181112

//eclipse 처음 설치할 때

바탕화면은 사실 경로가 되게 길어

그냥 c드라이브, d드라이브, 바로 만들어 놓는 게 좋음.

프로그래밍 관련 파일 또는 폴더들은 그냥 드라이브 바로 밑에 만들어 놓는 게 좋음.

만약 컴파일이 안 돼. 자바가. 왜일까. 뭐, 버전이 다르다든가, 경로 위치가 다르다든가, 그러면 폴더 이름이 바껴 버리고 경로를 잃어버려서 컴파일러 파일을 못 찾아.

설정 위치는 상단 설정 탭에서 Window - Preferences - Java - Installed JREs

여기서 JDK 파일 경로 지정. JDK 지정하면 컴파일러는 위치는 알아서 찾음.

//General - Appearance - Colors and Fonts - Basic - Text Font

//인터페이스에 떠 있는 네모 칸들, 탭들을 view 라고 함. 이것들 끄거나 키는 건

설정 탭에서 Window - view 에서 올리거나 내릴 거 선택 또는 기본 커스텀으로

설정 돼 있는 세트도 있음. 그건 Window - perspective - open perspective 에서

//자바 소스 만들기 -

Project Explorer 탭에 마우스 대고 우클릭 - new - project 만들고, 프로젝트 안에서 다시 src에 마우스 대고 우클릭 - new - class 클래스를 만들자는

workspace 폴더 명 - 프로젝트 폴더 명 - src에는 본 소스 파일 /bin에는 컴파일 된 class 파일

//단축키 목록 - ctrl + shft + l

//같은 파일 이름이나 클래스 이름으로 생성하지 못함. 그치만 경로가 다르면 됨. 예를 들어 같은 클래스 이름을 쓰더라도 프로젝트가 달라, 또는 프로젝트까지는 같아도 패키지가 달라, 그러면 생성 가능. 그래서 보통 회사에서 일 할 때는, 도메인 이름을 뒤집어서 뒤에서부터 씀.

예를 들어 abc.co.kr 에서 mygame 이라는 프로젝트를 진행 중이고, 그 안에 view 패키지에서 작업을 하고 있으면, 다음과 같음.

kr.co.abc.mygame.view.UserMenu

//

Refactor - 이거 끌어다 쓰거나 기타 연관된 다른 애들에까지 다 찾아서 바꿔줌.

//인스턴스 변수 / 멤버 변수

// 'A' < name && name < 'F' 이런식으로 해도

따옴표 안의 문자를 저절로 인트로 변환해서 인식.

(int)'A' < name && name < (int)'F'

굳이 이렇게 강제 변환 하지 않아도 됨.

181113

//패키지 간, 클래스 간, 메서드 간 호출 방법 모르겠

//s1.setName() 변수값을 변경하는 메소드

String s = s1.getName(); 변수값을 읽어주는 메소드

공개 메소드를 public으로 만들어서 간접적으로 private에 넣어줌.

pubilc void setEng(int e) //받아와서 대신 넣어주는 메서드

｛ //뭐 필요한 거 있으면 여기서 설정. 예)음수면 그냥 에러 or 종료

eng = e;

}

pubilc void getEng() //달라고 하면 대신 꺼내서 보내주는 메서드

{

return e;

}

//나중에 다른 라이브러리나 프레임워크에서 일하면 다 이런 관례

test => getTest or setTest

avg => getAvg or setAvg

work => getWork or setWork

이런 식으로 소문자로 시작하는 변수에 대해 get/set + 대문자로 시작하는 이름

(이런 식이다보니, 한자리 변수는 쓰지마. 에러.)

이렇게 해서 거의 다 getter / setter 만들어 놓음.

이클립스에서는 우클릭-source / alt + shift + s - getter,setter

에서 변수만 선택하면 자동 생성 가능

//project - package - class - method  
//package부터 선언, 그 다음에 import 도입, 그담에 class

//local 변수는 선언만 하면 쓸 수 없어. 초기화 해야줘야 해. string은 일단 null로 해놓으면 컴파일은 가능하지만 출력되는 건 없대.

// String c= new String("abc"); //healp 에 c라는 주소에 abc라는 데이터 넣어 둠.

String d= new String("abd");

System.*out*.println(a == b);

System.*out*.println(c == d);

이러면 a랑 b랑 주소가지고 물어보는 거. c랑 d랑 주소가지고 물어보는 거.

그 안에 내용은 아직 건들지도 않음.

//System.out.println(c.replaceAll("Java", "C");

이렇게 하면 모든 Java라는 글자를 찾아서 C로 바꿔줌.

이 연산의 경우, 기존의 c라는 주소를 가진 heap 인스턴스 자체를 수정하는 게 아니라, 새로운 heap을 만들고, 새로운 주소를 만들어서, 거기에 수정된 사항을 반영해서 출력함. 그리고 다음 연산 넘어가면, 새로 만들었던 애는 버려짐.

만약에 기존의 c를 바꾸는 연산을 진행해서 수정을 가하면, c를 수정하는 게 아니라 새로운 heap을 만들어서 c라는 주소에 덮어 씌움.

이런 식의 연산이다 보니, 많이 수정하는 코딩의 경우는 string 쓰지 않고 다른 형식으로 우선 진행해서 string으로 바꿈.

컨트롤 슬래스 or 컨트롤 쉬프트 슬래시

//일단 배운 거 까지 하자면 다른 패키지 꺼는 못 부름.

같은 패키지 다른 클래스 녀석을 불러 보자. 다른 클래스에서 지정해 놓은 형식의 변수를 만들고 싶을 때. 말하자면 새로운 변수의 이름을 가지는 인스턴스를 특정 형식으로 ‘선언’.

(클래스 이름) (여기서 쓰고 싶은 변수 이름) = new (클래스 이름);

예를 들어

BookList aaa = new BookList();

이거는 우리가 기존에 선언할 때랑 똑같애.

(자료형) (쓰고 싶은 변수 이름) = (넣고 싶은 값);

int aaa = 3;

int[] bbb = new int[10];

이것처럼. 앞에 new는 heap에 인스턴스로 만들어 달라는 뜻인 거고.

new 이후에 넣는 애들은, 만들 때 필요한 값. 같이 넣어줘야 하는 값. 예를 들어

배열 만들려고 할 거면, 그냥 자료형이랑 변수 이름만 선언할 거면

int[] aaa;

이렇게 하면 되겠지만 heap 에 생성까지 하려면 괄호 안에 필요한 정보(크기)까지 같이 넣어줘야 하잖아. 예를 들어

int[] aaa = new int[10];

이렇게 크기까지 같이 알려줘야 heap에 생성하든 말든 하잖아.

그거랑 똑같아. 클래스에서 지정한 형식으로 변수 선언하려면

클래스이름 변수이름 = new 클래스이름(필요한 변수);

이제 그 후에, 그 클래스에서 지정한 형식의 여러 값들을 넣어주는 거야.

int[] aaa = new int[10]; 이렇게 선언하고 나면

int[0] = 1; / int[1] = 2; / int[2] = 3; / 이런 식으로 하잖아.

BookList aaa = new BookList();

하고 나면

aaa.bookTitle = "Little Price"; / aaa.bookNum = 163261; / aaa.bookPrice = 12000;

이렇게 그 인스턴스가 갖는 값들을 하나하나 넣어주는 작업이 뒤따르는 것.

변수이름.세부형식이름 = 넣을값;

181114

//substring(a) / substring(a,b)

이러면 똑같은 기능을 하기는 하지만, 서로 다른 메서드야. 매개변수의 형식이나 갯수나 다르면, 같은 이름으로 다른 메서드를 만들어 놓는 거를 허용. 원래대로 하자면 에러지.

//Integer.parseInt(str aaa) - int 출력

aaa에 입력된 string 문자열을, 숫자로 변환해서 출력.

// scanner. next(); /탭이나 엔터, 스페이스바 등 화이트스페이스 전까지만 가져 옴.

scanner. nextLine() / 엔터 전 까지 다 가져옴.

// 그냥 charAt(7) == '-' 해도, 알아서 ‘-’를 알아서 int형으로 바꿔서 비교해 줌.

or 입력 받은 문자열에서 ‘-’를 찾아서, 그게 7에 있냐? 이래 물어봐도 되지.

//Student()하고 {}가 등장하면, 아 , 메서드구나, 인식.

//생성자를 선언해서 그 안에 기본 값으로 뭐 넣어 두면 생성할 때마다 그 값으로 초기화되는 값을 가지는 생성자 계속 나와

//생성자 메소드- 리턴 타입이 없음. 없다고 해서 void 쓰는 것도 안 돼. 그냥 리턴 타입 자체가 없어. 이름은 클래스명과 똑같이. 대문자까지 똑같이. 주로 하는 일은 초기화.

//이름, 국어점수, 영어점수, 수학점수 변수를 각각 갖는 생성자 Student에 대해,

Student s1 = new Student("홍길동“, 100, 100, 100);

이렇게 가능. 내 말 맞지. 배열 생성하는 거랑 똑같아.

자료형이름 앞으로쓸변수이름 = new 생성자이름(생성하면서필요한정보);

// Class 는 일단 상위 항목. 그 아래에 하위 항목인 메소드가 존재하겠지.

그 메소드의 일종으로서 생성자가 존재하는 것. 메소드 중에

뭐를 입력 받아서 특정 연산을 수행한다거나, 뭔가를 연산해서 출력한다거나,

뭔가 특정 형식을 갖는 객체를 생성하거나. 이것들이 메소드의 일종.

따라서 클래스 안에 속하는 것은 맞음. 다만, 특이한 기능을 하다보니,

이름은 클래스명과 똑같아야만 하고, 리턴값 자체는 공란으로 둬야 하고.

그래야 생성자로 인식.

//오버로딩 - 같은 이름의 생성자를 여러 개 생성. 근데 뭔가는 달라야 함.

받는 자료형의 종류나 갯수가 달라. 따라서

Student("홍길동“,100,100) / Student(100,100,100) / 이렇게는 선언이 되겠지만

Student(100,100)을 어떨 때는 영어,수학에, 어떨 때는 국어,수학에 넣는 식으로는 안 되겠지.

//해당 생성자의 여러 값들을 print해주는 메소드를 만들 거야.

System.out.print(name); / System.out.print(kor); / System.out.print(eng);

그냥 이렇게 하면 됨. 왜냐하면 해당 메소드에 같은 이름의 로컬 변수가 없어. mat 라든가

name 이라든가. 따라서 멤버 변수를 가져옴. 만약에 해당 메소드에서 멤버 변수와

같은 이름의 로컬 변수를 사용할 경우, this. 붙여서 멤버 변수를 지칭하고

안붙이면 로컬 변수고 하는 거지. 그런 용도지.

181115

//변수가 없는데 다른 방식으로 생성자를 여러 개 오버로딩 해놓으면,

더 이상 기본 생성자로 객체를 생성하지 못해. 다른 메소드가 사라져.

//한 메서드 안에서 같은 클래스 안에 있는 다른 메서드를 부르려면, this();

이건 바로 위에 있는 메서드 부르는 거. 괄호 안에 없을 때.

생성자 메소드에서 가장 첫줄에 한 번만, 하나만 부를 수 있음.

this(kor,eng,mat); 이런 식으로 위에 있는 생성자 메소드를

다른 데에서 쓸 때랑 똑같이, 괄호 안에 넣어줄 것들 똑같이 넣어 줌.

//Math 클래스도 클래스고 여러 기능을 가진 메소드가 존재. 그런데 객체를 만드는 게 아님.

만들 필요 자체가 없음. 따라서 객체 못 만들게 만들려고 해. 그럼 생성자 메소드의 접근지정자를 private으로 해놔. 그냥 놔두면 public으로 생성자 만들어지니까,

일부러 private으로 만들어 둠.

//생성자를 아무 것도 안 만들어 놓으면 기본 생성자 자동으로 생성.

생성자 하나라도 만들어 놓으면 기본 생성자는 안 만들어짐.

// final 붙이는 곳 - class 앞에 or 메소드 앞에 or 변수 앞에.

class 에 붙은 final - 상속 금지. / 변수에 붙은 final - 정의값이 변하지 않음

/method 앞에 final - 상속 받은 메소드를 수정해서 쓰는 것(over riding) 금지.

class는 상속 가능한데 메소드는 고쳐쓰지 마라 or

//static 은 고정, 안정된, 이라는 뜻. 뭐가 안정되고 고정 돼? 영역이. 인스턴스 자체가.

이거는 모든 메소드에서 공유하는 변수. 생성자 통해서 인스턴스를 만들 때,

그 생성자 메소드에 미리 설정해 놓은 대로 변수들의 자리를 만들어 주잖아.

그러니까, 그 인스턴스를 만들고, 그 인스턴스의 영역 안에 세부 변수들의 자리를

만들어 주는 거잖아.

근데, static은 모든 객체가 공유하는 값이기 때문에, 생성하든 안 하든,

생성하기 전이든 후든, 이미 가지고 태어나는 거야. 생성자가 만들어내는 인스턴스가 있고,

그 안에 포함돼 있는 변수가 아니라, 다른 인스턴스들과는 독립적으로,

이미 고유한 영역을 가지는 인스턴스로서 태어남. 그 주소를 가지고 모든 다른 메소드가

가져다 쓰는 거야. 원형 식탁(class)에 여러 명의 사람(mothod)이 있을 때,

각 사람이 자기 앞접시를 가지고 있기도 하지만, 동시에 원형 식탁 가운데에

메인 디쉬가 있는 것처럼. 누구나 손 뻗을 수 있는 곳에, 독립적 영역을 가지고,

활용, 수정할 수 있는 변수.

한 메소드에서 static 변수를 수정하면, 다른 메소드에서도 해당 static 변수가 수정 됨.

즉, 소통이 가능. a++ 이런 식으로 정의해 놓으면, 그 메소드를 읽을 때마다 증가 =>

객체가 몇번 생성 됐는지 알 수 있음

// Student[] sa = new Student[10] 이것 만으로는 Student 인스턴스가 1도 안 생김.

sa라는 배열에 담겨 있는 각 방의 값들은, 레퍼런스야. 그니까 주소야. 각 인스턴스들을

가리키는 ‘주소’가 10개 생겨 있는 것임. 실제로는 10번 일일이 인스턴스를 생성해 줘야 해.

주소만 미리 만들어 놓는 거지. 인스턴스 주소들의 배열은 만들어 놓고, 정작 생성은 아직

하지 않았으면, 그 주소들은 0 초기화 돼있어서 null 값이 돼 있음.

sa는 이제 배열 자체의 주소임. sa.length 만 가능하지, sa.kor 는 불가능. 이거는 이제

Student 의 변수 이름이 아니야. sa[5].kor 이런 식은 가능하지.

//switch(x)

{

case 1 = continue g; break; //이거는 unreachable code 라고 에러 떠. 왜냐면,

case 2 = break; int m = 7; //이거 보면 break 이후에 코딩이 더 있잖아. 그거랑 똑같아.

case 3 = break; break; //이거 처럼 break; 이후에 뭔가 코딩이 더 있긴 있는 거 같은데,

} //내가 거까지는 못 가겠다. 그 앞에서 이미 뭔가에 걸린다.

그 뒤에 뭐가 있는지는 보지도 않고, 그냥 뭔가라도 있으면, 거까지 못 가겠다,

그런 의미에서, unreachable code라고 오류.

//score.view / score.manage / score.vo 등등, 연관된 프로젝트, 패키지, 클래스 등은

같은 의미 단위로 구분할 수 있도록 ~~~. 을 통일해 줘. 그리고 저게 보기에는

src 밑에 다 따로 있어도, 실제 저장된 위치의 폴더로 가보면, src 안에 score 안에

view와 mange와 vo가 각각 있는 것임.

//새로운 파일을 목록에 불러오기

이클립스 상단 옵션 탭 - file - import - 불러오고 싶은 파일 형태 선택 - browse

이렇게 하면 경로만 지정해 놓는 것. 파일 자체를 가져오려면, copy projects int workspace

만약 다른 프로그램이나 이클립스가 알아볼 수 없는 형태로 생성했을 경우, 그 소스 파일만 열어서 이클립스 형태로 다시 저장, 가져오기, 그것도 안 되면 진짜 어쩔 수 없어. 켜서 자바 소스만 복붙.

//혹시 환경 설정이 다른 경우, String부터 다 에러남. 못찾아. 그럴 경우 해결 - 그 에러난 프로젝트 우클릭 - properties - 이게 그 파일의 환경 설정임. 아마 그걸 처음 짠 사람의 컴퓨터에 알맞게 설정돼 있겠지. 우리 껄로 맞추자 -

왼쪽에서 java build path - 상단에서 libraries - 오른 쪽에서 edit