자바
*라이브러리: 패키지 단위로 되어 있음
*패키지: 비슷한 기능의 클래스 파일을 모아 둔 디렉토리
*java API -> 자바개발자들에게 자바개발 도움 되기 위한 소스

- 주석 -> 설명 , 컴파일 안된다. -> 다른 프로그래밍 언어에서도 필수 사용
// 한줄
/*여러줄*/

-변수 -> 데이터 처리 -> 하나?
--- 값 하는 저장 -> 기본타입
--- 값 (주소) -> 객체 타입(Object) : 클래스, 배열, String,인터페이스

int i=10;

String str1=new String();

new 생성자();

1. 변수
1) 하나의 데이터를 저장할 수 있는 메모리(저장) 공간
2) 변수는 두가지 의미가 있다. 1) 주소 2) 주소의 참조값

*변수 명명 규칙 (변수 이름 정하는 규칙)
1. 키워드(예약어)를 사용 X
2. 숫자로 시작 X
3. 공백 X
4. _ $ 이외의 특수문자 X
5. 대소문자 구분(약속)
6. 소문자로 시작(약속)
7. 다른 문자가 시작되면 대문자로 시작(약속)
8. 한글 사용 안함(약속)

**** 클래스 맴버
1. 인스턴스 멤버 -> 클래스 내
2. 클래스 멤버 -> static -> 공유 (따로 패키에 저장)
3. 지역 변수 -> 매서드안에

4. 매개(인자형) 변수:　　메소드 선언할 때 괄호 안에 들어감 (지역변수)

****객체참조 변수: 객체의 주소값을 저장

*자바의 기본 자료형(primitive type)

	1	2	4	8
정수형	byte	short	int	long
실수형			float	double
문자형		char		
논리형	boolean			

*참조형(객체형 object) = reference type
: String, class, 배열, interface

*기본타입 변수는 값을 직접 대입 / 참조타입 변수는 객체의 주소값을 대입(new)
*class: 객체를 설계하기 위한 설계도, 사용자 정의 타입

new -> 객체 생성 연산자

String str1=new String("문자열");

*상수: 수식에서 변하지 않는 값
- 선언: final 상수타입 상수명;　　ex) final int MAN;
- 파이널 상수: 한 번만 초기화, 대문자로 씀, static과 함께 사용
public static final int ROLE_ADMIN=1;
public static final int ROLE_MEMBER=2;

*** 형변환 -> 프로젝트시 형변환이 자동으로 힌트를 준다

*연산자(operator): 연산기호
- 연산: 처리한다, 수행한다, 로직을 구현한다

1. 단항 연산자: 항 하나 ex) i++; ++i;　　증감 연산자

2. 이항 연산자: 항 둘 ex)
3. 삼항 연산자: 항 셋 ex) ? :

객체==객체
: 주소값을 비교한다.

객체.equal(객체)
:대상의 내용을 비교한다.

3) 논리 연산자 && || ^ ! , T(True), F(False)

4) 부정 연산자 ! (toggle, 스위치)

5) 대입연산 -> 오른쪽의 연산의 결과를 왼쪽 변수에 대입, 연산의 마지막에 설정

6) 비트연산자 0 1 & | ~

7) << >> 쉬프트 연산자

****

Wrapper클래스
박싱,언박싱

** 아스키 코드
..
65 A 'A' (int)'A' 65

97 a

**** 조건문 -> 문제를 생각하면 풀어본다. (필기도구)
*if
* switch ~ case

```
int key=1;
switch(key){
        case 1:
                처리문(1);
                break;
```

```
        case 2:
                처리문(2);
                break;
        case 3:
                처리문(3);
                break;
        default:
                나머지처리문;

}
```

if 범위
switch 모든 switch-> if

** 반복문
1.for -> 갯수가 정해 졌다. -> 객체형 반복몬(foreach문,forEach())
2.while
****DB에 DATA를 객체화 시킬 때 많이 사용
** 조건이 ture 이면 무조건 실행
**무한루프를 조심(무한반복)

3.do~while -> while문과 같다. 단. 무조건 한번은 실행
** 정보화 관련 시험

```
do{
        //처리문
        증감식;
}while
int i=2;
do{
        // 실행문;

}while(i<1);
```

* java 객체형 for

```
for (Object ob : Object List(ArrayList<Object> , List<Object>) {
        //객체형 리스트(반복문)
}
```

*참조타입(reference type)
- 생성된 객체의 주소값을 대입
- 객체.멤버 로 접근

*배열 -> 같은 타입을 그룹화, 순서가 있고 갯수가 정해졌다.
1. 같은 type의 data를 순서대로 그룹화
2. 인덱스(번지, 요소)는 0부터 시작
3. 배열의 개수는 고정한다 - 컬렉션...고정X
4. 배열 참조형 타입(배열의 전체 주소값으로 접근) - new 배열
5. 배열은 생성과 동시에 초기화해야 한다.
6. 값을 초기화하지 않으면 0으로 초기화된다
7. 반복문(for)과 같이 자주 사용
8. 전체 배열의 데이터의 개수는 전체 인덱스 총 수보다 1 크다.
9. 배열 이름에 전체 배열의 주소값을 저장
10. for~in: DB의 데이터를 가져올 때

*클래스
- 사용자 정의 타입 ex) DBMemberDto member;
- 객체를 설계하기 위한 설계도(틀)

*객체
- 클래스를 통해 만들어진 현실 사물체
- 현실 세계의 모든 것을 나타냄

1.배열(참조형타입) -> List<T> 배열형 컬렉션
  >같은 타입의 DATA를 순서대로 그룹화

  1)요소는 0부터 초기화
  2)배열은 이름에 전체 배열의 주소값이 저장
  3)배열은 선언과 동시에 초기화 해야된다.
  4)반복문과 함께 많이 쓰인다.
  5)초기화를 안할시 0으로 자동 초기화된다.

**초기화 방법
```
int[] arrInt=new int[5];
arrInt[0]=10;
arrInt[1]=20;
arrInt[2]=30;
arrInt[3]=40;
arrInt[4]=50;
```

//

```
int[] arrInt=new inr[]{1,2,3,4,5};
```

//

int[] arrInt={1,2,3,4,5};

** 모든 클래스는 Object클래스를 상속(inheritance) 받는다.
** 모든 클래스는 Object클래스의 모든 속성과 매서드를 상속 받았다.

*** get -> "가져온다~~"
*** set -> "설정한다~~"

매소드 오버로딩(Overloding) **자바에서만 허용

한 클래스 내에서 두 개 이상의 이름이 같은 메소드 작성
 >메소드 이름이 동일하여야 함
 >매개 변수의 개수가 서로 다르거나 ,타입이 서로 달라야 함
 >리턴 타입은 오버로딩과 관련 없음

---
class 맴버
필드 인스턴스, 클래스
매서드 인스턴스, 클래스

***static 맴버(클래스맴버) -> DB Connect(DB연동클래스등)
>>클래스당 하나만 생성
>>객체들에 의해 공유됨★★ 클래스이름.클래스맴버
>>프로그램이 종료될 때 사라짐

***instanse 맴버-> 객체를 생성한후(인스턴스화)에 사용 가능
>>객체별로 별도 존재
>>객체 생성 시에 맴버 생성됨
>>공유되지 않음★★
>>객체 사라지면 맴버도 사라짐

클래스의 맴버

필드 -> 하나의 값을 setting하거나, get
매서드 -> 어떤 기능, 처리, 구현 , 호출

** return문을 만나면 종료
** 구현부가 있으면 구현 하고 return값을 반환하고 종료
** call(호출)할 때마다 새로 생성 된다.

클래스
접근제한자 class 클래스명{
맴버;
}

필드
접근제한자 타입 필드명;

매서드
//선언부
접근제한자 반환타입 매서드명(입력인자){
// 구현부
return 반환값;
}

```
// 매서드 유형1
//void return값이 없다.생략가능
public void 매서드명(){
  //구현부
   // ruturn;
}
// 매서드 유형2
//void return값이 없다, 입력값이 있다.
public void method2(int num1,int num2) {
   // 입력 값이 있다.
  int sum=num1+num2;// 매서드 안에 선언된 변수(지역변수)
   // ruturn;
}
// 매서드 유형3
// 반환타입 -> 반드시 return값이 있어야된다.
// 입력값이 없다.
public int method3() {
       return 반환값(반환타입과 타입이일치);
}
// 매서드 유형4
// 반환타입 -> 반드시 return값이 있어야된다.
// 입력값이 있다.
public int method4(입력) {
        //구현부
```

```
        return 반환값(반환타입과 타입이일치);
}
```

== 접근 제한자(modify) ==
public 모든 클래스에서 접근 가능(공용)
private 같은 클래스내에서만 접근 가능

** 기본적으로 프로그래밍언어에서 이름(매서드,변수)같은 경우 같은 기억 공간에서는
허용 되지 않는다.
** 다만. 예외적으로 자바에서 메서드 오버로딩을 허용하고

== 매소드 오버로딩 == ** 매서드명, 타입, 매개변수 갯수가 완전히 일치 하면 허용 X
오버로딩(Overloading)
한 클래스 내에서 두 개 이상의 이름이 같은 메소드 작성
.매소드 이름이 동일하여야 함
.매개 변수의 개수가 서로 다르거나, 타입이 서로 달라야함
.리턴 타입은 오버로딩과 관련 없음
.구현부와 상관없다.

자바에서 허용..
오버로딩 "매서드의 이름이 같고 매개변수 유형이 다른 매서드를 허용"

생성자 오버로딩..

객체지향 ----------(정리 잘해야 됩니다~~)
*** 클래스, 맴버, 필드, static, instance, private , 매서드
*** 생성자(contruct) , 상속 , 추상클래스, 인터페이스
*** 패키지, 접근제한자(지정자), 오버로딩(매서드오버로딩), 오버라이딩, 다형성

github-> repository 생성
새로운 폴더 만들고 -> git 설치
오늘 배운 .java파일을 github에 add -> commit ->push

==========

[생성자(Construct)]
1. 클래스명과 이름이 같은 매서드

2. 반환 타입이 무조건 void >> void 생략 public 매서드명
3. 객체를 생성할 때 사용
4. 오버로딩 허용(매서드명이 같고 입력인자 구조 다른것)
5. 기본 생성자(default)는 자동으로 생성해준다.
6. 다른 생성자를 생성할 시 기본 생성자는 자동으로 생성하지 않는다.
7. 다른 생성자를 호출 시 this()로 호출
8. 다른 생성자 호출 시 반드시 생성자의 첫 번째에 위치

** 생성자는 객체 생성 시 한 번만 호출

<<상속(inheritance) >>
** 객체 지향 상속(자식이 부모 유전자(맴버)를 물려 받는 것과 유사한 개념)
1. “부모(슈퍼)클래스의 모든 맴버(필드, 매서드)를 자식(서브)클래스에 물려 주는 것”
2. 클래스간의 단일 상속만 가능 **
3. 자식클래스의 맴버의 수는 반드시 부모클래스보다 크거나 같다.
4. 자식 클래스 extends 부모클래스
5. 모든 생성자를 상속하지는 않는다.(생성자와 초기화 블록은 상속되지 않는다. 멤버만상속된다.)
6. 상속 시 기본생성자는 자식생성자 첫 번째 줄에 자동으로 생성해준다.
7. Object 클래스는 모든 클래스의 슈퍼 클래스
8. 자식클래스의 접근제한자는 반드시 부모클래스와 같거나 커야된다.
* 조상 클래스가 변경되면 자손 클래스는 자동적으로 영향을 받게 되지만,
자손 클래스가 변경되는 것은 조상 클래스에 아무런 영향을 주지 못한다.

**자바상속의 특징
자바는 클래스 단일 상속만 가능
C++은 다중 상속 가능(다중 상속으로 맴버가 중복 생성되는 문제가 있다)

***자바는 인터페이스(interface)의 다중 상속 허용
***모든 자바 클래스는 묵시적으로 Object클래스 상속받음
***java.lang.Object

**** 오버라이드(상속을 통해서)
매서드의 선언부가 똑같고 구현부를 다르게 프로그래밍하는 방식
```
//선언부
public void parentMethod()  {
//구현부
}
```

** 가장 나중의 매서드가 실행
** 슈퍼클래스 매서드를 사용 sper.매서드( );
** 오버로딩 “매서드명이 똑같고 매개변수 유형이나, 개수가 다른 방식
Override
*** 오버라이딩 ***
오버라이딩이란? 조상(부모) 클래스로부터 상속받은 매서드의 내용을 변경하는 것을 오버라이딩이라고 한다. 상속받은 매서드를 그대로 사용하기도 하지만, 자손 클래스 자신에 맞게 변경해야하는 경우가 많다. 이럴때 조상의 메서드를 오버라이딩한다.

오버라이딩 조건: ** 구현부를 다르게 구현
1. 자손클래스에서 오버라이딩하는 메서드는 조상 클래스의 매서드와 이름이 같아야 한다.
3. 매개변수가 같아야 한다.
4. 반환 타입이 같아야 한다.

final 매서드
더 이상 오버라이딩 불가능

오버라이드시 부모 매서드를 구현 하려면 자식 매서드안에
 -> super.부모매서드();

*** 이클리스가 자동으로 오버라이딩을 실행 할 수 있다.

Final필드, 상수선언

상수를 선언할때 사용
```
class SharedClass{
        public static final double PI=3.14;
}
```

상수필드는 선언 시

// 미완성 -> 다른 클래스에 도움을 추기 위하 클래스 , 인터페이스
추상클래스(abstract)
인터페이스(interface)

*** 다형성 polymorphism
1. 부모타입의 객체 참조변수로 자식객체를 참조 할 수 있다.
2. 하나의 타입으로 여러객체를 참조 할 수 있다.
3. 매서드, 맴버를 다양하게 변경하거나 설정 할 수 있다.

4. 오버로딩, 오버라이딩
-> **부모타입의 참조변수로 자손타입의 인스턴스를 참조할 수 있다
반대로, 자손타입의 참조변수로 조상타입의 인스턴스를 참조할 수는 없다.