Java study

프로그램 언어

- 고급언어 [소스파일]저급언어 [어셈블리어]사람이 이해하기 쉬운 - 고급언어컴퓨터가 이해하기 쉬운 - 저급언어

JAVA

- 운영체제 여부 상관없이 실행가능객체지향 프로그램메모리 자동정리무료 라이브러리 풍부자바의 소스파일 확장명 = .java컴파일 소스파일 확장명 = .class(운영체제 상관없이 똑같음)but 설치하는 법은 다름

변수 *variable*

- 메모리에 데이터 저장하는 방법과 규칙을 정하기 위해 사용하나의 값을 저장할 수 있는 메모리 번지같은 타입 같은 값만 가능.
    - ex)\
      float a = 10f; o\
      int a = 10f; x
    - 변수 명은 첫 글자는 소문자가 와야함.
      캐멀 스타일로 작성하는게 관례.
    - 캐멀스타일여러 단어가 섞일경우 대문자로 구분한다.
- #클래스명은 대문자 변수명은 소문자로 시작한다.

Primitive Type (기본 타입)

- 기본 타입은 8개 이며 **[byte, char, short, int, long, float, double, boolean]**
- 정수형, 실수형, 논리형으로 구분된다.

타입	메모리크기	저장되는 값의 허용범위
byte	1byte	-128 ~127
short	2byte	-32,768~32,767
char	2byte	0~65535(유니코드)
int	4byte	-2,147,483,648~2,147,483,647

long	8byte	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807

- 1byte = 8 bit
- 값의 허용범위보단 메모리크기를 알고 있는 것이 좋다.
- 문자 타입
  - 문자 타입은 유니코드로 변환되 저장이 되며 각국의 문자 0~65535 숫자로 매핑한 국제 표준 규약이다. 이러한 문자를 저장 할 수 있도록 char 타입을 제공" 이점은 컴파일 에러가 생겨 안에 공백을 추가 해줘야함.
- 실수 타입
  - float, double 이 있으며 32bit 64bit == 4byte 8byte에 메모리 크기를 가진다두 가지의 값의 허용 범위가 다르며 float < doublefloat 소수 7자리만 정확하며 double 15자리까지 정확하다정밀한 작업이 필요할경우 double 타입을 사용하면 된다.java 자체는 기본적으로 double타입으로 간주하기에 float를 사용하려면 접미어 f 를 입력해야한다.
- 논리 타입
  - true false 비교 및 논리 연산의 산출값으로 주로 사용
- 문자열 타입
  - " 사용, String 타입을 사용한다.문자열 내에 역슬래시(\) 문자 사용 -> 이스케이프 문자 사용[특정 문자, 출력에 영향].폰트로 인해 \ 표현이 다를 수 있지만 상관없다.
  - (Java 13 이후) *** 를 사용해 블록 문법을 제공한다.*** 사용하면 문자열 그대로 표출해줘서 이스케이프 라인피드를 사용할 필요가 없다.(Java 14 이후) 텍스트 불록은 \n으로 표현한다.줄바꿈을 안하고 이어서 작성하려면 끝에 \를 붙이면 된다.
- 자동 타입 변환
  - 작은 타입을 큰 타입 범위에 저장할 때 자동으로 변환을 한다.byte < short, char < int < long < float < double정수 타입을 실수 타입으로 대입할 때 무조건 타입 변환이 됨.
- 강제 타입 변환 (**casting**)
  - 큰 타입을 작은 타입에 변환할 때 사용 -> 연산자() 괄호 안에는 타입을 쪼대는 단위 타입 범위내에 초과가 되면 데이터가 누락됨.연산식 자동 타입변환서로 다른 연산식에 대한 값이 있을 경우 더 큰 타입으로 자동 형변환 되며 정수 실수 끼리 연산식 진행시에는 실수형으로 표현이 된다.실수타입에 정수형 연산식을 대입하면 정수결과가 나오며 이때는 실수타입으로 형변환을 해줘야 원하는 값이 나온다.

- 문자열 기본 타입 변환
    - String -> 기본형 타입 변환== 타입별 정의 = 타입.parse타입(문자열); 표현하며기본형 -> String 변환시에는String.valueOf(기본타입값) 메소드를 이용한다
- 변수 사용 범위
    - 블록 별로 나뉘어 있으며 특정 범위에서 사용하고 싶으면 블록내부에 선언하며 메소드 전체에 사용은 메소드 내에 선언한다.
- 콘솔 출력
    - System.out.println/ print / printf 등이 있으며printf 명을 사용할때는 형식화된 문자열을 표현할 수 있다.키보드 입력 데이터 변수 저장Scanner 타입 변수를 사용하며 이때는 import 문을 사용해야하며 Enter를 사용해 데이터를 입력해준다.기본타입 동일한지 비교 시에 == 사용String타입 비교 시에 equals()를 사용한다.

연산자 *Operator*

- 부호 증감 연산자.
    - +, - 주로 부호 변경시 사용 (-) 주로 사용부호를 변경 시에 타입은 int 이다. -> int 타입 변수에 대입을 해야함++ -- 변수의 값 1을 증가 감소 시킨다.증감 연산자가 앞에 있으면 우선 변수를 변환하고 연산 수행을 하며 뒤에 있으면 모든 연산을 끝낸 후에 변환함.
- 산술 연산자
    - +, -, *, /, % = 총 5 개더하기 빼기 곱하기 나누기 나머지로 표현함.
    - long을 제외한 정수 타입이 연산을 하면 결과는 int 타입 long 연산 시 long 타입 // 연산자 중 실수 타입 포함 시 결과는 실수 타입.
- 오버플로우 언더플로우
    - 타입의 최대 허용값 벗어나면 오버플로우 최솟값을 벗어나면 언더플로우 라고 말한다.실행시에 에러는 발생 안함 but 벗어나면 최소값, 최대값으로 돌아간다.발생시에 타입을 변환해줘야한다.
- 나눗셈 연산 후 NaN, Infinity 처리 (**예외처리 하는 법**)
    - 무한대의 값은 정수 표현 x -> 표현시 ArithmeticException 발생/, % 결과가 NaN,Infinity 확인 후 다음 연산 수행 한다.
    - Double.isInfinite(), Double.isNaN()를 사용해 확인한다.boolen 을 사용해 true false 산출함.
- 비교 연산자

    - 동등 == !=, 크기 < <= > >= 를 사용하며 boolen 타입으로 true false를 산출.주로 제어문에 사용 -> 조건문 if 반복문 for, while 에서 실행 흐름 제어할 때실수형은 값의 정밀도에 따라 false를 반환하기에 값 변환을 해줘야한다.문자열은 동등 연산자 대신 equals()와 !equals()를 사용한다.
- 논리 연산자
    - 논리곱 && 논리합 || 배타적 논리합 ^ 논리 부정 ! 연산을 수행논리 연산 -> 제어문 반복문에서 주로 사용한다.
    - && & => 모든 값이 true 시 true (& 전체 값 확인 후 출력 && 첫 값이 false면 바로 false 산출)|| | => 하나의 값이 true 시 ture (위 내용과 동일)^ => 두개 의 값이 true false 반환시 true! => 피연산자의 논리값을 바꿈 (true 시 false 반환)
    -
- 비트 논리 연산자
    - bit 단위로 수행 -> 정수 타입만 가능 실수는 부동소수점이기에 불가능.피연산자와 산출 결과가 1, 0 이라는 점에 주목을 하며 1은 true 0은 false를 나타낸다. [논리 연산자와 같다]
- 비트 이동 연산자
    - << >> >>> 이렇게 세가지를 나타내며 방향에 표시된 곳으로 이동을 하며 a는 기준점 b 는 이동 수를 나타낸다.
- 대입 연산자
    - 우측의 값을 좌측에 피연산자 변수에 대입을 하는 의미이며 단순 대입과 복합 대인 연산자가 있다단순 대입은 '=' 복합 대입은 '+= -= *= /= %= &= |= ^= <<= >>= >>>=' 이 존재한다.
- 삼항(조건) 연산자
    - 3개의 피연산자를 가지며 ? 앞의 피연산자는 boolean 아니면 조건식 형태가 와서 조건연산자라고도 한다.

제어 [조건문, 반복문]

- 코드 실행 흐름 제어
    - 코드 실행은 위에서 아래로 흐르며 각 중괄호({}) 흐름이 진행된다. 제어문에 따라 다양한 실행 흐름이 생성되는 점도 있다.제어문에는 조건문과 반복문이 있다.반복문일 경우 처음으로 다시 돌아가 반복 실행을 하는것을 루핑이라고 한다.제어문 블록 내에서도 또다른 제어문을 사용할 수 있어 복잡한 흐름 제어도 가능하다.
- 조건문 [if, swich]

    - 중괄호 블록은 생략하지 않고 작성하는 것을 추천하며 이유는 가독성이 낮아지며 버그 발생원인 이 된다.[조건문과 상관없게 되는 식이 발생할 수 있다.]

- if(조건식)
    - 조건식 안에는 참, 거짓을 산출하는 연산식이나 boolean 변수가 온다. 참이면 실행하고 거짓이면 안한다.거짓일 때 실행을 하려면 else 블록을 추가를 할 수 있다.임의의 정수를 뽑기 위해 Math.random() 메서드를 활용한다.if 블록 내부에 또다른 if 문을 사용할 경우 이 것을 중첩 if 문이라고 부르며 중첩의 단계는 제한이 없다.경우의 수가 많아지면 else if문을 반복적 추가로 코드가 복잡해진다

- switch
    - 변수의 값에 따라 실행을 해서 if else 문보다 코드가 간결해진다.변수의 값에 해당하는 case로 가서 실행문을 실행시킨다 해당하는 값이 없을 경우에는 default로 실행문을 실행한다. 이 경우 필요가 없을시 생략이 가능하다.(Java 12 이후) Expressions 를 사용할 수 있다. break 문을 없애고 화살표와 중괄호를 사용한다.switch(grade){case 'A':System.out.println("우수 회원임");break;}위 코드를 Expression 을 사용 하면switch(grade) {case 'A' -> {System.out.println("우수 회원임");}중괄호 안에 실행문이 하나만 있으면 생략이 가능함.위와 방식처럼 표현하며 코드 가독성이 좋아진다.

- for
    - 똑같은 실행문을 반복적으로 실핼할 경우에 사용한다.for(초기화식 조건식 증감식) { 실행문 } 으로 진행한다조건식이 거짓이 될때까지 반복 실행을 한다.초기화식에서 선언된 변수는 for문 안에 사용하는 로컬 변수라서 벗어나서 사용할 시에 for문 이전에 선언해야한다.

- while
    - for문은 정해진 횟수만큼 한다면 while은 조건식이 참일 때 반복한다. 거짓이 되면 반복문은 종료가 된다.whelie(조건식){ 실행문 }조건식에 참을 실행한다면 무한루트에 빠질 수 있다. 빠져나가기 위한 코드가 필요하다.if문을 사용해서 재정의를 통해 빠져나갈 수 있다

- do-while
    - do { 실행문 } while (조건식);실행문이 실행한 뒤 조건식을 평가하며 결과가 참일 경우만 반복한다.do while 문 같은 경우에는 while 마지막에 세미클론(;) 을 넣어야 한다.break continue 는 마지막 단에 작성을 하며 조건식에서 나가냐 더 실행하는 가로 나뉘며break 를 했을 시 실행을 중지 하거나 조건식 종료를 한다.continue 는 특정 조건에 만족을 할 시에 그 이후 문장을 실행하지 않고 다음 반복으로 넘어간다.continue 는 번호를 나열해 짝수 홀수를 구분하기에 좋다.

객체 지향 프로그래밍

참조 타입

- 데이터 타입 분류
    - 기본타입 (Primitive type) 참조타입 (Reference type) 두가지로 분류 한다.참조 타입은 객체의 번지를 참조하는 타입ㅇ미ㅕ 배열, 열거, 클래스, 인터페이스 타입이 있다.데이터(필드) + 메소드 = 객체기본과 참조의 차이점은 저장되는 값이다. 기본은 값의 그자체를 저장하고 참조는 객체의 생성된 메모리번지를 저장한다.변수들은 스택 메모리 영역에 생성이되며 참조 타입 변수는 힙 메모리 영역에 객체 번지를 저장한다. 힙 메모리 번지를 통해 객체를 참조한다.
- 생성된 객체를 참조 받는 경우에 같은 배열을 참조 받아도 대입되는 번지가 다르다. 불러온 참조값을 새로운 변수에 대입하는 경우에는 같다고 할 수 있다.
- null, NullPointerException
    - 참조 타입은 null 값을 가질 수 있다 여기서 null 이란 번지를 갖고 있지 않은 상태로 정의한다.null 값도 초기값 사용 가능해서 초기화된 값은 스택 영역에 생성이 된다. 참조 타입이 null 값을 갖는지 확인 시에 == != 연산을 수행 할 수 있다.자바는 프로그램 도중 발생하는 오류를 예외(Exception)라고한다.
      int[] intArray = null;
      intArray[0] = 10; >> NullPointerExceptionNullPointerException 위 코드예시처럼 intArray에 참조하는 배열 객체가 없어 10을 저장할 수 없기 때문이다.배열 참조 자체가 null 값이라 그 값은 참조된 값이 없기에 넣을 수 있는 장소가 없기 때문이다.
      다음 예시코드는
      String str = null;
      System.out.println(str.length());str 변수가 참조하는 String 객체가 없으며 문자열의 길이를 구할 수 없기에 NullPointerException이 발생한다.그래서 앞으로 NullPointerException이 발생하면 그 곳에 null 상태의 참조 변수가 사용하고 있다는 점을 알게 됐다.해결법은 참조 변수가 객체를 정확하게 참조하도록 번지를 대입해야한다는 해결법을 배웠다.참조하지 않는 객체는 사용할 수 없는 객체로 구분되며 자바에서는 이러한 객체를 쓰레기 취급을 하며 Garbage Collector를 실행시켜 자동으로 제거시킨다.자바에서 메모리를 관리해준다는 얘기가 어떤 의미인지 알게 되었다.코드를 이용해 객체를 제거하는 방법을 제공하지 않아 객체의 모든 참조를 제거해서 없애는 것이다.
- 문자열(String) 타입
    - 자바의 문자열은 String 객체로 생성이 된다.
    - 문자열 리터럴이 동일하면 객체는 공유하도록 설계 되있다.

- 다만 new 연산자를 이용해 직접 생성시에는 객체 생성 연산자라고 부른다. 이경우는 각자의 번지를 가지게 된다.

```
String name1 = "홍길동";
String name2 = "홍길동";
String name3 = new String("홍길동");

name1 == name2 // true
name1 == name3 // false
```

** 빈 문자열도 대입이 가능하다. 비교시에 equals() 메소드를 이용해야 한다.

- charAt() 으로 통해 문자 추출이 가능해진다.

```
String subject = "자바 프로그래밍";
char charValue = subject.charAt(3); // 프
```

- length() 로 문자의 개수를 알수 있다

```
String subject = "자바 프로그래밍";
char length = subject.length(); // 8
```

- 문자열 대체(replace), 문자열 잘라내기(substring), 문자열 찾기(indexOf), 문자열 분리(split)

```
String subject = "자바 프로그래밍";
String newStr = subject.replace("자바", "JAVA"); // 자바 > Java

String ssn = "880815-1234567";
String firstNum = ssn.substring(0, 6); // 0번째부터 6번째까지 잘라
String lastNum  = ssn.substring(7); // 7번째 이후로 잘라내기

int index = subject.indexOf("프로그래밍"); // 3
```

```
String board = "번호,제목,내용,성명";
String[] arr = board.split(","); // , 제공하면 분리된 문자열 배열을
```

배열(Array) 타입

Main 메서드 String[] 매개변수 용도

프로그램 수행 > 문자열 취급 > main 메서드로 매개값 전달

타입 변환시에는 강제적으로 바꿔줘야한다x`

열거 enum 타입

한정된 값을 가지는 타입을 열거 타입이라고 한다

열거 타입 이름으로 소스파일 생성하며 이름 첫 문자는 대문자, 캐멀스타일로 작명

열거 상수 값은 알파벳 정의하며 대문자로 작성 단어 사이를 언더바(_)로 연결

하나의 데이터 타입 이여서 변수 선언후 사용해야 한다.

변수에 열거 상수를 대입할 수 있으며 '열거타입.열거상수' 형태로 작성한다,

열거 타입은 참조형이기에 null 값을 대입이 가능한다.

비교할 시에 ' == != ' 연산자를 사용한다.

객체지향프로그래밍(Object Oriented Programming)

- 간단한 예시로는 각자의 부품을 만들어 하나의 완성품으로 조립하는 과정을 설명한다

객체는

속성과 동작으로 구성되며 이를 필드와 메서드라고 부른다

객체의 대표 속성과 동작을 정의하는 과정을 객체 모델링이라고 한다.

객체는 서로 상호작용을 하며 이에 사용하는 수단은 메서드이다

다른 객체의 기능을 이용할 때 이 메서드를 호출한다

메서드 호출의 형태는

메서드 (매개값1, 매개값2)

메서드 호출로 통한 데이터 공유 할때 데이터는 괄호 안에 기술한다

int result = add(10,20);

리턴한 값들을 int 변수에 저장한다.

객체는 단독 존재 가능하며 대부분은 객체간의 관계를 맺고 있다.

집합, 사용, 상속 세가지가 있으며

집합은 완성품과 부품의 관계이다

사용은 다른 객체 필드를 읽고 변경하며 메서드를 호출하는 관계이다.

상속은 부모와 자식관계를 말한다.

객체 지향 프로그래밍의 특징은 캡슐화 상속 다형성 이다

- 캡슐화

필드와 매서드를 하나로 묶어 실제 구현 내용을 외부로 감추는 것을 말한다. 객체의 손상도를 줄이기 위해 사용하며 이때 사용하는게 접근제한자 이다.

- 상속

코드의 재사용성을 높이고 유지보수 기간를 최소화 시켜주기에 상속을 사용하며 부모객체에서 사용하는 필드와 메서드를 물려줘 자식 객체가 사용한다

- 다형성

사용법은 같고 결과만 다양하게 나오는 것을 말한다.

구현하기 위해 자동타입변화, 재정의 기술이 필요하다

객체 생성시에 설계도가 필요한데 설계도는 클래스이며 클래스로 객체를 만드는 과정을 인스턴스화 라고 한다.

클래스 선언에는 객체의 생성과 가져야하는 데이터 그리고 동작이 무엇인지 본다

클래스 선언시에

접근제어자 클래스 클래스명 방식으로 선언을 한다

= public class JavaProject {}

복수의 클래스 선언시에는 소스파일명과 동일한 클래스만 공개클래스로 선언할수 있다

클래스 부터 객체 생성시에는 new 연산자가 필요하다

스택영역에 참조해 힙영역으로부터 불러온다.

클래스에는 두가지 용도가 있으며 라이브러리와 실행 클래스가 있다

실행을 못하고 다른 클래스를 이용하면 라이브러리

메서드를 가지며 실행하는게 실행 클래스다

자바는 일반적으로 하나의 실행과 여러 개의 라이브러리로 구성된다

클래스 선언 시 구성하는 멤버가 있다 - 생성자,필드,메서드

필드는 데이터 저장역할

생성자 = new 연산자 사용해 객체 생성시 객체의 초기화 역할

메서드는 객체가 수행할 동작

- 필드

데이터 저장 역할을 한다 변수 선언과 비슷하다

데이터는 고유, 현재 상태, 부품 데이터가 있다

선언방법은 = 타입 필드명[= 초기값];

- 타입은 필드에 저장할 데이터 종류이다. 기본과 참조형타입 둘다 가능 첫 문자는 소문자로 하며 캐멀 스타일로 작성한다.
- 필드 사용

외부 객체에서 사용시 도트 연산자를 사용해야한다.

***

변수와 필드 차이점은

변수는 생성자, 메서드 블록에서 선언이 되며 생성자자와 메서드 호출시에만 사용을하지만 필드는 클래스 블록에서 선언이되어 객체 내부 외부에서 사용이 가능하다.

- 생성자

new 연산자 객체 생성 시 초기화 역할 메서드와 비슷하며 리턴 타입이 없고 클래스 이름과 동일

생성자는 new 연산자를 사용해 객체 초기화하는 역할을 한다

객체 주소로 리턴되며 변수에 대입이되 필드와 메서드 접근 할때 이용한다

모든 클래서에 생성자가 존재하며 선언을 안하면 컴파일러에서 자동으로 추가시킨다

- 필드초기화

객체마다 동일한 값이면 필드 선언 시 초기값 대입이 좋고

객체마다 다른 값이면 생성자에 필드 초기화 하는게 좋다.

Korea 클래스를 선언 가정을 하면 국적은 같지만 각자 이름과 주민등록번호는 다르기에 생성자에 초기화 하는게 좋다.

매개 변수의 이름이 너무 짧으면 코드 가독성이 현저히 떨어져 필드명과 동일한 이름을 사용하는 것이 좋다고 한다.

생성자 오버로딩

매개값으로 다양하게 초기화하려면 생성자 오버로딩이 필요하다.

- 생성자를 여러개 선언 하는 것을 말한다.

생성자 오버로딩이 많아질 경우 중복된 코드 발생 가능성이 있다.

- 이 경우 한 생성자에 집중적 작성후 나머지 생성자에 this를 사용해 코드를 가져오는 방법으로 개선 할 수 있다.

메서드 오버로딩
메서드명은 같지만 매개변수 값을 다르게해 다양한 조건을 해결하기 위해 사용을 한다.

인스턴스 멤버
인스턴스멤버 클래스멤버로 나뉠수 있다.
객체로 구분을 하며 객체가 소속되면 인스턴스 객체없이도 사용가능한 클래스멤버가 있다.
객체에 소속된 인스턴스와 클래스에 고정된 클래스멤버라고 한다
객체 외부에서 접근을 하려면 객체를 생성해 참조변수 이용해서 접근을 한다.
객체 내부에서의 접근은 this 키워드를 이용해서 접근을 한다.

정적 멤버
정적 멤버 사용은 static 키워드만 추가를 해주면 된다.
도트 연산자를 이용해 클래스명과 같이 사용된다.
클래스를 새로 생성해 사용하는 방법도 있지만 정석은 클래스명 도트연산자 를 이용하는 것이다.

정적멤버가 복잡한 구조가 생길시에는 블록을 이용해서 작성하는 방법도 있다.

final 필드와 상수
인스턴스 정적 멤버의 데이터 변환을 막기위해 사용되는 키워드는 final 이다.
읽기전용이므로 변경은 불가한다.
상수는 단어끼리 혼합이되었으면 언더바로 구별을하게한다.

패키지 import문
패키지는 클래스의 일부분이며 도트연산자로 구분을한다 클래스의 맨 상단에 위치를한다.
클래스를 식별하는 용도로 사용이 된다.
import문은 다른 패키지에 있는 클래스를 사용할때 사용을 하며 클래스의 경로를 입력을 하면 된다.
같은 클래스 명이 있을때 사용을 하려면 하나는 전체경로를 작성해주는게 좋다.

접근제한자.
public, private, portected, default 가 있다.
클래스 내부 외부 객체 내부 등 접근을 제한하는 역할을 한다.
외부에서 접근을 하기 위한방법이 있다.

Setter Getter
데이터의 검증을 하기위해 Setter 메서드로 불러들인다.
유효한 값만 필드에 저장을 한다.

메서드 값을 적절한 표현으로 변환시켜서 사용하는 것은 Getter 메서드로 불러들인다.

싱글톤 패턴
핵심은 생성자를 private 접근 제한으로 부터 new 연산자 생성자를 호출할 수 없도록 막는것이다.

외부 객체에서 생성할 수 없어서 싱글톤 패턴으로 간접적으로 객체를 얻는다.

- 상속
부모가 자식에게 물려주는 행위
가장 큰 장점은 중복된 코드를 줄여줘 개발 시간을 단축시켜준다.
수정의 이점을 보인다. 상속을 받은 자식 클래스는 부모 클래스에서 정보를 수정하면 전부 반영이 되기 때문이다.

클래스 상속 키워드는 extends 이다.
상속 클래스는 하나만 지원한다. 다중상속이 안된다.

부모 객체를 먼저 생성해 자식 객체에 상속하고 그것을 자식생성자에 불러오는 시스템이며 자식 생성자에 super() 키워드를 통해서 호출이 되며 컴파일 과정에서 자동으로 추가를 해준다.
부모 클래스에 생성자가 없으면 컴파일 과정에서 에러가 발생한다.
매개변수 값이 있을 경우에는 무조건 키워드를 작성해야한다.

부모 클래스가 모든 자식클래스에 맞게 제작되면 좋지만 그게 가능성이 희박해 오버라이딩을 한다.
정확한 오버라이딩을 확인할 시에는 애너테이션을 추가하면 컴파일오류가 발생시에 알려준다.

클래스에 final 키워드를 작성을 하면 해당하는 클래스는 상속이 불가하다.
메서드도 final 키워드를 작성하면 오버라이딩이 불가하다.

protected 는 priveate 과 public에 대한 중간쯤 해당하는 접근 제어자이다.
같은 패키지에서는 기본값으로 접근이 가능하지만 다른 패키지에서는 자식 클래스만 가능하다
필드 메서드 생성자에 사용할 수 있다.
- 직접 객체를 사용하는 것은 안된다.
- 자식 생성자를 사용해 상속 super() 키워드를 사용해 접근을 한다.

타입 변환

- 타입을 다른 타입으로 변환하는 것을 말한다.
- 클래스에도 타입 변환이 이뤄진다. → 상속 관계의 클래스 사이에 발생한다.

자동 타입 변환

- 의미적으로 자동적으로 타입 변환이 이뤄진다.
- 부모 타입 변수 ← 자식타입 객체 조건에서 발생한다.

강제 타입 변환

- 캐스팅 연산자를 사용해 강제로 타입을 변환 시킨다.
- 자식 타입 변수 ← 부모타입객체 조건에서 사용한다.

다형성

- 사용 방법은 동일하게 하지만 실행 결과는 다양하게 나오는 성질이다.
- 다형성을 구현하려면 자동타입 변환 과 메서드 재정의가 필요하다.

필드 다형성

- 필드 타입은 동일 대입하는 객체가 달라져 결과가 다양하게 나올 수 있는 것을 말한다.

```
class Car{
        public Tire tire;

        public void run(){
            tire.roll();
        }
    }
    public class Tire{
        public void roll(){
            System.out.println("회전 중");
        }
    }
```

```
public class HankookTire extends Tire{
    // 메서드 오버라이딩
    @Override
    public void roll(){
        System.out.println("한국타이어 회전중");
    }
}
```

- 다형성은 필드보다는 메서드 호출 시에 많이 사용된다.

매개변수 다형성

- 메소드가 클래스 타입의 매개변수를 가질 경우
- 호출 시 동일한 타입 객체 제공이 정석
- 자식 객체도 제공할 수 있음 → 여기서 다형성이 발생한다.

객체 타입 확인

- 어떤 객체가 매개값으로 제공하는지 확인하는 방법은 instanceof 연산자를 사용한다.
- 좌항의 객체가 우항의 타입이면 true를 산출 아니면 false를 한다.

```
boolean result = 객체 instanceof 타입
```

- Java12 부터는 instanceof 연산 결과가 true 이여야 사용 가능하기에 강제 타입 변환을 해야한다.

추상 클래스

- 추상은 실체 간에 공통되는 특성을 추출하는 것이다.
- 동물은 추상적인 단어이다.
- 추상 클래스는 실제 클래스에서 공통되는 필드와 메서드를 추출해서 만든것이기에 new 연산자를 사용해서 직접 생성할 수 없다.
- 추상 클래스는 부모 클래스로만 사용한다. 즉 extends 뒤에만 올수 있다.

추상 클래스 선언

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

- 자식 객체 생성시 super()로 추상 클래스 생성자가 호출되서 생성자는 반드시 있어야한다.

추상 메서드와 재정의

- 추상 클래스를 작성할 때 메서도 선언부만 동일하고 실행 내용은 자식 클래스마다 달라야하는 경우가 있다.
- 이럴 경우 다음과 같이 추상 메서드를 선언한다.
- 추상메서드 선언

```
abstract 리턴타입 메소드명(매개변수, '');
```

- 일반 메서드와 차이점은 abstract 키워드가 붙고, 메소드 실행 내용인 중괄호{}가 없다
- 공통 메서드라는 것을 정의하는 것이기에 실행 내용을 갖지 않는다.

```
public abstract class Animal{
        public void breathe(){
            System.out.println("숨을 쉰다");
        }

        public abstract void sound();
    }

    public class Dog extends Animal {
                // 추상 메서드 재정의
        @Override
        public void sound(){
            System.out.println("멍멍");
```

```
        }
    }
```

봉인된 클래스

- final 클래스를 제외한 모든 클래스는 부모 클래스가 가능하다
- Java 15 부터 sealed 클래스가 도입됐다.
  - 무분별한 자식 클래스 생성 방지를 위해서
- 봉인된 클래스 선언문

```
public sealed class Person permits Employee, Manager {'''}
```

- sealed 키워드를 사용하면 permits 키워드 뒤에 상속 가능한 자식 클래스 지정해야 한다.

```
public final class Employee extend Person{''}
public non-sealed class Manager extends Person {'''}
```

- final은 더이상 상속 금지 non-sealed 는 봉인 해제 한다는 뜻
- 그래서 Manager 클래스는 자식 클래스를 만들 수 있다.

인터페이스

- 인터페이스는 두 장치를 연결하는 접속기를 말한다. → 서로 다른 객체를 본다면 이 두 객체를 연결하는 역할을 한다.
- 다형성 구현에 주된 기술로 이용한다. → 상속으로 구현할 수 있지만 인터페이스로 다형성 구현하는 경우가 더 많다.
- A객체가 인터페이스 호출하고 인터페이스가 B객체를 호출해서 정보를 전달할 때 중간 과정이 있어야하는 이유는 B객체가 C객체로 변경시에 유연함이 높기 때문이다.

인터페이스 선언

- 인터페이스 선언

```
interface 인터페이스명 {'''} // defalut
public interface 인터페이스명 {'''} // public
```

구현 클래스 선언

```
public class B implements 인터페이스명 {’’’}
```

-

```
public class Television implements RemoteControl{
    @Override // 인터페이스에 선언된 turnOn 추상 메서드 재정의
    public void turnOn(){
        System.out.println("TV를 켭니다.");
    }
}
```

- 인터페이스는 하나의 타입이기에 변수 타입으로 사용 가능하다.
- 인터페이스 참조 타입에 속해 null 값을 대입 가능하다.

상수 필드

- public static final 특성 갖는 불변의 상수 필드를 멤버로 가질 수 있다.
- public static final 생략해도 컴파일 과정에서 자동으로 붙는다.
- 상수명은 대문자이며 단어 끼리 연결 시에는 언더바(_)로 연결한다.

추상 메서드

- public abstract를 생략해도 컴파일 과정에서 자동으로 붙게 된다.
- 추상 메서드는 리턴타입, 메소드명, 매개변수만 기술되고 중괄호를 붙이지 않는다.
- 추상 메서드 선언문

```
[public abstract] 리턴타입, 메서드명, (매개변수, ''');
```

- 구현 클래스에서 추상메서드를 재정의 시에 주의할 점은 기본적으로 public 접근 제한을 갖기에 더 낮은 접근 제한은 재정의 불가하다 그래서 재정의 메서드는 모두 public 이 추가 되어있다.

```
pubcli class RemoteControlExample {
    public static void main(String[] arge) {
            // 인터페이스 변수 선언
            RemoteControl rc;

            // Television 객체를 생성 후 인터페이스 변수 대입
            rc = new Television();
            rc.turnOn();
            rc.setVolume(5);
            rc.turnOff();

            // Audio 객체 생성 후 인터페이스 변수 대입
            rc = new Audio();
            rc.turnOn();
            rc.setVolume(5);
            rc.turnOff();
```

디폴트 메소드

인터페이스 디폴트 메서드 선언이 가능하며 실행부가 있다. 큰 차이는 default 키워드가 리턴 타입 앞에 붙는다.

```
[public] default 리턴타입 메소드명(매개변수, ''') {'''}
```

정적 메소드

구현 객체 없이 인터페이스만으로 호출 가능하다.

인터페이스 선언된 정적메소드는 구현 객체 없이 인터페이스명으로 접근해 호출이 가능하다.

RemoteControl의 changeBattery() 메소드를 RemoteControl.changeBattery() 이것으로 호출을 할 수 있다.

작성 시 주의점은 상수 필드 제외 추상 메소드, 디폴트 메서도 private 메소드 등은 호출 할 수 없다.

위 메소드들은 구현 객체가 필요하기 때문이다.

다중 인터페이스 구현

- 클래스 상속과 달리 인터페이스 다중 상속이 가능하며 그래서 다중으로 구현이 가능하다.

```
public class 구현클래스명 implements 인터페이스1, 인터페이스2 {
        // 추상 메서드 재정의
}
```

중첩 선언과 익명 객체

- 객체간 상호작용이 발생하며 여러 클래스 관계 맺으면 독립적으로 사용이 좋고
- 특정 클래스의 관계를 맺을 경우 중첩 클래스를 선언하면 유지보수에 도움이 좋다.

중첩클래스

- 클래스 내부에 선언하는 클래스
- 멤버 클래스와 로컬클래스로 나뉘며
    - 클래스 멤버 선언은 멤버클래스 메소드 내부 선언시는 로컬 클래스라고 한다.

익명객체

- 익명객체는 이름이 없는 객체를 말한다.
- 필드, 로컬변수, 매개변수 들의 값으로 주로 사용된다.

라이브러리와 모듈

- 프로그램 개발 시 활용할 수 있는 클래스와 인터페이스들을 모아 놓은 것을 라이브러리라고 한다.
- 패키지 관리 기능까지 포함하는 라이브러리가 모듈이라고 한다.

- Jar 압축 파일 형태로 라이브러리가 있으며 이 파일에는 바이트코드 파일인 (.class)들이 압축되있다.

- Jar 파일을 사용하기 위해서는 환경변수 설정을 해줘야 경로에 맞춰 사용이 가능하다.

예외처리

- 잘못된 사용 혹은 코딩으로 인한 오류를 예외라고 말한다.
- 예외에는 일반 예외랑 실행 예외가 있다.

```
// 처리 코드
try {
    // 예외 발생 코드
} catch (예외클래스 e){
    // 예외 처리
} finally {
    // 항상 실행
}
```

리소스

- 데이터를 제공하는 객체는 리소스라고 한다.

- 예외를 넘기는 방법이 있다. 이것을 예외 떠넘기기 라고 하는데 메소드를 호출 한 곳으로 예외를 떠넘기는 것이며 키워드는 throws 이다.
-

```
리턴타입 메소드명(매개변수,'') throws 예외클래스1, 예외클래스2 {
}
```

라이브러리 활용

java.base 모듈

java.lang	자바 언어 기본 클래스 제공
java.util	자료 구조와 컬렉션 클래스 제공
java.text	날짜 숫자 원하는 형태 문자열 만들어 주는 포맷 클래스 제공
java.time	날짜 시간 조작하거나 연산 클래스 제공
java.io	입출력 스트림클래스 제공
java.net	네트워크 통신과 관련된 클래스 제공
java.nio	데이터 저장을 위한 Buffer 및 새로운 입출력 클래스 제공

- 객체 동등 비교와 객체 해시코드
- 객체 동등 비교는 객체의 번지를 비교하며, 객체 해시코드는 객체 식별하는 정수이다.
- 해시코드에는 HashSet 키워드가 있는데 동등한 데이터는 저장을 안한다.

레코드

- 데이터 전달을 위한 DTO (Data Transfer Object) 작성시 반복 사용 코드를 줄이기 위해 도입이 됐다.
-

```
public record Person(String name, int age){
}
```

- 선언 시 컴파일 과정에서 private final 필드, 생성자, Getter 메서드가 자동 추가 된다.

- hashCode(), equals(), toString 메소드 재정의한 코드도 자동 추가 된다.

```
public record Member(Stinrg id, String name, int age) {}

pubcli class RecordExample {
    public static void main(String[] args){
        Memeber m = new Member("winter", "눈송이", 25);
```

롬북

- 레코드와 같이 자동 생성 라이브러리다.
- 레코드와 같이 메소드를 자동 생성해 코드의 양을 줄여준다.
- 레코드와 차이점이라고 하면 필드가 final이 아니고 값을 읽고 값을 변경하는 Getter, Setter 명이 getXxx, setXxx 로 생성이 된다는 점이다.
- @(어노테이션) 을 class 선언 위에 붙인다.