Child1(); Child1 c1 = (Child1) p1; --> 컴파일가능.실행가능 ->상속관계가 존재하면 형변환 연산자 사용 가능하므로 컴파일 가능. 실행시 생성된 객체 타입(new Child1())을 체크하여 실행가능.

- super와 this

```
class A { int i = 10; }
class B extends A{
int i = 20;
void test(){
 super.i 출력 -->10
 this.i 출력---> 20
  i 출력 ---> 20
  B.i 출력 --> static변수 아니므로 불가능
}
```

- 제네릭

컴파일시 타입 미리 정적으로 결정 타입 매개변수를 가지는 클래스 - 클래스명<타입> 명시적 형변환 작성안해도 됨.

- String 동등비교

== 연산자	equals() 메소드

String s1 = new String("java"); String s2 = new String("java"); s1 == s2 ==> 두 문자열 객체 주소 비교	String s1 = new String("java"); String s2 = new String("java"); s1.equals(s2) ==> 두 문자열값 비교

- 생성자

```
class Test{
  public static void main(String args[]){
     A a1 = new A(); ==> 컴파일오류
    a1.i = 20;
    a1.mul();
  }
}

class A{
  int i = 0;
  A(int i) {
  this.i = i;
  }
  int mul(){
     return i *i;
  }
}
```

- 기본 생성자 자동 정의
- 사용자가 생성자 정의하면 기본 생성자 자동 삭제

- 메소드 overloading

```
public void method(int i){ }

public void method(int i, int j){ } -> overloading 가능
public int method(int i, int j){ } -> overloading 가능
public String method(int i){ } -> overloading 불가능
public void method(int j){ } -> overloading 불가능
```

- 1개의 클래스에 같은 이름의 메소드 여러개 정의
- 매개변수 갯수, 타입, 순서 다르다
- 리턴타입 상관없다
- 매개변수 이름 상관없다

- JDBC SELECT 순서

JDBC DRIVER 로드
Connection 생성
Statement or PreparedStatement 생성
ResultSet 리턴
Connection close

- JDBC 구현 순서

java.sql 패키지 사용

JDBC DRIVER 로드

Connection 생성
Connection con =DriverManager.getConnection("jdbc url", "계정", "암호");

Statement or PreparedStatement 생성
-Statement 이용한 dml 실행
Statement st = con.createStatement();
int rows = st.executeUpdate("insert | update | delete ");

- PreparedStatement 이용한 dml 실행
PreparedStatement pt = con.prepareStatement("? 포함한 insert | update | delete ");
pt.setXXXX(물음표인덱스, 값);
int rows = pt.executeUpdate();

-Statement 이용한 select 실행
Statement st = con.createStatement();
ResultSet rs = st.executeQuery("select..");

- PreparedStatement 이용한 select 실행
PreparedStatement pt = con.prepareStatement("? 포함한 select ");
pt.setXXXX(물음표인덱스, 값);
ResultSet rs = pt.executeQuery();

ResultSet은 최초에 첫번째 레코드 이전을 참조하므로
next 호출후 첫번째 레코드로 이동한다

Connection close

-sql

create table float(10, 2) --> number(10, 2)
select where 문자타입컬럼= '문자열값' (db에 따라 where 문자타입컬럼= "문자열값" 도 가능) where 숫자타입컬럼= 숫자값