15. 07 07

컴파일러와 인터프리터

h/w & s/W(os) = platform "실행환경

실행환경속에서

고급언어(c,java)를 컴퓨터가 읽을 수 있는 이진수로 변환 해주는 거

compiled / OS종속 / 성능 좋음 / 번역(통째로) / 인스톨 방식,게임, 오피스
interpreted / 독립 / 성능 나쁨(상대적) / 통역(한줄,한줄) / 웹

자바

자바
그린프로젝트 oak 동적

고급언어(high-level)
java source code -> .java
compiler javac.exe -> javac ???.java
*(자바)컴파일과정에서 성능을 높이기위해서, 에러, 자주 사용되는 것들을 미리 기계어로

--->>> ???.class (클래스파일 자바 실행파일(바이트코드, 자바 인터프리터가 이해하는 언어)

실행 : java.exe -> java ???(클래스파일명, 확장자를 주지 않는다)

*자바실행과정에 컴파일이 있지만 실행할 때가 인터프리트이기 때문에 인터프리트 방식이다.
*os입장에서는 인터프리트가 실행
*인터프리트가 컴파일된 bytecode를 실행

-jdk 설치
-패스 설정 자바 홈
HelloWorld.java 작성

JDK (java development kits) : jre + 개발도구
JRE (java runtime environment) : 자바실행환경
JVM(java virtual machine) : java interpreter, 메모리나 cpu를 내부적으로 가지고 있음, 통역기(바이트코드->기계어로)
실제 cpu와 메모리를 구현하는 os 같은 역할을 함

- API(abstract(application) programing interface) : 미리작성해서 제공되는 code
  +core API : sun(oracle)에서 제공해주는 API -> jre에 기본적으로 제공됨
  +확장 API : 제 3자가 배포하는 API

-1.2 java2
-1.5 java5

---

15. 07 08(수)

java 특징
- platform 독립적 JVM을 통해 어떤 환경에서도 사용
- 객체지향(object oriented programming)OOP

- 문법구분-> c++ // 개념 smalltalk

- 상대적으로 쉽다.
  + 메모리 관리(cpu, ram)

- 단일 상속 지향
- 다양한 API 지원
- 멀티 쓰레드 지원

객체지향언어

- p/g: 데이터를 관리하는 것, 데이터 값을 생성, 삭제, 수정 하는 동작기능을 수행
- 객체지향 <-> 절차지향(순서)
- 모듈화, 소분류를 나누거나 쪼개서, 같은 주제나 연관된 데이터를 모듈화해서 만들 순차실행
  고치는데 용이, 새로운 것을 추가하기 쉬움, 재사용성, 확장성이 좋음
  /// 만드는 것은 어려움

  절차 : 큰 틀에서 한 번에 만들고 실행
  객체 : 큰 틀을 쪼개서 부분적으로 모듈화해서 만들고 실행, 작은 단위의 프로그램들

- 객체 구성 1) attribute(속성) data
  2) behavior(동작) 제공 기능

class : 객체의 설계로 ->객체가 가지는 속성과 동작으로 설계한 것
관계?

class명/속성/동작

java 규칙

- ; 한실행문 , 구문이 완료
- 공백 : 프로그램 수행과 관련 없음, 컴파일러 실행 시 제거됨
- 공백, 엔터, 탭 코드의 가독성을 높여줌.
- {} 중괄호 : code block, (연관성 있는 동작을 다음 위한 실행 문들을 묶을 때 사용.

- 주석 : code에 대한 설명, 한줄 주석 //~~~~ ->>>
  block 주석/* ~~~~~~ */ (java doc.exe 주석을 html로 볼 수 잇게 만들어줌)
  문서 주석 /** ~~~~ */

15.07.08(목)

변수(variable) 변수의 타입 p20

: 객체를 저장하는 값의 저장소, 메모리 내에 program에 실행 중 사용할 값
(value)를 저장하기 위한 공간

-data저장>> 메모리_ ram / HDD
-저장할 공간을 메모리 내에 생성하고 저장소에 이름을 지정

- DataType 값의 형태(저장할)를 지정 해주어야 한다.

- 변수선언(*제한자는 옵션) : 변수 생성 => [제한자]Datatype 변수명;
- 변수 초기화 : 변수에 값을 최초로 대입 => 변수명 = 값;
- 변수 값 조회 : 변수명 사용 *값이 대입될 때 마다 바뀜
- ,(쉼표) 추가 * int a, b,c ;

식별자(identifier)

- 변수, 메소드(동작), class의 이름
- 규칙
  ① 일반 문자(알파벳, 한글), 숫자, _ , $
  ② 숫자는 첫 글자로 올 수 없음
  ③ 같은 식별자는 사용할 수 없음
  ④ 대소문자 구별을 한다.
  ⑤ 글자 수 제한은 없다 -> 식별자가 의미하는 바를 충분히 설명할 수 있도록 이름을 주는 게 좋다.
  ⑥ 예약어(Keyword)는 식별자로 사용 못함 ex) int 이미 의미가 부여되어 있는 단어들

*카멜(camel)표기법(낙타)
- 기본 소문자로 명명, 단어와 단어가 합쳐질 때 뒤 단어의 첫 글자를 대문자로
- 규칙은 아니지만 프로그래머들 사이에서 관례적으로 사용

*파스칼 표기법- 카멜 표기법 + 첫 글자도 대문자 **클래스제목은 첫 글자를 대문자

UML - 변수 명 : DataType = 값 ex) name : String, age : int =30
클래스다이어그램

Class 선언

[제한자] class 이름(식별자)
: 자바
-> attribute : instance member variable
-> behavior : instance member method

객체 : instance : 프로그램을 사용해서 클래스를 만들어진 객체
object : ?통상적으로 객체지향?

- class 식별자는 첫 글자 대문자(파스칼 표기법)

- 제한자
+[] 옵션
+modifier 클래스의 기능의 제한을 거는.. public
+class, 변수, 메소드, 생성자 선언자 그 역할이나 기능 제한/수식하는 keyword를...

_접근 제한자(access modifier) : 접근 범위 제한/수식
public / protected / private

_사용관련 제한자(usage modifier) : 기능, 역할을 제한/수식
final

+제한자는 여러개 올 수 있음

매서드(method)

- instance(객체)가 제공하는 기능(동작)
- 기능을 처리하기 위한 하나 이상의 명령문들의 묶음
- (다른 메소드의) 호출에 의해서 실행

호출하는 쪽 - Caller 메소드, 매개변수(parameter)/전달인자(argument)
호출당하는 일하는 쪽 - Worker 메소드, return값(value)

Caller 메소드가 호출하고 Worker 메소드가 결과 값을 응답, 서로간의 값을 줄 수도 있고 안줄 수도 있음
피연산, 연산

- class는 public 제한자만 사용가능!!

메서드 구문
- [] : 옵션
- [제한자] returnType(datatype) 식별자(이름) ([변수선언]){
매소드 실행구문(구현)
[return 값] **마지막 구문
}
{} : 구현
**[변수선언] : parameter(매개변수), 0 ~ N개, caller(호출하는 곳) 전달할 값을 저장할
**매개변수(parameter) : 괄호()는 필수, Caller가 호출 시 전달한 값(argument을 받을 변수 선언, 있으면 넣고 안 넣어도 됨
0~N개까지 선언가능
값을 대입(초기화) 할 수 없다. ex) int sum(int i = 20) 의미 없고 사용불가
메서드 내에 리턴 값을 콜러가 받는 것이기 때문에

**returnType : 작업 종료 후 Caller에게 전달 할 값(리턴 값)의 datatype 지정
리턴 값이 없을 경우 void 및 실행구문에 리턴 값 생략 가능
리턴 값은 1개만 가능
** 식별자 관례 : CamelI표기법, 변수 이름 주는 것과 동일, 첫 글자 대 문자임.

식별자 쓰는 방식으로 구분이 가능하다
abc() ()메소드 식별자
abc 변수 식별자
Abc 클래스 식별자

선언 : JVM과 사람에게 설명, 이름만 봐도 알 수 있도록 하는 것이 좋다.

ex)

public int sum(int a, int b)
{
return a + b;
}
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
 	
int i = sum(10,20)
 	
변수 = 메소드이름([전달인자])
	person
	----- 	
	name:string
	age:int
	address:string
	===>>>>
	class Person{
	String name;
	int age;
	String address;
	}

15.07.10

구현

1)속성 : instance 변수
2)동작 : 생성자(constructor), instance 메소드

상수화 된 변수 -> 모두 대문자
final int taxRate ->
TAX_RATE

UML

class person
---
age : int= 2-4
name : string
---
sum(op1:int, op2:int)

class로부터의 객체생성
1) (객체를저장할) 변수 선언, 변수선언과 동일 다만 클래스이름이 데이터타입이됨 - Class이름(datatype) 변수이름
2) 객체 생성 및 대입 - new class이름()

**Class이름(datatype) 변수이름 = new class이름()

객체의 instance 변수/메소드 호출

**변수호출 - 값 변경/ 조회

객체의값을 넣을때

호출하려는 책체.intance 변수 / 메소도
(객체를 부를때 쩜을 찍어서 객체의 값을 입력할 수 있음)

jvm 메인 메소드를 실행
메인 -> 메소드들 실행 했다 메인메소드로 귀환, -> JVM으로 보내고 실행

(2주차)
15.07.13 __ 교재 191p

생성자(Constructor) : 객체의 동작중 하나, 객체 생성 시 한번 호출되는 동작
- 주로 instance 변수 초기화 작업,(attribute 초기화)
- 구문 : [접근 제한자] class 이름 ([매개변수]) **식별자를 클래스이름으로 해야됨

```
{
    생성자 동작 구문
}
```

- 제한자 - > 접근 제한자만 쓸 수 있다.(public, protected, private)
- 리턴type이 없다. (리턴타입이 있으면 메소드)
- 식별자(이름)은 class이름과 동일해야함
- 매개변수(parameter)는 n~ 0개 선언가능 ,로 구분
- 호출 : new Student();

-*기본 생성자(Default Constructor) : class내에 생성자가 없으면 compiler가 만들어 주는 생성자
※생성자가 하나 이상 있으면 default 생성자는 안 만들어진다.

jvm ram 저장
heap(객체) / 실행 stack(지역 변수) / 메서드영역

heap : 객체가 저장되는 메모리 영역
객체 생성 시 instance 변수 초기화 3단계
o) 변수를 저장하기 위한 변수 공간이 heap 메모리에 생성
1) default (묵시적)(기본, 디폴트 초기화)초기화 : 변수 type을 default값을 대입
**type의 기본 값 숫자 - 0, Strong - null
2) 명시적 초기화 : instance 선언부에 값 대입을 했으면 그 값을 넣는다.
3) 생성자 수행에 의한 초기화 -> 생성자 실행

**생성자에서 넣은 값이 최종적으로 나옴.

overloding 생성자/ 메소드, 동작의 대한

오버로딩

생성자 오버로딩
- 한 클래스 안에 매개변수가 다르다면 여러 개의 생성자를 만들 수 있음
- 매개변수의 선언부의 타입과 개수가 다를 때 생성자를 여러 개 만들 수 있다.
  매개변수의 수가 같고 변수가 달라도 타0입이 같으면 에러 발생

메소드 오보로딩()

메소드의 매개변수(type이나 개수) 가 다르면 같은 이름을 사용할 수 있다.
why. 하는 일은 같은데 매개변수가 다른 경우 overloading을 이용해 메소드 구현
ex)println 12개 메서드를 여러 타입으로 입력해도 1개로 사용할 수 있음, 12개를 1개처럼 쓸 수 있는 것이 오버로딩의 장점

String타입과 다른 모든 타입을 합치면 String타입으로 합쳐진다.
- 문자열(String) + String => 두 문자열을 붙인다.
- 다른 타입 + STring => 다른 타입을 String으로 변환

java 변수의 종류
filed - instace 변수 : 객체의 data를 저장할 변수, 선언위치(class block 내에 선언), 호출 방법(객체. 변수명), 디폴트 초기화 : ㅇ, 저장메모리 위치 Heap

- static 변수 :

- 지역(local) 변수 : 메소드가 일하는 도중 사용할 변수, 메소드 BLACK 내에 선언, 매개변수도 지역변수, 매개변수는 메서드 호출하는 곳에서 값 대응

  디폴트 초기화가 없다, 저장메모리 위치 CALL EXECATION STACk(실행중인 메소드의 지역변수들이 저장되는 영역
  기본적으로는 값을 외부에서 쓸 수 없으나 예외적으로 매개변수가 변수를 받는 것은 됨

**class block안이냐 method안이냐로 지역변수인지, 인스턴스변수인지를 구분할 수 있음.
**로컬변수는 디폴트초기화가 안되기 때문에 변수에 값을 바로 지정하는 것이 좋음, 즉 선언초기화를 초기에 하는 것이 효과적임
**변수는 선언된 {}블록안에서 사용될 수 있는 것을 기본으로 한다.

execation stack 실행 frist in last out

- 지역벽수와 instance 저장 영역이 달라서 같은 식별자를 써도 된다.

- 로컬 변수는 선언한 구역이 다르면 같은 식별자명을 사용해도 무방하다 한 구역(메서드)내에 동일한 식별자는 사용불가하다.

this. instance 변수를 가리키는 메소드안에 로컬변수와 구별을 주기위에서, 해당 객체의 변수로

(2주차)
15.07.14

메소드와 생성자의 차이 : 생성자는 객체가 만들어지는 시점에서만 딱 한번 이루어져야 하는 동작, 객체가 만들어지고 필요한 자원들을
객체가 가져야 되는 instance변수를 초기화를 주로한다.

default 생성자 : 생성자가 하나도 없을 때(실제 코드 상 생성자가 없을 때) 컴파일러가 자동으로 기본 생성자 넣어서 처리해줌,
객체 생성만할 수 있게 해주기 때문에 하는 일은 실상 없음
public class(명){}

생성자 구현

```
class A{
int var = 20; (명시적 초기화)

A(int v){
var= v;
}
}
```

heap 메모리내 객체가 영역을 만들고

주소 100
값 0(묵시적, 디폴트)->20(명시적)->50(생성자 값)
객체 A(int v)

**객체를 생성 했을때 실제 변수에는 들어가는 값은 메모리의 주소(위치값)으로 들어감?

```
s = new students();  <- Heap 한 영역의 주소
```

int v
0(묵시적, 디폴트)->20(명시적)->50(생성자 값)

로컬변수와 달리instance 변수는 묵시적 초기화 이루어져서 처음에 값을 안 넣어도 에러가 일어나지 않음

this.구문

- 메소드/ 생성자 block내에서만 사용가능한 구문
- 메소드/생성자 의 객체가 가리키는 local변수.
- instace변수와 local변수의 이름이 같을 때 구분을 위해 사용
- 구분이 필요 없을때 생략가능

- this. 이 객체 _를 호출(요게 더 많이 쓰임)
  +this. instace member varialbe/method

  String name;

  Student(String name //String n){
  name = name //name = n
  }

  +인스턴스변수와 로컬변수의 식별자가 다를 때는 컴파일러가 알아서 name을 코드 중 찾아서 name을 인스턴스함수로 인식해주지만
  똑같을 때 this를 안 넣으면 로컬 name값에 로컬 name값을 그대로 넣는 것으로 인식하기 때문에this 사용

- 메소드 안에 메소드 정의가 안됨.
  public void Sleep(){

  eat();
  this();

  public void Eat(){}
  }
  public void Eat(){}

this(), 이 객체 CLASS의 생성자를 호출₩

- this([값...])
- 수행중인 생성자가 다른(overloading된) 생성자 호출 시 사용
- 생성자 block에만 첫 명령어(실행문)으로만 올 수 있다.
- 새로운 생성자를 생성하는 것이 아닌 객체안에 다른 생성자를 호출

super

- super(), super.

선언 호출 사용 접근

접근 제한자

: 객체 지향이 되면서 각 클래스 간 접근의 대한 범위를 지정할 수 잇는 환경이 마련되어, 호출(접근)의 접위를 지정하는 제한자이다.

접근범위:

넓다 1) public : 접근의 제한이 없다.

↑ 2) protected

↓ 3) package friendly : 같은 페키지 안에서 접근 가능함, 제한자를 안붙이면 자동으로 적용, 디폴트 접근제한자 값

좁다 4) private : 같은 Class 내의 멤버들만 접근 가능, 데이터를 보호하기 위해서,

class : public, X

instance var

local var : 애초의 메소드 내에서만 접근하기 때문에 접근제 한자가 의미가 없다. 그래서 X

생성자

메소드

정보은닉(information hiding)

- instance변수는 private로 선언하는 것이 관례임, instance변수를 변경, 조회 하는 메서드를 만듦
- 객체의 멤버(instance member 변수 / 메소드) 중 다른 객체서 접근 해서는 안되거나 접근할 필요 없는 것은 private 을 붙인다.
- instance변수를 priavte으로 해서 접근을 막고 변수에 값을 변경(setter) / 조회(getter) 하는 public 메소드를 제공

=>직접접근을 막고 =>메소드를 통해 접근해라.

- 통상적으로 getter메소드는 제공되는 경우가 많음.
- 관례적으로 get은 조회를 set은 변경하는 것을 나타낸다
- 메소드 동작으로 할 수 있어서 변경하는 데이터에 대해 조건을 주어서 검증할수 있다.

```
public class Student{
private String name;

public void setName(String name){ // 변경 메소드, 네임 변수이름에 set이 규칙화된 관례적으로 사용
this.name = name
}

public String getName(String name0{ //조회 메소드, 관례적으로 get을 붙임
return this.name;
}
```

0715₩Data.java

(2주차)15.07.15

캡슐화

객체가 캡슐?, 접
encoapsulaion 객체를 만드는 것 속성과 동작이 유사한 객체를 묶은 것.

package

- = directory = folder 폴더같은
- class파일들을 묶어주는 directory
- 비슷한 목적이거나 의므 가지는 class를 묶는다.
- package 선언: class파일이 포함된 package를 소스코드에서 알려 줘야 한다.-> class파일은 선언된 디렉토리에 저장되야 한다.
  소스코드의 첫 라인에 해야 한다, 소스 코드당 한번만 가능.

구문 : package root경로)[하위경로, 하귀경로..]0;
규칙 - 시벽자 규칙, 관례 - 소문자사용
domain주소를 거꾸로 사용, kosta.or.kr --> kr.or.kosta = kr₩or₩kosta₩
** . 경로를 나타냄

kosta.or.kosta ko = new kr.or.kosta();
**Class 풀네임: package명, class이름

import

: 다른 package의 class를 사용할 때 그 class의 package를 미리 선언하는 것
->사용 시, package명 없이 class이름만 가지고 사용가능하게 해준다.
-> import 없이 사용가능한 class들의 package
java.lang (core api)

- 구문 : import package명.class
  import package명.*;

-(경로가 아니라 클래스를 import한다)

->import 구문은 class 선언보다 먼저 나온다.

**디렉토리(package_는 다르지만 클래스의 명이 같은 2개를 임포트는 안 된다, 하나만 사용가능
java.ant.list
java.q.list (X) -> 하나만인식, 즉 하나는 임포트로 쓰고 하나는 전체를 다써서 list. / javaq.list

java.lang은 core api로 import안해도 사용가능 하다 System.out.println

source code 작성 순서 및 규칙

1) 하나의 소스파일에 여러개 class 작성가능
   ->단 public class는 하나만 간으
   ->public class의 이름이 파일명 되야 한다.

2) package선언 - 파일당 1번 선언
   import 구문w
   class 선언 및 구현

**class파일은 class당 하나씩 생성

class path

: class가 있는 경로 package directory class파일'

-> jdk에게 compiler 또는 실행시 classpath를 알려줘야 한다.

1) compile 또는 실행시 option으로 저장
   javac -class path 경로 ???.java
   java -cp 경로 ???.java
2) o/s환경변수로 지정
   변수명 : Claspath 변수값 class가 있는 경로

3) JDK설치경로 ₩jre₩lab₩ext -) jar로 압축해서 저장(디렉토리 체 압축)
   확장apI directory

.jar
.war
.ear

javac -d 경로 source.java
경로에 소스 파일을 이동시키고(패키지가 선언된 경우 패키지 디렉토리도 만든다.)

압축 할 때 : jar cvf 압축파일명 압축대상
압축 풀 때 : jar xvf 압축파일명

실행시 옵션으로
C:₩Program Files₩Java₩jdk1.8.0_45₩jre₩lib₩rt

jar파일을 받았을 때 C:₩Program Files₩Java₩jdk1.8.0_45₩jre₩lib₩ext 에다가 jar파일을 넣으면 클래스가 잡힘

변수 타입

- primitive datatype(기본형)

1) 정수형(Integral) : -byte : 1byte(8bit), 0~255, 첫번째 비트를 음수인지양수인지를 구분하는 부호 비트로
사용_0은 양수, 1은 음수, 나머지 7비트로 숫자 표현
-short : 2byte(16bit),
-int : 4byte(32bit),
-long : 8byte(64bit), 값L, 값l

**-2의 n-1제곱 ~ 2의 n-1제곱 -1
8진수 : 0값
16진수 : X0값

용량가지고 별로 고민 안해도 된다 예전에는 하드웨어가 상대적으로 비싸서 중요 했지만 지금은 메모리가 저렴해져서
용량을 가지고 타입을 고르는 큰 이슈는 아니다. 통상적으로 int를 주고 대략 +-26억이 넘어가는 변수라면 long으로
int가 정수의 기본type 대부분 32bit단위로 데이터를 처리하기 때문에
byte, short는 기본도 안 되는 놈(?)

long l = 365* 365* ..... //임시 변수가 int로 형성되는데 어느 순간 26수치 이상을 넘어가면 계산이 깨져서 올바른 값이 나오지 못한다.
long l = 365L* 365..... //숫자에 L을 붙여서 임시 변수도 L로

2) 실수(Floating point) - float : 4byte , 소숫점 이하 7자리 까지 보장
값 F 값f
- double : 8byte, 소숫점 이하 15자리까지, 실수형의 기형
l 값D 값d 값e 값2
값E3 = 값*10(3승)
**계산 시 부적한 값이 나올 수 있어 부정확한 값이 노올 수 있어서 계산X
**소수부의 정밀도

3) 문자(Character) 한 글자 char
- 유니코드(unicode) 한 글자
- 1글자 2byte / 값은 ' ' 로 감싼다.
-stirng

- escape 문자 : 역 슬러쉬는 역 슬러쉬가아니라 escape 문자로 본다

₩t : tab
₩n : enter
/₩₩ : ₩
₩' : '
₩" :
₩u : unicode를 직접 입력 16bit 16진수 ex)₩u325a

**경로의 경우 ₩₩를 넣어주어야 인식된다.

4) 논리형(Logical) boolean : 참(tue) / 거짓(java는 이외 값은 가질 수 없다.)
- boolean type instace 변수의 getter메소드는 is변수명();
- ex) isMail 메일을 받을거냐

refoence datatype

String
String은 클래스 자주 사용되기 때문에 코어api에 java.lang //임포트 없잉 사용
0글자 이상의 문자열, " " 로 표현
String 예외 클래스, ___ String 객체임,
덧셈연산이 가능(String 타입끼리 다른 타입과도 가능 => String 값으로 변경)
new 없이도 객체 생성가능 한 몇 안되는 클래스 하나

(2주차)15.07.16

속성 위주의 Class
-Value Object -ov
-Data Transfer Objecet - DTO
-Domain Object - DO

형변환 (Type casing) : 연산 시 피연사자의 type이 다른 경우 type을 맞추는 것
- 기본 작은type => 큰type : up casting(자동)
- 큰 type => 작은 type : down castiong -->형변환 연산자를 이용해서 명시적으로 해야 한다.
대입 연산할 때
=> int i = 10ㅣ; 요거는 작은 int값은 이미 선언된 것이라서 변환이 안되서 10L을 변환해야되는 상황이다.
long i = 10L;이 코드를 변환 하는게 좋지만 형 변환을 해야 되는 상황이라면 형변환 연산자를 이용한다..
- (변환할type)값; --> (int)10L
- 형변환연 산자 : primitive type에서는 다운 케스팅을 하지 않는 것이 좋다, reference type에서는 자주 사용 되서 이 개념은 꼭 알고 있어야함

- 형변환은 피연산자간 type의 상하관계가 구분하고 작은 쪽이나 큰쪽 은 변경한다.
- 논리형 f와 t만 사용 되서 변환이 안 된다.

- primitive와 reference type은 변환이 안 된다 (예외 primitive -> String, +연산 시 자동으로 일어나는 것)

- primitive 순서 : byte < short < int < long < float < double
char
(**boolean은 변환 안됨)

<----- down casting -----> (up casting)

비교 연산자

- 연산결과가 boolean
- 기준 : 왼쪽 피연산자

== : 같다

!- : 다르다

>

<

>=

<=

맞으면 true 틀리면 false

논리연산자 : 피연산자가 boolean -> 결과도 boolean

- and : &, && ┐
- or : |, || ├ 이항연산자 , and or는 두개짜리 씀
- xor : ^ ┘
- not : ! (단항 연산자)

1) and : true & true => true
   나머지 => flase

2) OR : false | false => false
   나머지 => true

3) XOR : false ^ true ┐
   true ^ false ┘ =>true
   t ^ t ┐
   f ^ f ┘ => false

4) NOT : ! 반대로 바꾼다
   !true => false
   !false => true

&는 앞에서 결론이 나도 뒤에 피연사자를 보지만
&& 앞에서 결론이 나면 피연사자를 안봄

|

||

앞에 값만으로 결과가 나올 경우 바로 도출 안 나올 경우만 뒤에도 봄
xor는 앞뒤 다 봐야 되기 때문에 쇼?가 없다.

상항연산자(조건연산자)

- 조건 ?값①:값②
- ture/false

a<10 ?T:F

if(조건){
값1
else
값2
}

대입연산자

- 변수 = 값, 상수, 연산식, 변수, 메소드 호출(리턴값 대입)
  <----------
- 변수 연산자 = 값

int i = 10;
i += 5 i= i+5
i *= 5 i= i*5

(2주차)
15.07.17

제어문 **모든언어의 공통적 사항(분기문과 반복문)

- 분기문(조건문) -if + else - else if, switch +case + default
- 반복문 for문, while문, do while문
  (유한 무한 0~N번) (1~N번)

- data type = boolean
- 비교/논리 연산자를 많이 사용

IF문

- :

```
if(조건){
        실행구문
}
```

**조건은 boolean-> 값, 연산자(결과가 boolean), 메소드(리턴타입이 boolean)
**조건이 true이면 실행구문({}) 을 실행 조건이 false이면 실행X

IF ELSE문

```
if(조건){
실행구문A
}else{
실행구문B
}
```

조건 : true이면 A를 실행하고 조건 faluse이면 B를 실행

IF ELSE IF문

```
if(조건A){

실행구문A

}else if(조건B){
실행구문B

}else if(조건C){
실행구문C

}.....

else{
실행구문D
}
```

else값은 맨 뒤에 트루 값이 이후 뒤는 읽지 않음

// 문자열은 ==으로 동등비교하지 않는다.
String A.equals(StringB) =>A문자열과 B문자열이 같은 값인지 맞으면 true ㅇ틀리면 false flxjs

Switch문 case : 동등비교조건

Switch(변수){
case 값1:
실행구문들!1
case 값2:
실행구문들!2
.....
default:
실행구문들!3
}

**변수 : 비교할 값을 가진 변수, datatype int만 가능(즉. char, short, byte는 int보다 작아서 형변환이 자동을 가능해서 사용가능)
**1.7번전이 나오면서 String 지원하기 시작했지만 아직 이 버전 이상을 쓰는 현장은 그리 많지 않다고 함
**default

swithc(x){
case 1:실행구문들1

case 2:실행구문들2

default 3:실행구문들3
}

x가 1이면 구문1 실행
2면 구문2실행
다 아니면 3실행

첫 번째 case의 충족해도 그다음 나머지 case들의 값은 비교하지 않고 구문도 그냥 다 실행한다.
switch 한 블럭으로 함
케이스마다 break문을 걸면 if문과 동일한 식으로 값을 만족하만 해당 실행구문 실행하고 빠져나오게 해줄 수 있다.

break -> block을 빠져나오라는 ... switch, 반복문

default 문법상 위치는 상관없으나 의미상 다른 case이 맞지 않았을 때 실행되기 때문에 논리 구조상

마지막에 두는 것이 바람직하다.

하나의 변수로 등등비교, 한조건에서 ||연산시
if(x=5 || x=6 || x=7) 이럴때
요렇게 표현이 가능
switch(x){
case5:case6:case7:

(3주차)15.07.20

math.random()

0 ~n : (int)Math.random() *(n+1)
Math.random() : double - o.o<= 임의 숫자 <1
실행시 마다 다른 값이 리턴

동작위주의 클래스 : 카드
속성위주의 클래스 : 학생

api₩index.html 에서 API를
cmd
doc

반복문 p93
- 특정조건이 true인 동안 특정 구문 반복실행
- 동일한 코드 반복
- 동일한 업무처리_값이 규칙적으로 바뀌는 코드의 반복

- **for** : 반복의 횟수를 알고 잇을때 유한 반복
구문 : **for([초기화];[조건식];[증감식]){**
반복 구문 }
**초기식
초기식에서 사용할 변수 선언 및 초기화
같은 TYPE인 경우 여러개 변수 선언가능, (쉼표),구분
변수는 for문 내에서만 사용가/능
**조건식
boolean 값 만 가능, 값 한개만 간으
-> 이 조선식이 true 인 동안 만 반복 조건식은 (쉼표)가들어가면 안됨
**증감식 : 조건식에서 사용한 변수의 값을 증감시킬때(변화)
여러개 올 수 있다. (쉼표),구분
** 셋다 생략 가능(생략하면 true)

①초기식②조건식-->true ④반복구문 ③증감식
②조건식-->true ④반복구문 ③증감식
②조건식-->true ④반복구문 ③증감식
.....
②조건식-->false, for문 종료
-for문 밖에서 선언한 변수도 사용할 수 있다.

- **while** : 무한반복
구문 : **[초기식]**
while(조건식){
실행구문
[증감식]
}

**조건식 : boolean 값만 가능, 생략 불가능
** 초기식과 증감식을 block 안에 알아서 넣어야 한다.

- **do while** : 무조건한번 이상 1~n번 반복

- 구문 : **[초기식]**
do{

반복 할 실행구문
{while(조건)
**그 실행구문을 먼저 실행하고 while문으로 가서 true, false 판단 어

조건문에 블럭 없이 ;로 바로 조건문을 닫을 수도 있다. 하나의 실행 문만 있을 때만 가능하다
while(조건){} = while(조건);

중복반복문 같은 블록 안에 변수는 중복 선언 불가함

1) break -반복문,switch문 / block를 빠져나가는.. / if문에서 주로 사용
2) continue -반복문에서만 쓰임 / 조건을 다시 비교하라는.. / 조건을 다시 비교하라고 올려 보냄.
사실상 while 맨 끝에 생략되어있는 것, while문의 경우 증감식을 한 다음에 컨티뉴를 실행해야한다.
end 블록을 만났을 때와 같은 일 처리

(3주차)15.07.21

배열

- 배열(array) : **같은 TYPE**의 VALUE(값)을 모아서 관리하는 **객체,** 같은 의미의 Value들을 보아서 하나의 변수로 관리하기 위해 사용
- 구문 : 모을 dataype[] 변수 명'

- 배열객체생성 및 변수에 대입 : 변수명 = new DataType[length]
  + length : 이 배열의 크기, 배열에 저장 할 수 있는 갑의 최대 개수

- 배열 초기화 및 사용(값 대입,)
  + 대입 : 변수[index] = 값
  + 조회 : 변수[index]

- 인덱스 : 배열의 값을 저장하기 위한 공간의 **식별 값**, **0부터(자바에서)** 시작하는 번호

type 기본 값 숫자 : 0, 0.0 / char : /u0000(공백) / boolean - false / 나머지 - null

배열.length -> 배열

배열 초기화 3가지

1) int [] a = {10,20,30,40,50}; //반드시 선언을 하면서 넣어야 한다.
2) int [] a;
   a = new int {10,20,30,40,50};
3) int I = new int[]{10,20,30,40,50};

Product[] pl = new Product[3]; : product 배열은 3개의 배열 값을 갖는다.

(3주차)

15.07.22

향상된 for문(jdk 1.5 추가)

->배열, 컬랙션내의 모든 값들을 조회 시 사용

0~n의 익덱스를 모두 출력한다, 값을 변경할때는 사용X

구문 : for(변수**선언** : **배열, 컬렉션 객체**);

{ **선언된 변수를 이용해 조회한 값 사용**

}

ex)

int	product

int [] ages = {1,2,3,4,5}
for(int i = o ; i<ages.length ; i++){
int age = ages[i];
s.o.p(age);
 	
▼▼▼▼▼▼▼▼▼▼▼▼▼▼▼▼▼▼▼▼▼▼
 	
for(int age:ages){
s.o.p(age)
}
	for(int i = 0 ; i < list.length ; i++){
	Product a = list[i];
	// Product a 배열 아닌 하나의 값을 새로 생성해서 받아온 Product list[] 배열 값 중의 하나를 초기화
	if((a.getName()).equals(pname)){
	// if((list[i].getName()).equals(pname)){b
	return a;
	}
	}
	
	▼▼▼▼▼▼▼▼▼▼▼▼▼▼▼▼▼▼▼▼▼▼
	for(Product i : list){
	if((i.getName()).equals(pname)){
	return i;
	}
	}

**제약 조건 jdk 1.5버전 이상에서 사용 조회시 사용이 용이

for문안에 변수는 밖에서 선언한 변수도 사용가능하지만 반복해서 돌릴 변수는 조회 값

for()안에서 변수와 타입을 선언해주어야 한다.

가변인수

매개변수 : datatype... 변수명 -> 가변인수(VarAgrs)
-메소드 구현부에서는 배열로 처리
-호출 시에는 원하는 개수의 값을 전달하면 된다.(0~n개)
- 가변인수는 항상 마지막 매개변수로 선언해야한다. (가변인수 다음에 다른 변수를 선언하면 암됨)
- 여러 개 선언 불가
- jdk. 1.5이상부터 사용 가능

배열 값 정의
int [] i : int 배열
i[n] : int 값

Stirng [] i : String 배열
i[n] : Stirng 값

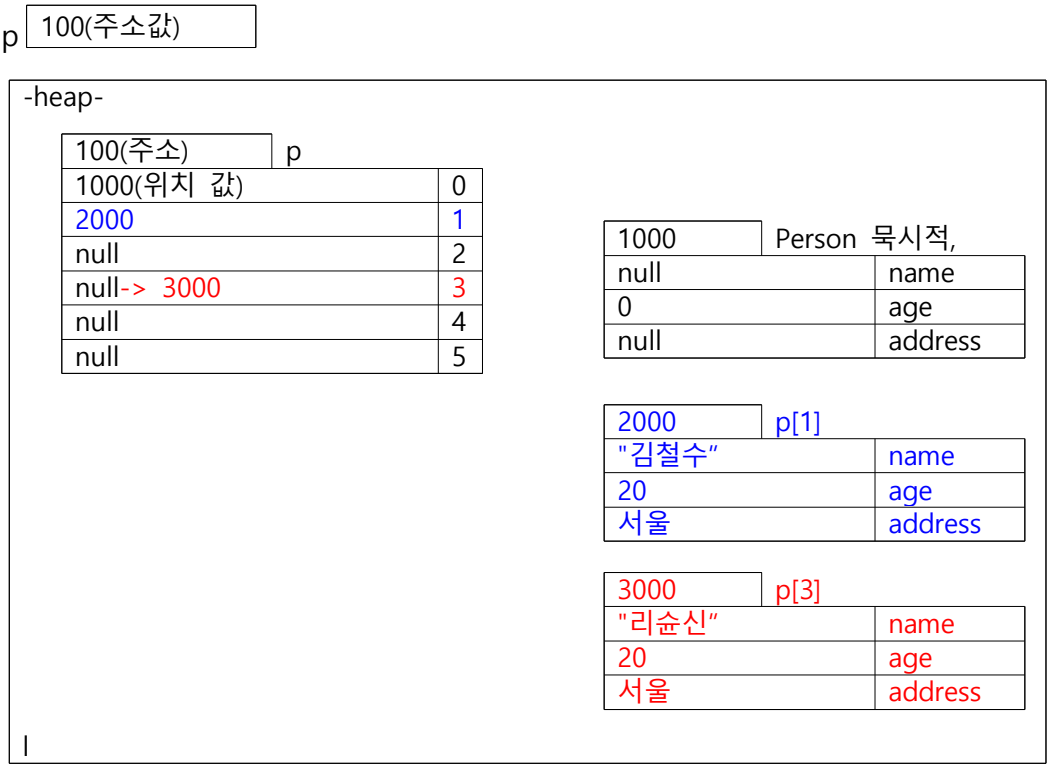
public static void main(String[] args)
jvm이 호출
메인 메서드를 다른 메서드가 부르면 계속 반복될 수밖에 없다.
커멘드 라인 아규먼트
java class명 [arg값 O O]
java hw A^B^C //^ 공백
-사용자로부터 프로그램 실행 할 때 입력 받는 값

배열

person
name : String age : int address : String

1) 변수선언
Person[] p;
2) 배열객체 생성
p = new Person[5];
**p : Person배열 타입
**p[n] : person 타입
3) 배열 초기화
p[n] = new person();
p[0] = new person();
p[1] = new person("리슌신", 20, "서울")
p[1].name = "김영수";

p[0] = new Person();



p[1].

p[1].name = "김철수";

System.out.println(p[1].name); => 출력 "김철수"

p[3] = new person("리슌신", 20, "서울");

(3주차)15.07.23

4차원 배열

- 배열의 배열
- 변수 : 하나의 값
- 배열 : **같은 의미**를 가지는 값들을 모을때 사용'
- 다차원 : **의미**가 여러 차수로 구분되어질 째.... 대분류, 중분류, 소분류
  분류하는 차수에 따라서 배열을 만듦
- 대개 2차원 내에서 해결한다.
- 1)선언 int []{} age;
  int [] age[];
  int age[][];
- 2) 배열객체 생성
  **2차수배열의 크기들이 동일한 경우

age = new int[3][5];

1차수 length 2차수 lenth

**2차수 배열의 크기가 다른 경우(1차수 크기는 무조건 정해 놓고)

age = new int[3][]; // 먼저 1차수 배열을 생성하고

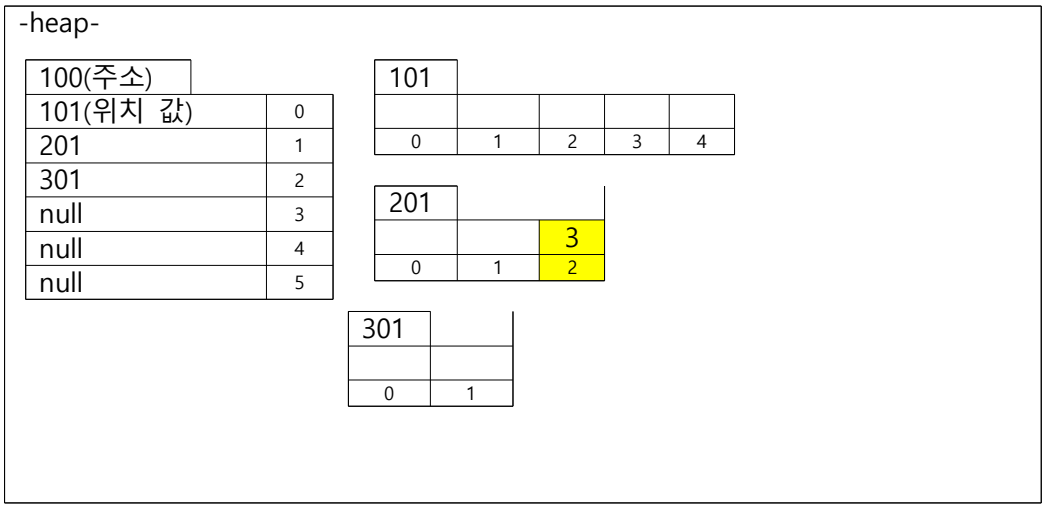
age[0] =new int[5]

age[1] =new int[3]

age[2] =new int[2] // **2차수 배열은 각각 생성해서 인덱스 별 배열생성

호출 ; age[0][2] : 1차수index, 2차수index

ex) A학원에 다니는 2015홍길동



age -> int[][], int 2차배열 값

age[n] -> int[], int 배열값

age[][]-> int 값

int age[1][2] = 3;

int I[][] = {{1, 2, 3}, {3, 4}, {34, 1, 45}} --> 1차수배열, 2차수배열

배열 변수 선언 시

int [][] a / int [][] / in a [][] 3가지 모두 같은 의미로 들어간다

int a[] / int []a 도 같은 의미

- 배열의 length는 불변

- 값을 바꿀 순있지만 length는 늘릴수 없기 때문에, 값을 늘리기 위해서는 복사해서 새로 만든 arraycopy

System.arraycopy(원본배열i, 시작 index, target 배열j, 시작 index,개수)I

원본 배열은 타겟배열의 수보다 같거나 작아야 한다.

eclipse
intellij기반 / Netbeans / 무료툴

Integrated
Development
Enviornmnet

plugin 오픈

다양한 환경 사용가능
project단위로 저장 및 보관
perspective 특정 작업을 하기 위한 뷰들

(3주차)
15.07.24

의존관계 상속관계

데이터 위주의 메소드
일 위주의 메소드
Dependency(의존관계) : 객체와 객체간의 관계
- 사용하는 관계(use a relationship) : 메소드내에서 사용
- 가지고 있는 관계(has a relationship) : 의존하는 객체를 instance 변수로 가지고 있는 관계
whole(의존하는 객체) - part(의존당하는 객체)
└ ┌Aggregation 관계 : whole-part객체의 life cycle이 다른 관계
->whole 객체가 생성될 때 part의 객체가 반드시 instance변수에 대입될 필요가 없다.(부억-토스트기)
└Composition 관계 : whole-part객체의 life cycle 같은 관계
-> whole 객체가 생성될 때 part의 객체가 instance변수에 반드시 대입되 있어야 하는 관계(차-엔진, 부엌-싱크대)
** lifecycle :

ex) 부엌 | 싱크대, 식탁, 가스렌지, 냉장고, 토스트기
(whole) (part)
- 부엌이 생기고 나서 나중에 싱크대가 셋팅 되면 Aggregation .. 싱크대가 null값이 었다가 나중에 메소드를 통해 가져올때
- 부엌과 싱크대가 같이 객체 생겼다가 같이 살아지는 것 Composition
** composition이 필요한 이유는 부엌에서 싱크대가 하는 역할이 대부분이어서 싱크대 없이 부엌이 실행되지 못하는 경우가 있기에

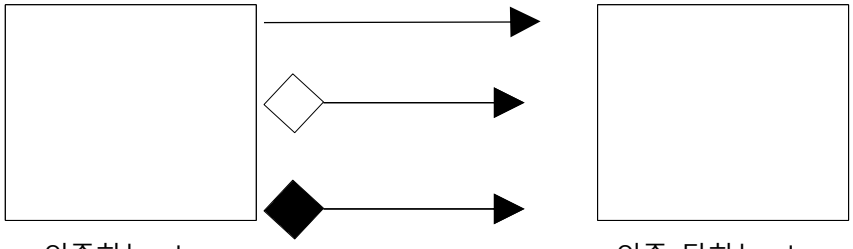
ex) 차 ; 엔진, 핸들 네비게이션, 방향제

-엔진과 핸들은 차를 움직이는 주요요소로 엔진고 핸들이 없으면 차는 실행이 안 된다.->life cycle이 같다

-네비게이션이나 방향제는 없어도 차가 실행이 안되는 건 아니다 -> life cycle이 다르다.

.uml

- 의존 ◇->어그리게이션관계 / ◆->컴포지션관계



-고객관리 프로그램

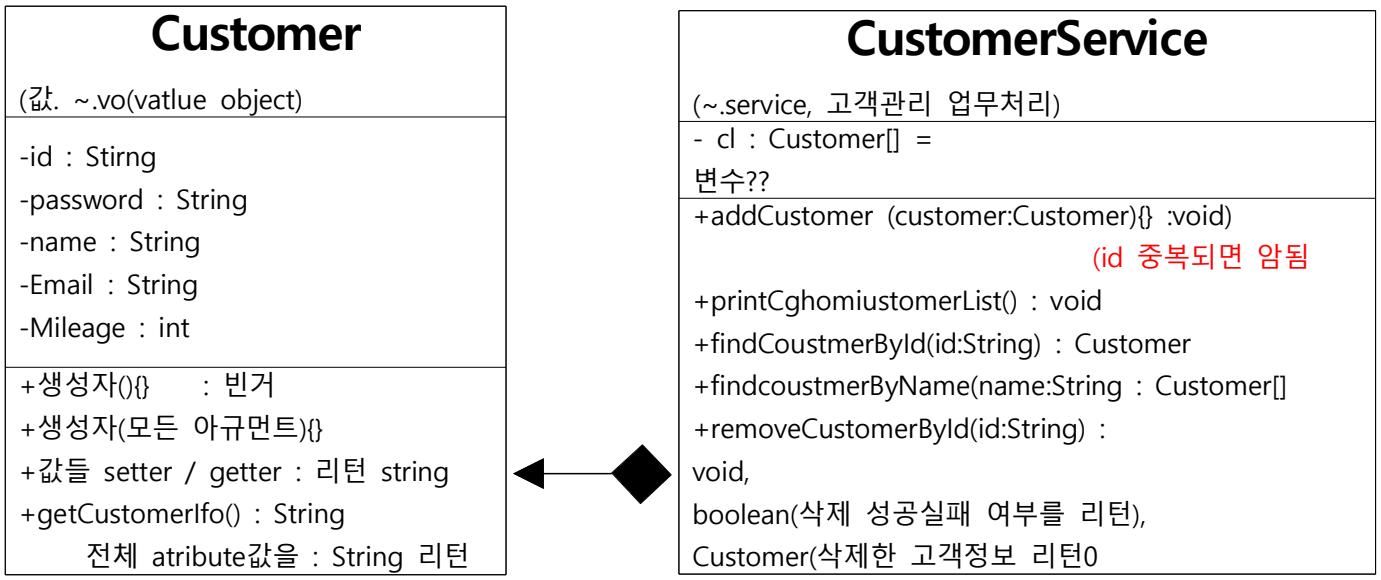
고객을 관리하는 프로그램을 만든다.

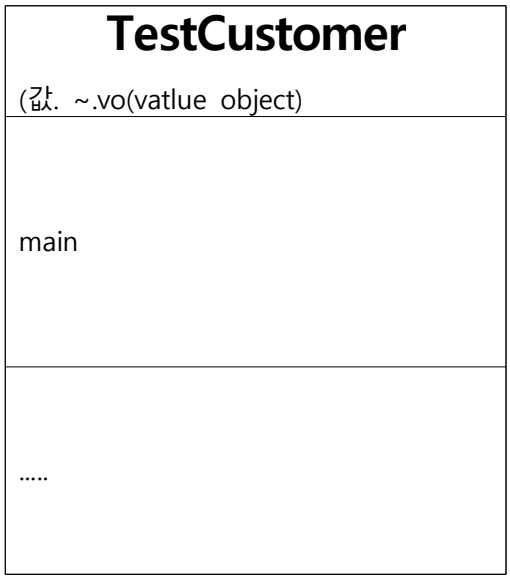
고객은 id, password,이름, email, 마일리지를 가진다. - 고객 값

고객관리 클래스는 다음과 같은 기능을 제공한다. - 고객 관련 업무처리

1. 고객을 최대 100명까지 저장할 수 있도록 한다.
2. 고객관리 클래스는 한명의 고객을 저장한다.
   - 저장 시 중복된 ID의 고객은 저장할 수 없다.
3. 저장된 모든 고객의 정보(id ~ 마일리지)를 출력한다.
4. 저장된 고객들 중에서 ID로 고객을 찾아 알려준다.
5. 저장된 고객들 중에서 이름으로 고객을 찾아 알려준다.
6. 저장된 고객들 중 ID로 찾아 삭제할 수 있다.

UML - private + public





-heap-

100(주소)	list : Customer 값을 갖는 배열
id, password, name, email, mileage	0
a, 123,홍길동,n@n.co.kr, 767777	1
null	2
null	3
null	4
null	5
...	....
null	99

101				
0	1	2	3	4

201		
		3
0	1	2

301	
0	1

(4주차)

15.07.27

oop

-캡슐화

-상속 : 코드의 재상용적인 측면

-다형성 " datatype

동물 클래스

개 클래스

동물
나이, 색, 크기 성별, 무게, 종,
먹는다, 싼다, 잔다

추상화(abstraction), 공통적인 것

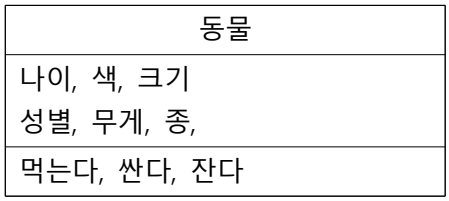
↗ **상속** ↖ **(물려 받는다)**

dog
짖는다, 문다, 달린다

cat
할퀸다, 털갈이

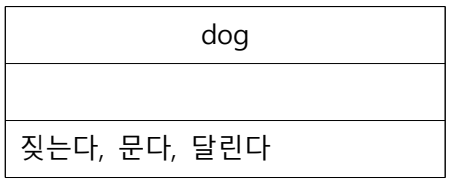
- 여러 클래스들이 가지는 공통점을 하나의 클래스로 정의
- 다중 상속 : 여러 군데서 상속을 받으면...(java에서는 interface를 통해서 지원한다)

- 단일 상속 : 한 군데서(java는 단일 상속 기준으로)
**언어마다 상속의 기준은 다르지만 보통 다중 상속을 공유



물려주는 class - parent class(부모) / super class(상위)

↑



상속하는 class - child class(자식 / sub class(하위)

**구문 : public class Animal{구현}{
public class Dog **extends Animal**{~~
**java 상속구문 -> 자식(sub) class 선언에 표시
[제한자] class 식별자 extends SuperClass이름{구현}

다중상속은 상속구조가 복잡해서 단일 상속에 비해 상대적으로 같은 메소드등이 엉킬 수 가 있다.
자바는 다양성보다 안전성을 택해서 단일 상속을 주로 한다. 다중상속은 인터페이스를 활용해서 한다.

[extends SuperClass명]
->생략시
-> extends java.lang.object
가 Class선언문에 추가됨,lang
[extedns object]는

(4주차)15.07.28(화)

상속 - 다형성 - 캡슐화

상속

1) 코드 재사용 기존클래스<-새로운 클래스

dog, lion, cat ---> animal class (공통점을 묶어서 **추상화**해서 animal class(공통적 속성/동작)

A is a B / a: sub, b: super / a:dog b:animal

2) 타입의 통일 **중요!!**

개나 사자나 고양이나 동물이다. 즉 개나 사자나 같은 동물 타입으로 객체를 생성 가능하다.

Animal d = new dog();

Animal c = new cat();

객체를 생성 하면 상속받는 상위 클래스들도 공간도 같이 생성된다.

- 단 접근 제한자가 맞아야 상속클래스의 변수와 메소드도 사용할 수 있음

this : 현재 일하는 동작의 객체

- this(값) : 객체내에 생성자
- this. : 객체내에 변수나 메서드

super

super : this의 부모객체, 상위객체

- super(값) : 상위 객체의 생성자

Super([값....])

--> 생성자에서 부모객체의 생성자를 호출

--> [값] : 부모생성자에 전달할 인자(argument 값)

--> 생성자 구현부의 첫 구문으로 와야한다.(this()도 동일)

***[]는 옵션임

** 생성자 구현부에 super()나 this()가 없으면 부모객체 생성자, overldading된 생성자 호출코드 super()(부모의 no- arg)가 첫 구문 으로 들어간다(컴파일러가, 명시적으로)

상위 클래스에 생상자 선언되어있어야 함???????????

- super. : 상위 객체의 변수나 메서드

메소드 오버라이딩

- 상속한 메소드를 하위 class에서 재정의 하는 것 구현을 재정리
- 규칙 : 선언부에 대한 규칙 1) 접근 제한자-> 부모의 것과 같거나 더 넓어야 한다.

메소드 오버로딩

부모클래스에서 받은 거를 다시 재정해서 사용?
- 상속한 메소드를 하

public - public
protected -> public protected

접근자 이름 매개변수는 같아야한다.

**생성자 오버라이딩은 없으므리다., 생성자는 상속이 안됨 class명이랑 같아야 하기때문임뫙

접근 제한자(258p)

UML	구분	같은 Class내	같은 패키지 다른 Class	하위 Class에서 상위 Class에 접근	어디에서든지	비고
-	private	○	×	×	×	IV
	package frendly (default)	○	○	상위 하위 클래스 패키지가 같은 경우 ○ 아닌 경우 ×	×	
#	protected	○	○	○	×	overridng용 method
+	public	○	○	○	○	method

Class - public, X
instance variable - public, protected, private
local value - X

final(254p)

변경이 안 됨.
class, method, variable에 다
final class 상속이 안 되는 class ex)animal // 상속을 받을 순 있다 ex_dog
method - 하위 class에서 overriding을 할 수 없다.
variable - 상수화, 변화가 안 되는, 한번 값이 들어가면 못 바꾼다.
약속된 불변의 값(코드 값) 파이

final_instance 변수, static변수는 묵시적 초기화(default) 초기화가 발생하지 않는다.
fianl instance변수는 -> 명시적 초기화 or 생성자 초기화가 이루어져야 한다.

final 변수의 관례 모두 대문자 단어_단어 (대문자이고, 단어와 단어를 언더바(_)로 구분)

(4주차)
15.07.29(수)

ctrl + shift + x : 블록 영역 대문자로
ctrl + shift + y : 블록 영역 소문자로

static (182p, 254p)

- class의 멤버 -> 객체와 상관없다.
- class block에 선언한 변수
- method
- local변수, class에 는 붙일 수 ㅇ벗다.

객채가 아니라 객체들이 공통적으로 가진 속성

-> 구문 변수 : [접근 제한자] static type 변수명([=값])
메소드 [접근 제한자] static returntype 메소드([[매개변수,,,])

외부에서 : 사용할 때
Class.명.ㅅ변수 // Class이름. 메소드

java 변수 정리

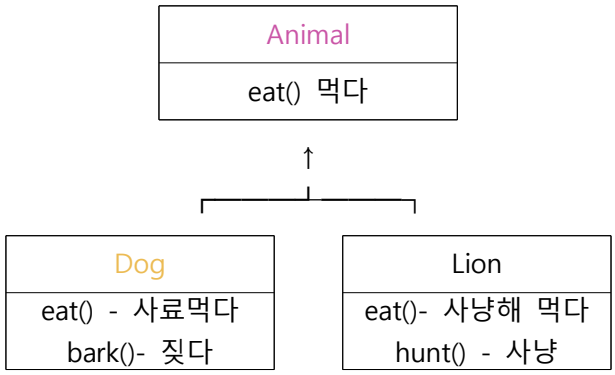
구분		선언위치	사용범위 scope	디폴트 초기화	저장 메모리 영역
Filed	static	class block내 (statkc 제한자)	program 시작~ 종료	일어남	class(method)영역
	instance	class block내	객체에 접근 가능할 때까지 (생성 ~)		Heap
Local(지역) method	method block 안에서 사용	method block내	메스드내에서만 사용가능 외부에서 호출이 안됨	사용 전에 명시적 초기화가 필요하다	execution(call) stack

다형성(Polymorphism) 265p

- 하나의 메시지에 여러 다른 응답이 나오는 것.
- 구현 부모type변수 = 자식type 객체
- 변수에 대입가능 객체가 여러종류가 될 수 있다.
- 부모는1 이지만 자식은 n개
- 하위에서 부모에 정의된 메소드를 재정의 오버라이딩
- 같은 메소드 호출시 대입된 객체애 따라 결과가 다르게 나올 수 있다.

reference 타입 상속

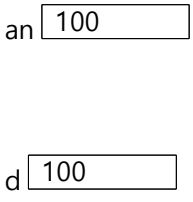
ex)
person >dog
dog c = new cat();
animal d = new dog();

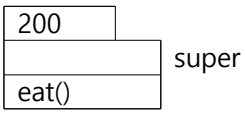
<table>
<tr>
<td> 
</td>
<td>Animal an - new Dog();
부모변수 > 자식객체

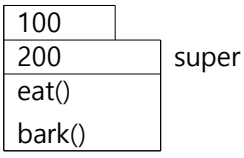
an.eat(); ->animall.eat()에서 확인 ok

an.bark() // animall bark이 없기에 compile error

Dog d = (Dog)an;
//an이 animal type으로 dog type보다 더 크기 깨문에 dog타입으로 형변환 해준다.

*animal type으로 만들었으면 각 상속받는 원래의 클래스의 메소드는 호출이 아니라 선언한 type의 클래스로 안 된다.</td>
</tr>
<tr>
<td> 
</td>
<td>Heap
object

-Animal객체-
 

-Dog객체-
 
</td>
</tr>
</table>

다형성이 필요한 이유

1) 배열

- 다형성 -> 배열 적용 : 부모 type의 배열안에 모든 자손type의 객체들을 저장.
- object 배열에는 모든 객체들을 대입할 수 있다.
  -> Collection API

**object 클래스는 자바에 존재하는 모든 데이터 타입을 받는다.

즉 다양한 상속된 클래스들로부터 데이터를 배열[]로 넣을 수 있다.

ex) 동물들[] 다양한 동물들을 넣을 수 있음 다양한 동물들을 추상화한 animal 부모클래스를

2) 매개변수

부모type변수를 메소드의 매개변수로 선언하면 호출하는 곳에서 모든 자산type의 객체를 받을 수 있다.

overloding이 필요 없다.

최소한의 수정으로 새로인 class 추가가능 -> **확장성**

ex) 개변수를 선언할 때 animal type으로 선언하면 밑에 하위 상속받는 클래스들을 다 쓸 수 있다. 고양이 강아지 사자 클래스를 각각 오버로딩 하지 않고...

****변수.메소드() 시점의 차이**

compile : 변수 type의 메소드

컴파일은 소스코드를 검사하는 것이지 실제로 공간을 만들고 실행하는 것이 아니다,

실행 : 객체의 메소드

컴파일 시점과 실행시점의 차이

(4주차)15.07.30(목)

homogeneous Collection : 같은타입 의 값 들을 모아서 관리- 배열
heterogeneous Collection : 다른타입 의 값들 모아서 관리- 배열에다형성적용

object - 모든 클래스(객체)의 최상위 클래스(객체) -> 모든 객체의 타입

객체 instanceof type

- 객체가 type의 객체냐? return true or false'
- 객체의 type : 부모(super) type
- Type : 자식(sub) type

추상클래스

- obstract ;/./ /.제한자 -> 미완이다.
- instance 메소드, class 제한자로 사용(변수 사용X)
- 추상클래스는 추상메서드를 가짐
- 추상메소드는 실행구문이 X

- instace 메서드 -> 선언만 한 메서도 -> 구현부가 있으며 안 됨 {}가
  ex) public abstract void go()
  하위 class에서 overrideing을 강제하기 위한 메소드
  하위 class에서 오바라이딩을 하지 않으면 abstarct 메소드를 받게되서 abstract class로 객체도 생성 못하게 될 수 있다.

- class : 객체 생성을 못하는 class
  super(부모) class 역할용 -> 자식객체들의 type의 역할, (다형성)
  자식클래스에서 마들어야 하는 (공통된) 동작이나 속성을 제공
  자식클래스들의 틀의 역할

- abstract class는 일반 class들이 가지는 것들
  (instance/static 변수, 매소드, 생성자) + abstract메소드를 가진다.
- abstarct 메소드가 있는 class는 반드시 abstract class가 되어야 한다.
- 생성자: 하위에서 super() 호출한 생성자 new ???()는 안되지만 메모리상 생성은 된다.
- 구현된 instance 메소드 - 하위 class들의 구현이 같은 경우 추상 클래스에서 구현해서 제공
- abstract 메소드 - 구현은 하위 class에서 하되 그메소드 구문은 통일 시켜야 하는 경우

(4주차)15.07.31(금)

인터페이스(204p)

- 추상class의 역할 => 완벽한 추상
- **type의 역할**만하기 위한 새로운 구문. ==> 하위 클래스들의 type
- public 추상 메서드와 public static final 변수 만 가질 수 없다.
  static이 안 붙으면 객체?
- 객체 생성이 안 된다.

=- 다중상속가능

class <- class 단일

interface class - 다중상속

- interface구문: [public] interface 식별자이름 [extends Interface ....]{

//구현

[public static final] 변수 = 값;

[public abstract] 메소드 선언();

(interface) 생략가능? 생략하면 컴파일러가 자동으로 붙여준다.

final 변수는 값 초기화 반드시 해야 함.

메소드는 중괄호들어가면 안됨(abstract)

}

++[public]: option이지만 거의 100%붙인다.

++식별자 관례 - class와 동일(파스칼 표기법)

++extends : 다른 interface상속 시 추가->다중상속가능(콤마(,)로 연결)

Class 상속 안 됨.

- class가 interface 상속 구문

[제한자] Class 식별자이름 [extends Superclass] [implements interface이름, 이름,.....]

단일, 생략 : object 상속 받을 인터페이스_다중상속이 가능

{ 구현 }

**자바가 상속을 단일상속을 막은 이유는 설계상 여러 부모를 상속받으면 코드가 엉킬 여지가 많기 때문에 단일상속으로 정렬함, 인터페이스는 어차피 메서드를 하위클래스에서 메소드를 오바라이딩해서 쓰기 때문에 메소드가 엉키지 않기 때문에 인터페이는 다중 상속이 가능하다.

같은 개념을 확장 받을 때는 extends

다른 종류를 상속 받을 때는 implement

-상위 type의 역할 -> implement한 class들의 type 통일 ㄱ

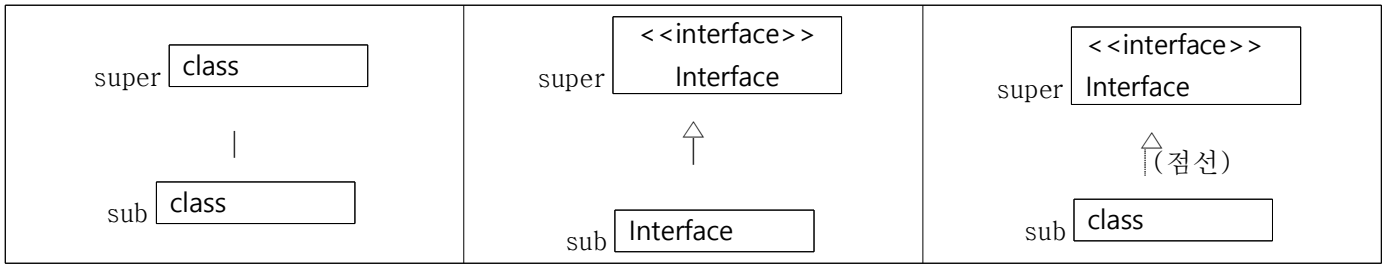
-> implement한 class들이 구현할 메소드 제공 ㄴ 다형성

-다중상속지원 -> 크게 연관 없는 class들을 하나의 type으로 묶을 수 있다.

비행기, 헬리콥터, 새 => 난다()라는 하나의 인터페이스 선언가능

클래스들의 공통적??? 인터페이스

UML



(5주차)
15.08.03(월)

java.lang 패키지(9장 p.366)

- **object** → 모든 Class의 최상위 Class. (모든 객체의 type이 될 수 있다.)
- toString() : String → 객체를 String 객체(값) 으로 변환.
  object → "Class이름@hashcode값"(16진수)

→하위에서 overriding : instance변수 값들을 하나의 문자열로 만들어 retrun.

- public boolean equals(Object obj)

→ boolean b : 객체1.equals(객체2)
객체1 == 객체2 --> 같은 객체인지 비교 // 기준에 따라 overriding
같으면 true 다르면 false.

equals() 오버라이딩
**같은 class에서 생성된 객체를 → instance 변수의 값들이 같으면 같은 객체
== 같은 객체인지를
equals는 객체의 값들이 같은지를
equals()를 overriding하면 반드시 **public int hashcode() 메소드를 같이 overriding 한다.
why.....
ex) a.equals(b) → true
int j = a.hashCode();
int i = a.hashCode();
a == b → true여야 함

a.equals(b) && a.hashCode() == b.hashcode();

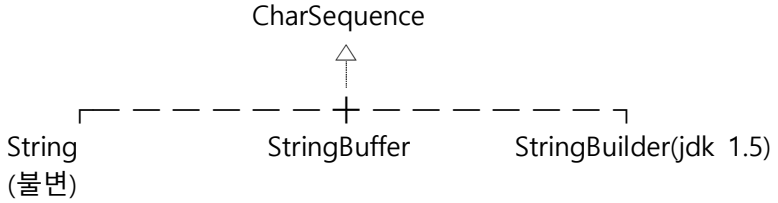
**관례적으로 사용되는 메서드 명
add~~() : 추가
set~~() : 변경/수정
get~~() : 조회
to ~~ () : ~~로 변경

Value Object :

=data Transfer Object(dto),
=domain object(DO)
→(객체)값의 type으로 사용될 클래스, instance위주의 class
- class 이름 : 값의 type명 : Customer
값의 type명 VO : CustomerVO
값의 type명 DO : CustomerDo
값의 type명 (D)TO : CustomerDTO
패키지에 표현하기도 함 - kr.or.kosta.**vo**.Student

- Class 구현
◎ public Class 로 선언
◎ implement java.io.Serializable
◎ private instance 변수 선언
◎ public setter/getter 메소드 제공
◎ 생성자 No-argument 생성자
모든 instance 변수 초기화하는 생성자 + a
◎ toString() overriding → 모든 instance변수들의 값을 하나의 문자열로 합쳐서 리턴.
◎ equals()/hashCode() overriding →instance 변수의 값들이 같으면 true 리턴.(기준 : instance 변수들 값
◎

String



공통점
1) 문자열을 다루는 Class들
2) 문자열 char[] 저정해서 관리.
배열방식으로 저장함, 즉 각각의 문자들이 index값을 가짐

- String : 불변객체, 한번 객체가 생성되는 그 상태(값)은 절대 안 바뀐다.
    1?.concat(2??) : 1?에 2??추가

comcat으로 추가해도 String 본래 값은 바뀌지 않음

- 문자열 조작 (값 변경)StringBuffer/StringBulider : mutable(불변객체가 아니다.)
여러 번에 걸쳐
조작 후 String으로 변환
    String Buffer
비효율적임 저정공간에 있어 때문에 buffer를 활용해

String 메서드

~.length() 글자수 세기
~.equalsIgnoreCase(~) : 대소문자 상관없이 비교
~.startsWith(??) : ??으로 시작하는지 조회(boolean)
~.endWith(??) : ??d으로 끝나는지 조회(boolean)
~.indexOf(??,*) : ??이 몇 번 인덱스에 있는지 조회 0번부터 조회해서 처음 찾은 값 리넡ㄴ
*번부터 조회시작
~.LastIndexOf(??) : indexOf의 반대방향에서 조회시작
~.charAt(?) : index의 char(한글자)를 리턴
~.toLowerase : 대문자를 소문자로 바꾸어 리턴
~.toUpperCase : 소문자들을 대문자로 바꾸어 리턴
~.substring(숫자) : 넣은 숫자 index부터 마지막 인덱스까지를 리턴
(숫자1,숫자2) : 숫자1부터 숫자2까지 출력 숫자2는 제외하고 리턴
**대개 시작부터 끝 숫자를 명시했을 경우 끝 숫자는 제외한다.
~.trim() : 문자열 기준 좌우 광백 제거 하고 리턴
~.replace('a,'가') : char 'a'를 char '가'로 변경, 한 글자를 쏵 다 바꿀 때.
~.replaceAll("aaa","bbb") : 문자열 aaa를 bbb로 변경하여 리턴
~.split("?") : ?를 기준으로 쪼개서 배열을 넣는 메소드
₩" 이스케이프 문자로

(5주차)15.08.04(화)

object - 최상위 - 모든 객체의 type

StringBuffer

- String과 달리 변하는 값을 사용할 때

StringBuffer a = new StringBuffeer("~~")

StringBuffer a = new StringBuffeer(숫자) : 숫자만큼 문자열 저장할 공간의 최초 크기 지정

StringBuffer 메소드

~.length() : 글자 수 리턴

~.setCharAt(2,'c') : ~에 2번째 인덱스 값을 c 로 변경해서 리턴

~.setLength(5) : 글자 수 변경 . 5번째 인덱스 이후 삭제하고 리턴

~.append("hi") : 문자열 뒤에 추가(다양한 타입가능)

~.insert(1,"lee") : 1번index "lee" 문자열을 삽입

~.capacity() : 글자를 저장할 공간- 버퍼의 크기 조회28

~.delete(2, 5) : 2~5 인덱스의 글자들을 삭제. 5번은 포함 안됨.

~.replace(0, 2, "영일오") : 인덱스의 글자들을 "영일오" 로 변경, 2번 인덱스 포함 안 됨.

~.trimToSize() : 버퍼의 사이즈를 글자 수의 맞추어서 줄여줌.

**stringBuffer는 equals를 오바라이딩 하지 않아서 Object의 것을 그대로 사용해서 equals()사용 시 == 으로 객체를 비교한다. 비교하려면 toSting으로 String값으로 변환 후 equals()로 비교하면 됨

**나머지는 index참고

메소드 체인

- 객체를 변경하는 메소드가 처리 후 변경된 객체(자신)을 리턴 하도록 해서 연속적인 변경이 가능하도록 메소드를 구현하는 기법.
- 연속적으로 변경시 여러 명령문에 걸쳐 할 필요 없이 하나의 명령문으로 처리 가능.

ex) sb.append("c").append("f").insert(1,"안녕

Math 클래스(395p)

- 생성자가 private
- 변수/ 메서드는 public static
- 객체를 생성 할 필요 없이 클래스 객체로 접근 가능
- PI , E (final)

wrapper 클래스(397p)

- object o = 모든 객체(Primitive/Reference)
- Primitive Type의 값을 객체화하기 위한 Class들
- primitive sdtype 8개이고 이에 맞추어 wrapper 클래스도 각각 8개
-

기본(Primitive)	래퍼클래스
boolean	Boolean
char	**Character**
byte	Byte
short	Short
int	**Integer**
long	Long
float	Float
double	Double

boxing

- 생성자를 통해 boxing

ex) new Integer(30)
new Double(3.5)
new Long("30");

**문자열 값을 넣을 경우 반드시 변환 가능한 값이어야 한다.

unBoxing
Wapper 객체 -> primitive (unBoxing)
Warpper 객체. xxxValue().xxx
xxx : primitive type명

맞는 타입의 값을 넣어 주어야 한다. 아니면 실행할 때 에러 발생

<table>
<tr><th colspan="2">AutoBoxing</th><th colspan="2">AutoUnboxing</th></tr>
<tr><td>Integer in = 20;</td><td>20 -> new Integer(20);
20은Integer 객체</td><td>int k = in</td><td>int i = in.intValue()</td></tr>
<tr><td>Long l = 500000L;</td><td>500000L ->new 500000L</td><td>long ll = l;</td><td>long l = lon.longValule();</td></tr>
<tr><td>Character c = 'a';</td><td></td><td>double dd = d</td><td>double d = dou.doubleValue();</td></tr>
<tr><td colspan="2" rowspan="2">autoBoxing으로는 문자열을 넣을 순 없다.
Integer in2 = "20"; error
Integer in2 = new Integer("20") 可能</td><td colspan="2">객체타입을 다른 primitive 타입으로 전환할 경우 형변환을 해야 한다.</td></tr>
<tr><td>Object o = 50;</td><td>int k = (Integer)o
//형변환
int r = (int)o</td></tr>
</table>

- 문자열을 String를 primitive 값으로 변환 → Wrapper Class들이 static메소드 제공(Character제외)
  (각각의 랩퍼 클래스에 있는 **parse 메서드**를 이용해서 변환)
- 구문 : WapperClass.parseXXX(String) :
  **XXX는 primitive type,
  **String 값은 변환 가능한 문자열이어야 함

ex) String → long

```
long l = Long.parseLong("30");
```

String → boolean

```
boolean b = Boolean.parseBoolean("true");
```

**parseChar은 없음

java.util (515p)

- Date
- Calendar(GregorianCalendar)

→ 날짜,시간(일시)를 관리

Date()

```
//년도 조회
    int q = d.getYear(); //줄거진 것은 퇴화된 메서드표시
}
```

- deprecated 퇴회된 메서드,생성자, class : 없애고 싶으나 기존 코드와 호환성 때매 없애지 못함
- 줄 그어진 게 퇴화 되었다는 표시, 구현은 가능함
- year값의 경우 1900년대에 나와서 1900이 + 되어있다.
- Date d = new Date(); : 실행시점 의 일시를 가진date객체생성
- int값 형태로 리턴 받는다.
- Date d = new Date(year,month,date,hour,minute,second);
- Date 생성자 : 노아큐, 연월일, 연월일시분, 연월일시분초
- Date 메소드

    int year = d.getYear() : -1900한 값 +1900
    int month = d.getMonth() : 0~11월 +1
    int date = d.getDate() : 일 조회
    int hour = d.getHours() : 시간
    int minute = d.getMinutes() : 분
    int second = d.getSeconds() : 초
    int day = d.getDay() : 요일 일:0 ~ 토:6

- Long millsec = d.getTime(); : 1970-1-1 0:0:0 ~ date 객체가 가진 일시를 밀리초로 계산해서 리턴
- Date a = new Date(5000000000L); : 1000분의 1초를 단위로 계산해서 리턴

Calendar()

```
Calender d = new Calendar(); 추상클래스
Calendar c = Calendar.getInstance(); //실행시점에 일시를 가진 객체를 리턴.
```

**캘린더 클래스는 추상클래스이기 때문에 추상메소드 객체를 리턴

- ?.get(Calendar.추상메서드) : ?객체의 get 형식으로 리턴받음

    Calendar.YEAR : 년도
    Calendar.MONTH : 월 0~11월 +1
    Calendar.DATE : 일or DAY_OF_MONTH
    Calendar.HOUR_OF_DAY : 24시간제
    Calendar.MINUTE : 분
    Calendar.SECOUND : 초
    Calendar.DAY_OF_WEEK : 일 : 1 ~ 토 : 7
    Calendar.AM_PM : 오전/오후, 0/1로 리턴

** 월의 경우 0~11이기 때문에 +1을 해줘야 1~12로....**
** Calendar 나 date나 값들을 모두 int로 받기에 요일이나 달을 문자로 표현하는 Calendar의 경우 이런
- AM, PM // 0,1
- SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, or SATURDAY // 일:1 ~ 토:7
- JANUARY, FEBRUARY, MARCH, APRIL, MAY, JUNE, JULY, AUGUST, SEPTEMBER, OCTOBER, NOVEMBER, DECEMBER // 0~11월

EX)
System.out.println(month == 7);
System.out.println(month == Calendar.AUGUST);
둘이 같은 거

GregorianCalendar

요넘은 추상클래스 아님, 서양시계인가
생성자 : 노아큐, 연월일, 연월일시분, 연월일시분초 등등

cd.add(Calendar.MONTH, -5); : 숫자 추가?.add

boolean flag = cd.isLeapYear(2011); 윤년 인지를 리턴booelan으로

(5주차)
15.08.05(수)

형식화 클래스(561p)
java.text.format →값을 원하는 형태의 문자열로 변환

- DecimalFormat : 숫자(정/실수)를 변환
  # : 10진수 숫자, 표현한 자릿수보다 적은 값일 경우 무시
  0 : 표현한자리수보다 적을 값일 경우 0으로 채움 00 5 → 05
  . : 소숫점 표현
  , : 단위구분자
  % : 페센트 표시. 값*100 결과에 %를 붙인다.
  $: 달러표시
  ₩u00A4 : 원화표시

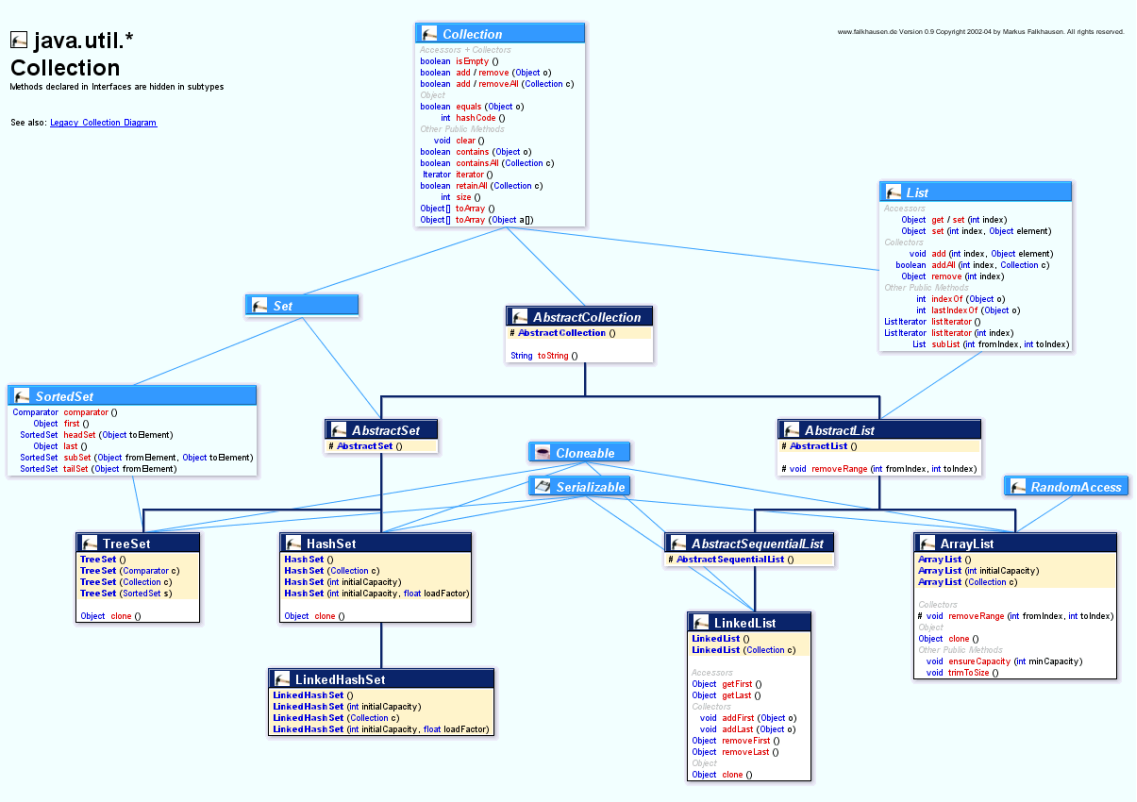
- SimpleDateFromat : date(일시) 객체를 변환
  + 주요전환 문자
  y : 년 - yyyy
  m : 월
  d : 일
  h : 시간 (1~12)
  H : 시간(1~23)
  s : 초
  E : 요일
  a : 오전/오후
  *월, 일, 시간, 분, 초 - 한 개를 사용하면 일 단위는 한자리로 표시,
  두 개를 사용하면 일 단위를 두 자리로 표시.
  ex) mm한 경우 5를 05로 표기
- 전환문자를 제외한 문자는 그대로 그 자리에 표시

parseObject

- Foromat : 객체 생성 시 변환 패턴을 지정해줌
  format(값) : String, 값을 문자열
  parseObject("문자열") : Object → 값
  **값의 type은 object이나

Collection api(428p)

- 객체를 모아서 관리하는 객체→ object[] 기능을 추가한 class들



- ①Collection : 객체(값)만 모아서 저장

  ①set : **중복을 허용**하지 않는다.(같은 객체 저장 시) equals() && hashCode();

  ⓒ~set / HashSet

  ①List : **순서**가 있다._추가한 순서대로 관리

  ⓒ~list

  ⓒArrayList

  ⓒVector

- ①Map : **key-value 쌍으로 모아서 저장.**

  object

  ⓒ~Map

  ⓒHashMAP

  ⓒHashTable

  └→ⓒProperties key

  value String

** set / list / map 이냐를 각각의 특징에 따라 선정

주요메서드-**set**

add(Object) : boolean - 객체를 Collection에 추가.

└→ 추가 성공여부

add(Collection) : boolean → 매개변수로 받은 Collection 만의 객체를 추가.

└→하나 이상 추가되면 true

contains(Object) : boolean Collection 안에 매개변수로 받은 객체가 있는지 여부.

size() : int → Collection 내의 요소들의 개수를 리턴 (마치 배열의 ~.length와 유사한 기능)

remove(object) : boolean → 매개변수로 받은 객체를 삭제

remove(Collection) : boolean → 매개변수로 받은 Collection안의 객체와 같은 것들을 제거

clear() : 쏵다 밀어버림

(5주차)

15.08.06(목)

주요메서드-list

- list : 계열 추가순서가 있음.
  + 저장된 객체들 마다 index(순번을 붙이고 그 순번을 객체들을 관리한다.
  + 중간 순번이 비어있지 않도록 내부적으로 처리된다.
  + 주요 메소드 : Collection에 정의된 메소드 + 순번과 관련된 메소드들.
    * add(Object) : boolean - 추가(순서대로 들어간다)
    * add(int idx, Object) - void - idx 인덱스Object를 **삽입(index들의 공백이 생길 수 없다.)**

    * remove(Object) : boolean - Object와 같은 객체를 리스트에서 제거
    * remove(int idx) : Object - idx의 인덱스의 객체를 리스트에서 제거 후 리턴.
      └──> 삭제 후 비게 된 index는 뒤의 객체들로 채운다.

  * size() : int - 저장된 객체의 개수를 리턴.
  * indexOf(Object), lastIndexOf(Object) : int - Object가 몇 번 인덱스에 있는지 리턴

  * contains(Object) : boolean - 매개변수로 받은 객체의 존재 여부 (collection과 동일)
  * get(int idx) : Object - idx인덱스에 저장 돼있는 객체를 조회.
  * set(int idx, Object) : Object - index의 객체를 매개변수로 받은 객체로 변경하고 기존 객체를 리턴

- 향상된 for문 → 배열내의 모든 요소들 조회
  → Collection(set,list)도 사용가능 map은 불가
  for(Object ? : Collection객체)
  요소를 저장할 변수 선언 Collection, 배열

MAP: Key value 쌍으로 객체를 모아서 관리하는 컬렉션API
- **key**가 모으려는 객체(value)에 대한 **식별자**로 사용된다.
- 주요 클래스 : HashMap, Hashtable
- 주요메서드 _ map
  * put(Object key, Object Value) : Object - 추가와 변경하는 메소드.
    └──> key로 저장된 객체가 없으면 추가, 있으면 변경.
    └──> 변경일 경우 기존 value객체를 리턴, 추가일 경우는 null를 리턴.
    └──> key : 중복이 안된다. Value는 중복가능.
    └──> key는 주로 String을 사용,
  * remove(Object key) : Object(Value) - 매개변수로 받은 key의 객체를 제거하고 삭제한 value객체를 리턴
  * clear() : Map내에 저장된 모든 객체들을 제거

* get(Object key) : Object(value) - 매개변수로 받는 key의 값(객체)를 조회.(없으면 null)
* containsKey(Object key) : boolean - 매개변수로 받은 key가 있는지 여부.
* containsValue(Object Value) : boolean - 매개변수로 받은 value가 있는지 여부.

* **keySet()** : Set - Map내의 key객체들을 Set에 모아서 리턴.(set이 중복허용을 하지 않기 때문에 list로 안받음)
* **entrySet()** : Set - Map내의 Entry 객체들을 Set에 모아서 리턴.
  └──> entry : key-value - 쌍으로 모은 것

**map은 index값이 없어 for문을 조회할 수 없기에 keyset과 entryset을 활용한다.

Entry Class
  └──> key : Object / value : Object

Entry객체를 새로 만들어서 넣어주는 것. key 값과 value값을 묶어서 entry 객체 하나로

hashMap hsatable

(5주차)15.08.07(금)

Iterator

Iterator(Interface)
- Collection(list, set) 내의 요소(element)들을 전체 조회 할 때 사용.
- Collection객체.iteratro() 메소드를 이용해 사용합니다.
  └──>전체 조회할 Collection 객체로부터 객체를 얻어온다.
- 주요 메소드
  next() : Object - 컬렉션내의 하나의 객체를 조회 없으면 에러
  hasNext() : boolean - 조회할 객체가 남아 있는지 여부 조회
  remove() :

Enumeration 1.0 ->
HasMoreElemets() / nextElement()

업글

Iterator 1.2 ->
HasNext(); / Next(); / **remove() 추가**

listItarator 1.2
previous() : 이전 값
hasprevious() :

(6주차)15.08.10(월)

Generics(제너릭스)

- class내에서 사용할 type을 class 구현 시 지정하지 않고 사용 시 지정하는 기법.

  <구문> Class 선언 시 가변type 선언 :

  public Class 이름 <가변type A[, type B,....]>{

  Type(변수들의 type)을 가변type으로 선언

  }

- 사용 시 가변TYPE에 들어갈 type을 지정

  Class이름 **<type지정>** 변수 명

- 가변 type 지정
  - ↳식별자 규칙
  - ↳여러 개 지정 가능.
  - ↳관례: 대문자, 한 두 글자 이용해 표시
  - ↳Type의 제한 : <T extends Super> → Super의 Type으로 제한
- Caller에서 가변 type에 들어갈 type을 지정하지 않으면 Object가 된다.
- 장점 1) 불필요한 casting이 줄어든다.

  2) casting시 casting하는 type값에 대한 불안전적인 요소가 줄어든다.

예외처리(exception handling)(330p)

1. 오류 : 프로그램이 정상적으로 실행될 수 없는 상태.
   - ↳ ⓐ Error(에러) : 프로그램 내에서 처리가 불가능한 오류
   - ↳ ⓑ Exception(예외) : 프로그램 내체서 처리가 가능한 오류
   - ↳ java는 모든 오류를 객체로 처리한다. → Class로 정의해 사용.

2. 예외처리(Exception Handling)
   - ↳ 발생한 예외(Exception)을 처리해 프로그램을 정상화 시키는 것.

3. Exception(예외)의 종류
   - ↳ ⓐ unchecked 계열 Exception
     - * Exception 처리 여부를 compiler가 체크하지 않는 예외.
     - * 보통 프로그램 코드를 잘못 만들어서 발생하는 경우가 많다.
       - + 업무 흐름에서 나오는 예외가 아니라 언어의 core적인 부분에서 발생하는 오류
       - + 발생확률이 100% 이므로 처리보다는 고치는 것이 낫다.
     - * 최상위 클래스 : RuntimeException
   - ↳ ⓑ checked계열 Exception

     *Exception 처리 여부를 컴파일러가 체크한다.

     + 발생 Exception을 처리하지 않으면 컴파일 에러가 발생한다.

     * 주로 업무 흐름상 발상 가능성 있는 예외를 표현한다.

+ 실행 환경 상 문제.(환경이나 사용자)

* 코드 작성 시 점에는 오류 발생여부를 알 수 없다.
* 최상위 : Exception

___오류가 날수도 있고 안날수도 있으니 났을 때 처리할 것을 구비해라

* 오류 발생시 exception 객체가 생성이 된다. 그것을 잡아내는 게 catch임
* exception은 발생되면 상위 호출한 곳으로 계속 던져진다. catch 될 때까지.

- Throwable : 모든 오류의 최상의 type(상속구조)
  - ↳ ① Error
  - ↳ ② Exception - ⓐ. 모든 Exception의 최상위(발생 <- throw)
    - ⓑ checked Exception의 최상위
    - ⓒ RuntimeException : unchecked Exception의 최상위
    - ⓓ 기타
- try - catch - finally : 직접척리
- throws : 간접처리

- 예외처리 구문

```
try{
Exception 발생 가능성이 있는 code
}catch(발생Exception type의 변수 선언){
예외 처리코드.
}.....[catch]
finally{
 무조건 실행되는 코드들(예외상황과 관련 없이)-- 대개 자원을 반납하는 코드 close()
}
```

**try문 하나의 여러 개의 catch를 통해 여러 개의 예외처리를 할 수 있다. 여러 개의 catch문이 있어도 하나의 catch문의 충족하면 try-catch문을 빠져 나온다.

- 발생 exception 객체를 받을 변수를 catch 선언 시 상위 타입의 변수로 선언가능
- catch 여러 개 잡을 경우(다중 catch) 하위 타입부터 잡아야 한다.
- 발생한 exception 들마다 처리방식이 다른 경우는 다중catch
- 동일한 예외처리를 하는 경우 상위타입으로 한 번에 처리한다.

- catch문 실행되고 elseIf처럼 try문 완전 탈출

- 순서는 try - catch(다중가능) - finally(다중 그런 거 없음)
- try단독사용 불가, try -catch / try -finally / try - catch - finally로 사용가능.

-close() 무조건 해주어야한다 이유는 시스템 상 노드의 영역을 한정 지어 놨을 수도 있고 한 번 스트림이 형성되고 닫지 않으면 계속 남아있어서 영향력을 끼치거나 메모리를 먹는당 엿나?

(6주차)15.08.11(화)----------

Caller메소드로 돌아가기
ⓐ return : 정상 완료
ⓑ 예외발생 : 비정상 발생

- 보통 예외처리는 발생한 곳보다는 호출한 곳에서 하는 경우가 다반사임.

Callstack 매커니즘
- 결국 호출한 곳으로 돌아간다 jvm까지
jvm - main - a() - b() - c()
a() b() c() exception

throws
- 메소드 안에서 발생한 Exception을 호출한 곳으로 던지겠다는 표시 할 때 사용.
- 메서드 선언부에 온다.
- 메소드 구문 : [제한자] 리턴type 식별자이름([매개변수, ...]) [throws Exception 명,...]
- 생성자 구문 : [제한자] Class 이름 ([매개변수]) [throws exception, ...]

method overriding - 상위 클래스에 정의된 메소드를 하위클래스에서 재정의
↳ 규칙
① 접근제한자의 규칙 : 부모의 것보다 더 넓거나 같은 접근 제한자를 가진다.
② method리턴타입, 이름, 매개변수(타입)가 같아야한다.
③ Exception관련 규칙 - throws 구문, 부모클래스에서 throws한 Exception(TYPE)만 throws 할 수 있다.
↳ 부모클래스에 throws 한 것 안 던질 수 있다.
*unchecked Exception RuntimeException하위는 예외

- Exception Class 구현

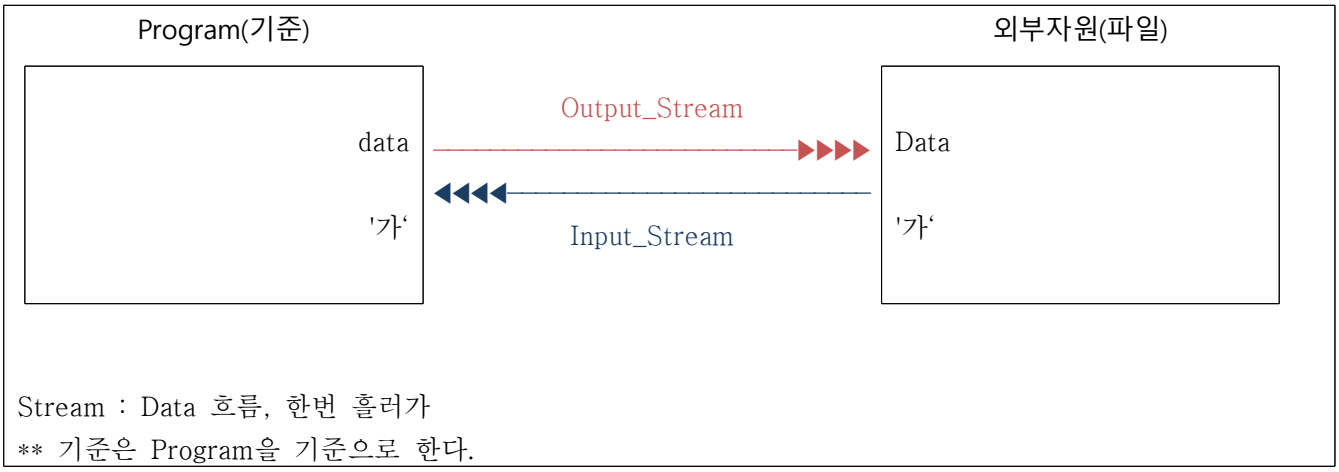
```
public class Exception 이름 extends Exception{
        no arg생성자 public 생성자(){}

        String 받는 생성자 public (String message){
        Super(message);
        --> Exception 처리하는 곳에 전달할 메시지 설정
        }
}
```

**이름 : 예외 상황을 표현 할 수 있도록 이름을 준다.
~Exception을 관례적으로 붙인다.
**상속 : -checked 계열 : Exception 상속
- unchecked 계열 : RunTimeException 상속

(6주차)15.08.12(수)

I/O

Input(입력)
Output(출력)



- java.IO 패키지
- Input(입력) / Output(출력) 둘 중 하나로 구분됨
- Node계열 / Filter계열
- byte 계열 / character 계열.

** 3가지 속성 값을 가진다.

Node 계열 Stream Class
+ 목적 : 외부차원과 연결
+ 다양한 입출력 기능은 제공 안함

Filter 계열 Stream Class(Decoration Stream)
+ 목적 : 기존 연결된 Stream에 다양한 입출력 기능 제공(추가)
+ 연결 기능은 없다.

***Node로 연결하고 거기에 Filter를 추가한다.

FileOutputStream - 연결은 되지만 기능은 X
DataOutputStream _ wirteInt(?) : 원하는 데이터 크기로 읽/쓰
-->filterStream은 연결기능은 없지만 node Stream을 감싸않아서 사용될 수 있으므리다.

- Byte 계열

-> 입출력 단위가 1byte

-> binary data 입출력용

-> I : InputStream / O : OutputStream

- Character 계열

-> 입출력 단위가 유니코드 문자(2byte) 한 글자

-> I : Reder / O : Writer

unicod 2byte 8*2bit = 16 65536가지

최상위 class -> 추상 Class

	Byte 계열 - binary Data		Character계열 - 문자	
	Input	Output	Input	Output
최상위 Class(추상)	InputStream	OutputStream	Reader	Writer
node	~InputStream	~OutputStream	~Reader	~Writer
filter		DataOutptStream		

Node / Filter -> 생성자의 매개변수로 구분가능(최상위클래스를 받음)

node : 연결대상 / Filter : Stream type

int i = fis.read(byte[]) : byte배열의 크기 만큼 데이터를 읽어서 배열에 넣는다.
읽은 바이트 수를 리턴 EOF를 읽으면 -1리턴

int i = fis.read(byte[], int startIdx. int length) : 배열의 startidx부터 읽을 데이터를 length만큼만 저장해라

fos.wirte(byte[]) : byte[] 내의 모든 데이터를 한번에 출력

fos.wirte(byte[], int startIdx, int length) : 배열의 startIdx번 인덱스부터 length만큼 출력해라

(6주차)15.08.13(목)`----------

PrintWriter(___)

PrintWriter(File file)
Creates a new PrintWriter, without automatic line flushing, with the specified file.

ㄴ> 명시된 file을 printwriter와 같이 생성한다.

- PrintStream의 업글
- printWriter(모든 type을 문자열로 출력하는 Stream)
- 매기변수로 file, outputStream, Writer, String
- byte/character 계열의 FilterStream
- File과 직접 연결이 가능??? filter? node?
- 파일이 있으면 덮어씀

- 메소드 print, printf, println, append 등

-BufferdReader

: Filter 계열 스트림 - 라인단위로 읽는 기능을 제공.
- readLine() : String - 엔터문자(₩n)를 기준으로 ㅇ릭는ㄴ다. (단 엔터는 읽지 않는다.) EOF를 읽으면 null을 리턴.

-나머지는 api doc문서 참고

flush

- 버퍼에 있는 거 롸잇 나우 출력

Interface Flushable

All Known Subinterfaces:

JavaFileManager, StandardJavaFileManager

All Known Implementing Classes:

BufferedOutputStream, BufferedWriter, ByteArrayOutputStream, CharArrayWriter, CheckedOutputStream, CipherOutputStream, Console, DataOutputStream, DeflaterOutputStream, DigestOutputStream, FileOutputStream, FileWriter, FilterOutputStream, FilterWriter, Formatter, ForwardingJavaFileManager, GZIPOutputStream, InflaterOutputStream, JarOutputStream, LogStream, ObjectOutputStream, OutputStream, OutputStream, OutputStream, OutputStreamWriter, PipedOutputStream, PipedWriter, PrintStream, PrintWriter, StringWriter, Writer, ZipOutputStream

public interface **Flushable**

A Flushable is a destination of data that can be flushed. The flush method is invoked to write any buffered output to the underlying stream.

Since:

1.5

PrintWriter(OutputStream out, boolean autoFlush)
Creates a new PrintWriter from an existing OutputStream.

↑↑ 이걸 쓰려면 노드 연결을 FileWriter 잡아주고 해야 한다. printWriter가 직접연결도 가능하지만 기능적인 부분을 활용할 때는 노드를 받아야한다.

printf

* 구문 -**printf(String fomat, Object...args)**
* 1.5 이상
* - 출력패턴(형식화된 문자열)을 이용해 출력하는 메소드.
* - format : 출력하고자 하는 내용, 전환문자(패턴문자)를 이용해 패턴화 해서 전달,.
* - args : format 문자열의 전화문자에 들어갈 값. 전달 순서대로 대입된다.
*
* 전환문자 - 패턴 : %전환문자, %[갑순번]전환문자,
* %d - 정수
* %f - 실수, 실수의 경우 소수점 이하를 6자리로 맞춘다. 자리는 0dmfh
* %s - 문자열
* %n - 엔터기능
* %% - % 출력

%(숫자)d : 숫자만큼 공간을 주고 숫자 자리 내에서 넣은 값을 오른쪽 정렬
%-(숫자)d : 숫자만큼 공간을 주고 숫자 자리 내에서 넣은 값을 왼쪽 정렬
%(숫자)$d : 달러 앞 (숫자) 번째 arg 값을 넣어준다

Date나 Calender도 출력가능
"%tY - %1$tm - %1$td %1$tA %1$tH:%1$tM:%1$tS %n", (Date or Calender)
%tY : 년 $tm : 월 $td : 일
$tA : 오전/ 오후
$tH : 시 $tM : 분
%tS초

(6주차)

15.08.14(금)

Java.io.file

- File

생성자 매개변수로 Directory 객체(절대 경로)

File 객체(상대경로) : class 파일을 기준으로 찾아감

Directory, File(절대경로, 상대경로)로 넣을 수도 있음

*절대 경로 : 루트디렉토리(최상위)를 기준으로 시작

*상대 경로 : 현재 시점을 기준점으로

메서드 : getPath() : Sring - 경로 리턴

getAbsolutePath() : String - 절대경로 리턴

exists() : Boolean - 파일 존재 여부

isDirectory() : Boolean - 디렉토리인지 여부, 없으면 false

isFile() : Boolean - 파일인지 여부, 없으면 false

canRead() : Boolean - 읽기 가능 여부

canWrite() : Boolean - 쓰기 가능 여부

.length() : Long - 파일의 크기(byte)

.lastModified() : Long - 수정된 시간

.mkdir() : Boolean - 디렉토리 생성 메소드, 못 만들어거나(권한 없음), 이미 존재할땐 false

.mkdirs() : Boolean - 위에 꺼+ 부모 디렉토리가 없으면 부모 디렉토리도 만들어줌

.createNewFile() : Boolean - 파일생성, 0byte , 파일생성여부 리턴

.delete() : Boolean - 파일 삭제, 삭제여부 리턴

.renameTo() : Boolean - 파일 이동 및 이름 변경, 성공여부 리턴, 존재하고 있으면 실패임

표준 입출력

<table>
<tr><th colspan="3">System</th></tr>
<tr><td>표준입력장치
(키보드)
------Input----></td><td>p/g
------Output----></td><td>표준출력장치
(화면)</td></tr>
<tr><td colspan="2">- `
InputStream,
byte계열, 입력, 최상위</td><td>- System.out
- System.err
: 오류 출력용 변수

**두 가지는 변수명의 차이
**printStream -> PrintWirter -> print, println, printf</td></tr>
</table>

da

Byte계열을 Character계열로 바꾸주는 Filter

InputStreamReader(InputStream) : Reader 객체로, Chartacter계열의 Filter

OutputStreamWriter(OuputStream) : Wirter 객체로, Chartacter계열의 Filter, 거의 안씀

ex) InputStreamReader ir = new InputStreamReader(System.in);
-> byte계열을 chat계열의 필터인 InputStreamReader로 전환해서

컨트롤 z로 EOF를 보낼 수 있다.

(7주차)15.08.17(월)

연결 - 출력 - 닫기
http://shin6666.tistory.com/90

Serialization
* 직렬화(Serialization,객체 출력) **<->** 역직렬화(객체 입력) : 객체(대상) 입출력

* 객체 출력 대상 -> **java.io.Serializable(**Serializable은 Class가 아닌 **Interface**) Type의 객체만 가능
즉 class fff implemets java.io.**Serializable**
-> 객체의 속성이 (입)출력 대상이 된다.(생성자,메소드,static멤버X)

* 입출력은 값을 목적으로 하기 때문에 Serializable Interface를 써야함
스트림을 통해서 값이 나가서 저장되고 또 불려지기 때문에 직렬화가 필요해서

* 역직렬화(ObjectInputStream) - readObject() : Object -> 객체 입력 **FilterStream**
* 직렬화(ObjectOutputStream) - writeObject(Object) -> 객체 출력 **FilterStream(**Serializable)
객체의 **속성=instance 변수가 (입)출력** 대상이 된다. (생성자, 메소드, static멤버 x)

* 입출력 대상 - 객체(Serializable)
속성(Instance 변수) : 직렬화대상
- Primitive type변수
- Reference Type변수(객체) - Serializable Type

* **transient** : instance변수에 붙이는 키워드(제한자 -> instance변수)
객체 직렬화 대상에서 제외할 instance변수에 붙인다.

이유 : 1.보안적 이유
2.Serializable 타입이 아닌 instance변수

* Serializable을 구현하지 않은 클래스드의 객체(인스턴스)들도 직렬화 대상에서 제외 즉 출력 불가
* sub클래스가 Serializable을 구현했어도 부모클래스가 받지 않으면 부모클래스내 멤버들은 직렬화가 안됨

_class not found exception (생성자, vo)
**serialVersionUID : 모든 serialzation 이 필요한 클래스에는 명시적으로 serialVersionUID 를 선언해 주는 것이 좋다. 그 이유는 디폴트 serialVersionUID 계산은 클래스의 세부 사항을 매우 민감하게 반영하기 때문에 컴파일러 구현체에 따라 deserialzation(serialization 했던 객체를 복구하는 과정) 과정에서 예상하지 못한 InvalidClassException을 유발할 수 있다는 게 이유란다
** private static final long ***SerialVersionUID*** 1L;

오늘한 코딩

(4. 저장, 3. 종료)버튼을 누르면 지금까지 등록된 회원 정보(회원객체)를 저장
- instance변수 ArrayList list객체를 파일에 저장
프로그램이 시작되는 시점(MemberInput 객체 생성시점) - 저장된 회원정보를 읽어서 ArrayList에 저장 (4)

- 전체조회, 등록 -> instance변수 list를 가지고 처리
Hashmap 했다가 저장에서 List 옮겨 타게 해놨어

시점 : 3,4 출력
2 입력

1. MemberInput 생성자 (list 초기화) -> 파일에서 저장된 ArrayList객체를 읽어서 int 대입
-> 단, 대입값이 없으면 직접 생성

2. 종료, 저장메뉴에서 save() 메소드 구현 -> list instance 변수의 값(ArrayList 객체)를 파일에 저장

(7주차)

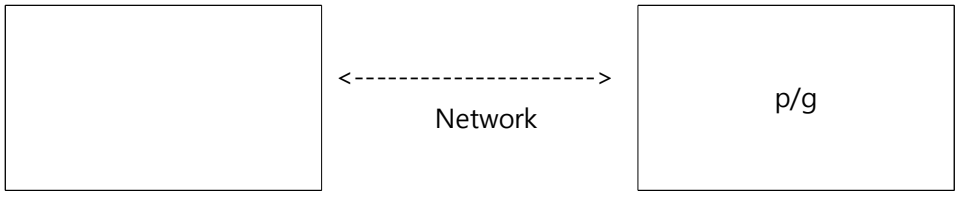
15.08.18(화)

try(연결자원 = 연결){코드} **--> 자동으로 닫아줌(close)**

ex) try (bufferedReder br = new BufferedReader){} **--> 자동으로 닫아줌(close)**

jdk 1.7만 가능

네트워크



- computer <-> computer
- data를 주고받기 위함
- internet(인터넷) : 전세계 computer들이 연결된 것.

1) 네트워크연결 node
2) data 입/출 IO
3) 프로토콜(protocol) : 통신규약, 통신을 하는 두 개의 program간의 공통된 통신언어
   TCP : http, ftp : 연결기반, 통신을 하려면 연결부터...
      -> 데이터 전송의 신뢰적이다.(양방향성
      -> ip 주소 : 인터넷에 연결된 컴퓨터들을 구분하기 위한 고유 식별주소
         ↳ 4byte 체계 : 1byte, 1byte, 1btye, 1byte, 형식을 가진다.(각 구역은 0 ~255)
         ↳ domain 주소와 연결된 문자열로 구성된 가상의주소

      -> port번호 : 컴퓨터 내에서 network 서비스를 하는 실행중인 프로그램들(process)을 구별하가 위한 식별번호
         ↳0 ~ 65535(2byte) 중 하나를 사용한다.
         ↳0~ 1023번은 사용하지 않는 것이 좋다.(예약된 번호)
         ↳ 서비스를 제공하는 프로그램은 자신이 몇 번 port번호로 서비스 할지 지정해야 한다.
            ** 이미 사용중인 port번호는 사용할 수 없다.
   ** 서비스를 받는 프로그램(client)은 연결할 프로그램이 있는 ip주소와 port번호를 이용해 연결해야 한다.

* TCP - Socket을 이용해 network프로그램 작성.
   ↳ Socket : Network 연결점
   ↳ Socket통신 : 소켓을 이용한 통신

- java.net 패키지에 네트워크 프로그램 API를 제공한다.
- Client p/g 패턴

① 서버와 연결 : Socket 생성(ip, port)
Socket s = new Socket(ip, port);
**domain을 써도 된다?
② Socket으로부터 I/o stream을 조회
socket.getInputStream() : inputstream
socket.getOutputStream() : OutputStream
③ I / O 작업
Filter Stream 추가
입출력 작업
④ 연결 닫기 : I/O Stream.close();
Socket.close();

- Server Program패턴
**Client의 요청을 대기
① SeverSocket 생성 : Client의 연결 대기

② Serversocket.accept() : Socket - Client가 연결시 연결된 Socket을 return

③ I / O Stream 조회 (Socket)에서
④ IO 작업
⑤ Close - Stream 먼저 닫고 다음 Socket을 닫음

Server	Client
① ServerSocket 객체(port) <--------3000) ② ServerSocket.accept() I / O	③ Socket객체생성(ip, port) Socket s = new Socket(ip, 3000) I / O

**server가 한번 client를 받고 끝나기 때문에 while문을 넣어서 무한 반복으로 Client를 여러번 받을 수 있게 처리할 수 있음.

(7주차)
15.08.19(수)

(7주차)

15.08.20(목)

쓰레드

- 멀티 쓰레드, 멀티태스킹 : 동시에 여라작업 수행
- 쓰레드 : 프로그램의 한 흐름, 데이터와 코드가 실행하는 흐름을 쓰레드
- 원래의 하나의 흐름에서 쓰레드,, 여러개의
- Process - 실행중인 Program : data / code / Thread
- Multi-Thread : 한 Process 내에서 동시에 여러 Thread(실행흐름)이 실행되는 것.
  한 Process내에서 실행중인 Thread들은 code/Data를 공유할 수 있다.
- 구문 ① java.lang.Thread를 extends 하고 run()을 Overriding
  ↳run() → Thread가 실행시킬 코드
- Thread로 실행
  ① Class 객체 생성, start() 호출

- Thread 구현2
  ① java.lang.Runnablie를 implements 하고 public void run()을 overriding
  ② 쓰레드로서 실행
  Thread객체를 생성하면서 생성자에 1의 객체를 전달한다.
  Thread 객체의 start() 메소드 호출

runable 인터페이스
- thread에게 실행할 코드를 알려주는 것.

상속을 받느냐

방법 1

object ↓ Thread ↓ main class -Thread 객체를 만들어서 start()

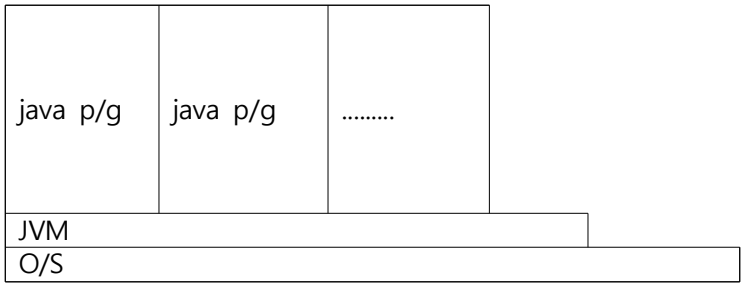
방법2

main class threadclass 객체 사용 start();	object ↓ Thread ↓ threadclass extends Thread public void run() 실행구문

implement
extends

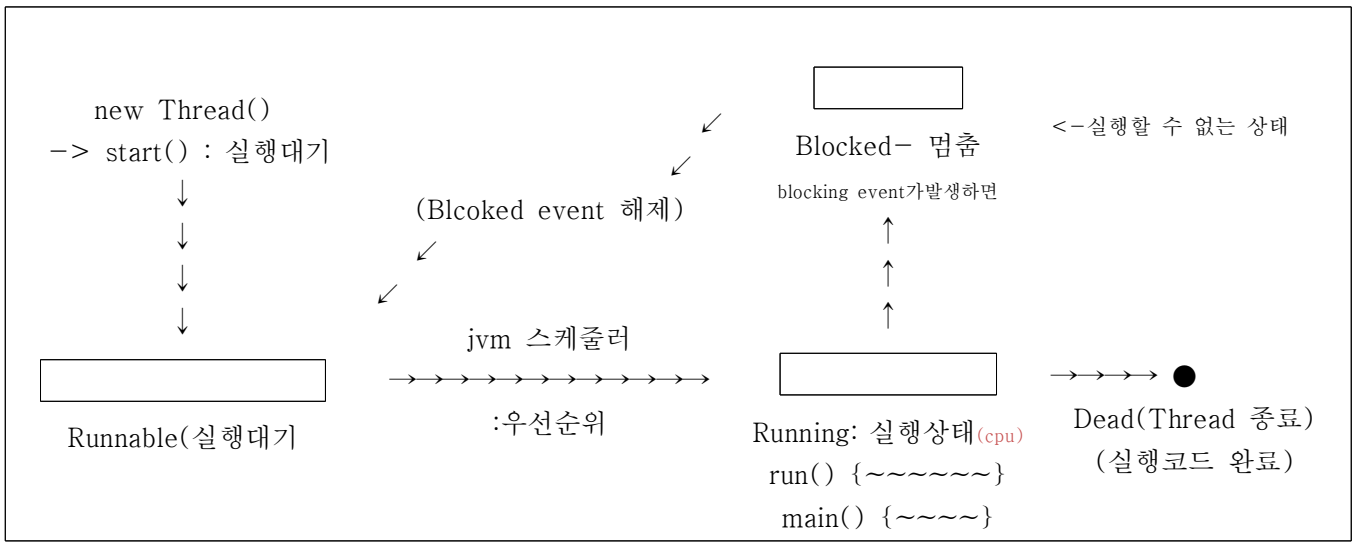
cpu는 여러 개의 프로그램을 동시에 실행하는 것이 아니라 돌아가면서 여러개를 쪼개고 쪼개서 처리하는 식으로 실제 한 시점에서 실행되고 있는 프로그램은 하나이다.

- Thread 스케줄링 : 멀티 스레드 사황에서 실행 대기 중인 Thread들 사이의 실행순서를 정하는 것.
  → O / S 담당 : java는 jvm이 담당 한다



- 시 분활 방식 - 동일한 시간만큼 돌아가면서 수행하는 방식.
- 우선순위 방식 - 각 Thread 마다 우선순위를 주고 우선순위가 높은 Thread가 먼저 실행 되도록 하는 방식
  → 우선순위가 높으면 실행될 확률이 더 높다.

**jvm도 우선순위 방식임



- Thread 제어
  Sleep(long 밀리초) - static

  Thread.slepp(1000) : 1000밀리초(1초)동안 멈춘다. -> Blocked 상태가 됨

(7주차)

15.08.21(금)

thread

sleep

- Thread.yield() - Running 중인 Thread가 Runnable 상태가 된다.
 -> Sleep은 block상태로 가서 지정된 time동안 실행이 불가능하지만 yield는 raunnable상태로 보내서 다른 쓰레드가 실행 될 수 있게 양보해주는 개념

- Thread객체.join() 현재 실행중인 Thread가 특정 thread가 dead가 될 때 까지 blocked 상태가 된다.
 join(숫자) : 숫자 만큼 기다려주는 것 만약에 10000(10초)를 넣었는데 1초만에 끝났다면 바로 실행하고 10초가 지나도 안 끝났으면 실행

- Thread 우선순위 낮다(1) ⟷ 높다(10)
- Thread.MAX_PRIORITY : int - 10
- Thread.NORM_PRIORITY : int - 5 (기본 우선순위)
- Thread.MIN_PRIOIRITY : int - 1

- 우선순위 설정 - t객체.setPriority(int 우선순위)
- 우선순위 조회 - t객체.getPriority() : int

동기화

하나의 데이터를 여러 군데서 활용 시 데이터를 가져올 때 문제가 발생할 수 있다.

한 쓰레드에서 클래스로 호출 한 값을 리턴 받기 전에 쓰레드가 RA상태로 들어가고 다른 쓰레드가 클래스에 호출해서 값을 받아가고 먼저 호출한 쓰래드가 RN이 된다면 클래스로부터 받는 리턴값 이 달라질 수 있다.

Synchronized(객체){ *객체는 this 또는 해당 객체를*
실행코드
}

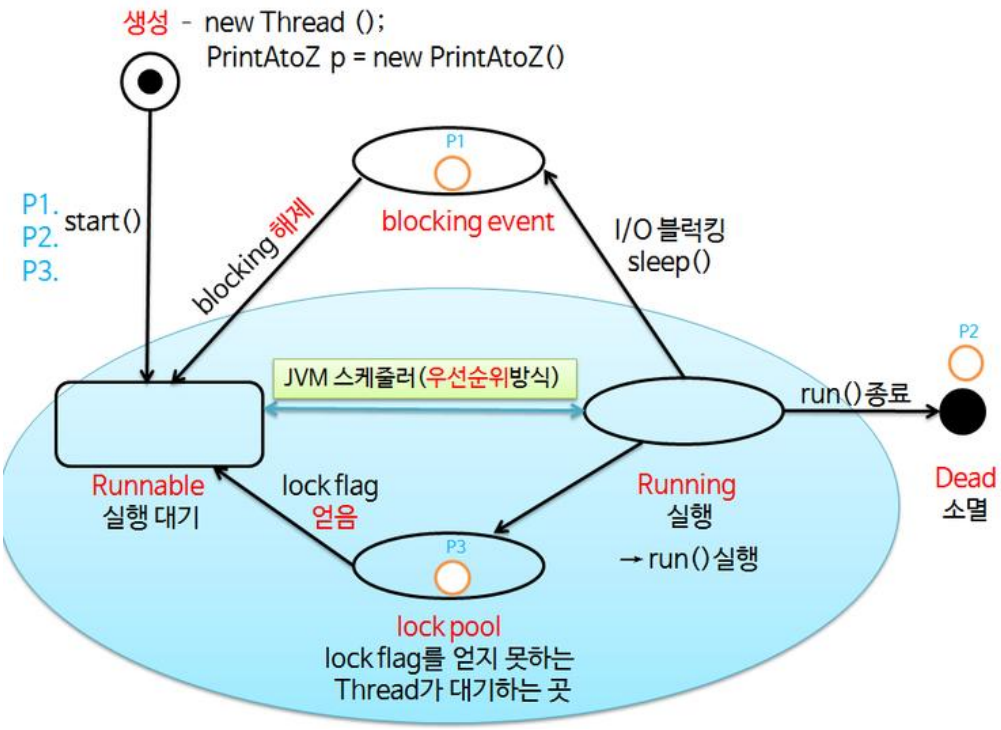
-> *실행코드*를 실행하기위해서는 객체의 lockflag가 필요하다.

** lockflag가 없는 thread는 얻을 때까지 대기.

Shrar OBJ	

method(){
 	
synchroonized(해당하는 객체){
 	
(rockflag를 쓰레드에 줌
그 쓰레드가 끝났을 때 다시 돌려 받음)
즉 rockflag를 돌려 받지 않은 상태에서 다른 쓰레드가 현재의 클래스 메서드의 실행코드를 실행안함
 	
sleep(10000)
 	
}
}
	(쓰레드들)t1,t2,t3
	
	t1
	so.method()
	RA-RN rockflag를 받음
	받고나서 RA가 되더라도 rockflag를 돌려주지 않고 대기 쓰레드에서 해당 메서드가 끝나야 돌려줌
	
	t2
	so.method()
	해당 so.method가 rockflag가 없다면 실행하지 못함
	
	t3
	so.method()

프로세스 코드와 DATA가 있는데 이것들이 실행되는 것이 쓰레드



(8주차)15.08.24(월)채팅 프로그램

(8주차)15.08.25(화)pool - 동일 종류의 객체들을 모아서 관리
awt ui 챕터는 안드로이드할 때..