[ JAVA ]

1. Datatype(자료형)

int(정수), double(실수), char(문자), String(문자열)

1. 연산자

+, -, \*, /, %(나머지),

+=, -=, \*=, /=, %=

==, !=, ++, --

1. 입출력

출력 : System.out.println();, System.out.print();

입력 : Scanner scan = new Scanner(System.in); + int n = scan.nextInt();

문장을 입력하기 위해서는 nextLine();를 사용해야 한다.

next.Int(), nextDouble(), next() 다음에 nextLine() 사용할 때에는 반드시 엔터 처리가 필요하다!!!

1. 조건문 (if문, switch case문)

if(조건식){조건이 true 일때 실행되는 문장};else if{조건이 false 일때 실행되는 문장}else{이전 조건이 false 일때};

switch(변수이름){

case 변수: 해당 값 break;, case 변수: 해당 값 break;, case 변수: 해당 값 break;};

* Break를 사용하는 이유는 더 이상의 불필요한 실행을 정지하기 위함.

If 와 switch case 문의 차이는 조건식에 해당하는 구간에 해당하는 변수들과 정해진 변수이름을 사용하는 지의 여부임. 12달의 해당 일자 수를 지정하는 상황에서는 switch case문이 유용하지만 나머지 조건으로 인해 분류하는 상황은 if문이 유용함.

1. 반복문(while문, for문)

While(조건식(true or false)){조건식이 true 일때 실행되는 문장};

* 조건식에 true or false 만 나눌 수 있는 식이 주어짐

For(초기값;조건식;증감식){( )안의 내용을 충족시키는 문장이 실행};

While 와 for 문의 차이는 초기값과 증감식이 있는지의 여부임.

1. 제어문(break문, continue문)

Break와 continue문은 모두 제어문 안에서 사용되어야 하고 이는 모두 제어문을 탈출하고 종료시키는 역할을 함.

Break는 반복문에서 해당 break문 이후부터는 반복을 중지하는 것을 의미하고

Continue는 반복문에서 continue문 이후부터의 문장을 수행하지 않고 다음 반복문으로 넘어감(점핑).

1. 배열

Int[], double[], String[], int[][], double[][], Stiring[][]…

기본적인 형태는 int[] arr = new int[5];

* arr이라는 배열변수에 5칸짜리 배열을 만들겠다(int자료형으로). 이를 출력하면
* arr[0], arr[1], arr[2], arr[3], arr[4]가 되고 각각의 값에 일반 변수처럼 값을 지정할 수 있다.
* 하나하나 값을 지정할 수도 있지만 배열에 해당하는 값들을 모두 알고 있을 경우에는
* Int[] = {10,20,30,40,50};으로 한번에 지정할 수도 있다.
* 2차원 배열
* Int[][] = new int[a][b];
* (정수타입) 배열을 만드는데 [a][b]에서 b개짜리 배열을 총 a개 만들겠다는 것을 의미함.
* Int[][] = new int[4][2];
* Int list = {{1,2}, {2,3}, {3,4}, {4,5}};

1. 랜덤클래스

기본적인 형태는 Random ran = new Random(); + int n = ran.nextInt();이다.

Int n = ran.nextInt(5);는 (정수타입의 변수) n에 0부터 4까지의 5개 숫자 중 하나를 랜덤으로 지정함.

Int n = ran.nextInt(45)+1;은 로또 번호 처럼 1 ~ 45까지의 숫자 중 하나를 랜덤으로 지정함.

1. 문자열 데이터 -> 숫자로 변환

String str = “0”;

int num = Integer.parseInt(str);

* str문자열로 저장되어 있던 0이 숫자 형식으로 num에 변환되어 저장됨.

-------------------------------------------------------------------------------------------------------------------------------

\*클래스

클래스는 객체를 만들어 내기 위한 설계도이다.

클래스는 속성과 기능으로 이루어짐.

속성은 클래스가 만들어 낼 객체의 자료형이나 변수이름을 의미하고

기능은 클래스에서 생성된 변수가 실행하는 메소드를 의미

class Pokemon{

String name;-> Pokemon이라는 클래스를 임의로 만들고 그 안에서 사용할 변수들과 타입을 사전에 지정한다.

**int** lv;

**int** exp;

}

**class** Student{

// 클래스 변수 -> 클래스변수는 Student에서 변하지 않는 고정변수(static)

**static** String *school*;

**static** String *prin*;

// 인스턴스 변수 -> 인스턴스변수는 값이 변하는 변수

String name;

String id;

**int** age;

}

**public** **class** Ex07 {

**public** **static** **void** main(String[] args) {

Student.*school* = "메가고등학교";

Student.*prin* = "홍길동";

->변하지 않는 값이기 때문에 객체를 생성하기 이전에 클래스에 값을 지정해 놓는다.

Student s1 = **new** Student();

s1.name = "둘리";

s1.id = "123";

s1.age = 18;

System.***out***.println("이름 : "+s1.name);

System.***out***.println("ID : "+s1.id);

System.***out***.println("나이 : "+s1.age);

}

\*return문

값을 반환하는 문장이다

return문을 호출한 자리에 해당 메소드의 계산 결과 값을 저장하고 사라진다

return문이 사용되면 그 메소드는 사용이 종료된다(값을 남기고)

**class** Student{

String name;

**int** math;

**int** eng;

**int** kor;

**void** setStudent(String name, **int** m, **int** e, **int** k) {

**this**.name = name;

**this**.math = m;

**this**.eng = e;

**this**.kor = k;

}

* 클래스에 구성될 각각의 변수를 지정하고 입력 받는 변수 값을 인스턴스 변수에 저장

**int** getTotal(){

**int** total = 0;

total = math+eng+kor;

//System.out.println(total);

**return** total;

//System.out.println("hello"); -> 이미 종료되었기 때문에 작동하지 않는다!

}

* int는 return을 통해 남겨지는 값에 대한 자료형을 지정하는 것이고 return을 통해 값이 남겨지기 때문에 void를 사용하지 않는다.

**double** getAvg() {

//int total = 0;

//total = math+eng+kor;

**int** total = getTotal();

**double** avg = total/3.0;

**return** avg;

}

* int total = getTotal();로 지정한 이유는 위에서 이미 getTotal();이 return 값을 남기고 갔기 때문에 return값을 바로 int total에 저장해서 새로운 메소드를 계산함. 그리고 다시 나온 값을 avg로 저장해서 return값으로 저장하고 종료.

\*메소드 오버로딩(Overloading)

동일한 이름의 메소드를 정의할 수 있다.

하지만 이름이 동일하기 때문에 매개변수의 타입이나 개수가 달라야 한다.

같은 메소드로 입력되더라도 입력되는 매개변수의 타입이나 개수가 클래스에서 지정된 같은 형식의 메소드가 실행된다.

public void setUser(int a, int b){1출력};

public void setUser(int a, int b, int c){2출력};

u1.setUser(1,2,3); -> 2출력

u2.setUser(1,2); -> 1출력

* 여기까지 JAVA1

객체지향프로그램 JAVA

기존처럼 main클래스를 바로 만들지 않고 클래스와 메소드를 직접설정하고 만들어서 그에 대한 객체를 만들어 냄.

이러한 객체들은 블록화 되어 있기 때문에 필요한 클래스끼리 묶을 수 있고 유지보수 등이 용이하다.

기본적인 개념을 이해하고 있는 것도 중요하지만 전체적인 JAVA라는 언어에 있어서 큰 그림을 이해하여야 하고 다른 사람들의 코드를 보면서 어떻게 그러한 코드를 짜는지에 대해 따라하고 익혀야 한다.

클래스에는

1. 요소(속성, 멤버변수, 필드)

2. 기능(메소드)가 있다.

기존까지는 멤버변수에 따로 접근제한자를 설정하지 않아서 public였다.

\*접근제한자

default : 같은 패키지안에 클래스 멤버에 접근가능

protected : 상속관계에 있어서 자식클래스가 부모클래스의 멤버변수에 접근가능

private : 따로 getter을 설정해야만 접근가능

public : 모두가 접근가능

\*getter / setter

getter은 private 되어 있는 멤버변수 값을 사용하기 위해 getter을 통해 우회하여 접근

private String maker = “apple”;

public int getMaker(){

return this.maker;}

* System.out.println(cellphone1.getMaker()); 하면 apple출력

setter은 private 되어 있는 멤버변수 값에 새로운 값을 입력하기 위해 사용

public void setMaker(String maker){

this.maker = maker;}

cellphone1.setMaker(samsung);

* cellphone1의 멤버변수 maker에는 Samsung 값이 입력되어 있다.

\*생성자 클래스

생성자 클래스는 객체를 인스턴스화하는 동시에 실행되는 것이다.

리턴타입이 없다.

생성자 클래스는 클래스 명이 기존 클래스 명과 같다.

생성자 클래스 또한 각각 다른 매개변수(parameter)를 통해 생성자 오버로딩이 가능하다.

객체를 구현할 때 생성자에 바로 변수를 입력하기 위해서는

public Person(String gender, String name){

this.gender = gender;

this.name = name;

}

Person p1 = new Person(“남자”, “임건호”);

* 결과는 p1이라는 객체의 gender에 남자가 name에 임건호가 입력저장되어있음.

메소드를 만들 때 결과값에 대한 저장이 필요하면 return을 활용한다.

\*상속

상속은 부모클래스에서 지정해 놓은 멤버변수나 클래스 메소드를 자식클래스가 사용하는 것

따라서, 부모클래스보다 자식클래스가 더 많은 내용을 가지고 있음

부모클래스의 멤버변수는 protected로 자식클래스에서는 이용가능

public class Student(자식) extends Person(부모){}

자식클래스는 하나의 부모클래스만 상속할 수 있음

\*메소드 오버라이드(Override)

오버라이드는 부모클래스에 존재하는 메소드를 자식클래스에서 같은 이름의 메소드로 다시 실행하는 것이다.

대신 자식클래스에서 사용할 때에는 @Override라고 표시해주며 사용하면 되고

부모클래스의 메소드를 사용할 때에는 super.showInfo();라고 하면된다.

\*업캐스팅과 다운캐스팅(upcasting, downcasting)

Animal a1 = new Dog();

a1.eat(); -> 부모의 클래스에 있기 때문에 가능

a1.cry(); -> 부모클래스의 메소드를 오버라이드했기 떄문에 가능

a1.lash(); -> 자식클래스에만 있는 메소드이기 때문에 불가능

업캐스팅(Animal a1 = new Dog();)

1. 부모클래스에 있는 메소드를 사용할 수 있다.

2. 부모클래스에 있는 메소드를 오버라이드 한 메소드를 사용할 수 있다.

3. 자식클래스에 있는 메소드는 사용할 수 없다.

다운캐스팅(Dog d1 = (Dog) a1;)

* Animal의 객체인 a1 을 Dog의 객체로 만들기 때문에 자식클래스의 메소드 사용가능

부모클래스 -> 자식클래스 다운캐스팅은 캐스팅을 명시로 해주어야 한다. -> 자식클래스의 메소드를 사용가능(d1.lash();가능)

a1.lash(); -> 사용불가 이유는 부모의 클래스에 upcasting 했기 때문에 독단적으로 생성한 자식의 lash는 사용할 수 없다(부모의 클래스에는 없고 모름)

instanceof를 사용해서 다운캐스팅이 올바르게 사용되었는지 확인하기 위해 사용(업캐스팅은 상관x)

\*추상클래스, 추상메소드

public abstract class Animal{}

1. 추상클래스는 추상메소드가 하나 이상 포함되어있는 클래스이고 안에 일반적인 메소드도 포함할 수 있다.

2. 추상클래스는 객체를 따로 생성할 수 없고, 대신 추상클래스를 상속받은 자식클래스만 객체화 할 수 있다.

public abstract void cry();

-> 이상태로 끝남

-> 자식클래스에서 구현해야함

public class Dog extends Animals{

public void cry(){

System.out.println(“멍멍”);

}

}

3.추상메소드는 자식클래스가 반드시 구현해야하는 메소드이다.

\*인터페이스

인터페이스는 클래스에 의해 구현되고 인터페이스에 나와 있는 메소드는 반드시 해당클래스의 의해 구체화되고 실행되어야한다.

마치 여러가지 기능이 클래스에 존재하지만 타인에게 특정기능만을 사용하게 하기 위해 인터페이스를 통해 지정한 기능만을 사용하게 할 수 있다.

ex) 핸드폰의 게임기능만 사용하게 하기 위해

게임 phone = new Phone();

Phone에는 전화기의 다른 기능들도 있음

게임은 인터페이스이고 안에는 게임한다는 메소드만 구현

1. 새로운 클래스를 만들 때 new -> interface를 선택하면 만들어진다.
2. 기존 클래스에서 인터페이스를 사용하기 위해서는

public class JobApplicant implements Cprogramming, Javaprogramming

1. 인터페이스는 상속과는 다르게 동시에 여러 개의 인터페이스를 구현가능
2. 인터페이스 내에 메소드는 해당클래스에서 추상클래스와 동일하게 구체화해서 구현

\*wrapper 클래스(boxed type)

기본자료형을 객체로 만들어주는 것

Integer, Character...

int n = 5; 기본자료형(primitive type)

Integer n2 = new Integer(5);

\*Collection framework

기존에 배열을 통해 항목들을 저장했지만 이후에 수정이나 보충을 용이하게 하기위해 사용

배열 (Array)

1. 크기를 미리 알고 있어야 한다. 크기가 고정되어 있다.

2. 중간에 있는 값을 빼내면 비어있는 채로 유지된다.

3. index 기반으로 접근이 가능하다.

4. 기본자료형(primitive type), 객체도 담을 수 있다.

List

1. 크기가 고정되어 있지 않고, 동적으로 할당된다.

2. 중간에 있는 값을 빼내면 앞으로 당겨진다.(메모리 낭비가 없다)

3. 여러 방법으로 데이터를 다룰 수 있다.

4. 객체만 담을 수 있다.(기본 자료형 안됨)

5. 타입 안정성을 보장하는 generic을 쓸 수 있다.

List<Integer> list1 = new ArrayList<>(); -> list에 Integer만 저장

list1.add(1); -> list1에 1이 저장

List<String> list2 = new ArrayList<>(); -> list에 String만 저장

list2.add(“aaa”); -> list1에 aaa가 저장

* 배열은 맨처음에 지정하는 수만큼 저장하는 공간의 개수가 지정되지만 ArrayList에는 맨처음에 10개의 공간 그 다음에는 저장된 개수의 반만큼씩 저장 공간이 생겨난다.

List<Integer> list3 = new LinkedList<>();

* 저장된 값의 중간에 값이 제거되면 뒤에 있는 값이 앞으로 밀려와서 저장되고 중간값을 추가하고싶으면 저장되어 있는 값들 사이에 추가저장가능

많은 메소드들이 있다

list3.add(); -> 새로운 값을 추가

list3.add(1,”a”) -> 1번째 자리에 문자a를 추가

list3.addAll(); -> ()안에 있는 것을 전부 추가

list3.contains(); -> ()안에 있는 것을 포함하고 있는지?

list3.remove(); -> ()안에 있는 것을 제거

list3.clear(); -> 전부삭제

list3.isEmpty(); -> list3가 비어있는지?

list3.sort(Comparator.naturalOrder()); -> list3에 있는 값을 오름차순으로

list3.sort(Comparator.reverseOrder()); -> list3에 있는 값을 내림차순으로

일반적으로 collection framework 의 출력은

for(int i=0; i<list1.size(); i++){

String element = list1.get(i);

System.out.println(element);

}

* list1.length;를 쓰는 것이 아니라 list1.size();를 사용
* 추가적으로 list1의 값을 String element에 저장할때 list1.get()로 불러온다.

\*향상된 for문(=foreach문)

for(String element : list1){

System.out.println(element);

}

* 기존의 for문 대신에 사용하는 향상된 방법으로 ()안에 String element변수를 입력하고 = 대신에 :를 사용해서 우항에는 객체를 입력한다.

\*Iterator(반복자)

Iterator<String> iter = list1.iterator();

while(iter.hasNext()) {

String element = iter.next();

if(element.startsWith("b")) {

iter.remove();

}

}

System.*out*.println(list1);

\*3항 연산자

-> 조건, 참, 거짓을 한번에 나타낼 때 표현하는 식

->   
boolean isSuccess = fruit.add(scan.nextLine()); // boolean 을 통해서

// 3항 연산자 : 조건? \_\_참\_\_ : \_\_거짓\_\_

System.*out*.println(isSuccess? "입력 성공" : "입력 실패");

* isSuccess ?를 통해서 조건을 물어보고 이에 대한 답이 참 일 때 “ ”, 거짓일때 “ ”의 값을 한번에 나타내는 식

System.*out*.println(fruit.isEmpty()? "비어있습니다" : fruit);

* fruit.isEmpty() ? 를 통해 결과값의 참, 거짓의 유무에 따라 출력되는 값을 한번에 나타냄.

\*참고사항

1. String 특성의 객체 끼리 비교할 때는 .equals() 처럼 앞에서 배운 String객체 비교 메소드를 활용한다.

2. ArrayList.asList(100, 101, 102); -> 리스트에 한번에 입력하는방법(하지만 추가나 수정불가

\*Set(집합)

1. Collection의 하위 항목 중 하나이다.

2. 입력값이 중복되지 않는다.

3. index 기반이 아니다.

4. 전반적인 특징은 list와 같지만 중복이 되지 않는다는 특징을 갖고있다.

Set<Integer> set1 = new HashSet<>();를 사용하고 (HashSet를 가장 많이 사용)

set1.add();

set1.addAll(set2); -> 합집합

set1.contains();

set1.retainAll(set2); -> 교집합

set1.removeAll(set2); -> 차집합

\*Map

1. Collection 과 다른 것

2. key – value 쌍으로 이루어져 있다.

3. key는 중복되지 않지만, value는 중복가능(주민번호 - 이름)

4. key를 통해서 해당 value를 아주 빠르게 찾을 수 있다.

5. value로는 key를 찾기 어렵다.

형태는

Map<String, Integer> map1 = new HashMap<>();

String타입의 key와 Integer타입의 value를 map1의 값으로 입력

map1.put(“a”, 1);

map1.put(“b”, 2);

map1.put(“c”, 3);

map1.put(“d”, 4);

map1에 각각 key값과 value값을 입력(List나 Set에서는 add를 사용했지만 Map는 put)

map1을 출력하면 {a=1, b=2, c=3, d=4}가 나온다.(중괄호로…)

map1.containsKey(“a”);

map1.containsValue(1);

map1.size();

key만 출력하려면

Set<String> keys = map1.keySet();

를 통해 keys안에 key값들을 저장

value만 출력하려면

Collection<Integer> values = map1.values();

를 통해 values안에 value값들을 저장

map1안에 들어있는 내용을 제거할 때는

1. key값을 이용하는 방법

* map1.remove(“c”);

1. key값과 value값을 이용하는 방법

* map1.remove(“d”, 4);

\*예외처리(Exception)

->예상치 못한 상황을 대처하는 것

자주 발생하는 예외

1. NullPointerException = NPE
2. ArrayIndexOutOfBoundException
3. ClassCastException

형태는

* try{예외의 발생이 예상되는 로직}
* catch(예외클래스 인스턴스){예외가 발생했을때 실행되는 로직}

try{}catch(NullPointerException e){System.***out***.println("NPE가 발생했다.")}

catch는 예외가 발생했을 경우 뒷수습하는 것

catch(){}를 연결해서 다중catch문 사용가능

catch는 else if문의 else와 비슷한 맥락!

\*finally{예외여부와 관계없이 무조건 실행되는 로직}

앞에서 try-catch문에서 예외가 발생하더라도finally문에서는 기존의 로직이 실행된다.

\*파일 입출력(Input, Output)

1. 파일쓰기

FileOutputStream output = new FileOutputStream(“output.txt”,true);

() 안에 있는 파라미터의 의미

앞에 있는 output.txt 는 내가 저장한 파일의 이름이고

뒤에 있는 true, false (true는 기존에 있던 내용에 이어서 내용을 추가, false는 기존의 내용에 입력하는 새로운 내용을 덮어씀)

String text = “반갑습니다.”;

output.write(text.getbyte()); -> output파일에 String text를 byte로 입력한다.

output.close() -> 스트림을 닫음.

2. 파일입력

FileInputStream inputStream = new FileInputStream(“src/output.txt”);

* inputStream에 “src/output.txt”의 경로에 있는 파일을 가져온다.

InputStreamReader streamReader = new InputStreamReader(inputStream, “utf-8”);

* streamReader에 inputStream파일을 utf-8양식으로 인코딩해서 가져옴

BufferedReader br = new BufferedReader(streamReader);

* br에 streamReader을 한줄한줄씩 가져옴

String readline = br.readline();

readline에 br을 한줄씩 가져와서 저장함.

이후에는 for문을 통해 한줄씩 br을 출력한다.