**JAVA프로그래밍 및 실습 - 중간고사 공부 2276219 이서현**

[Week 01]

* 컴퓨터와 소프트웨어
* 컴퓨터: 메인프레임, PC, 태블릿, 스마트폰… // - 앱: 게임, 메신저 등등..
* 프로그래밍 언어
* 기계어(machine language): 0과 1의 이진수로 구성. 컴퓨터 CPU는 기계어만 이해하고 처리가능.
* 어셈블리어: ADD, SUB, MOVE 등과 같은 기계어 명령을 표현하기 쉬운 상징적인 단어인 니모닉 기호로 일대일 대응시킨 언어.
* 고급언어: 사람이 이해하기 쉽고 복잡한 자료구조, 알고리즘을 표현하기 위해 고안된 언어 (ex. Pascal, Basic, C/C++, Java..) 절차 지향 언어/객체 지향 언어로 나뉨.
* 컴파일
* 소스: 프로그래밍 언어로 작성된 텍스트 파일
* 컴파일: 소스 파일을 컴퓨터가 이해할 수 있는 기계어로 만드는 과정

(소스 파일 확장자와 컴파일 된 파일의 확장자 비교: .java 🡪 .class)

* 자바의 플랫폼 독립성, WORA(자바 가상 기계가 설치된 모든 컴퓨터에서 실행)
* 바이트 코드와 자바 가상 기계
* 바이트 코드: 자바 가상 기계에서 실행 가능할 바이너리 코드. 바이트 코드는 컴퓨터 CPU에 의해 직접 실행되지 않고, 자바 가상 기계가 작동 중인 플랫폼에서 실행된다. 자바 가상 기계가 인터프리터 방식으로 바이트 코드를 해석한다.
* 자바 가상 기계(JVM: Java Virtual Machine): 동일한 자바 실행 환경을 제공한다. (JVM자체는 플랫폼마다 각각 작성되어 플랫폼 종속적)
* 자바 응용프로그램 실행: JVM이 응용프로그램을 구성하는 클래스 파일(.class)의 바이트 코드를 실행한다.
* 자바와 C/C++의 실행 환경 차이
* 자바는 컴파일러가 소스 파일을 바이트 코드로 변환하고, 자바 가상 기계 상에서 실행된다. 링크 과정이 없다. 바이트 코드는 JVM에서만 실행 가능. 자바는 필요한 클래스들을 프로그램 실행 중에 동적으로 로딩한다.(동적 로딩은 JVM에 포함된 클래스 로더에 의해 이루어짐)
* C/C++은 컴파일러와 링커가 소스 파일을 바이너리 실행 파일로 변환하고, 운영체제 위에서 프로그램이 실행된다. 컴파일 시 컴파일러가 중간 단계인 목적 코드를 생성하고, 링커가 목적 코드와 라이브러리를 연결해 실행 가능한 최종 실행 파일을 생성한다. 정적 라이브러리는 실행 파일에 포함된다(->실행 파일 크기 커짐) 동적 라이브러리는 실행 중에 동적으로 링크된다. 목적 코드 및 실행 파일은 플랫폼에 따라 다르다. 다른 플랫폼에서 실행시키려면 다시 컴파일 및 링크 필요.
* JDK와 JRE
* JDK(Java Development Kit) : 자바 응용 개발 환경, 개발에 필요한 도구 포함. 컴파일러, 컴파일된 자바 api 클래스들이 들어 있는 모듈 파일들, 샘플 등 포함
* JRE(Java Runtime Environment) : 자바 실행 환경, JVM 포함. 컴파일된 자바 API들이 들어있는 모듈파일…
* JDK의 bin 디렉터리에 포함된 주요 개발 도구
* javac, java, javadoc, jar, jmod, jlink, jdb, javap…
* JDK 설치 후 디렉터리 구조 (bin, conf, include, jmods, legal, lib
* 자바의 배포판 종류 (SE, ME, EE)
* 자바 통합 개발 환경: 이클립스(IDE 통합개발환경: 편집, 컴파일, 디버깅 한번에)
* 자바 프로그램 개발
* public class Hello //클래스 선언문. 클래스는 {} 사이에 정의. 자바는 하나 이상의 클래스로 구성
* public static void main(String[] args) //자바 프로그램은 main()에서 실행 시작. 반드시 하나 존재
* int n = 2023; //지역 변수 선언
* System.out.println //System.out 객체는 JDK 제공
* 자바 소스 확장자 .java
* 명령 프롬프트에서 컴파일: javac (파일명).java / 실행: java (실행파일명-.class 확장자 붙이지않는다)

**JAVA프로그래밍 및 실습 - 중간고사 공부 2276219 이서현**

[Week 02] 자바 구조

1. 자바 프로그램의 기본 구조

* 자바 프로그램의 기본 구조
* 자바 프로그램은 하나 이상의 클래스로 구성됨. 각 클래스의 프로그램 코드를 별도의 소스 파일에 저장하고 각 소스 파일명을 소스 파일에 정의된 클래스명과 동일하게 지정해야 함 (만약 이름이 다르면 컴파일 과정에서 오류 발생)
* 모든 자바 소스 파일의 확장자: .java
* 프로그램에 public static main 메소드가 최소 하나 있어야함. 하나의 패키지에는 main 메서드 딱 하나 존재해야함/
* 소스 파일(클래스명.java) : 패키지 > 클래스 > 메서드 > 처리문(실행문)

(명명법 관례: 패키지명은 소문자, 클래스명은 단어 첫글자 다 대문자, 메소드명은 2번째 이후 단어 첫글자 대문자, 상수는 다 대문자. 대소문자를 구별함.)

* 패키지: (기능을 기반으로) 서로 관련된 클래스의 모음.
* 패키지문: 클래스가 저장되는 네임스페이스.(ex. package 패키지명; 🡪 ‘프로젝트명/src’안에 패키지명으로 폴더가 생성됨) 패키지문을 생략하면 이름이 없는 기본 패키지에 클래스명만 선언된다.
* 클래스: 객체지향 언어에서 프로그램을 개발하는 단위. class 키워드를 사용해 클래스를 선언한다. 전체 클래스 내용은 {}안에 포함되어야함. public 키워드 사용시 패키지 외부에서 클래스 접근가능(나중에 접근지정자 추가 공부)
* 클래스명과 자바 파일명이 반드시 동일해야 한다.
* main() 메서드: 모든 자바 애플리케이션의 시작점이자 진입점. 애플리케이션이 시작될 때마다 가장 먼저 호출되는 메서드임. 하나의 자바 프로그램엔 main()메서드 가진 클래스 반드시 존재해야 함. 중괄호{}내부에 메서드의 내용 담김. String[] args: main()메서드가 문자열 배열을 입력받을 수 있음을 의미
* 메서드: 수행할 작업을 나열한 코드의 모음
* 처리문(statement): 자바 프로그램의 동작을 명시하고 이러한 동작을 컴퓨터에 알려주는 데 사용되는 문장. 자바의 모든 처리문은 반드시 ;로 끝나야함. 프로그램을 실행하면 main() 메서드 안의 처리문이 순차적으로 실행된다.
* 처리문: 작업을 지시하는 변수 선언, 값 저장, 메서드 호출 등의 코드
* 주석(comment): 코드에 대한 이해를 돕기 위해 프로그램에 설명을 덧붙이거나 디버깅을 위해 작성하는 일조으이 메모. 주석은 컴파일되지 않는다. 한줄 주석: //, 블록 주석: /\* \*/

1. 변수

* 변수의 기본 개념
* 변수: 특정 자료형의 값을 저장하는 메모리 공간의 위치. 데이터가 저장되는 메모리 주소가 복잡하기 때문에 메모리 주소 대신 메모리 주소의 이름을 지정한다. 이러한 메모리 주소의 이름, 즉 메모리 위치의 이름을 변수라고 한다(여기에 지정된 값은 변경될 수 있다)
* 변수는 이름과 자료형을 가지고 있으며, 자료형에 따른 값이 저장된다.
* 변수 정의 시 반드시 자료형 필요. var키워드 쓰려면 반드시 초기화해줘야. var sum; 불가능, var sum=0; 가능(유추함)
* 변수 선언의 기본 형식: ‘자료형 변수명’
* 자료형(data type): 변수에 저장할 수 있는 데이터의 유형
* 변수명: 메모리 주소를 대신하는 이름 지정
* 변수 초기화: 지정된 메모리 위치에 값을 입력하는 것 (변수 초기화는 변수 선언과 동시에 값을 할당하는 것)
* 변수명 생성 규칙
* 변수명은 영문자, 숫자, 밑줄\_, 달러$로만 구성할 수 있음
* 변수명은 숫자로 시작할 수 없음. 공백을 포함할 수 없음. 키워드를 이름으로 사용할 수 없음.
* 변수 작명시 맨 앞은 영문자 소문자로 하자ㅎㅎ
* (변수의 유형) 지역 변수, 정적 변수(static 키워드로 정의된 클래스변수), 인스턴스 변수(클래스 내부에서 정의)  
  정적 변수는 클래스와 생명 주기를 같이한다.
* 상수
* 반드시 선언과 동시에 초기화해야 하며, 프로그램이 실행되는 동안 저장된 데이터를 변경할 수 없다.
* final 키워드를 이용해서 선언한다. (ex. final int SPEED=20;)

(입출력 시 주의 – 출력은 System.out.println이다. 스캐너 입력받을 때가 Scanner(System.in)이다.

1. 자료형

* 데이터 형태를 식별하는 분류.
* 기본 자료형: 프로그램이 실행될 때 인식할 수 있도록 이미 컴파일러에 하드 코딩 되어있음  
  정수형(byte, short, int, long), 실수형(float, double), 문자형(char), 불리언형(boolean)
* 객체 자료형: 프로그램에 해당 정의가 포함된다  
  문자열형(String), 배열, 클래스, 인터페이스 등. //자료형 중에 문자열만 참조형이다. (String 클래스)
* 정수 자료형 – byte(1바이트), short(2바이트), int(4바이트), long(8바이트-자바에서!)  
  long l = 123456789L; //long임을 유추할 수 있도록 맨 뒤에 L 쓴다
* 실수 자료형: 소수 부분이 있고 음숫값을 허용, float(4바이트), double(8바이트)  
  float fl = 12.25f; //맨 뒤에 f 붙임

double d = 123.45d; //맨 뒤에 d 붙임

* 문자 자료형: char, 문자 집합의 기호를 나타내는데 사용. 작은따옴표로 값을 묶어 표현. 단일 문자를 포함하는 문자열인 “A”와 다름.
* 문자열 자료형: String, 문자열 저장에 사용. 큰따음표로 값을 묶어 사용한다.  
  System.***out***.println(123+123+123+"123"); //결과: 369123  
  System.***out***.println("123"+123+123+123); //결과: 123123123123
* 불리언 자료형: 값은 true와 false 두 가지뿐. if 또는 while과 같은 제어문을 제어하는 조건식에 사용.

1. 출력 처리

* 데이터 출력
* print() 메서드: System 클래스를 사용해야 한다.

System.out.print() : 따옴표 안의 문자열, 변수 값을 출력

System.out.println() : 따옴표 안의 문자열, 변수 값 출력 후 다음 행으로

System.out.printf() : 문자열 포맷팅(형식화) 제공

+문자열과 변수 결합 출력 가능

* 이스케이프 문자 : \b, \t, \n, \r, \”, \’, \\
* 증가 및 감소 연산자 : ++ 또는 --. 증감 연산자 전위/후위 따라 출력 결과 달라지는 문제~
* 형변환 : 자동형변환(암시적형변환), 캐스팅형변환(명시적형변환)
* 자동형변환: 작은 자료형을 큰 변수에 할당할 때 더 큰 자료형으로 자동 변환. ex-int형 변수에 short값 할당
* 캐스팅형변환(강제,명시적): 작은 자료형에 변수에 더 큰 자료형의 값을 할당할 때. 데이터 일부가 손실될 수 있음.

자료형 변수명 = (자료형)변수명;

(형변환 시의 연산결과 주의. 정수/실수->결과 실수)

* Scanner 클래스
* 키보드로 데이터를 입력받을때 사용함. java.util.Scanner;패키지를 임포트해서 쓴다
* (메서드) nextInt(), nextFloat(), nextDouble(), nextLong(), nextShort(), next(), nextLine(), nextBoolean();

next()는 공백문자 불포함 문자열=즉 단어별로 받는다. nextLine()은 한 줄을 읽어온다.

* 입력처리과정

Scanner 클래스의 패키지 임포트🡪 Scanner 클래스의 객체 생성 (System.in: 키보드 입력 스트림 객체, 키보드로 데이터를 입력받기 위해. //Scanner sc = new Scanner(System.in))🡪입력 메서드 호출

(\*객체 접근 시 레퍼런스 변수(sc)를 통해 간접적으로 접근한다.)

import java.util.\*;하면 java.util 패키지 전체의 파일들이 참조된다 (폴더는 참조 안됨)

[week02-2]

1. 제어문

제어문 : 조건문, 반복문, 분기문.

* 프로그램 코드 실행시 코드의 처리 순서를 변경할 수 있는 문장. 명령의 흐름 처리 방법을 지정할 때 사용.
* 조건문: 조건에 따라 처리하는 명령을 달리함(if~else, switch)
* 반복문: 일정한 조건이 충족될 때까지 동일한 명령을 반복적으로 처리(for, while, do~while)
* 분기문: 강제로 명령의 처리 순서를 바꿈(break, continue, return)

1. 조건문

주어진 조건의 참.거짓 판별하고 그에 따라 특정 명령을 수행

* if-else, switch, + break, continue
* if문: if (조건식) {//조건식이 참이면 실행되는 코드} //이때 조건식에는 결과가 참 또는 거짓인 연산식이나 불리언 변수가 들어감
* if-else문: 조건식의 결과에 따라 특정 코드 또는 코드 블록의 실행을 선택. 이때 else는 따로 조건식을 작성하지 않음
* if-else if문: else if 수 제한없음
* 중첩 if-else문
* switch문  
  switch(연산식 또는 변수) {

case 값1:

//실행 코드

break;

default:

//실행 코드

}

1. 반복문

일정 횟수만큼 어떤 코드를 반복 실행. 반복적으로 실행되는 동작 루프(loop)라고 함.

* for문, while문, do~while문 //for(;;)무한루프
* for문: 특정 횟수만큼 실행해야 할 때 좋음. 자바는 초기화식에서 변수 선언해도 됨 int i=0; 이렇게

for (초기화식; 조건식; 증감식) {…}

* while문: 조건식이 참이면 루프를 계속 반복. 조건식은 대개 비교 또는 논리 연산식.

while (조건식) {…}

* do~while문 (while문과 for문처럼 조건식에 의해 반복 실행된다는 점은 비슷, 하지만 조건식을 확인하기 전에 코드 블록의 코드가 한 번 실행된다는 점이 다르다)
* 중첩 반복문: 외부 루프 후 내부 루프 (별표로 역삼각형 만들기 등의 문제에서 조건식 주의)

1. 분기문

* break, continue, return
* break: 반복문 중지 혹은 종료
* continue: 아래부분을 건너뛰고 싶을 때 – 반복문 블록 내부에서 for문의 증감식 또는 while문의 조건식으로 이동

1. 메서드

* 코드의 집합. (재사용성, 가독성, 유지보수성 향상)
* 메서드의 유형: 표준 라이브러리 메서드, 사용자 정의 메서드
  + 표준 라이브러리 메서드: 자바 클래스 라이브러리에 이미 정의되어있는 메서드. 사전 정의 메서드. 프로그램에서 호출하기만 하면 언제든 사용 가능.
  + 사용자 정의 메서드: 사용자 또는 프로그래머가 작성한 메서드, 필요에 따라 추가/보완/수정/삭제 할 수 있음  
    - 정적 메서드 / 인스턴스 메서드
* 메서드의 기본 구조  
  [접근제한자] 반환유형 메서드명 ([매개변수목록]) //선언부

{ //메서드 본문… } //구현부

* 메서드 호출: 메서드명 뒤의 괄호 안에 인수(매개변수 목록)가 있는 경우🡪인수를 사용해 메서드 호출

1. 사용자 정의 메서드 생성

* 반환 유형이 있는 메서드: 메서드명 앞에 자료형을 정의함, 메서드 내부 마지막 행에 return 키워드를 사용하여 메서드명 앞의 자료형과 동일한 값을 반환함
* 반환 유형이 없는 메서드 : 메서드명 앞에 void 키워드 사용, 메서드 안에 return이 없음
  + [입력=매개변수/출력=반환유형] : o/o, o/x, x/o, x/x 가능

1. 메서드 오버로딩

* 메서드명이 같지만 매개변수가 다른 메서드를 하나의 메서드명으로 정의하는 것
* 메서드 오버로딩을 위한 조건: (프로토타입 부분) 메서드명이 같음, 매개변수의 자료형이나 개수가 다름

**JAVA프로그래밍 및 실습 - 중간고사 공부 2276219 이서현**

[Week3] 배열과 문자열

1. 배열

* 배열은 같은 유형의 데이터를 모아둔 집합. 인덱스를 통해 각각의 데이터에 접근.
* 모든 배열을 동적으로 할당된다.
* 배열의 길이는 ‘배열변수명.length’로 계산
* 배열 변수를 자료형 뒤에 대괄호를 사용해 다른 변수와 같이 선언할 수 있다.
* 배열 내부의 변수는 인덱스 0부터 시작한다.
* 배열은 정적 변수, 지역변수 또는 메서드 매개변수로도 사용 가능
* 배열의 크기는 int값으로 지정해야 한다
* 배열의 슈퍼 클래스는 Object다
* 인덱스 오류: ArraysIndexOutOfBoundsException
* 한계를 벗어난 인덱스에 접근하려고 시도할떄 발생하는 오류. 프로그램이 배열 범위 밖의 요소에 액세스하려고 시도할 때 인덱스 오류 발생.

1. 1차원 배열과 2차원 배열

* 1차원 배열: 단일 인덱스를 사용해 요소를 저장. 인덱스 1씩 증가시켜 사용. 실제 배열은 new 연산자를 사용해 생성.  
  int[] myArr : 배열 선언 / =new int[5]; : 배열 생성(5개의 정수를 저장할 메모리 공간이 배열에 할당되었음을 나타냄)
* 1차원 배열 초기화: 배열을 선언하고 초기화할 때 배열의 요소를 직접 설정함. {} 안에 요소를 안 넣으면 길이가 0인 배열생성
* 배열의 크기: .length로 확인가능
* for-each 루프 : for(자료형 변수명 : 배열명) { //이 안에서 변수명을 list[i]처럼 사용, 루프마다 그다음 요소}  
  **int**[] list = {1, 2, 3, 4, 5};

**for**(**int** e : list) {

System.***out***.println(e);

}

* 2차원 배열 : 행과 열이 여러 개. 표 형식으로 데이터를 나타냄.  
  자료형[][] 배열명 = new 자료형[행크기][열크기]; //new는 메모리 할당 키워드  
  int[][] myArr = new int[3][4];
* 2차원 배열 초기화: 각 내부 중괄호는 하나의 행을 나타냄. 선언과 동시에 값 선언하고 할당해 초기화 가능.
* 래그드 배열: 배열의 배열. 1차원 배열 요소에 배열을 저장하는 방식

**int**[][] ragged = **new** **int**[3][];

ragged[0] = **new** **int**[1];

ragged[1] = **new** **int**[2];

ragged[2] = **new** **int**[3];

**for**(**int** r=0; r<ragged.length; r++) {

**for**(**int** c=0; c<ragged[r].length; c++) {

ragged[r][c]=c;

System.***out***.print(ragged[r][c]);

}

System.***out***.println();

}

* ArrayList: 자바에서는 ArrayList라는 클래스를 제공하는데 이 클래스를 사용하면 배열의 크기를 동적으로 변경하면서 사용할 수 있다.

ArrayList<String> list; //String은 ArrayList에 저장할 자료형. list는 참조변수

list = **new** ArrayList<>(); //ArrayList 생성

list.add("Apple");

list.add("Grape");

System.***out***.println(list); //출력: [Apple, Grape]

* 문자열: 문자의 배열. 공백도 문자. 항상 “”안에 작성. String클래스의 객체임. new키워드로 문자열을 메모리에 동적으로 할당. 아래의 코드에서 s1==s2이다 (새롭게 메모리 공간이 할당되지 않음)

String s1 = "A";

String s2 = "A";

String s3 = **new** String("A");

String s4 = **new** String("A");

System.out.println(System.identityHashCode(s1)); //s1, s2의 주소값은 같다

System.out.println(System.identityHashCode(s2));

System.out.println(System.identityHashCode(s3)); //s3, s4는 다르다

System.out.println(System.identityHashCode(s4));

[JVM 메모리] 문자열을 저장할 때 s1, s2는 힙 내의 ‘스트링 풀’에 저장된다.(s2 만들 때 “A”가 새롭게 메모리 공간 할당되지 않음) s3, s4는 그냥 힙에 저장. //더 찾아보기

* 객체 비교와 .equals() 메소드: (feat.week07자료)

‘==’는 주소값이 같냐는 의미. 두개의 레퍼런스를 비교한다.

boolean equals(Object obj)는 객체 내용이 같은지를 비교한다.

* String 클래스의 메서드
* length(): 문자열 길이를 반환 / isEmpty(): 문자열이 비어 있는지 확인 / charAt(int index): 특정 인덱스에 대한 char 값을 반환. / substring(int startIndex, int endIndex): 주어진 시작 인덱스와 끝인덱스에 대한 부분 문자열을 반환 / concat(String str): 두 문자열을 결합 / indexOf(char ch): 지정된 문자의 인덱스를 반환 / equals(Object anotherObject): 문자열과 객체가 같은지 확인. 데이터를 비교할 땐 equals를 사용해야 한다. / comparTo(Object obj) : 문자열을 객체와 비교 / trim(), replace(char oldChar, char newChar) 등…

String s1="Ja";

String s2="va";

String s3=s1.concat(s2);

String s4=" Java ".trim();

System.***out***.println(s3.equals(s4) + " " + s4.compareTo(s3)); //출력: true 0

* 문자열 연산자 :  
  + : 두 문자를 연결한다

-- : 두 문자열의 메모리가 같으면 true를 반환, != : 두 문자열의 메모리가 다르면 true를 반환

equals() : 두 문자열 내용이 같으면 true를 반환

\*문자열-""로 초기화한 것들은 스트링 풀에 저장

String s1 = **new** String("Java");

String s2 = **new** String("Java");

String s3 = s1;

**if**(s1 == s2) System.***out***.println("s1==s2");

**else** System.***out***.println("s1!=s2");

**if**(s1 == s3) System.***out***.println("s1==s3");

**else** System.***out***.println("s1!=s3");

**if**(s1.equals(s2)) System.***out***.println("s1==s2");

**else** System.***out***.println("s1!=s2");

**JAVA프로그래밍 및 실습 - 중간고사 공부 2276219 이서현**

[week04] 객체

1. 객체 지향 특성: 캡슐화

* 자바의 클래스와 객체

클래스: 객체 모양을 선언한 틀(캡슐화)

* 필드(멤버 변수)와 메소드(멤버 함수)는 모두 클래스 내에 구현

객체: 클래스 모양대로 생성될 실체=인스턴스. 객체 내 데이터를 보호, 외부 접근 제한 (객체 외부에서는 비공개 멤버(필드, 메소드)에 직접 접근할 수 없고 공개된 메소드를 통해 비공개 멤버를 접근한다.)

클래스를 선언하면 클래스 모양으로 객체를 만들 수 있음 (클래스는 필드 부분과 메소드 부분으로 나뉨)

* 상속
* 자식 클래스가 부모 클래스의 속성 물려받고 기능 확장. is-a관계 성립.
* 부모 클래스: 수퍼클래스 / 자식 클래스: 서브클래스. 수퍼 클레스 재사용 및 새로운 특성.

\*서브 클래스 객체는 슈퍼 클래스의 멤버와 서브 클래스의 멤버를 모두 가진다.

\*자바는 다중 상속이 없음.

2. 객체 지향 특성: 다형성 (같은 이름의 메소드가 클래스나 객체에 따라 다르게 동작하도록 구현.)

* 다형성 사례: 메소드 오버로딩, 메소드 오버라이딩

[절차 지향 프로그래밍, 객체 지향 프로그래밍]

\*클래스: 객체를 만들어내기 위한 설계도 혹은 틀. 객체의 필드와 메소드 포함.

\*객체: 클래스의 모양 그대로 찍어낸 실체. 프로그램 실행 중에 생성되며 메모리 공간 가짐. 인스턴스라고도 부름

객체들은 클래스에 선언된 동일한 속성을 가지지만 객체마다 속성값은 다를 수 있다.

* 클래스 구성- 접근지정자 클래스선언 클래스이름 필드변수 메소드

생성자: 클래스의 이름과 동일한 특별한 메소드. 객체가 생성될 때 자동으로 한 번 호출된다. 개발자는 객체를 초기화하는데 필요한 코드를 작성.

* 객체 생성 및 접근
* 객체 생성: 반드시 new 키워드 이용해서 생성. new는 객체의 생성자들 호출한다.
* 객체 생성 과정: 객체에 대한 레퍼런스 변수 선언 (Circle c)->객체 생성 (c = new Circle();하면 클래스 타입 크기의 메모리를 할당, 객체 내 생성자 코드 실행)
* 객체의 멤버 접근: 레퍼런스변수.멤버 (c.radius = 10; c.getArea(); 이렇게)

//예제

* 생성자(constructor) 개념: 객체가 생성될 때 초기화를 위해 실행되는 메소드

//예제

* 생성자의 특징: 생성자는 메소드, 생성자 이름은 클래스 이름과 반드시 동일하다. 생성자 여러 개 작성 가능(오버로딩)

->오버로딩 시 매개변수의 개수/타입/타입순서 가 서로 달라야 한다. 생성자는 new를 통해 객체 생성 시 객체당 한 번 호출된다.

\*\***생성자는 리턴 타입을 지정할 수 없다.** (접근 지정자만 씀) 생성자의 목적은 객체 초기화이다. 생성자는 객체가 생성될 때 반드시 호출된다. 그러므로 하나 이상 선언되어야 함(작성하지 않으면 컴파일러가 자동으로 기본 생성자 삽입)

//예제

* 기본 생성자
* 매개변수x, 아무 작업 없이 단순 리턴. 클래스에 생성자가 하나도 없다면 컴파일러 의해 자동 삽입.  
  그러나 개발자 클래스에 생성자가 하나라도 작성된 경우 기본 생성자 자동 삽입되지 않음, 컴파일 오류

//예제

* this 레퍼런스
* this: 객체 자신에 대한 레퍼런스. 다른 메소드 호출 시 객체 자신의 레퍼런스를 전달할 때 사용
* **this.멤버**: 객체의 멤버 변수와 메소드 변수의 이름이 같은 경우 사용 (+ this.멤버메소드())
* **this()**: 생성자가 클래스 내의 다른 생성자를 호출할 때 사용 (생성자 내에서 사용 가능, **반드시 생성자 코드의 제일 처음에 수행**) super()와 같이보기

//예제

* 객체의 치환: obj1=obj2; 같은 경우. 객체가 복사되는 것이 아니며 레퍼런스가 복사된다~!!
* 객체 배열: Circle[] c; //Circle 배열에 대한 레퍼런스 변수 c 선언🡪레퍼런스 배열 생성. c[i]=new Circle(i);하면 배열의 각 원소 객체를 생성.