문법정리

2020년 5월 9일 토요일 오후 10:01

오후것 3시부터 시작함~~ 영상보고 다시 정리하기!!!! ****혼자 다 말할 수 있을 정도가 되어야함****

<생성자>
클래스명과 이름이 동일
반환타입이 없다.
디폴트 생성자 : jvm이 자동으로 만들어줌. 매개변수 없고 바디 내용 없는 것. 클래스명(){}
생성자 사용 이유 : 객체를 생성할 때 속성의 초기화를 담당한다.

오버로딩 지원 // 클래스 내에 메소드 이름이 같고 매개변수 타입 또는 개수가 다른 것
this키워드 활용. this.멤버변수 this.메소드 // this생성자
this 생성자의 특징 : 오직 생성자 안에서 오버로딩 생성자를 호출 // 생성자 내에서만 호출이 가능
:생성자의 맨 첫 라인에 위치!!!!
: static 영역에서는 사용이 불가능

<static>
로딩 시점
static은 클래스를 인식하는 순간 만들어
non : new 로 인스턴스 객체 생성 시 로딩

메모리 할당
static 클래스당 1개씩-----> "공유"하는 특징. 여러 객체가 공유한다.
클래스 해석하는 순간 메모리 할당. 클래스 이름으로 접근.
static에 접근시 클래스명.멤버변수명 혹은 클래스명.메소드명으로 접근
생성시점은 static이 더 빠름. static 메소드에서 nonstatic 변수 사용 못하고 this도 의미가 없음.

<상속>
자식클래스는 부모 클래스에 선언된 멤버변수, 메소드들을 사용가능하나,
접근제한자에 따라서 부모클래스에 선언된 것 중 내가 접근하지 못하는것이 있을수도 있다.
super.멤버변수명(parent에 선언 된 member변수와 메소드) 이렇게 접근 가능
this.멤버변수명(child에 선언 된 member 변수와 메소드) 이렇게 접근 가능

extends 키워드 사용
new 하는 순간 부모의 공간과 자식의 공간 같이 생성되기에 부모꺼 사용가능.
내거 this로 사용가능
부모의 것으로 super 사용 super.멤버변수 super.메소드 등..
오버라이딩 가능
오버라이드 : 부모의 메소드 중 메소드의 매개변수 메소드명 리턴타입이 같으면서 내용부를 재정의하는 것
사용이유 : 다형성 때문에!! 무조건 상속관계에서
*오버로딩 : 같은 클래스 내에 메소드명은 같은데 매개변수 타입 갯수 다를때

x instanceof y ==> x가y의 인스턴스 객체니? 상속관계일때 물어볼수있음 true, false 반환

<접근제한자>
public 어디서든 접근 가능
protected 같은패키지(단, 다른패키지의클래스와상속관계가있을경우접근가능)
default 같은 패키지
private 자신의 클래스만
멤버변수는 private 메소드는 public 쓰면 중간 이상은 간다~~

<<private 멤버 어찌 접근할까??>>
내가 가진 멤버변수 보여줄때 get 사용~ getter 메소드
getName
getPrice 이렇게. get 이후의 단어에서 첫알파벳은 대문자

멤버변수의 값 수정시는 주로 set사용~~ setter 메소드
setName
setPrice 이렇게.

<추상클래스>
클래스 내부에 1개 이상의 추상 메소드 가지고 있는 클래스
목적 : 표준. 다형성을 위해. 자식이 같은 이름의 메소드를 갖게하려는 강제성
구현된 메소드와 멤버변수 보유 가능.
인스턴스 생성 놉!!!!! ! new 앙댐 . 자식을 객체로 만들기 - 자식은 부모가 가진 추상 메소드를 반드시 오버라이딩 해야함
abstract 키워드 : 메소드와 클래스명 가능 .. 멤버변수에는 abstract 붙일 수 없다.
추상클래스를 상속받은 자식 클래스 인스턴스 객체를 만들다보니 묵시적 형변환 가능(다형성위해)

<객체의 형변환>
A=B : A와 B는 데이터타입이 다르면서 상속관계가 있어야함.
묵시적 : Super = Sub 인 경우. 부모가 자식의 인스턴스 객체 가지고 있더라도 부모의 멤버변수와 메소드만 접근 가능함.
명시적 : Sub= (Sub)Super 부모의 참조변수를 자식이 가지게함. Sub = Super 묵시적이 선행되어야 가능한것

<인터페이스>
추상과 다르게 추상메소드들의 집합.
인터페이스 : 상수변수와 추상메소드만!! 가질 수 있음
인터페이스의 변수 **public static final**이라는 키워드를 가짐
메소드의 접근제한자는 public이다.
인터페이스 상속받은 자식을 통해 인스턴스 만들기
implements 키워드
자식은 인터페이스의 추상 메소드 오버라이딩 해야함.
객체 형변환으로 다형성 가능

final : 금지가 키워드
변수 : 상수로 사용. 대입 금지
메소드 : 오버라이딩 금지
클래스 : 상속 금지

예외처리
try
catch
finally - 여기까지 직접처리
throws 간접처리
throw : 사용자 정의 예외 발생시킬 때 사용

다형성 : 같은 타입이지만 실행 결과가 다양한 객체를 이용할 수 있는 성질
New : 클래스로부터 객체를 생성시키는 연산자.
New 연산자는 힙 영역에 객체를 생성시킨 후, 객체의 주소를 리턴

Main
public static void main(String[] args){}

<문자열>
'홍'이 포함되어 있는 것 찾기 : str_name.contains('홍');
indexOf에서 -1은 없다는 의미.
IndexOf는 찾는 문자가 처음 나온곳 까지만 찾아봄. Hello면 첫번째 l 만 본다아
Substr(2,7) --> 2부터 6까지
Substr(2) = substring(2, str.length)

<배열>
Int [] arr = new int[3];
- 배열의 인덱스는 0부터 시작
- 배열 크기 : 배열이름.length()
- 마지막 요소 : 배열크기-1

int [] arr = new int[]{10,20,30};
int[] brr={10,20,30} ---> 이 방식은 선언과 동시에 초기화!!
new가 있기에 그 이후 arr = {1,2,3,4} 사용가능!
new가 없기에 그 이후 brr = {1,2,3,4} 못씀

Arrays.toString(배열명)==> "[첫번째요소, 두번째요소, ..., 마지막 요소]"
System.out.println(Arrays.toString(listname));

<List>
순서 있고 중복 허용 (배열과 유사)
장점 : 가변적인 배열 // 배열과 다르게 크기가 유동적으로 변함.

단점 : 원하는 데이터가 뒤쪽에 위치하면 속도의 문제가..

ArrayList!!
ArrayList list = new ArrayList(); - 모든 객체 받을 수 있음
ArrayList<String> list2 = newArrayList<String>(); - String만 받을 수 있음

List.add("봄"); 데이터 입력
List.get(0)=봄 데이터 추출
List.size() 크기 반환
List.remove 데이터 삭제. 1번지꺼가 0번지로온다. 2번지꺼가 1번지로 온다
List.remove("봄")
List.clear() 모든 데이터 삭제
List.contains("봄") 특정 데이터 있는지 체크
List.isempty() 데이터 존재하는지 체크
List.addAll(sub) 기존에 등록된 콜렉션 데이터 추가

<Map>
Key와 Value값 으로 나누어 데이터 관리, 순서 없으며, 키에 대한 중복 미허용!
- V put(K key, V value) : 데이터 입력
- V get(Object key) : 데이터 추출
- V remove(K key) : 입력된 데이터의 크기 반환
- boolean containsKey(Object key) : 특정한 key 포함 여부
- void putAll(Map<K key, V value> m) : 기존 콜렉션 데이터 추가
- Set<Map.Entry<K, V>> entrySet() :
  (key 와 value) 쌍을 표현하는 Map.Entry 집합을 반환

<Set>
순서가 없고, 중복 미허용
- add(E e) : 데이터 입력
- size() : 입력된 데이터의 크기 반환
- remove(Object o) : 특정한 데이터를 삭제
- clear() : 모든 데이터 삭제
- contains(Object o) : 특정 객체가 포함되어 있는지 체크
- isEmpty() : 비어있는지 체크(true, false)
- iterator() : Iterator 인터페이스 객체 반환
- toArray () : Set의 내용을 Object 형의 배열로 반환