디버깅 단축키

1. [Ctrl] + [Shift] + [B] : 현 커서의 위치에 브레이크 포인터 설정/해제

2. [F11] : 디버깅 시작

3. [F8] : 디버깅 끝까지, [F6] : 한 줄씩 실행,   
[F5] : 한 줄씩 실행하되 함수일 경우 그 함수 내부로 들어감

21.03.16

Public class의 이름은 반드시 파일명과 같아야 함

#main() 함수

-프로그램의 시작 지점

-사용자가 프로그램을 시작하면 가장 먼저 main() 함수를 찾아서 실행시킨다.

-main() 함수의 범위는 중괄호 {}로 결정된다.

#자바의 데이터들

1. “”사이에 적는 것들 : 문자열(String🡪**char[]의 배열 형태**), 문자를 여러 개 쓸 수 있는 데이터 타입
2. ‘’사이에 적는 것 : 문자,문자 단 하나만 쓸 수 있는 데이터 타입
3. 숫자를 적는 것 : 정수(Integer)
4. 소수를 적는 것 : 실수(Float, Double)

#문자들은 실제로는 숫자값을 가지고 있다, 우리 눈에 보일때만 문자로 보이는 것이다.

; : 한 명령어가 끝났음을 알린다.

#Escape 문자  
 -특수한 기능을 가진 문자들  
 -그냥 출력하려고 하면 문제가 발생하는 특수한 문자들  
 -앞에 \(역슬래시)가 붙은 문자는 Escape 문자로 간주  
 -앞에 붙은 \와 뒤의 문자가 합쳐져서 하나의 문자로 인식  
 -\\ : Escape 문자의 기능을 이용하지 않고 그냥 출력하고 싶을 때

#변수(variable)

-데이터를 담을 수 있는 공간  
 -자바에서는 변수를 반드시 선언한 뒤에 사용해야 한다.  
 -변수 선언시에는 그 변수에 저장할 데이터의 타입을 정해줘야 한다.  
 -“”나 ‘’가 없는 모든 문자열 또는 문자는 변수이다  
 -같은 이름의 변수는 다시 선언할 수 없다  
 -변수의 선언(declare)  
 -선언한 변수 에 최초로 값을 대입하는 것을 초기화(initialize)

=의 의미 : 왼쪽의 변수에 오른쪽의 값을 넣어라 (값을 대입)

변수 타입  
정수- byte(1byte), int(4byte), short(2byte), char(2byte), long(8byte)  
이때 char 타입에 저장된 값은 기본적으로 문자이므로 코드값을 확인하고 싶으면 앞에 (int)를 붙이면 됨  
실수- float(4byte), double(8byte)🡪실수는 주로 그래픽 계산에 사용, 부동소수점 방식  
참/거짓-Boolean🡪참과 거짓을 표현하는 데에 사용되는 타입  
참조형 타입(클래스 타입)- String(문자열), 그 외 모든 클래스들…(대문자로 시작)

21.03.17

#변수를 사용하는 이유

1. 값을 한번에 변경할 수 있다.(값을 일일이 찾아다니면서 변경할 필요가 없어진다.)

2. 값에 의미를 부여할 수 있다.(변수의 이름을 통해 해당 값의 용도를 쉽고 빠르게 파악할 수 있다.)

#변수 작명 규칙(권장 사항)

1. 두 단어 이상을 이어붙인 변수명을 사용할 때 \_를 활용

2. 두 단어 이상을 이어부틴 변수명을 사용할 때 대문자를 활용

3. 변수명 첫 글자는 소문자

4. 프로그램 내에서 값이 변하지 않을 변수(상수)는 모두 대문자를 사용한다.

5. 클래스명 첫 글자는 대문자를 사용한다

6. 변수에 들어있는 값을 추측할 수 있는 적절한 단어를 사용하자

#변수 작명 규칙(필수)

1. 변수의 첫 번째 글자는 숫자가 될 수 없다(1st, 2nd..)

2. 변수명 사이에는 공백을 사용할 수 없다.

3. 특수문자는 \_와 $만 사용할 수 있다.

4. 이미 자바에서 사용하고 있는 키워드는 변수명으로 사용할 수 없다.

#콘솔에서 출력하기 위한 여러가지 함수들

1. print() - 줄을 자동으로 바꾸지 않는 가장 기본 출력

2. println() - 줄을 자동으로 바꿔주며 출력하는 함수(()에 전달한 데이터 맨 뒤에 '\n'을 추가하여 출력)

3. printf() - 서식 문자를 이용할 수 있는 출력 함수

-문자열을 +로 이어붙이는 것 보다 서식을 사용하는 경우가 편리할때도 있다.

-줄을 자동으로 바꿔주지 않는다.(\n을 따로 입력해야함)

-출력의 형태를 미리 만들고, 서식문자 자리에 값을 채워서 사용한다.

#printf()을 날때에 %d 자리에 숫자를 하나씩 채워서 사용한다.

ex.)System.out.printf("%d년 %d월 %d일 / %d:%d:%d.%d\n", year, month, day, hour, minute, second, millisec);

#서식 문자의 종류

%d : 해당 자리에 전달한 정수를 10진수로 출력

%x, %X : 해당 자리에 전달한 정수를 16진수로 출력

%o : 해당 자리에 전달한 정수를 8진수로 출력

%s : 문자열

%c : 문자

%f : 실수

#서식 문자 옵션들

서식문자 자리에는 d만 넣어야 하는 것은 아님

%숫자d : 숫자만큼 칸을 확보한 뒤 오른쪽 정렬하여 출력(System.out.printf("사과 : %10d개\n", 10);)

%-숫자d : 숫자만큼 칸을 확보한 뒤 왼쪽 정렬하여 출력 (System.out.printf("사과 : %-10d개\n", 10);)

%0숫자d : 숫자만큼 칸을 확보한 뒤 빈 칸에 0을 채워서 출력(System.out.printf("%03d개\n", 5);)-->005개)

%+d : 양수 앞에도 부호를 붙여준다(System.out.printf("양수 : %+d, 음수 : %d\n", 5, -5);-->+5, -5)

%.숫자f : 소수 자릿수를 설정(System.out.printf("평균 점수 : %.2f\n", avg);-->123.13)

#연산자(Operator)

-계산할 때 사용하는 것 +, -, \*, /...

-정수끼리 나누면 몫만 나온다

-정수와 실수를 나누면 정확한 값을 계산한다.

-% : 나머지 구하는 연산자

-Math.pow(a,b) -->a를 b제곱

-Math.sqrt(a)-->a의 제곱근

-Math.abs(a)-->a의 절대값

-Math.round(a)-->소수 첫째 자리에서 a를 반올림

-Math.ceil(a)-->소수 첫째자리에서 올림

-Math.floor(a)-->소수 첫째자리에서 내림

-Math.max, min(a,b)-->a,b 중 더 큰, 작은 숫자 택하기

# Math 클래스

-수학과 관련된 기능들이 모여있는 클래스(ex. abs, round, ceil..)

#비트 연산(%, |, ^, >>, <<)-->숫자를 2진법으로 바꿔서 연산

#import

-다른 패키지에 정의되어있는 클래스를 사용하기 위해서는 해당 클래스의 정확하 경로를 import 해야 한다.

-같은 패키지에 존재하는 클래스는 import 없이도 사용할 수 있다.

-같은 패키지라면 임폴트없이 사용가능

-->(자바이름).클래스이름(대입해야할 변수)

-해당 패키지의 모든 클래스를 불러오고 싶을 때는 \*를 사용한다.

#java.util.Scanner 클래스

-프로그램의 외부로부터 입력을 받을 수 있는 기능들이 모여있는 클래스

-어디에서 입력을 받을지 결정한 뒤에 다양한 값들을 입력받을 수 있다.

-콘솔로부터 입력받고 싶을 때는 System.in을 사용한다 -->Scanner sc = new Scanner(System.in);

-System.in으로부터 입력받은 새로운 스캐너를 하나 생성, Scanner타입 변수 sc에 생성한 스캐너를 담아놓는다.

#Scanner들의 메서드

-sc.nextInt(); : 다음으로 들어오는 정수를 하나 꺼낸다  
 -sc.nextDouble(); : 다음으로 들어오는 실수를 하나 꺼낸다.  
 -sc.next() : 다음 단어를 String타입으로 받아오기 (공백, \t 기준) 🡪 apple banana orange 🡪apple  
 -sc.nextLine() : 다음 한 줄을 String타입으로 받아오기(\n 기준) 🡪 apple banana orange 🡪apple banana orange

21.03.18

-final이 함께 선언된 변수는 나중에 값을 변경할 수 없다.(이때 변수명은 무조건 대문자로)

#비교 연산자

-비교 연산의 결과는 참 또는 거짓이다. (Boolean 타입)  
 -산술 연산과 비교 연산이 함께 있으면 산술 연산을 먼저 계산한다.  
 - == : 두 값이 같으면 true, != : 두 값이 다르면 true

#논리 연산자

* Boolean 타입 값으로 하는 연산
* && : 양 옆의 값이 모두 true일 때만 true (AND)
* || : 양 옆의 값 중 하나만 true여도 true (OR)
* ! : true면 false, false면 true (NOT)

\*문자열(참조형 타입)은 ==으로 비교하면 안된다.(equals로 비교!)🡪 str.equals(“yes)

#기본형 타입과 참조형 타입의 차이

-기본형 타입은 내부 데이터의 크기를 예측하기 쉽다.(차곡차곡 쌓이는 메모리) **Stack**   
<-> 참조형 타입은 내부 데이터의 크기를 예측하기 어렵다.(여기저기 막 쓰는 메모리를 사용한다) **Heap**

#타입 캐스팅

* 타입이 자연스럽게 변하는 경우 – 타입이 변하더라도 문제가 없을 때
* 타입을 강제로 변환시켜야 하는 경우 – 타입이 변하면 문제가 생길 때

#타입의 크기

* 정수 타입 : byte(1) < char, short(2) < int(4) < long(8)
* 실수 타입 : float(4) < double(8)  
  정수 타입은 무조건 실수타입 보다 작다

1. 자연스러운 타입 캐스팅  
   -\_int = \_byte, \_long = \_int;
2. 값이 손상될 위험이 있는 타입 캐스팅  
   -\_int = 1000; \_byte = \_int(오류)  
   -\_short = \_int🡪이때 \_short = (short)\_int로 강제 형변환해주면 가능
3. 같은 값이더라도 다르게 해석될 수 있는 타입의 경우 타입 캐스팅이 필요하다.  
   -int = 80, (char)a = P

21.03.19

# IF문

* ()안의 값이 true일 때 {}안의 내용을 실행한다.
* ()안의 값이 false일 때는 {}안의 내용을 무시한다.
* {}안에 들어갈 내용이 단 한 줄이라면 {}를 생략할 수 있다.

#else if문

* 위에 있는 if문이 실행되지 않았다면 if문처럼 동작한다.
* Else if는 단독으로 사용할 수 없다.
* Else if는 여러 번 사용할 수 있다.

#else문

* 위의 있는 if, else if가 모두 실행되지 않았다면 {}안의 내용을 무조건 실행
* If 바로 다음에 쓸 수도 있음
* Else가 붙어있는 조건절은 최소 한번은 실행된다.

#charAt

* “문자열”.charAt(index);
* 해당 문자열에서 원하는 번째의 문자를 char타입으로 꺼내는 함수  
  🡪System.*out*.println("Hello, everyone!!!".charAt(10));🡪r
* 첫 번째 문자는 0번 인덱스에 있다.
* 빈칸(스페이스)도 문자로 침

#length

* “문자열”.length();
* 함수를 실행하면 해당 문자열의 길이를 int타입으로 알려준다.

#switch-case문(파이썬에는 이거 없음)

* ()안의 연산 결과에 따라 실행할 코드를 결정한다.
* ()안에 Boolean타입은 들어갈 수 없다.
* Break를 안 쓰면 break를 만날때까지 내려가면서 모든 case를 실행한다.
* Default는 if문의 else 역할을 한다.
* if문으로 완벽하게 대체 가능하다  
  -🡪case 'E': case 'e':  
  System.***out***.println("전체 관람가");🡪조건을 나열해주면 됨

#삼항연산자

* 간단한 if문을 한 줄로 표현할 수 있다.
* 조건 ? 예 : 아니요;

#대입 연산자

* = : 왼쪽의 변수에 오른쪽의 값을 대입한다.  
  🡪int num = 10;  
  🡪num = num + 20;🡪num = 30;

#복합 대입 연산자

* 변수의 결과를 누적시키는 연산자  
  🡪num = 10;  
  🡪sysout(num +=15)🡪 num = num + 15 🡪25  
  🡪sysout(num -=3)🡪 num = num - 3 🡪22  
  🡪sysout(num \*=2)🡪 num = num \* 2 🡪44  
  🡪sysout(num /=4)🡪 num = num / 4 🡪11  
  🡪sysout(num %=3)🡪 num = num % 3 🡪2

#단항 연산자

* 변수에 결과를 1씩 누적시키는 연산자  
  🡪num = 10;  
  🡪num++🡪num = num + 1; 🡪11  
  🡪num--🡪num = num – 1; 🡪10
* ++, --의 위치에 따라 결과가 달라진다.  
  🡪num = 3;  
  sysout(num++)🡪3  
  sysout(num)🡪4  
  sysout(++num)🡪5  
  sysout(num)🡪5(--도 규칙은 동일함)
* ++, --를 뒤에 붙이면 해당 줄의 모든 명령을 실행한 후에 값이 변한다
* ++, --를 앞에 붙이면 해당 줄의 모든 명령을 실행한 후에 값이 변한다

21.03.22

# 반복문 (Loop)

* For, while, do-while(X) …
* 똑같은 코드를 여러 번 반복하고 싶을 때 사용한다.
* for(초기화자리; 조건; 증가값){  
   가운데 있는 조건이 true인 동안 반복될 명령어들을 적는 곳  
  }
* for문에 대해서 초기값, 증가값, 조건은 마음대로 변경할 수 있다.
* 초기값과 증가값의 위치는 마음대로 변경할 수 있다.  
  for(; x<10;)
* 중괄호 안에 선언된 변수는 중괄호가 끝나면 선언된 변수는 사라짐(변수의 생명주기)
* 반복문 내부에 다른 문법들도 자유롭게 사용할 수 있다.
* for문은 조건을 비워놓으면 무한반복한다.  
  -반복문은 원하는 타이밍에 break문을 통해 탈출할 수 있다.(break를 만나면 반복문은 탈출🡪이때 break가 if 안에 있어도 for 탈출)
* 반복문 내부에서 continue를 만나면 바로 다음번 반복으로 넘어간다.
* 반복문으로 총합 구하기  
  (1) 반복문 바깥쪽에 총합을 저장할 변수를 하나 선언하고 0으로 초기화 해둔다.  
  (2) 반복문으로 진행하면서 해당 변수에 값을 누적시킨다.

21.03.23

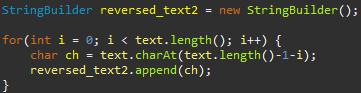
for(int i = 0, j = word.length()-1; i< word.length(); i++, j--)  
🡪동시에 변수 2개 선언해서 반복조건 가능

\*char를 합해서 string으로 만들 수 있음



#StringBuilder

* 문자열을 다시 조립하는 기능이 모여 있는 클래스  
  ex. 문자열을 거꾸로 생성해보기



* String 클래스는 누적연산이 매우 느린편이다.
* String 클래스로 잦은 연산을 하는 것은 비효율적이다.
* 즉 StringBuilder는 성능이 좋음

System.*currentTimeMillis*(); : 현재 시스템 시간을 밀리세컨드로 가져옴(long타입)

#InnerLoop(반복문 내부에 반복문 사용하기)

for (int i = 0; i < 10; i++) {

System.***out***.printf("========================바깥쪽 반복문 %d\n", i);

for (int j = 0; j < 8; j++) {

System.***out***.printf("-------------안쪽 반복문 %d-%d\n", i, j);

}

}

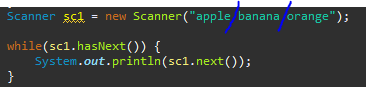
21.03.24

#while

* for문과 사용법이 약간 다른 반복문
* 초기값과 중가값의 위치가 정해져 있지 않다.
* ()안의 조건이 참인동안 {}안의 내용을 반복
* while문은 증가값의 위치에 따라 결과가 달라질 수 있다.

#while문과 사용하기 좋게끔 설계된 클래스들이 있음

* has, next등의 수식어가 붙은 함수(메서드)를 가지고 있다.  
  -sc.hasNext()🡪 다음 값이 있으면 true, 없으면 false  
  -sc.next() – 다음 하나의 값을 String 타입으로 꺼낸다  
  🡪apple, banana, orange



\*정수를 문자열로 바꾸는 방법들

* Sysout(“” + 123);
* Integer.toString(123)

\*문자(char)를 입력받는 방법

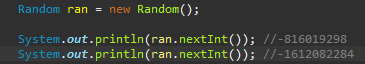
char alpha = sc.next().charAt(0);

21.03.25

#JAVA에서 무작위 숫자 생성하기

* Math.random()  
  -double 타입의 랜덤 소수를 생성한다.(0 <= x < 1)  
  -이 랜덤 소수를 이용하여 원하는 범위의 숫자를 만들 수 있다.  
    
  원하는 범위의 숫자를 만드는 순서(30~60)-31개  
  1. 생성된 랜덤 소수에 원하는 숫자의 개수를 곱한다.(31을 곱하면 0 <= x < 31 범위가 됨)  
  2. 원하는 숫자 중 가장 작은 숫자를 더한다(30)🡪(30 <= x < 61 범위가 됨)  
  3. 소수점 아래를 삭제한다🡪math.floor() 또는 int 타입 캐스팅

#Random 클래스  
-좀 더 편리하게 랜덤 값을 선택할 수 있는 기능들이 모여있는 클래스  
-Random ran = new Random();로 생성  
-생성할 때 랜덤 시드가 생성  
-원하는 랜덤 시드를 선택할 수도 있다.  
ex.)  
   
  
Random ran = new Random(11); 즉 ()안에 수를 넣으면 시드값이 정해져서 이 ran을 사용한 랜덤값은 고정임  
단 ()안의 값을 바꾸면 랜덤값은 바뀜(수를 안넣으면 무작위



* Random.nextInt(bound) 🡪0 ~ bound 미만의 정수를 반환



* Random.nextBoolean()  
  -Random 클래스는 원하는 타입의 랜덤 값을 쉽게 얻을 수 있다.  
  System.***out***.println(ran.nextBoolean() ? "앞" : "뒤");

21.03.26

# 배열(Array)

* 같은 타입 변수를 한번에 여러 개 선언하는 방법
* 같은 이름으로 변수를 여러 개 선언한 뒤 인덱스를 활용해 구분한다.
* Ex.) int[] score = new int[100];, score[0] = 90;, score[99] = 99;
* 배열도 배열.length를 통해 길이를 알 수 있다. score.length;
* 배열은 한번 크기가 정해지면 크기를 변경할 수 없다.
* 배열은 생성과 동시에 초기화가 되어있다.  
  (정수: 0, 실수: 0.0, boolean: false, 참조형: null)  
  🡪배열은 변수와 다르게 초기화가 자동으로 되어있다.
* String은 char형 배열로 변환할 수 있다.(.toCharArray())  
    
  🡪즉 msg3이랑 msg1이랑 같게 되어 msg3[0] = msg1[0]이다



#배열을 선언하는 방법

1. 타입[ ] 배열이름 = new 타입[배열크기];
2. 타입[ ] 배열이름 = {값1, 값2, 값3, .. };
3. 타입[ ] 배열이름 = new 타입[ ] {값1, 값2, 값3, ….};



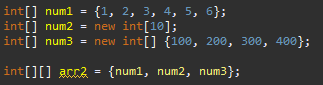
#문자열을 쉽게 대문자 또는 소문자로 변환하기

Text.toUpperCase();

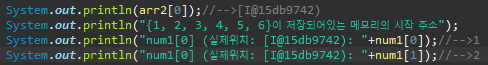
21.03.29

#배열 내부에도 배열을 넣을 수 있다.(2차원 배열)

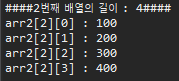
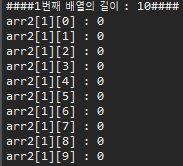
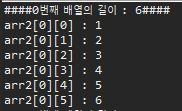
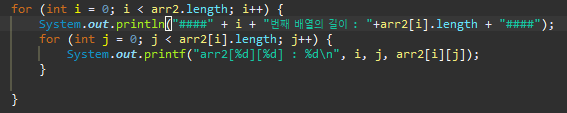
1. 2차원 배열의 값 사용하기  
     
     
     
   🡪arr2[ arr2안에 있는 인덱스 ][ arr2안에 있는 배열의 인덱스 ]



1. 배열 안에는 배열이 시작되는 메모리 주소가 있음



1. n차원 배열은 n중 반복문으로 모두 탐색할 수 있다  
     
   🡪



#Arrays.toString(배열의 이름) : 배열 편하게 출력하기

# row : 행, column : 열

**#함수(Function)-메서드**

* 기능을 미리 정의해두고 나중에 가져다 쓰는 것
* 자바에서 함수는 반드시 클래스 내부에 선언해야 한다.
* 정의해둔 함수 이름 뒤에 ()를 붙여서 함수를 호출하면 호출한 자리에 함수의 실행 결과가 반환된다.
* 함수를 정의할 때 해당 함수가반환할 값의 타입을 앞에 적어둔다.
* 나중에 재사용 할 가능성이 있는 기능들을 미리 만들어둠으로써 작업의 반복을 줄일 수 있다.
* 리터 타입 void는 해당 함수가 반환하는 값이 없다는 것을 나타낸다.

21.03.30

\*매개변수의 개수 혹은 타입이 다르면 같은 함수명을 사용할 수 있다(함수의 오버로딩)🡪function overloading

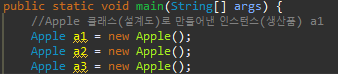
\*클래스 내부에 있는 함수를 메서드라고 부른다.(사실 JAVA에는 메서드밖에 업다)

**#객체지향 프로그래밍 (OOP : Object Oriented Programming)**

* 세상에 존재하는 모든 것들을 변수와 함수로 표현하려는 프로그래밍 방법론  
  ex:사과  
  -사과의 변수: 크기, 색깔, 당도…  
  -사과의 메서드: 던진다(크기 만큼의 데미지를 입힌다), 먹는다(먹은 만큼 칼로리가 올라가고 사과의 크기가 줄어든다)
* 변수와 함수를 무분별하게 사용하기 보다는 현실에 존재하는 객체 단위로 묶어서 표현하여 가독성을 올린다
* 객체 (Object) : 세상에 존재하는 모든 것을 말한다.(사물, 개념 …)

**#클래스 (Class)**

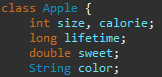
* 객체를 프로그래밍 언어로 표현한 것
* 클래스는 객체의 설계도이다 (정의한 시점에서는 실체가 없음)
* 클래스를 통해 생성하는 실제 객체를 ‘인스턴스’라고 부른다



* 클래스 명은 대문자로 시작한다.
* **클래스는 참조형 변수 타입이다**
* 프로그램 실행 시에 public class에 있는 main만 찾을 수 있다.  
  (public이 안붙은 class에 main을 해놓으면 찾을 수 없음)
* 클래스는 우리가 만든 타입

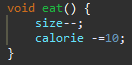
#인스턴스 변수

* 각 인스턴스마다 다른 값을 가질 수 있는 변수
* aka 필드, 속성, 멤버 변수, 상태…  
  를 지칭함



#인스턴스 메서드

* 인스턴스 변수를 활용하여 해당 인스턴스의 상태를 변화시키는 메서드  
  (Apple 클래스 내부에 있는 메서드임)



**\*다른 패키지에서 가져다 쓰고 싶을때는 public을 붙여야 한다.**

**#클래스의 생성자 (Constructor)**

* 클래스의 이름과 똑같은 이름의 **메서드**
* **메서드지만 리턴 타입을 정의하지 않는다.**
* 이 클래스의 인스턴스가 생성할 때 가장 먼저 호출되는 메서드
* 생성자는 가장 먼저 호출되기 때문에 주로 초기화 용도로 사용됨
* **인스턴스를 생성하기 위해서는 new를 통해 생성자를 호출해야 한다.**
* 생성자를 하나도 정의하지 않은 클래스에는 기본 생성자가 자동으로 생성된 기본 생성자가 존재한다.
* 생성자가 모두 실행된 후에는 생성된 인스턴스의 주소가 반환된다(sc에 Scanner 주소가 반환되서 sc.nextInt등을 쓸 수 있음)
* 생성자도 메서드이기 때문에 오버로딩 할 수 있다(대입하는 매개변수가 달라지면 메서드 이름이 같아도 된다는 의미)
* 필드를 초기화하는 기능을 한다.
* 객체가 생성될 때 호출된다.
* 하나 이상의 생성자를 만들었다면 기본 생성자는 생성되지 않는다  
  생성자를 호출하는 행위들



\*this: 각 인스턴스에서 자기 자신을 가리킨다.(지역변수와 멤버변수를 구분하고 싶을때)

\*생성자를 오버로딩한 클래스는 다양한 생성 방식이 생긴다

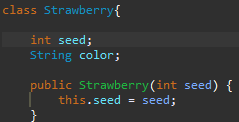
**클래스 및 생성자에 대한 설명**



\*클래스의 인스턴스 변수도 초기화를 하지 않으면 배열과 똑같은 초기값을 지닌다.

\*다른 오버로딩 타입의 생성자가 존재할때는 기본생성자를 자동으로 생성해주지 않는다.

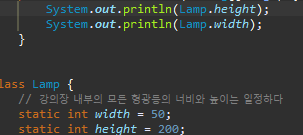
🡪이러면 Strawberry(){}라는 기본 생성자는 없는거임(하고싶으면 따로 생성하면 됨)



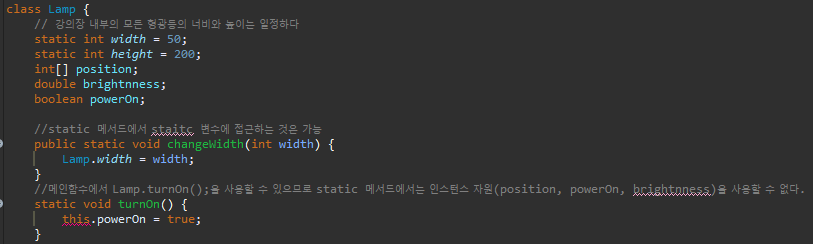
**기본 생성자 : 매개변수를 받지 않는 생성자**

**#static(정적 영역, 클래스 영역 <-> 인스턴스 영역)**

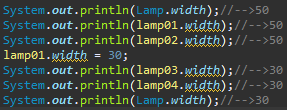
* Static은 같은 클래스의 모든 인스턴스가 공동으로 사용하는 영역🡪같은 곳을 바라보는 거임  
  이게 가능



* 앞에 Static이 붙은 클래스의 자원은 모든 인스턴스에 같은 값을 지니게 된다.
* Static 영역은 클래스당 하나밖에 생성되지 않으므로 메모리가 절약
* Static 영역은 클래스당 하나이기 때문에 클래스명에 .을 찍고 접근하는 것이 좋다.
* 인스턴스가 존재하지 않을 때도 static이 사용될 가능성이 있기 때문에 static 메서드에서는 인스턴스 자원을 사용할 수 없다.
* Static 영역의 자원은 인스턴스가 생기기 전에도 호출할 수 있다.-->아직 생성된 인스턴스가 없더라도 static 자원을 사용할 수 있다.  
    
  즉 이러하다



**또한 이게 가능 매우 중요!**



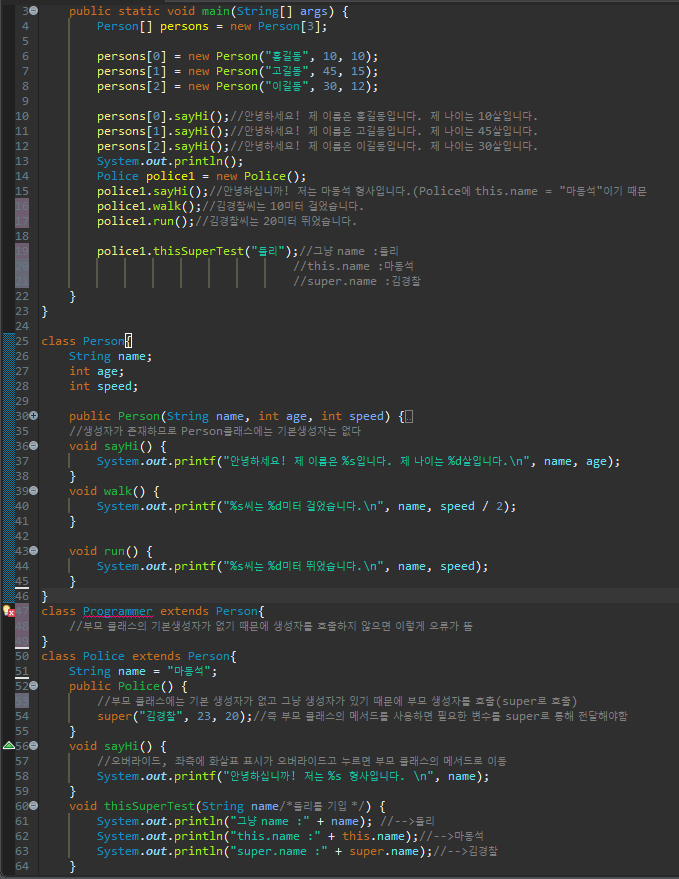
21.03.31

**#클래스 상속**

* 이미 존재하는 클래스를 그대로 물려받거나 고쳐서 사용하는 것(static으로 선언된 것도 상속 가능)
* 상속해주는 클래스를 **부모 클래스** 또는 **슈퍼 클래스**라고 한다.
* 상속받는 클래스를 **자식 클래스** 또는 **서브 클래스**라고 한다.
* 상속받은 클래스는 부모 클래스의 생성자를 호출해야함  
  단 부모 클래스의 기본 생성자가 존재할 시 호출을 생략해도 됨
* **자식 클래스의 생성자에서는 첫 줄에 super()를 반드시 호출해야 한다(필수)**
* Super(): 부모 클래스의 생성자를 실행
* 자식 클래스에서 생성자를 정의하지 않을 시 자식 클래스에도 기본 생성자가 생기고 부모 클래스의 기본 생성자를 사용한다
* Final을 붙인 메서드는 자식 클래스에서 오버라이드를 금지시킨다
* **부모 클래스가 들어가야할 것들에 대해서 상속받은 클래스가 대신 들어갈 수도 있다!(\*상속을 받는 큰 이유 중 하나)**

1. 상속받은 클래스는 생성자에서 가장 먼저 부모의 생성자를 호출해야 한다.  
   부모 클래스에 기본생성자가 존재하면 생략할 수 있지만(**super() 생략 가능**), 존재하지 않을때는 생략이 불가능하다.  
   **자식 클래스의 생성자에서는 첫 줄에 super()를 반드시 호출해야 한다(필수)**
2. 자식 클래스는 부모의 메서드를 마음대로 고쳐서 사용할 수 있다 **(오버라이드, Override)**
3. 부모 클래스와 자식 클래스에 똑같은 이름을 가진 자원이 있는 경우 super와 this로 정확하게 구분할 수 있다.

**상속에 대한 설명**



**이때 Police클래스의 생성자가 발동되면 super를 통해 부모클래스인 Person의 생성자가 실행되는거임(단순히 값 대입이 아님)**

**#객체의 다형성**

* 객체는 다양한 형태를 지닐 수 있다.  
  - 사과는 사과이기도 하고 과일이기도 하지만 과일은 과일이기도 하지만 사과는 아니다
* 자식 클래스는 부모 클래스가 될 수 있다.  
  -자식 클래스 타입이 부모 클래스 타입이 되는 것을 **업캐스팅**이라고 한다  
  -업캐스팅된 객체는 자식 클래스에만 존재하는 것들을 모두 잊어버린다.  
  -오버라이드된 메서드는 자식 클래스에서 구현한 내용이 실행된다.
* 부모 클래스는 자식 클래스가 될 수 없다 **(다운 캐스팅)**  
  -예외적으로 업캐스팅되었던 인스턴스는 다시 원래 타입으로의 **다운캐스팅**이 가능하다
* 객체의 다형성을 쓰는 이유는 수정하기 용이하고 요지보수에 쉽게 때문이다

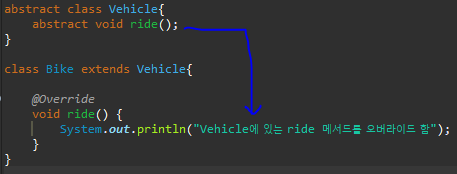


**#추상 클래스**

* 완성을 자식 클래스에게 미루는 클래스
* 클래스 내부에 추상 메서드를 지닌 클래스는 앞에 abstract 붙여 표시
* **인스턴스 화가 불가능하다**(완성 안된 메서드가 존재하기 때문)

**#추상 메서드**

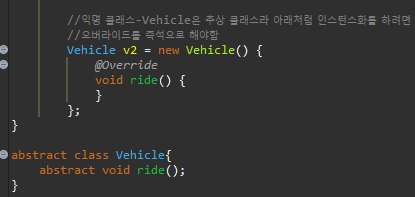
* 함수의 형태(리턴 타입, 접근 제어자, 매개변수)만 정해놓고 자세한 구현은 자식 클래스에서 오버라이드
* 자식 클래스에서 오버라이드 하지 않으면 컴파일 에러가 발생한다.



21.04.01

#익명 클래스(Anonymous class)

* 추상 클래스는 인스턴스화가 불가능한데 즉석으로 오버라이드 시키는 거임



21.04.02

this(): 기본 생성자, this(~~): 생성자

코드 짤 때 메인 함수에 부모 클래스를 불러와서 코드를 짜기!(자식 클래스에서 사용되는 메서드는 부모 클래스의 빈 메서드로 두고 오버라이딩!)

**#접근 제어자 (Access Modifier)**

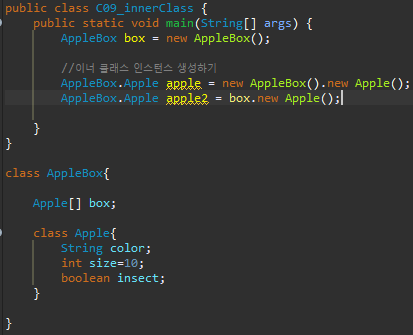
* 다른 클래스가 해당 클래스의 자원을 사용하려고 할 때 어디까지 허용할지를 설정한다
* 접근 제어자를 통해 구분할 수 있는 거리의 종류  
  -같은 클래스 내부의 다른 자원(가장 가까운 거리)  
  -같은 패키지 내부의 다른 자원  
  -다른 패키지에 있지만 상속받은 자원  
  -다른 패키지의 자원(가장 먼 거리)
* 종류  
  -private : 앞에 private이 붙은 자원은 같은 클래스 내부에서만 접근할 수 있게 된다.  
  -default(아무것도 안 씀): 같은 패키지 내부에서는 자유롭게 접근 가능(다른 패키지는 접근 불가능)  
  -protected:앞에 protected가 붙은 자원은 같은 패키지 내부에서는 자유롭게 접근할 수 있다.   
  단 다른 패키지에서는 상속받은 클래스만 접근할 수 있는 자원이 된다.  
  -public: 다른 패키지에서도 자유롭게 접근할 수 있는 자원이 된다

21.04.05

Arrays.sort : 배열을 정렬함

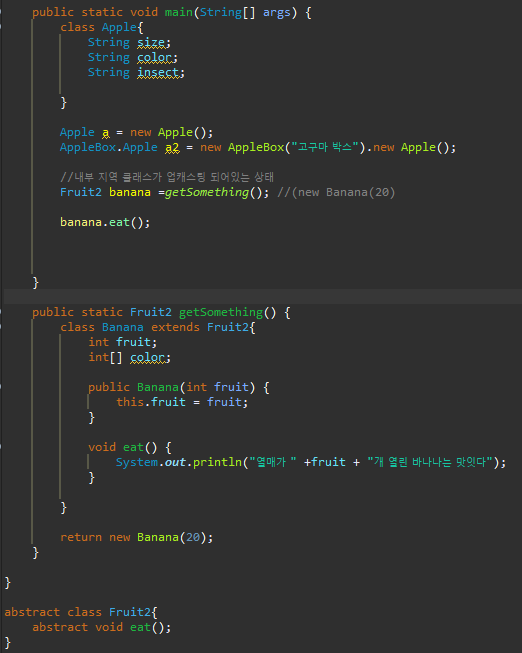
**#클래스 내부에 클래스 사용하기(InnerClass)**

* 클래스 내부에도 클래스를 정의하고 사용할 수 있다
* 클래스 내부에 존재하는 클래스는 인스턴스가 생성되어야 사용할 수 있는 설계도이다.



**#지역 내부 클래스**

* 메서드 내부에서 선언하여 사용하는 클래스
* 메서드가 끝나면 해당 클래스도 수명이 다한다



**#익명 내부 클래스**

* 클래스에 이름을 짓지 않고 구현한 다음 바로 사용하는 방식
* 일회용이다.

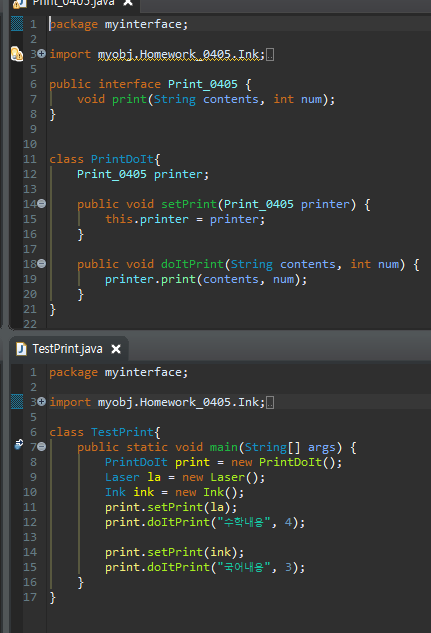
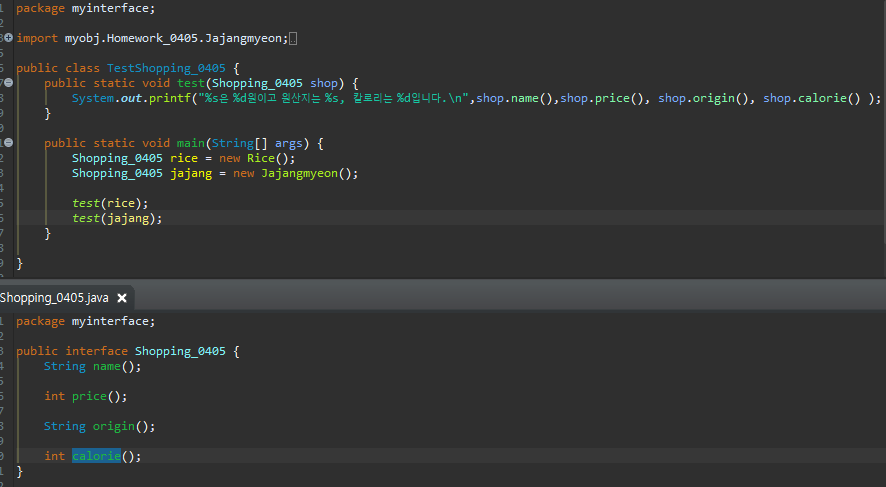
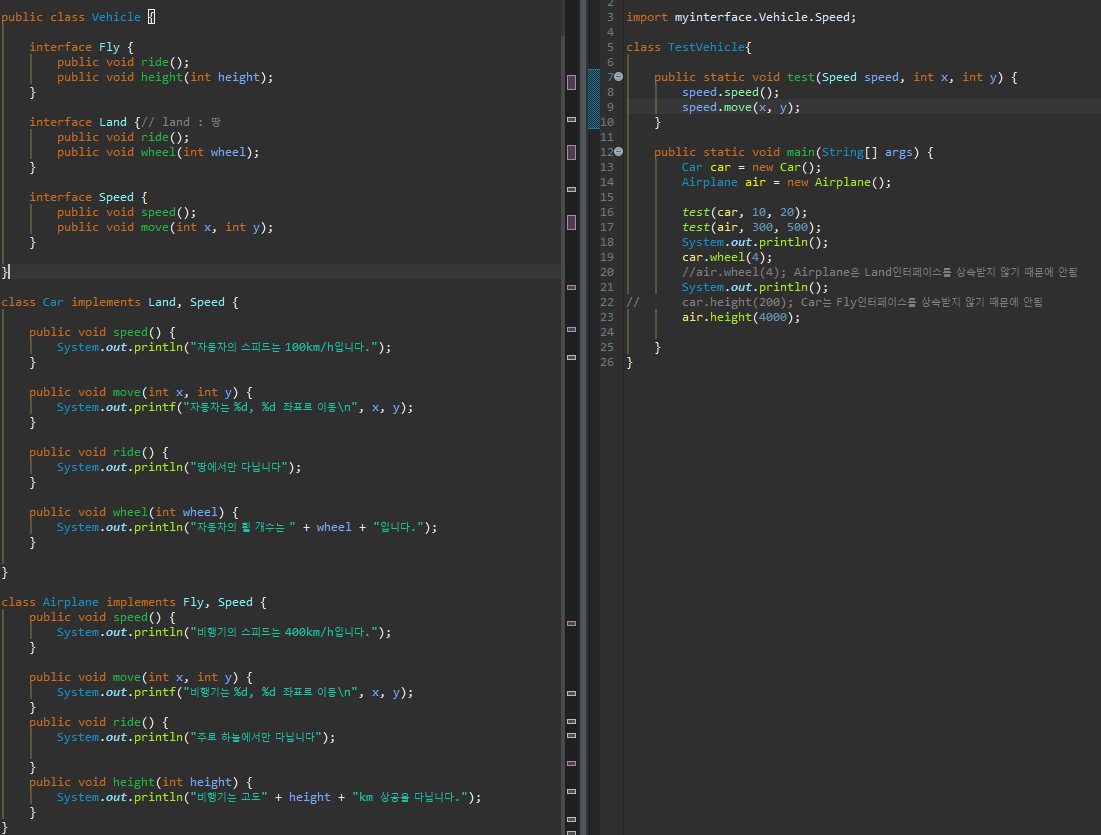


* 인스턴스화 할 수 없는 클래스도 익명 클래스를 통해 즉석으로 사용할 수 있다.



**#인터페이스 (Interface)**

* abstract class(추상 클래스)는 상속을 여러 개 받을 수 없다는 단점을 보완한 것  
  (일반 클래스도 상속을 여러 개 받을 순 없다 그러나 일반 클래스는 합칠 수 있기 때문에..)
* 인터페이스 내부의 메서드는 자동으로 abstract public이 된다.
* 인터페이스 내부의 변수는 자동으로 final static이 된다.
* 인터페이스 내부의 메서드를 정의할 때 default를 붙이면 기본 동작을 설정할 수 있다.  
  (인터페이스 내부에 메서드들은 모두 abstrac(추상 메서드)가 되므로 보통 메서드로 작동하기 위해서는 default를 써야함)



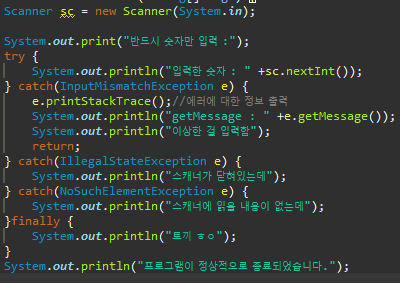
21.04.06

**#예외 (Exception)**

* 우리가 에러라고 알고 있던 빨간 글씨들을 사실 **예외**라고 부른다
* 프로그래머가 발생 가능한 예외를 미리 예상하고 대비해두는 것을 예외처리라고 한다.
* 자바의 문법을 틀리는 것은 예외가 아니라 컴파일 에러라고 부른다 (오타 등)
* 프로그래머가 예외에 대하여 적절한 조치를 취해준다면 프로그램을 강제종료하지 않고 계속해서 진행할 수 있다.
* 메서드에 마우스를 올려보면 어떤 예외가 어떤 상황에 발생하는지에 대한 설명이 Throws에 적혀있다.
* Catch(예외가 되는 경우)를 입력하면 예외가 되는 경우에 한해서만 예외저리가 됨  
  **Catch(Exception e)를 하면 모든 예외에 대해서 적용되는 거임**

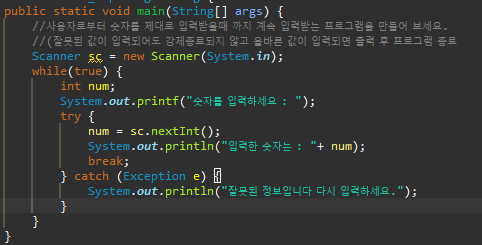
**#예외처리**

* 예외가 발생할 가능성이 있는 코드를 try문 내부에 포함시킨다
* Try문 내부에 있는 코드는 예외가 발생하지 않으면 모두 정상 실행된다.
* Try문 내부에서 예외가 발생하는 경우 실행이 즉시 중단되고 해당 예외를 담당하는 catch문으로 이동하게 된다
* Catch문에는 발생한 예외에 대한 정보를 담고 있는 인스턴스가 도착하게 된다.
* Catch문은 여러 개 중첩해서 사용할 수 있다.
* 예외가 발생하든 안하든 실행하고 싶은 코드는 finaly에 작성한다.(catch 부분에 return을 붙이면 fanally만 작동됨)

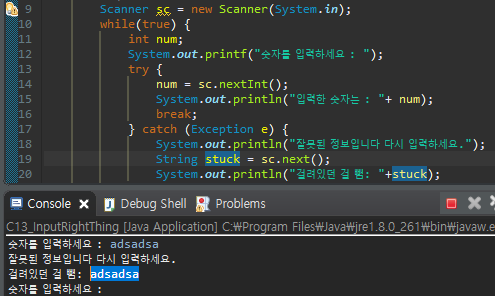


System.exit(0); 🡪프로그램 강제 종료

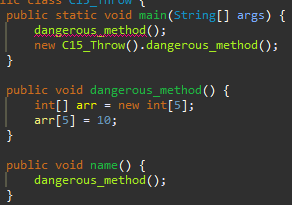
🡪이렇게 짜게되면 sc.nextInt()라는 메서드에 잘못된 찌꺼기가 남아서 try를 판단할때  
바로 catch로 넘어가게 됨 그렇게 때문에 sc를 while안에 넣어 계속 선언함으로써 찌꺼기를 없애야 함



혹은 🡪catch부분에 찌꺼기를 빼내면 됨

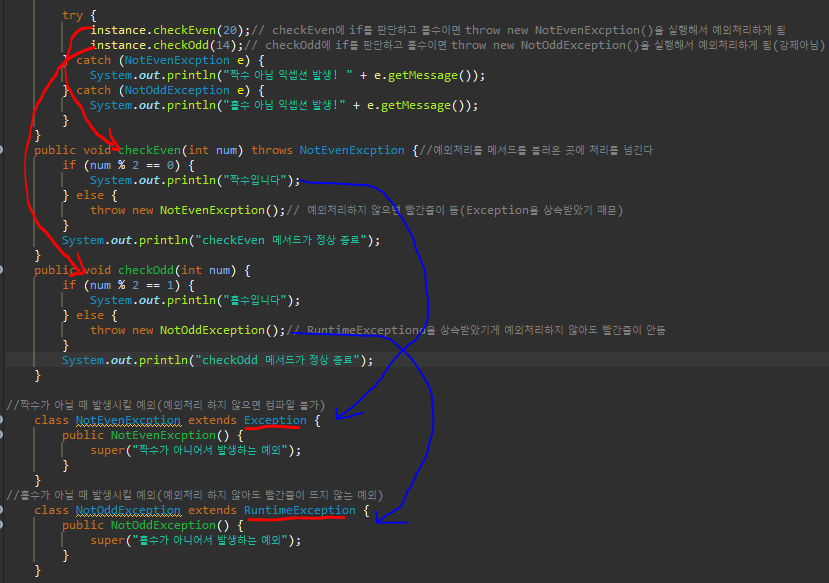
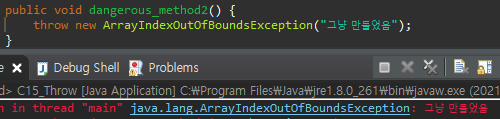


**\*static이 있는 메서드에서는 같은 클래스에 있는 메서드라도 인스턴스화를 해줘야함 <->static이 없는 메서드는 같은   
클래스 메서드는 인스턴스화 필요 없음**



**#throw**

* 원하는 예외를 발생시킨다



**#throws**

* 해당 메서드에서 이러한 예외들을 발생시킨다는 것을 알린다
* 해당 메서드를 호출하는 곳에 경고를 해줄 수 있다.
* 예외에 대한 처리를 호출하는 곳으로 전가

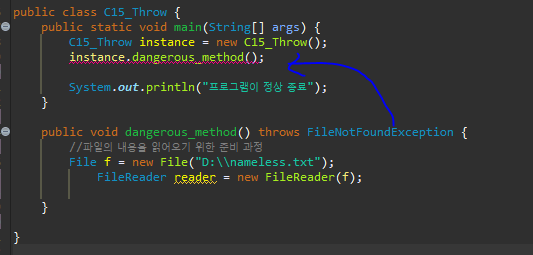
#Exception 클래스를 상속받은 예외는 “반드시 처리해야 하는 예외”가 된다.

* Exception은 예외처리하지 않으면 컴파일 에러를 발생시킨다
* 예외처리가 된 경우에만 컴파일을 허용해준다.

#RuntimeException 클래스를 상속받은 예외는 “꼭 처리하지 않아도 되는 예외”가 된다

* InputMismathException, ArrayIndexOutOfBoundException… 등등
* 예외처리를 하지 않아도 컴파일 에러를 발생하지 않는 예외들

🡪throws가 없으면 아래 메서드에 try/catch를 해야하는데 호출하는 곳에 경고를 넘김으로서 메인 메서드에 try/catch를 해줘야 함.(단 꼭 처리하지 않아도 되는 예외는 빨간줄이 안뜸)



**# Java collections**

* 자바에서 기본적으로 제공하는 자료구조 인터페이스
* Collection 인터페이스를 상속받은 인터페이스로는 List, Set이 있다.

**#Java.utll.ArrayList**

* 배열과 다르게 크기가 자동으로 조절되는 자료구조 클래스
* 배열과 유사한 형태이다
* 순차적으로 데이터에 접근해야할 때 가장 유리하다
* 타입을 지정할 때 사용하는 <>를 제네릭(Generic)이라고 부른다  
  ArrayList<String> fruits = new ArrayList<String>();어레이 리스트에 String타입만 들어감  
  이때 제네릭에는 기본형 타입을 사용할 수 없다 (int는 안됨, Integer)
* .add(item): 리스트에 원하는 데이터를 순차적으로 맨 뒤에 추가한다fruits.add("kiwi");
* .add(index, item): 리스트의 원하는 위치에 데이터를 넣는다.(이때 index는 0부터) fruits.add(2, "banana");
* .get(index): 리스트에서 원하는 인덱스의 데이터를 하나 꺼낸다(이때 index는 0부터) System.***out***.println(fruits.get(0));
* .size(): 해당 컬렉션의 크기를 반환한다. System.***out***.println(fruits.size());
* .remove(index): 해당 번째의 데이터를 삭제한다, 방금 지운 데이터를 반환한다(return 가능) fruits.remove(0);  
  \*contains(): 컬렉션에 해당 요소가 포함되어 있는지 여부를 반환
* Collections.sort(어레이 리스트) : 오름차순으로 정렬(숫자면 낮은 숫자부터 정렬) Collections.*sort*(fruits);  
  배열 정렬은 Arrays.sort(배열 이름)
* \*Collections.addAll(컬렉션이름, 추가1, 추가2, 추가3, 추가4, ….) : 해당 컬렉션에 한번에 추가하는 기능

21.04.07

String.format : 서식을 통해 내가 원하는 문자열을 만드는 것  
ex. String.*format*("서식 %d", 10);

**# Set**

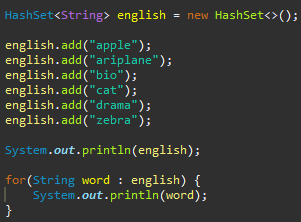
* 집합을 구현해놓은 클래스
* 요소로 같은 값이 들어오는 것을 허용하지 않는다(중복은 허용하지 않음)

**# Hash**

* 어떤 값을 넣었을 때 전혀 예측할 수 없는 값이 생성되어야 하는 알고리즘
* 같은 값을 넣었을 때는 항상 일정한 값이 나와야 한다
* 유사한 값을 넣었을 때 예측하기 힘든 전혀 다른 값이 나온다
* 생성된 해쉬값으로 원래 값을 찾는 것이 불가능해야 한다  
  즉 로그인할 때 비밀번호 같은거를 암호화해서 저장하게해야함
* 다시 원래값으로 돌아갈 수도 없는 단방향성 알고리즘
* 중복 체크시에 뛰어난 성능을 보인다  
  예를 들어 방대한 내용(책 같은거)도 도출되는 알고리즘 개수는 일정하므로 수정되었는지 중복되었는지 체크하는 데에 용이함  
    
  hello 🡪 **5d41402abc4b2a76b9719d911017c592**(즉 어떤 값을 넣었을 때 예측하기 힘든 전혀 다른 값이 나오는 알고리즘임) **—MD5**  
  hello1 🡪**1299118129ac98235697727158def400(즉 유사한 값을 넣을 넣었을 때 예측하기 힘든 전혀 다른 값이 나옴)—MD5**

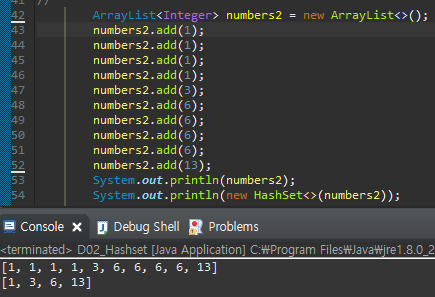
**\*Hash를 사용하는 알고리즘은 순서 혹은 index가 없다(hash 순서이기 때문에 예측 불가능)  
-즉 Collections.sort(정렬)을 못씀**

**\*같은 컬렉션 인터페이스이기 때문에 \*HashSet과 ArrayList는 같은 메서드들임**\*Set은 인덱스가 없기 때문에 get를 쓸 수 없고 for-each로만 내용을 꺼낼 수 있다.  
 🡪 String word : english에 들어있는 원소만큼 word에 하나씩 넘겨서 돌리겠다는 것



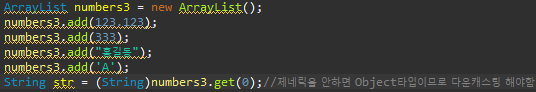
**\*컬렉션끼리는 변환이 자유롭다**-Set의 내용을 정렬해야 한다면 List로 변환하여 사용할 수 있다.  
-List의 내용들 중 중복을 제거한 목록이 필요하다면 Set로 변환하여 사용할 수 있다.  
ArrayList<String> english\_list = new ArrayList<>(english);-->set타입을 list로 받음  
Collections.*sort*(english\_list);🡪즉 set을 정렬하게 만드는 거임  
System.***out***.println(english\_list);🡪정렬된 set(실제론 list)를 출력

🡪즉 list타입에 대해 중복을 제거하기위해 set을 사용한 거임



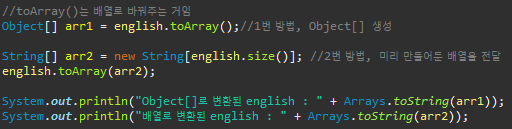
**\*제네릭<>을 요구하는 클래스에 제네릭을 사용하지 않는 경우 모든 타입을 넣을 수 있다.**

* 모든 타입이 Object 타입으로 업캐스팅 되어 저장된다
* 해당 값을 다시 제대로 사용하기 위해서는 다운캐스팅이 필요해서 번거롭다



**\*컬렉션들을 배열로 변환하기**

\*toArray() : 컬렉션 형태로 되어있는 것들을 객체배열로 반환해줌(리턴타입은 Object)

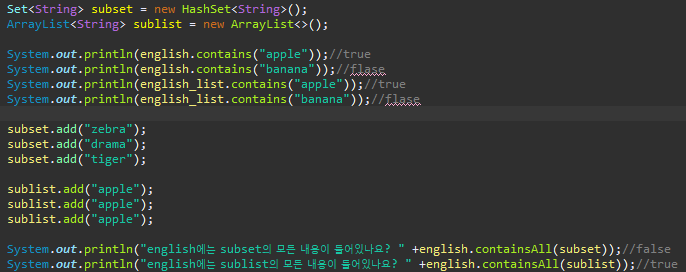


🡪첫번째 방법은 english라는 hashset을 배열로 변환(단 타입은 Object)

🡪두번째 방법은 hashset의 사이즈 만큼의 배열을 만들어서 미리 만들어둔 배열에 hashset을 전달  
**🡪즉 중복제거해서 배열로 추출하기 위함임(인덱스 혹은 정렬의 장점이 있음)**

\*contains(): 컬렉션에 해당 요소가 포함되어 있는지 여부를 반환

\*containAll(Collection) : 전달한 컬렉션이 모두 포함되어 있는지를 검사



\*addAll(Collection) : 전달한 컬렉션의 내용을 모두 추가(추가 성공 여부를 반환🡪리턴 타입 boolean)**🡪단 제네릭 타입이 같아야함**  
-추가 성공 여부를 반환  
-이미 모두 포함된 경우에는 false를 반환  
english.addAll(subset);

\*removeAll(Collection) : 전달한 컬렉션과 같은 내용을 모두 제거  
-지워진게 있으면 true 없으면 false를 반환  
english\_list.removeAll(subset);🡪english의 hashset과 subset의 set의 교집합에 해당하는 내용을 english에 대해 제거

\*retainAll(Colleciont) : 전달한 컬렉션과 일치한 내용만 남긴다.  
-변한 것이 있으면 true, 없으면 false를 반환  
english.retainAll(subset); 🡪english의 hashset과 subset의 set의 교집합에 해당하는 내용을 english에 남김

**\*isEmpty() : 해당 컬렉션이 비어있으면 true, 아니라면 false를 반환**

21.04.08

\*소문자로 시작하는 타입들(기본형)은 매개변수로 전달할 때 값을 복사하여 전달한다.

\*대문자로 시작하는 타입들(참조형)은 매개변수로 전달할 때 주소를 복사하여 전달한다.(String만 제외)

\*제네릭으로 설정해둔 클래스는 인스턴스 생성시 원하는 타입을 고를 수 있게 된다.-->고르지 않으면 Object로 설정  
public class D02\_2\_JiphapQuiz<T>

이때 <T>는 T만 들어올 수 있고 <? extends T>는 T를 상속받은 제네릭도 들어올 수 있음

\*다른 패키지에서는 dafault 패키지에 접근(임폴트) 불가능

**#Wrapper Class**

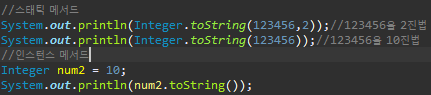
* 기본 타입들을 참조형 타입으로 사용하기 위한 클래스
* 각 타입들에 대한 유용한 기능들이 모여있다.
* 기본형 – byte, short, char, int, long float, double, boolean
* 참조형 – Byte, Short, Character, Integer, Long, Float, Double, Boolean

**#parse 시리즈**

* 문자열을 원하는 타입으로 변환시키는 기능  
  Integer.*parseInt,* ("ㅁㄴㅇ1234ABC", 16(진0법);//-->문자가 있어 오류가 남Double.*parseDouble*Character.*toString*(str.charAt(i) 🡪문자열을 문자(String)으로 변환해줌
* 각 타입에 스태틱 메서드로 존재한다.
* 데이터를 주고 받을때는 잠시 문자열로 변환하여 전송되는 경우가 많다
* 다시 사용하기 위해서는 받은 데이터를 원래의 타입으로 되돌려야 한다.
* Radix가 적혀있는 오버로딩은 진법을 의미한다

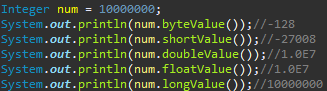
**#toString()**

* 해당 타입의 값을 손쉽게 문자열로 변환할 수 있다.
* toString은 Object의 메서드이기 때문에 모든 클래스에 존재한다
* 인스턴스 메서드 toString()은 해당 인스턴스의 값을 문자열로 반환
* 스태틱 메서드 toString()는 전달한 값을 문자열로 반환한다.



**#typeValue() 시리즈**

* 해당 인스턴스 값을 원하는 타입의 값으로 손쉽게 변경하는 메서드
* 타입캐스팅의 참조형 버전

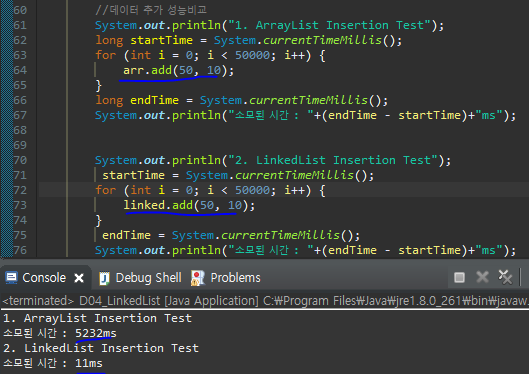


**#연결 리스트(LinkedList)**

* ArrayList보다 삽입/삭제에 유리한 리스트
* 데이터 중간에 삽입이 발생했을 때 ArrayList는 모든 데이터를 뒤로 밀어야 한다.
* 데이터 중간에 삽입이 발생했을 때 LinkedList는 연결된 노드만 변경하기 때문에 훨씬 빠르다
* ArrayList는 데이터 보관 및 읽기에 유리하고 LinkedList는 데이터 추가 수정 및 삭제에 유리하다(웹에서는 ArrayList를 많이 씀)

\*pop(), poll() : 리스트에서 맨 앞 데이터를 읽고 버린다🡪즉 변수명(LinkedList타입).pop();을 하면 index0이 삭제됨

\*pollLast() : 리스트의 맨 뒤 데이터를 읽고 버린다.



🡪즉 수정에 있어서는 LinkedList가 더 빠름

**#Comparator**

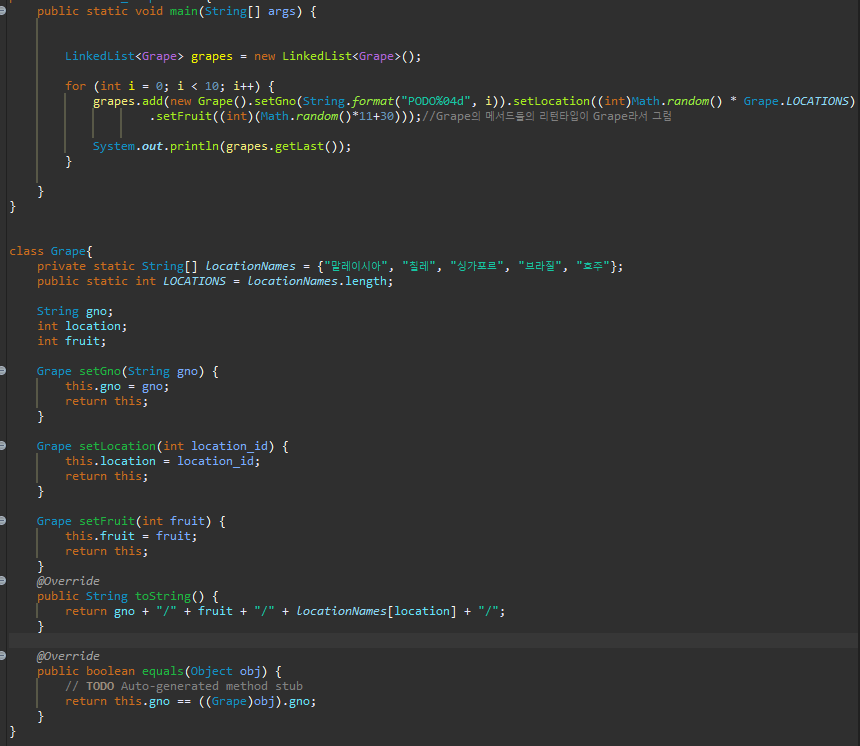
* 구현하면 두 객체의 크기를 비교할 때 사용하는 클래스가 되는 인터페이스
* 객체간의 크기를 정할 때 사용된다

**#toString**

* 모든 객체에 포함되어 있는 메서드 (Object 메서드)
* 이 클래스를 문자열로 표현한다면?을 구현해 놓은 메서드
* System.out.println()은 사실 전달한 인스턴스의 toString() 메서드의 결과를 콘솔에 출력하는 메서드이다.
* Object 클래스에 구현되어 있는 toString() 메서드의 기본 동작은 “클래스명@주소값”이다.

**#equals()**

* Object 메서드
* 같은 객체인지 비교할 때 어떤 동작으로 구분해야 하는지를 구현해 놓은 메서드
* 기본 동작은 주소값 비교



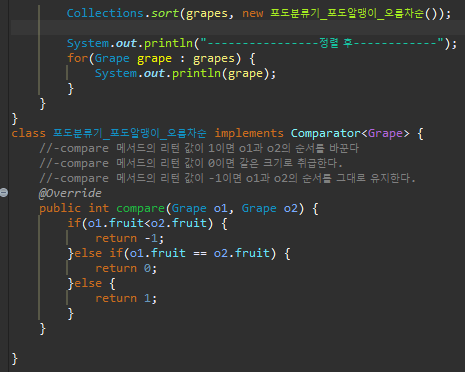
🡪즉 Object에 있는 메서드인 equals, toString을 오버라이드해서 객체를 출력하는(toString)   
또는 객체끼리 기준에 따라 같은지(equals) 판단하는 거임



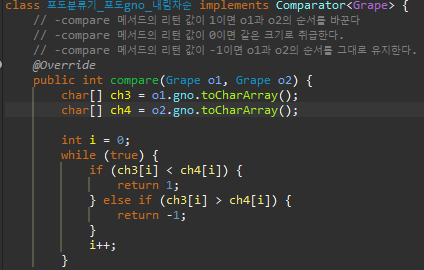
또한 toString()을 결정해주면 메서드를 호출하지 않아도 sysout만 해줘도 toString이 자동 발동됨

**#Comparator의 규칙  
\*반드시 Comparator를 상속받고 compare 매서드를 이용해야함!**

* compare 메서드의 리턴 값이 **양수**이면 o1과 o2의 순서를 바꾼다
* compare 메서드의 리턴 값이 **0**이면 같은 크기로 취급한다.
* compare 메서드의 리턴 값이 **음수**이면 o1과 o2의 순서를 그대로 유지한다.



🡪즉 compare 메서드를 통해 Grape 클래스의 크기를 결정 하고 Collections.sort(grapes(리스트 이름), Comparator 객체)를 해주면 정렬을 해줌



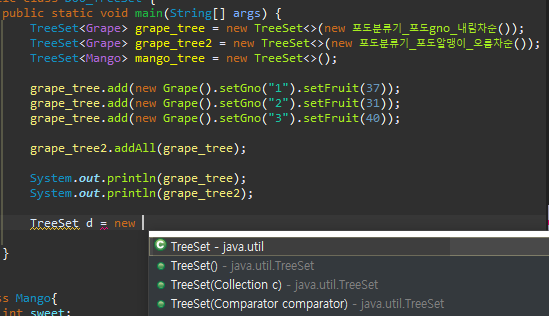
🡪String에 대해서 비교하는 로직임

**\***return o1.gno.compareTo(o2.gno); 🡪문자열의 알파벳 순서 비교는 이미 String클래스의 compareTo 구현되어 있다.

21.04.09

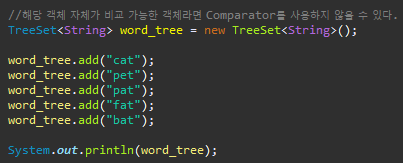
**# Treeset**

* 중복값을 저장할 수 없다
* 데이터를 저장할 때 Comparator를 이용해 미리 정렬한다.
* 요소로 null을 허용하지 않는다.
* 처음에 정렬해서 넣기 때문에 HahSet에 비해 저장 속도가 느리다
* 하지만 TreeSet에는 순서(인덱스)가 있기 때문에 원하는 데이터만 꺼낼 수 있다.
* 하지만 TreeSet은 정렬이 미리 되어 있기 때문에 정렬된 순서를 데이터를 꺼내는 것은 가장 빠르다.



🡪즉 TreeSet에 대해 정렬기준(Comparator)를 넣어주는 것이 좋음(안넣어주면 에러발생)

단 해당 객체 자체가 비교 가능한 객체라면 Comparator를 사용하지 않을 수 있다.  
🡪단 제네릭이 String이나 Integer 같은 경우에는   
자체 크기 비교가 가능한 객체이기에 따로 Comparator를 통해 compare가 필요가 없다.



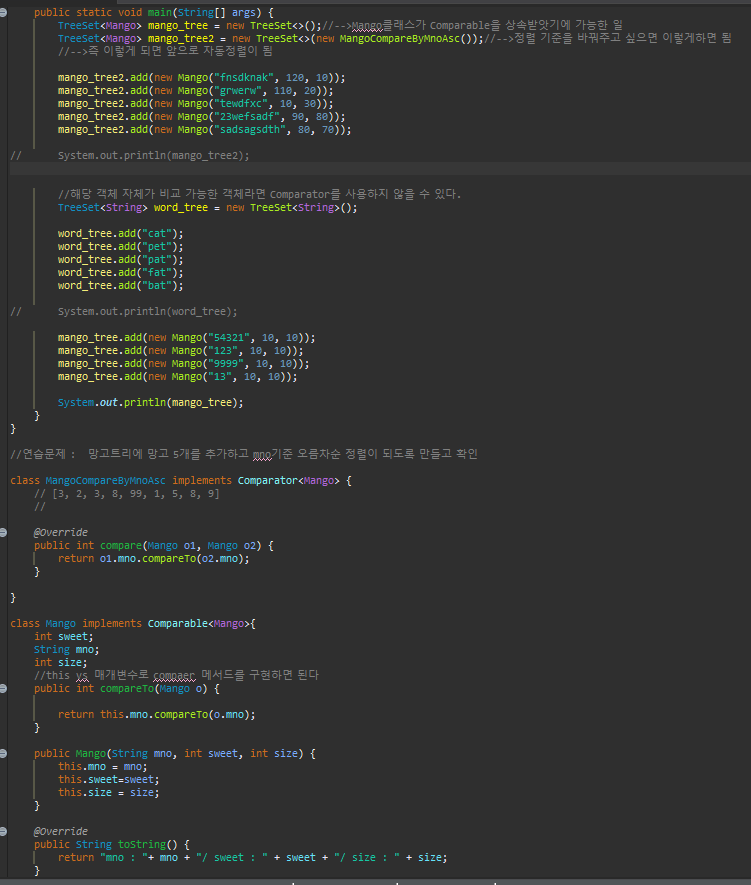
\*Comparator : 비교할때 사용하는 비교 도구 클래스로 만드는 인터페이스

\*Comparable : 이 객체가 비교 가능한 클래스임을 표시하는 인터페이스

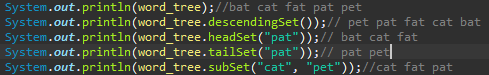
**#TreeSet의 두 가지 생성자**

* 정렬 기준을 정하면서 인스턴스를 생성
* 컬렉션을 전달(정렬 기준은 전달하는 컬렉션의 제네릭 타입의 Comparable을 사용)

**Comparator, Comparable에 대한 설명**



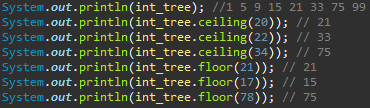
\*decendingSet() : 미리 정렬해 놓은 방향의 반대 방향의 Set 인스턴스를 반환  
\*headSet(toElement) : 맨 앞부터 원하는 요소까지 자른 Set을 반환🡪단 미리 정해둔 정렬기준에 맞게 기준을 넣어야 함  
\*tailSet(toElement) : 원하는 요소부터 맨 뒤까지 자른 Set을 반환  
\*subSet(fromElement, toElement) : 원하는 부분만을 자를 수 있음



**한편 TreeSet에 대해 제네릭이 String, Integer등 자체 비교 가능한 객체에 대해서는 따로 정렬기준을 해주지 않아도 자동 오름차순 정렬됨**

\*ceiling(element) : 원하는 가장 가까운 요스를 올려서 선택 (전달한 값도 포함)

\*floor(element) : 원하는 가장 가까운 요스를 내려서 선택 (전달한 값도 포함)



\*higher(element) : 원하는 가장 가까운 요스를 올려서 선택 (전달한 값도 **미포함**)

\*lower(element) : 원하는 가장 가까운 요스를 내려서 선택 (전달한 값도 **미포함**)

Out class에는 protected와 private을 사용할 수 없다.

Inner class에는 모든 접근 제어자를 사용할 수 있다.

21.04.12

Getter : public으로 private 필드를 꺼내가기만 하는 메서드

Collections.shuffle(컬렉션 이름) : 그 컬렉션에 대해 위치를 랜덤으로 섞음

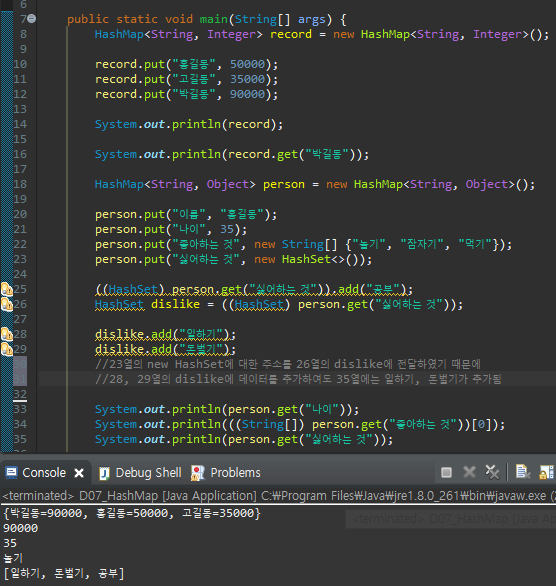
**#Map**

* Key와 Value가 한 쌍을 이루는 자료구조
* Key값을 통해 Value에 접근할 수 있다
* Key는 중복을 허용하지 않는다

put(key, value) : map에 데이터 추가

get(key) : key를 이용해 value를 꺼냄

🡪즉 위 화면처럼 단순 데이터 뿐만이 아니라 배열이나 컬렉션과 같은 것도 들어갈 수도 있음



21.04.13

이미 존재하는 키값에 put하면 값을 수정함

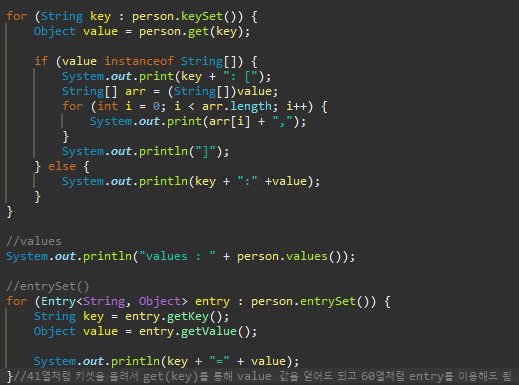


🡪이미 존재하는 키값에 새로운 value를 put으로 추가하면 value가 바뀌며 이전에 들어있는 값을 출력함

#반복문으로 Map을 활용하기

* keySet() : key들로만 이루어진 Set을 반환한다
* values() : value들로만 이루어진 Set을 반환한다.
* **entrySet() : Entry<key, value>로 이루어진 Set을 반환한다  
  (entry란 하나의 key와 value의 쌍이라고 생각하면 됨)**
* containsKey(key) : 맵에 해당 키가 포함되어 있는지 검사한다(boolean 타입)
* containsValue(value) : 맵에 해당 값이 포함되어 있는지 검사한다(boolean 타입)

**A instanceof B : A가 B의 인스턴스인지를 판별한다. 업캐스팅된 상태에서도 판별된다.**



**\*String1.contains(String2) : String2이 String1에 포함되어있는지 판단해주는 메서드(boolean 타입)**



🡪Entey를 이용해 키와 값을 TreeSet에 넘겨 기준에 맞게 정렬하는 코드(참고)

21.04.14

**#String**

* split(delimiter) : 문자열을 원하는 기준으로 자른 후 문자열 배열로 변환



* join(delim, 이어붙이고싶은 것(배열, 컬렉션이 들어갈 수 있음)) : 뒤에 나오는 문자열들을 모두 delim으로 이어붙인다.



* Substring(start(int), end(int)) : start부터 end미만까지 문자열을 잘라서 반환



* Replace(“바꿔야할 문자열”, “바꾸고싶은 문자열”) : 문자열을 치환한다.(**정규표현식도 가능**)



* Format(format, value) : 서식을 이용해 원하는 문자열을 생성



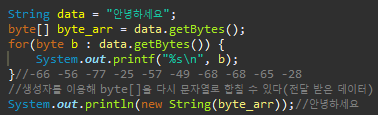
* Contains(seq) : 문자열에 해당 문자열이 포함되어 있는지 검사한다



* Stateswith(seq) : 문자열이 해당 문자열로 시작하는지 검사한다(경로 검사에 유용)
* endsWith(seq) : 문자열이 해당 문자열로 끝나는지 검사한다(확장자 검사에 유용)



* getBytes(): 문자열을 byte[]로 변환한다 (데이터 전송 준비)  
  이때 byte[]로 나뉠 때와 같은 인코딩 타입을 사용해야 한다.



* trim() : 문자열 바깥쪽의 공백을 제거해 준다.



* toUpperCase() : 문자열을 모두 대문자로 변환한 새로운 인스턴스 반환
* toLowerCase() : 문자열을모두 소문자로 변환한 새로운 인스턴스 반환
* equalsIgnoreCase() : 대소문자 상관없는 equals()
* getChars(start, end, 담을곳, 담을곳 시작지점) : char[]에 문자열의 원하는 부분을 담을 수 있다



* Arrays.fill(char[], 문자) : 해당 배열에 문자를 꽉 채움

21.04.15

T… elements : 가변인자(여러 개 넣을 수 있음)

**#정규표현식(Regex Expression)**

* 문자열의 패턴을 표현하는 표현식
* 특정 프로그래밍 언어에 종속되지 않는다.
* 프로그래머가 원하는 문자열 패턴과 일치하는지 검사할 때 사용한다

**# Pattern 클래스**

* 정규표현식을 다루는 클래스

**#Matcher 클래스**

* 패턴 인스턴스를 이용하여 문자열을 검사할 때 사용하는 클래스

Pattern.matches(regex, input) : input이 regex에 매치되는 문자열인지 검사해주는 메서드

[] : 해당 위치의 한 글자에 어떤 문자들이 올 수 있는지 정의하는 표현식(regex 자리, char 혹은 int형 즉 한 단어밖에 못들어옴)  
System.***out***.println(Pattern.*matches*("s[lh@]eep", "sleep"));//ture

[]내부에 정의할 수 있는 것들

1. abc : 해당 자리에 a 또는 b 또는 c를 허용
2. ^abc : 해당 자리에 a,b,c를 제외한 모든 것을 허용  
   System.***out***.println(Pattern.*matches*("s[^lh@]eep", "s@eep"));//false
3. a-z : a부터 z까지 모두 허용  
   System.***out***.println(Pattern.*matches*("s[0-9a-z]eep", "s2eep"));//true 0~9, a~z까지 허용
4. && : 교집합



1. | : or임(&&와는 달리 1개만 써도 됨)



여러 문자를 나타내는 것들([]안에 안쓰는거임)

1. . : 모든 문자



1. \d : 모든 숫자(이스케이프 문자이기에 역슬래쉬 2번 해야함)  
   System.***out***.println(Pattern.*matches*("s\\deep", "s9eep"));//true // 모든숫자
2. \D : 숫자를 제외한 모든 것  
   System.***out***.println(Pattern.*matches*("\\D\\D\\D", "ㅊat"));//true // 모든 문자
3. \s : 모든 공백 (\t, \n .. 등의 모든 공백들  
   System.***out***.println(Pattern.*matches*("s\\seep", "s eep"));//true // 공백만 허용
4. \S : 공백 제외  
   System.***out***.println(Pattern.*matches*("s\\Seep", "s eep"));//flase // 공백비허용
5. \w : 일반적인 문자들을 허용 [a-zA-Z0-9\_]
6. \W : 일반적인 문자들을 제외한 모든 것을 허용

**#해당 패턴이 적용될 문자의 개수를 정의하는 방법**

1. [ex]{n} : {}앞의 패턴이 n개 일치해야 한다



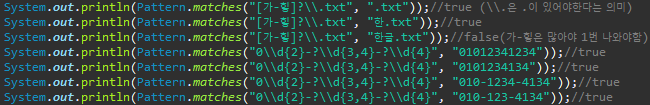
1. [ex]{n,m} : {} 앞의 패턴이 최소 n개 이상 최대 m개(m개 포함) 까지 일치해야한다.  
   **단 {n, m}처럼 ,뒤에 띄어쓰기는 금지!!**



1. [ex]{n,} “ {} 앞의 패턴이 최소 n개 이상 일치해야한다.



1. [ex]? : 0번 또는 한번 허용



1. [ex]+ : 최소 한번 이상
2. [ex]\* : 0번 또는 여러 번(얘는 길이제한이 없다)(\*뒤에 오는 문자는 마지막 문자로 할 수도 있음)



이런식으로도 가능!

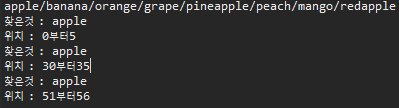
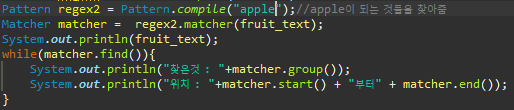
**#Pattern.compile(regex)**

* 전달한 정규표현식을 해석하여 Pattern 인스턴스를 생성하고 반환한다.

**#split() : 컴파일된 패턴 인스턴스로 문자열 split()을 진행한다.  
🡪/을 기준으로 잘라줌**



**#matcher() : Matcher 인스턴스를 생성하여 반환한다**



🡪즉 기준을 찾아줌



**^ : 처음, $ :마지막**

Pattern regex = Pattern.*compile*("[\\w]+@[\\w]+[.](com|net|co.kr)");  
🡪이메일을 찾아주는 정규표현식

String regex = "(?=.\*[a-z])(?=.\*[0-9])(?=.\*[!@#$%^&\*\_]).{8,16}"  
🡪숫자, 문자, 특수문자를 최소 1번씩 써야하는 정규표현식

나의 해석

(?=패턴) : 전방탐색 🡪이후에 나오는 문자들이 ‘패턴’에 매치되어야 하지만, 매치 결과에는 나타나지 않는다

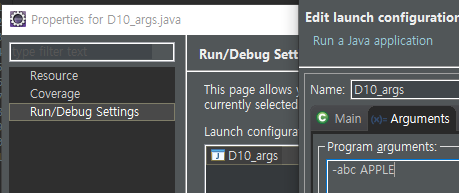
즉 (?=.\*[a-z])을 보면 ?= 뒤에 있는 패턴은 .(아무문자 가능)이며 \*이 적어도 0번 이상이니 따라서 패턴 [a-z]이 최소한 한번 이상 나와야 한다는 의미??

.\* : 아무 문자가 있지만 수량제한은 없음

21.04.16

main 메서드 첫 번째 매개변수(args)의 역할

* 프로그램 실행시 사용자가 지정한 옵션을 main의 첫 번째 매개변수에 받는다  
    
  🡪String[] args에 -abc, APPLE이 기본적으로 들어가있는 상태에서 프로그램이 시작됨  
  (설정은 클래스파일 우클릭-Properties-Run/Debug-Edit-Arguments-Program Arguments에 기입)



* 사용자가 지정한 옵션에 따라 다르게 동작하는 프로그램을 만들 수 있다.

**#Date, Calendar, Time**

* 자바로 날짜와 시간을 다루는 클래스

\*유닉스타임 – 1970년 1월 1일 이후로 시간이 몇 밀리초나 흘렀는지 센 것

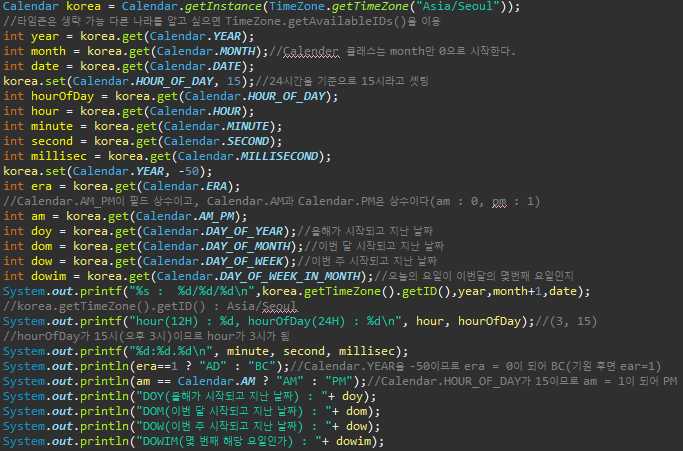
\*System.currentTimeMillis(); : 시스템의 유닉스 타임을 가져오는 메서드**(여기에 1000을 나누면 초단위가 됨)**

**#Date(이거 옛날 방식이야)**

* Date 클래스 내부의 메서드들은 요즘 버전에서는 거의 업데이트가 중단되어 있다
* 시간을 계산 할 때는 **다른 클래스를** 사용할 것을 권장
* Date 클래스는 주로 시간 인스턴스를 보관하는 용도로만 사용한다.

**#Calender(이것도 옛날 방식)**

* 달력을 표현한 클래스
* 날짜를 쉽게 계산할 수 있는 기능들이 모여있다.
* 시간을 가져올 때 생성자가 아닌 getInstance()라는 스태틱 메서드를 사용한다  
  Calendar now2 = Calendar.*getInstance*();
* getInstance() 메서드에 경도를 나타내는 TimeZone을 전달할 수 있다.
* TimeZone : 여러 지역의 시차를 보관하고 있는 클래스  
  -getAvailableIDs() 스태틱 메서드를 통해 사용할 수 있는 타임존들을 볼 수 있다
* Calendar에 저장된 시간 중 필요한 값을 get()메서드를 통해 꺼낼 수 있다
* Calender에 저장된 시간을 set() 메서드를 통해 수정할 수 있다.



int maxDay =korea.getActualMaximum(Calendar.***DATE***); 🡪현재 월의 가장 큰 일수 (1월-31, 2월-28, 3월-31…)

add(field, int) : 원하는 필드를 원하는 수치만큼 더하거나 뺄 수 있다.

🡪Calendar 클래스는 어떤 필드를 변경하면 나머지 필드도 상황에 맞게 알아서 업데이트 된다.

#SimpleDateFormat – 간편하게 날짜 인스턴스를 문자로 변환할 수 있는 클래스

#날짜 서식  
y: 년, M: 월, d: 일(1~30), D : 월 구분 없는 일(1~365) H: 시(24시), h: 시(12시), m:분, s: 초, S: 밀리언 초, E: 요일, a : 오전/오후,   
w : day of week in year(올해 시작 기준 몇번째 주?), W : day of week in month(이번 달 시작 기준 몇번째 주?

SimpleDateFormat f1 = new SimpleDateFormat("yy/MM/dd HH:mm:ss.SSS a EEEE" System.*out*.println(f1.format(Calendar.*getInstance*().getTime()));

**#java.time.Time**

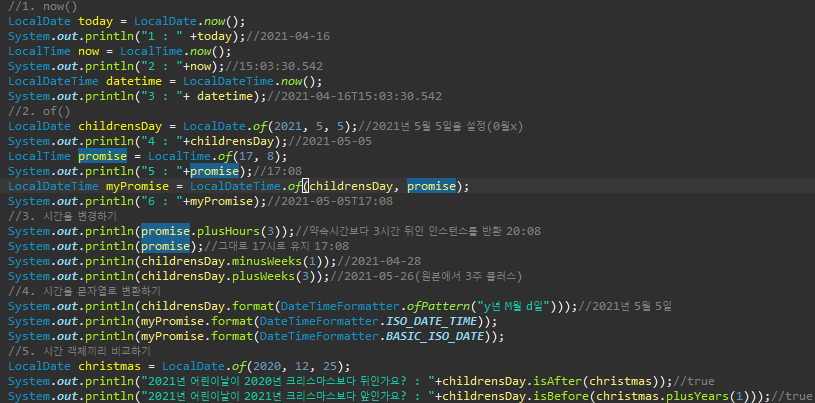
* Calendar의 인스턴스를 set()을 통해 변화시킨다는 단점을 보완한 클래스
* Java.time 패키지의 클래스들은 시간을 계산한 뒤 새로운 인스턴스를 반환한다.  
  즉 원본 인스턴스는 그대로 유지한다(calendar와의 차이점)
* LocalDate, LocalTime, LocalDateTime이 있다.

1. now() : 지금 시간의 인스턴스를 생성하는 메서드
2. of() : 원하는 시간의 인스턴스를 생성하는 메서드
3. 시간을 변경하기(원본 인스턴스가 변경이 안됨)  
   -Calendar는 원본을 훼손하는 안좋은 설계였기 때문에 여기서 차이가 난다.  
   -java.time 패키지의 클래스들은 시간이 변경된 **새로운 인스턴스를** 반환한다.
4. 시간을 문자열로 변환하기  
   -DateTimeFormatter 클래스를 사용한다  
   -다양한 static 메서드와 표준들을 이용해 문자열 형식을 지정할 수 있다.  
   -.ofPattern(srt) : 포맷 직접 만들기  
   -.다양한 표준 이름들 : 표준 사용하기
5. 시간 객체끼리 비교하기  
   객체a.isAfter(객체b) : a날이 b날보다 뒤인지 판별(Boolean)  
   객체a.isBefore(객체b) : a날이 b날보다 앞인지 판별(Boolean)

\*LocalDate: 날짜만 저장하기 위한 클래스

LocalTime : 시간만 저정하기 위한 클래스

LocalDateTime : 날짜와 시간을 모두 저장하기 위한 클래스



20.04.19

Github 계정 : 이름 : xoghkscc, email : [xoghkscc@naver.com](mailto:xoghkscc@naver.com)

<https://blog.naver.com/kyoungin90/222180641615>

**#Git**

* .git 폴더가 위치한 폴더의 모든 내용을 관리해주는 프로그램
* .git 폴더가 위치한 폴더를 Repository라고 부른다.
* .git 폴더 내부에는 해당 폴더의 현재 상태가 모두 기록되어 있다.
* .git 폴더를 니용하여 언제든지 이전의 상태로 돌아갈 수 있다 (Windows의 복구 기능과 비슷)
* 해당 폴더 내부의 무엇인가가 조금이라도 변하면 알아채고 관리한다
* Eclipse에는 Git이 내장되어 있다.  
  **windows-show view -Git-Git Repositories에 있음**

**#Github**

* Repository는 내 컴퓨터에 저장해놓고 사용할 수도 있고 원격지에 존재하는 컴퓨터에 저장해놓고 필요할때마다 내 컴퓨터로 복사하여 사용할 수도 있다.
* 전자를 로컬 Repository라고 하고 후자를 원격 Repository라고 한다
* **Giuhub는 원격 Repository를 업로드 해놓을 수 있는 사이트다.(클라우드 같은 것)**

**#Github에서 Repository 생성 후 작업하는 컴퓨터로 가져오기**

1. Github에서 새로운 Repository 생성해줄 것 같은 버튼 누르기(프로필 아이콘 누르고 your repository-new 클릭)
2. 레포지토리 이름 정하기
3. Readme.md와 .gitignore 추가하기(.gitignore는 사용하는 언어의 템플릿 고르기)
4. 라이선스 고르기(내가 만든 이 코드를 가져다 사용할 때 돈을 내야하는지 여부
5. Code 버튼 클릭후 https에 있는 주소 복사
6. 이클립스에 **windows-show view -Git-Git Repositories-**Git Repositories-Clone a git repository 클릭 후 uri에 붙여 넣기
7. 하단에 git hub 아이디랑 비번 입력하고 저장할 폴더 설정 -finish
8. Eclipser에서 clone으로 가져오기  
   🡪이렇게 되면 github에 있는 것들을 다 가져올 수 있음
9. 만약 github와 연동된 상태에서 업데이트 하고 싶으면 pull로 땡겨오면 됨

**#Repository에 Eclipse 프로젝트 관리 대상으로 등록하기**

1. 새 프로젝트 생성
2. 우클릭 🡪Team 🡪Share Project 🡪원하는 Repo 선택  
   🡪이렇게 됨



**#commit**

* 레포지토리 내의 변경 사항을 저장하는 명령어
* Git은 언제든지 레포지토리 내의 상태를 commit 했었을 때의 시점으로 돌릴 수 있다.  
  (스냅샷을 찍는다)
* Checkout을 통해 원하는 commit으로 head를 계속 옮겨다닐 수 있다.  
  (head는 현재 git이 보여주고 있는 상태를 말한다.), (우클릭-show in-history에서 하면 됨)

**#.gitignore**

* .gitignore 파일 내부에 등록된 패턴과 일치하는 파일들은 git의 관리 대상에서 제외된다.
* Eclipse 프로젝트를 사용할 때는 Eclipse 관련 설정 파일들을 등록해두는 것이 좋다.  
  .settings/, \*.project, \*.classpath

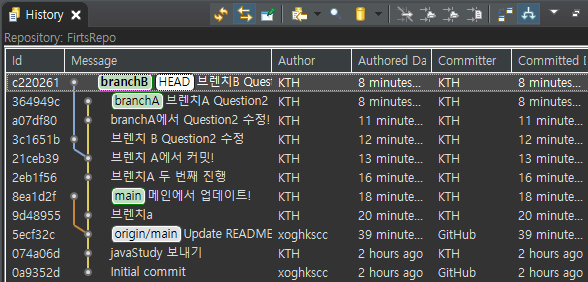
**#unstaged와 staged의 차이**

* 가장 최신 커밋 이후로 변화가 있었던 파일들은 모두(.gitignore에 등록된 것들은 제외) unstaged 상태가 된다
* 모든 변화들 중 이번 커밋에 확정짓고 싶은 변화들만 staged 상태로 올린 후 커밋을 진행한다
* 즉, unstaged는 변화는 있었지만 아직 더 만들어야 하는 상태이고 staged에 포함시킨 것은 변화가 있었고, 다 만든 상태인 코드들을 말한다.

\*서버에 있는 **Repository**를 우클릭해서 pull을 누르면 쭉 땡겨옴

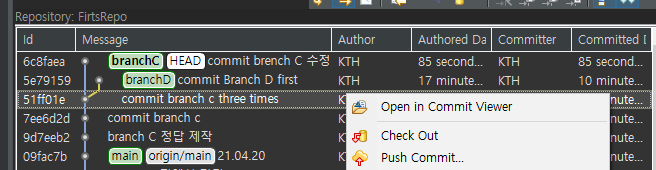
**#branch**

* 하나의 레포지토리를 다양한 상태로 보관하기 위한 기능
* Repositories 🡪switch to … 🡪 New Branch
* Checkout을 통해 언제든지 원하는 branch로 전환이 가능하다
* **Branch를 전환하면 커밋하지 않은 작업 내용들은 모두 사라진다.**
* **history에서 branch들의 상황을 알 수 있다**



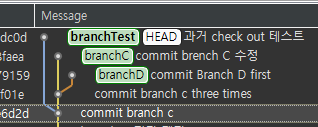
**#checkout**

* Checkout을 하면 그 brench로 이동하거나 history에서 그 시점으로 이동함



* 단 이전버전에서 Check out을 하면 그 시점으로 돌아가고 그 이후의 brench가 남으며 그 이후 다른 brench를 따서 commit을 해야함

**🡪이렇게**



**#merge**

* 현재 HEAD(현재 보고 있는 상태)로 다른 브렌치의 작업 내용을 합치는 것
* 합칠 때 같은 부분을 수정한 곳이 있다면 둘 중 하나를 선택하는 과정을 거친다.
* Merge 충돌이 해결되지 않으면 다음 커밋을 진행할 수 없다.  
  충돌을 해결하려면 merge를 할 때 우클릭-merge-합칠 파일을 택하고 충돌이 일어난 파일에서 남길부분 혹은 제거할 부분을 수정  
  후에 커밋을 하면 됨 그리고 흡수된 branch는 제거하는게 좋음
* 충돌이 없는 경우 쉽게 합쳐진다

21.04.20

**#rebase**

* 브렌치를 나눈 시점을 바꾸는 기능

Main ㅁ-ㅁ-ㅁ-ㅁ-ㅁ-ㅁ(HEAD)

\

ㅁ-ㅁ-ㅁ (branchA)

**\*rebase 후**

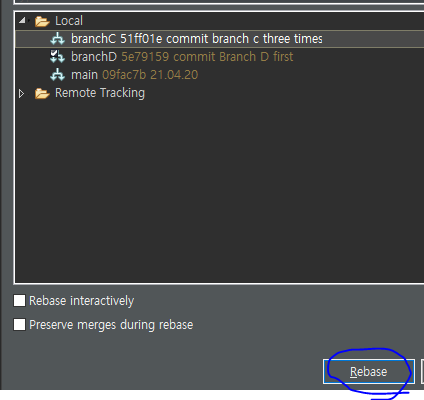
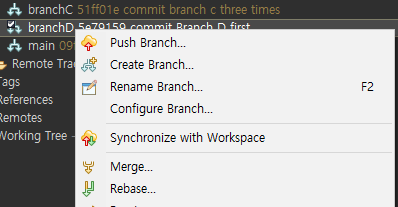
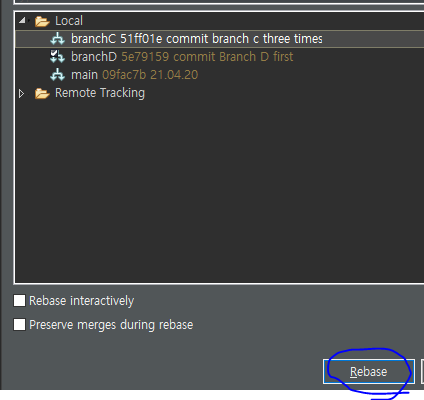
Main ㅁ-ㅁ-ㅁ-ㅁ-ㅁ-ㅁ(HEAD)

\

ㅁ-ㅁ-ㅁ (branchA)

🡪즉 brench C, D를 나누고 C를 작업한 후에 D를 base로 C와 rebase를 하면 C의 내용이 D에 덧붙여지며 C와 D는 나눠진 상태

\*D를 기본으로 C와 rebase 하는 방법 :



🡪즉 brench C는 brench D의 과거가 되기 때문에 brench C는 지워도 무방

**\*다른 사람 계정에 있는 repo 추가하는 방법**

**다른 사람의 경로를 받아서 들어간 뒤 Fork를 통해 다른 사람의 repo를 내 repo로 복붙해서 가져옴 그리고 위와 같은 방법으로 하면 됨**

**#JAVA IO (Input/Output)**

* 자바는 스트림이라는 개념을 이용해 입출력을 다룬다
* 프로그램으로 들어오는 모든 데이터들은 입력이라고 부른다.(Input)
* 프로그램이 연산해서 내보내는 모든 데이터들을 출력이라고 부른다 (Output)

**#Stream**

* 데이터들이 오고가는 통로
* 입력 받을 때 데이터들이 프로그램으로 들어오는 통로를 InputStream이라고 부른다
* 출력 할 때 데이터들이 나가는 통로를 OutputStream이라고 부른다.
* 데이터는 Stream으로 이동하기 위해서는 해당 데이터를 byte 타입으로 변환해야 한다.(String.getBytes 메서드를 통해 가능)

\*FileOutputStream(File file(파일의 경로), Boolean append)타입이며 append가 true일 경우에는 파일의 데이터를 계속 쌓고   
false거나 비어있으면 데이터를 덮어씌움

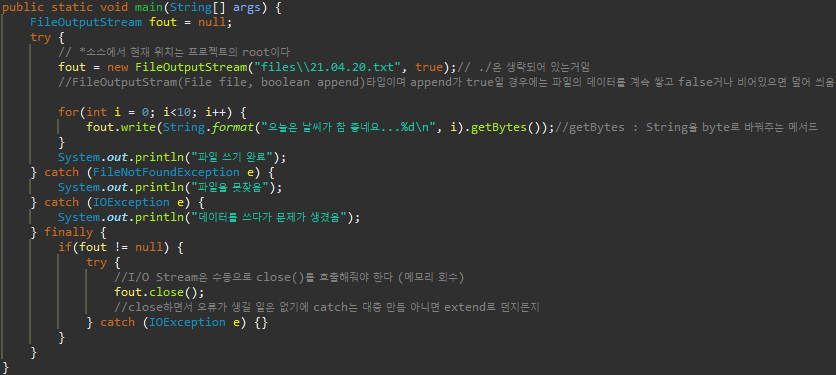
**\*I/O Stream은 수동으로 close()를 호출해줘야 한다 (메모리 회수)**

\*상대경로로 할 경우 **./**(현재위치)을 쓰고 원하는 경로를 쓰면 됨(이때 ./은 해당 workspace의 해당 클래스 파일이 있는 project 폴더가 됨  
(현재는 JavaStudy폴더가 됨)

**#절대 경로와 상대 경로**

* 원하는 자원까지의 경로를 뿌리(root)부터 모두 적는 것을 **절대경로**라고 한다  
  (C:/, D:/, E:/, http://, https://, ftp://, file://)
* 원하는 자원까지의 경로를 현재 위치부터 찾아가는 것을 상대경로라고 한다  
  상대경로는 ./로 시작한다(**./는 생략이 가능하다**)  
  (./) : 현재 위치부터 시작  
  (./)../ : 상위 폴더

**#입력에 대한 예시**



🡪이러면 JavaStudy\\files\\21.04.20.txt에 데이터가 채워짐

**#java.util.File**

* 파일 시스템의 파일 정보를 담을 수 있는 클래스
* 인스턴스에 담겨있는 정보를 바탕으로 새로운 파일이나 디렉토리를 생성할 수 있다.
* 이미 존재하는 파일 또는 디렉토리의 정보를 불러오기도 한다

File a = new File("files\\21.04.20.txt");

a.canRead() : 읽기 권한이 있는지 판단해주는 메서드 (boolean 타입)

a.canExecute() : 실행 권한이 있는지 판단해주는 메서드 (boolean 타입)

a.canWrite() : 쓰기 권한이 있는지 판단해주는 메서드 (boolean 타입)

a.getPath() : 파일 a(21.04.20.txt)의 절대 경로를 반환해주는 메서드 (String 타입)

a.getPath() : 파일 a에 담긴 경로가 절대 경로인지 판단해주는 메서드 (boolean 타입)

a.exists() : 파일 a(21.04.20.txt)가 존재하는지 판단해주는 메서드 (boolean 타입)

File dir2 = new File("D:/files2/"); (실제로 D드라이브에 files2라는 폴더는 없는 상황)

dir2.mkdir(); :해당 경로에(D:/)에 files2라는 디렉토리를 만들어주는 메서드

21.05.12

**#Stream**

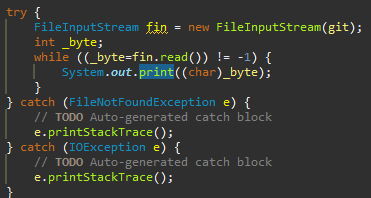
* 데이터들이 오가는 통로
* 프로그램으로 들어올 때 사용하는 통로를 InputStream이라고 한다
* 프로그램에서 처리된 후 나갈 때 사용하는 통로를 OutputStream이라고 한다

**#IntputStream의 공통 메서드**

* read() : 스트림으로부터 다음의 한 바이트를 읽어온다. EOF(End Of File)에 도달하면 -1을 반환
* read(byte[]) : byte[]을 만들어서 전달하면 거기에 담아준다
* read(byte[], off, len) : byte[]을 만들어서 전달하되 원하는 만큼 위치를 읽는다
* close(): 다 사용한 스트림을 닫는다. 닫지 않으면 계속 메모미를 차지하고 있게 된다.
* available() : 용량의 사이즈를 반환

\*1바이트씩 읽기

🡪file에 대해 끝까지 읽게하는 코드



\*원하는 바이트 크기만큼씩 읽기

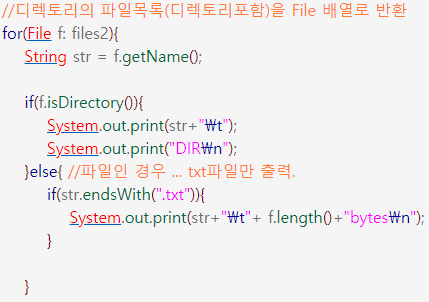
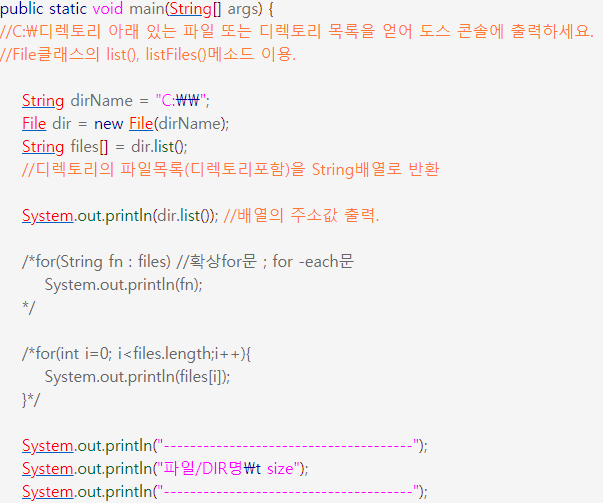
**#OutputStream의 공통 메서드**

* write(int) : 한 글자씩 쓴다
* write(byte[]) : byte[]에 담긴 내용을 한번에 쓴다
* flush() : write로 모아놓은 내용들을 모두 내보낸다.
* close(): 다 사용한 스트림을 닫는다. 닫지 않으면 계속 메모미를 차지하고 있게 된다., 남은 데이터가 있다면 flush()를 한다.



해당 폴더에 있는 파일 목록 출력

\*https://codechacha.com/ko/java-list-files/ 자세한건 여기 사이트에서 확인!



21.05.13

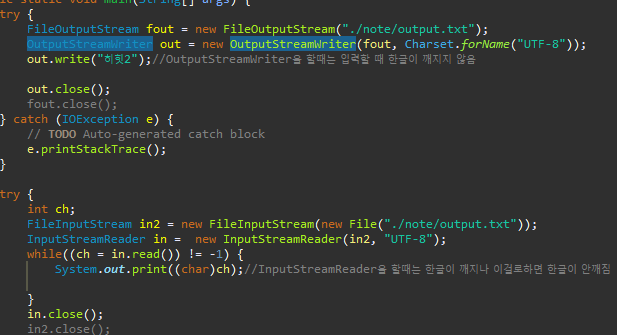
Collections.frequency(객체를 담고 있는 컬렉션 인스턴스, 찾고 싶은 객체) 🡪Collection 안에 있는 객체가 몇번 등장했는지 리턴해줌

**# Reader/Writer**

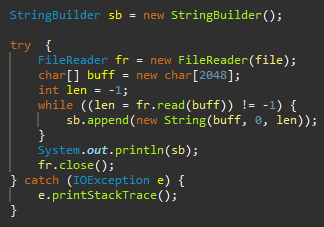
* InputStream과 OutputStream은 데이터를 바이트 단위로 읽고 쓴다.
* 바이트 단위로 데이터를 사용하기 때문에 원하는 타입으로의 재구성이 필요하다
* Reader와 Writer는 데이터를 문자 단위로 읽고 쓰는 클래스들이다.
* InputStream/OuputStream을 Reader/Writer로 한번 더 감싸서 사용한다.

**# abstract class Reader**

* BufferedReader, InputStreamReader, FileReader… 등은 모두 Reader 클래스를 상속받아 구현할 클래스들이다.
* Stream으로 읽은 데이터를 문자로 재구성하여 프로그래머가 사용하기 편하게 해준다.(한글이 가능)
* 문자로 합치거나 바이트로 변환할 때 사용할 인코딩 타입을 결정할 수 있다.



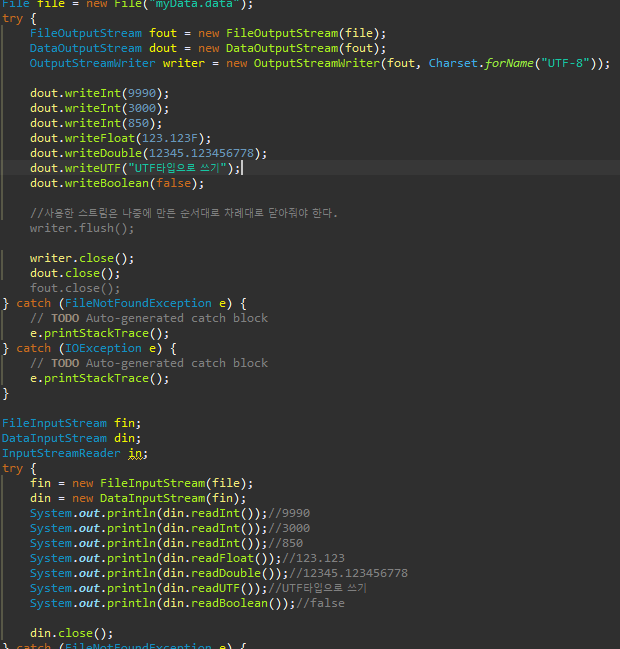
🡪단 밖에 것만 close해도 안에 있는 것도 자동 close 됨



🡪배열(char[])로 텍스트를 읽는 방법

**# DataInputStream/DataOutputStream(주로 이걸 쓸거임)**

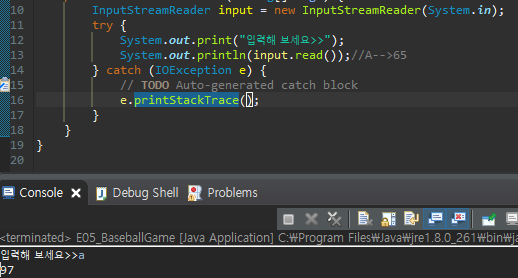
* Reader/Writer는 Stream을 문자로 손쉽게 다루기 위한 클래스였다면   
  DataInputStrea/DataOutputStream은 기본형 타입으로 손쉽게 다루기 위한 클래스다(단 저장될 때에는 데이터 형식으로 깨져서 보임)
* writeInt로 썼던 것은 readInt로 읽어야 한다(즉 쓴 형식에 맞게 읽을 때도 형식에 맞게 읽어야함)
* **문자열을 입력 및 출력은 writeUTF/readUTF 메서드(위의 방법은 char를 쌓아야 하나 이건 String 형식을 그대로 받아올 수 있음)**



21.05.14

System.in도 inputStream이다

따라서 InputStreamReader sc = new InputStreamReader(System.in)도 가능



**#Serializable interface**

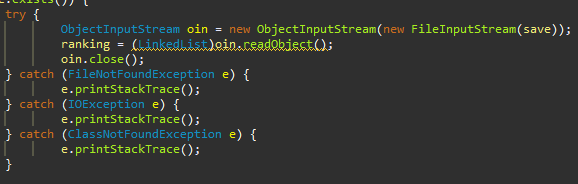
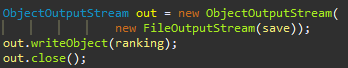
* **해당 클래스로 생성된 인스턴스를 직렬화(코드화 : 0과 1로 바꾸는 것) 할 수 있다고 표시하는 인터페이스(즉 Serializable 인터페이스를 달아놓으면 직렬화 가능함**
* serialVersionUID에 원하는 값을 넣으면 된다.
* **ObjectOutputStream으로 자바 객체를 파일에 저장할 때 해당 객체를 직렬화(코드화) 할 수 있어야 한다.**
* 직렬화 코드 ***serialVersionUID*** 일정해야함 처음에 직렬화해서 보낼 때 직렬화 값이랑 추후에 직렬화 값이 다르면 오브젝트를 불러올 때  
  오류가 뜸(암호 같은 느낌)



* 즉 ***serialVersionUID*** 의 값이 1L인 상태에서 직렬화 했고 추후에 값을 수정하면 오류가 뜸

**\*ObjectOutputStream**

* **오브젝트 타입 자체를 파일에 저장하는 기능임(즉 arrayList나 hashSet, hashMap, 클래스의 인스턴스 등을   
  통째로 파일에 저장 및 출력이 가능)**
* 단점은 Java가 아닌 언어에 대해서는 그 오브젝트를 불러올 수 없음
* **리턴타입이 Object이므로 다운캐스팅을 통해서 가져와야 함**



**# JSON**

* JavaScript의 Object타입과 굉장히 유사한 형태의 데이터 표현 방식
* 문자열만으로 다양한 데이터를 간결하게 표현할 수 있어서 **인기가 많다**
* {“Key” : Value, “Key” : Value, …}
* 프로그래밍 언어의 구애받지 않음
* JSON 안에 JSON이 들어갈 수도 있음
* 즉 내가 JSON을 만들어놓고 가져다 쓰는 거임  
  JAVASTUDY에서 E07\_JSON을 보면 map을 문서로 바꾸거나 문서에 있는 데이터를 map으로 불러올 수 있게 만들었음

**# JSON의 Value에 표현 가능한 타입들**

* ‘’, “” : String
* 숫자, 소수 : Number
* [] : Array
* {} : Object (JAVA의 Map)

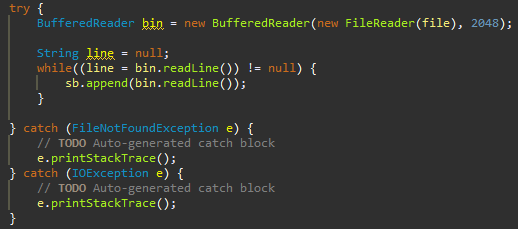
21.05.17

**A instanceof Object[] : A의 인스턴스가 배열인지 판단해주는 거임**

Try auto close ( Autoclose 인터페이스가 구현되어 있어야 함) : 자동으로 close(); 해주는 거임  
try(OutputStreamWriter dataOut = new OutputStreamWriter(new FileOutputStream(file))) {

StringTokenizer st = new StringTokenizer(String, “분해할 문자열”);->문자열 split을 좀 더 객체지향적으로 활용할 수 있는 클래스

\*BufferedReader로 파일을 읽어오는 방법 (\*얘가 배열로 받는 것 보단 더 좋음)



\*BufferedWriter 로 파일을 쓰는 방법



**#암호화**

* 평문을 알아볼 수 없는 형태(암호문)로 변환하는 것
* 전달 과정에서 내용을 누가 훔쳐가더라도 알아볼 수 없게 하는 것이 목적이다.

**#복호화**

* 암호화 된 암호문을 다시 평문으로 되돌리는 것

**#키**

* 암호화 또는 복호화에 사용되는 값

**#시저 암호**

* “Hello, world!” ---- key:5 --🡪 “Mjqqt, bqwpi!”  
  ABCDEFGHIJKLMNOPQRSTUVWXYZ
* 암호화 : key만큼 오른쪽으로 문자를 이동

21.05.18

**-Character.isAlphabetic(char) : char이 알파벳인지 검사해주는 메서드(bollen 리턴)**

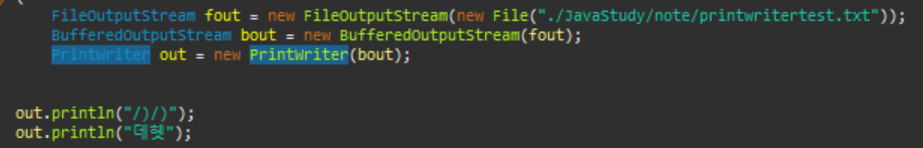
**-Character.getType(char) : char이 어떤 타입인지 (int 리턴-대문자 : 1, 소문자 : 2)   
(**Character.***UPPERCASE\_LETTER*: 1,** Character.***LOWERCASE\_LETTER*: 2)**

🡪위 내용을 활용한 거임



**#PrintWriter**

* Writer를 한 번 더 감싸서 편리한 기능(print 메서드)까지 추가해놓은 클래스
* Printf(), println()등이 구현되어 있다



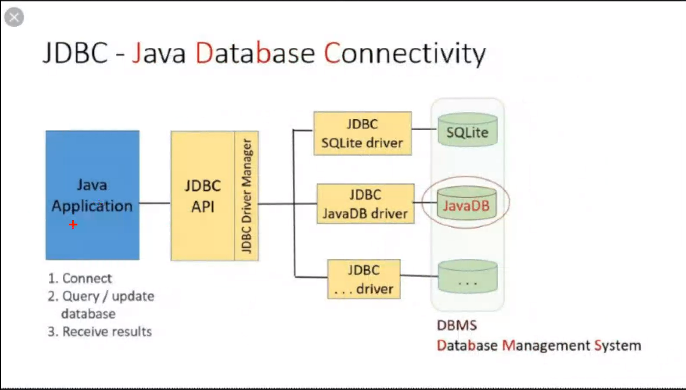
**#PrintStream**

* Stream을 한 번 더 감싸서 편리하게 print를 할 수 있는 클래스  
  (Stream을 감싸지 않으면 123을 넣으면 문자로 인식하여 넣을 수도 있음)

21.05.31

**# JDBC**

* Java Database Connectivity
* DB에 쿼리문을 전달하여 실행하고 결과를 받아오는 Java API
* 어떤 DB와도 호환할 수 있도록 설계되어 있다
* Java 부분의 소스는 어떤 DB를 사용하더라도 항상 일정하다
* DB를 제작한 쪽에서 jdbc에서 요구하는 형식을 맞추어 개발하기 때문에 프로그래머들은 새로운 DB도 기존의 jdbc로 편리하게 사용할 수 있다.



**# OJDBC**

* 오라클 데이터베이스가 JDBC API와 소통하기 위해 만든 라이브러리
* 오라클 DB가 설치된 폴더에 포함되어 있다

프로젝트 우클릭-Build Path- Configure Bulid Path - Libraries -Add External JARs - sqldeveloper/jdbc/lib에 잇는 jar를 추가  
🡪라는 폴더가 해당 프로젝트 안에 생성됨  
🡪이게 있어야 오라클과 연동 가능(없어도 되긴 하나 경로가 복잡함)

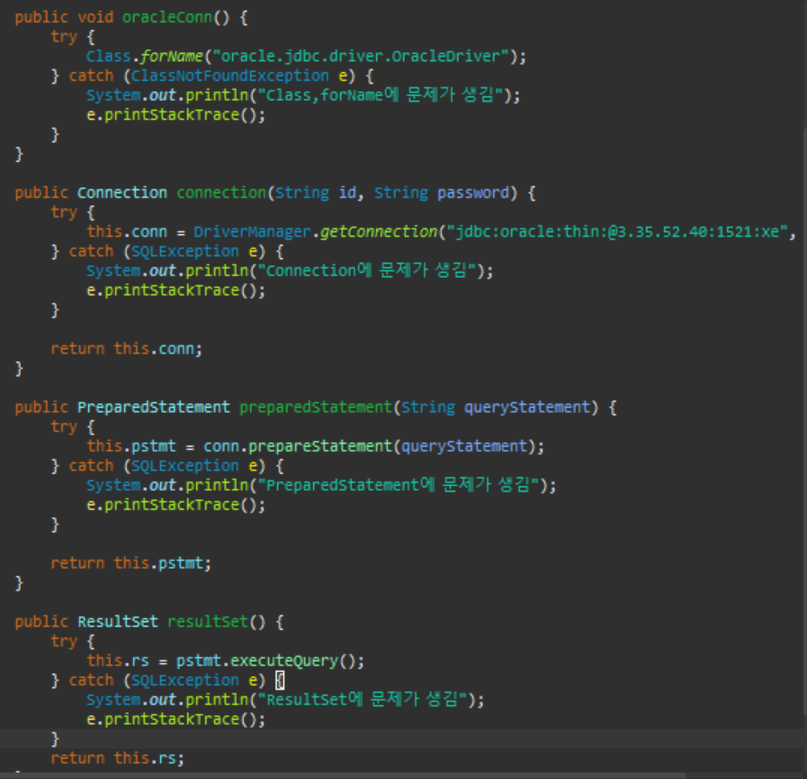
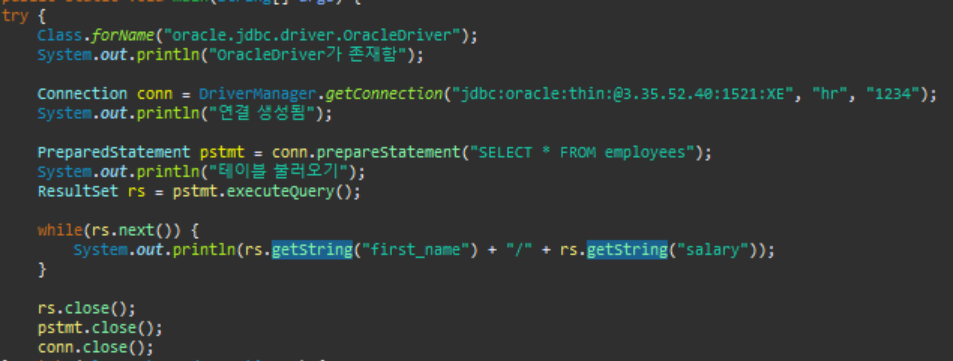
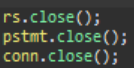


\*어떤 회사 DB를 사용하더라도 거쳐야 하는 똑 같은 절차

1. Class.forName(driverName)으로 해당 DB사의 jdbc드라이버를 로드한다(driverName에 class이름을 써서 클래스도 불러올 수 있음)  
   Class.*forName*("oracle.jdbc.driver.OracleDriver");  
   **🡪생략가능!!**
2. DriverManager 클래스를 통해 DB와의 연결을 생성한다  
   DriverManager.*getConnection*(url, user, password);  
   Connection conn = DriverManager.*getConnection*("jdbc:oracle:thin:@3.35.52.40:1521:XE", "hr", "1234");
3. 생성된 연결을 통해 전송할 쿼리문을 생성한다  
   PreparedStatement pstmt = conn.prepareStatement("SELECT \* FROM employees");
4. 생성된 쿼리문을 DB로 보낸다  
   ResultSet rs = pstmt.executeQuery();
5. 받아온 결과를 마음대로 사용한다   
     
   **🡪즉 rs.getString(“컬럼이름”);을 통해 순차적으로 받아온다**

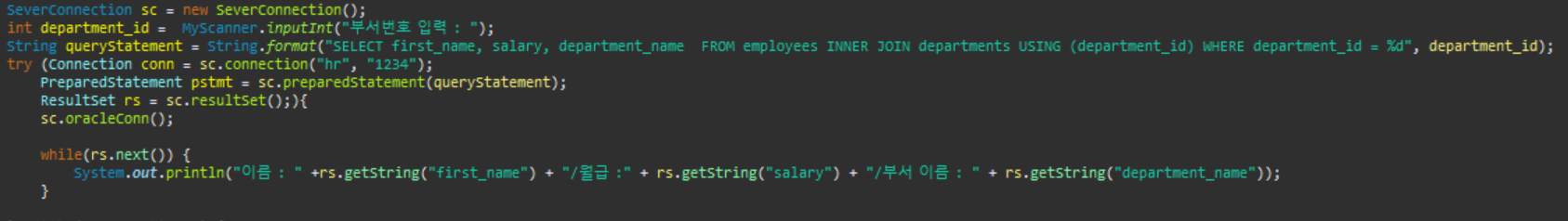


1. 다 썼으면 연결을 잘 끊어준다(가장 나중에 열었던 순서대로 닫는다  
     
   예시)  
    🡪서버 관련 메서드들을 만든거임

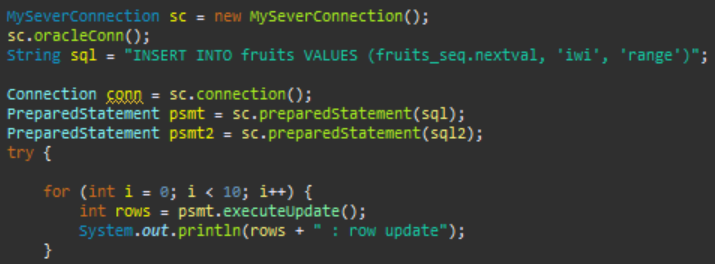


\*JDBC에서 SELECT는 ResultSet을 반환한다.그 외의 DML(INSERT, UPDATE, DELETE)는 적용된 행의 개수를 반환한다

🡪이를 통해   
**이와 같이 만들어진 메서드를 사용하며 try close를 이용하여 자동으로 close하게 한다!  
또한 컬럼의 순서를 안다면**System.***out***.println("이름 : " +rs.getString(1) + "/월급 :" + rs.getString(2) + "/부서 이름 : " + rs.getString(3));  
**처럼 컬럼 이름을 넣는 것이 아닌 컬럼 순서 넘버로 출력이 가능하다**



\*SELECT는 PrepparedStatement의 executeQuery()메서드를 사용하고 나머지는 executeUpdate()메서드를 사용한다



🡪입력은 executeUpdate로 하며 1행이 입력될 때마다 1이 반환됨

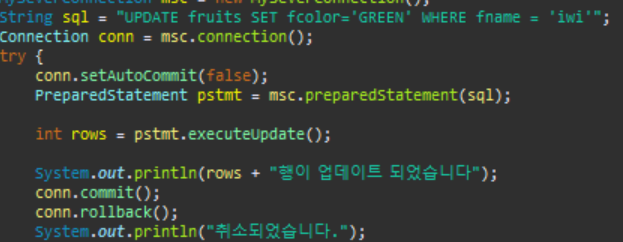
또한 시퀀스를 통해 FID를 1씩 증가시키며 입력(이때 나머지 데이터는 동일하게 들어감…)

\*JDBC로 처음 열결을 생성하면 기본적으로 AutoCommit이 활성화 되어있다.

\*트랜젝션을 직접 제어하기 위해서는 자동 커밋을 비활성화 해야 한다.  
conn.setAutoCommit(false);  
**🡪따라서 트랜잭션을 직접 제어하기 위해서는 위와 같은 코드는 필수**

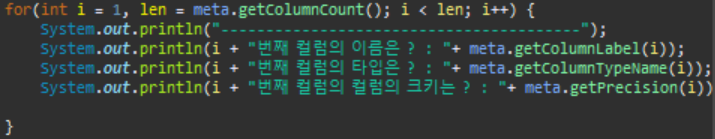
conn.commit();🡪커밋해주는 코드

conn.rollback();🡪롤백해주는 코드



21.06.01

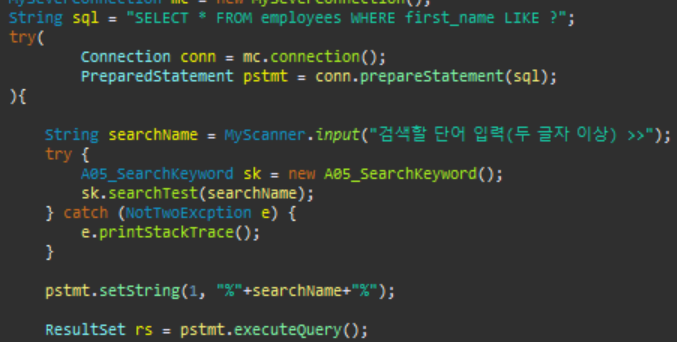
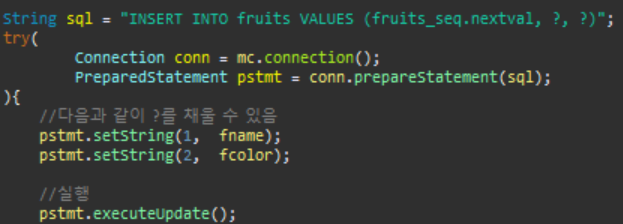
\*가져온 데이터의 메타 데이터들을 보고 싶을 때  
ResultSetMetaData meta = rs.getMetaData();  
System.***out***.println("컬럼이 몇 개인가? : "+ meta.getColumnCount());  
  
\*getColumnLabel : 컬럼의 이름, getColumnTypeName : 컬럼의 타입, getPrecision : 컬럼의 컬럼 크기(NUMBER(5)이면 크기는 5)



\*   
🡪이렇게 쓰면 보완성에 취약함



**\*PrepareStatement에 준비시킬 쿼리문에 변수가 필요하다면 변수자리는 ?로 채운다  
  
🡪단 setString으로 넣을땐 자동으로 ‘’처리를 해줌!**



**#ConnectionPool**

* DB에 DML을 적용할때마다 새로운 접속을 생성하는 것은 리소스가 많이 낭비된다.
* 연결을 이미 여러 개 생성해두고, 사용중이지 않은 커넥션을 꺼내주는 방식을 커넥션 풀이라고 한다.

**\*구글-hikariCP-밑으로 내려가면 OR download from here 들어가서-최선꺼 jar 다운-넣고 싶은데 넣기**  
(난 C:\Users\xoghk\OneDrive\바탕 화면\공부\국비\git\FirtsRepo\JavaStudy\files다가 넣음)

그리고

HikariConfig config = new HikariConfig();

config.setJdbcUrl("url");

config.setUsername("id");

config.setPassword("pw");

config.addDataSourceProperty("cachePrepStmts", "true");

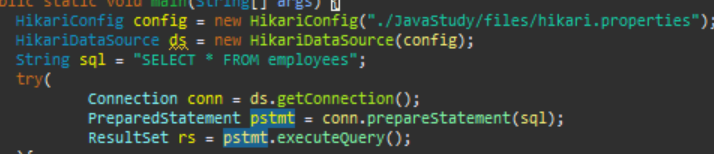
config.addDataSourceProperty("prepStmtCacheSize", "250");

config.addDataSourceProperty("prepStmtCacheSqlLimit", "2048");

를 복사해서 붙여아함

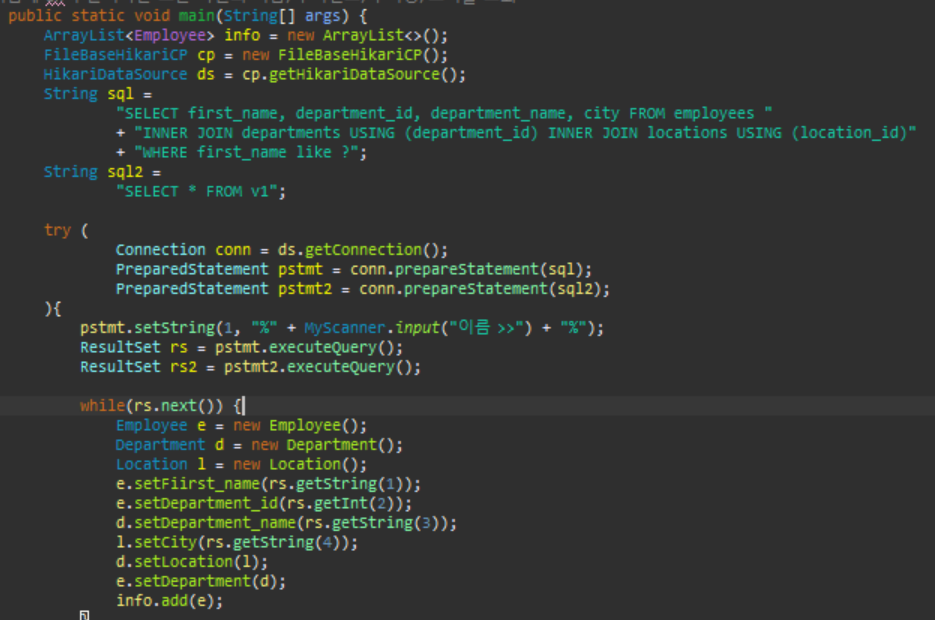
HikariDataSource ds = new HikariDataSource(config);

그리고 org.slf4j.LoggerFactory 라이 브러리를 다운 받아야함  
<https://mvnrepository.com/artifact/org.slf4j/slf4j-api/1.7.30>  
🡪여기 있음  
--slf4j : 로그를 찍는 용도로 사용하는 인기 라이브러리  
  
🡪이렇게 쓰는 거임



21.06.02

\*뷰를 통해 원하는 데이터들로만 만들어서 모델 클래스에 넣기  
  
🡪조인을 하여 Employee, Department, Location 모델 클래스에 넣는 방법(Employee안에 Department 클래스가 존재하고 Department 안에 Location 클래스 존재)  
🡪혹은 pstmt2는 뷰를 만들어 원하는 정보만 얻는 방법



\*tip – 원래 모델 클래스를 상속받아서 뷰에 맞는 생성자만 만들어 내는 방법도 있음  
  
🡪원래 모델 클래스를 상속받은 자식 클래스  
🡪

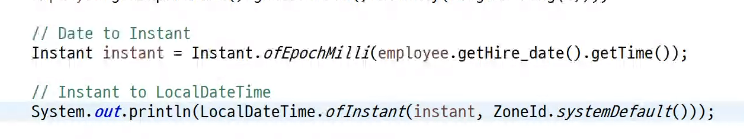


🡪자식 클래스에 뷰의 내용을 집어넣어서 toString으로 출력  
🡪  
🡪다시 공부하기(구글에 date to localdatetime 검색 ㄱㄱ)



Instant instant = Instankt.ofEpochMilli(employee.getHire\_date().getTime()));

LocalDataTime.of(Instant(instant, ZoneId.systemDefault()));



**#Swing**

* Java로 윈도우 GUI 프로그램을 만들 수 있는 API

**#Container**

* 컴포넌트를 추가할 수 있는 판
* Jframe, Jpanel, Jdialog …등
* 컨테이너마다 원하는 레이아웃을 지정할 수 있다.

**#Component**

* 컨테이너에 추가할 수 있는 부품
* Jbutton, Jtable, Jslider, Jmenu …등

**#Jframe**

* 프레임 컨테이너. 다양한 컴포넌트들을 추가할 수 있다.

**Jframe을 상속 받고 하자**  
public class S02\_JButton extends JFrame{

\*X버튼을 눌렀을 때의 동작 설정  
setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

\*프레임을 보이게 설정  
setVisible(true);

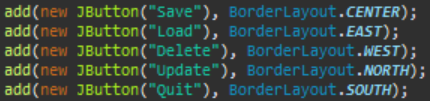
\*프레임 크기 설정  
setSize(600, 600);

\*위치 설정  
setLocation(200, 50);

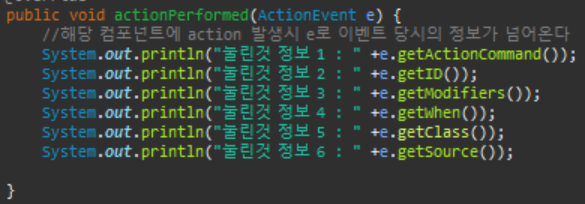
\*프레임의 레이아웃 설정  
setLayout(new BorderLayout());  
(border말고 다른 것도 있음)

**#BorderLayout**

* 동서남북과 가운데로 이루어진 레이아웃
* 컴포넌트를 추가할 때 위치를 함께 지정해 줄 수 있다
* 수평 여백과 수직 여백을 설정할 수 있다.  
    
  quit\_btn.setFont(new Font("궁서체", Font.***BOLD***, 35));  
  🡪이런식으로 각 컴포넌트를 추가할 수 있으며 컴포넌트에 대한 설정도 하면 됨



\*버튼에 이벤트를 추가  
quit\_btn.addActionListener(new ActionListener()

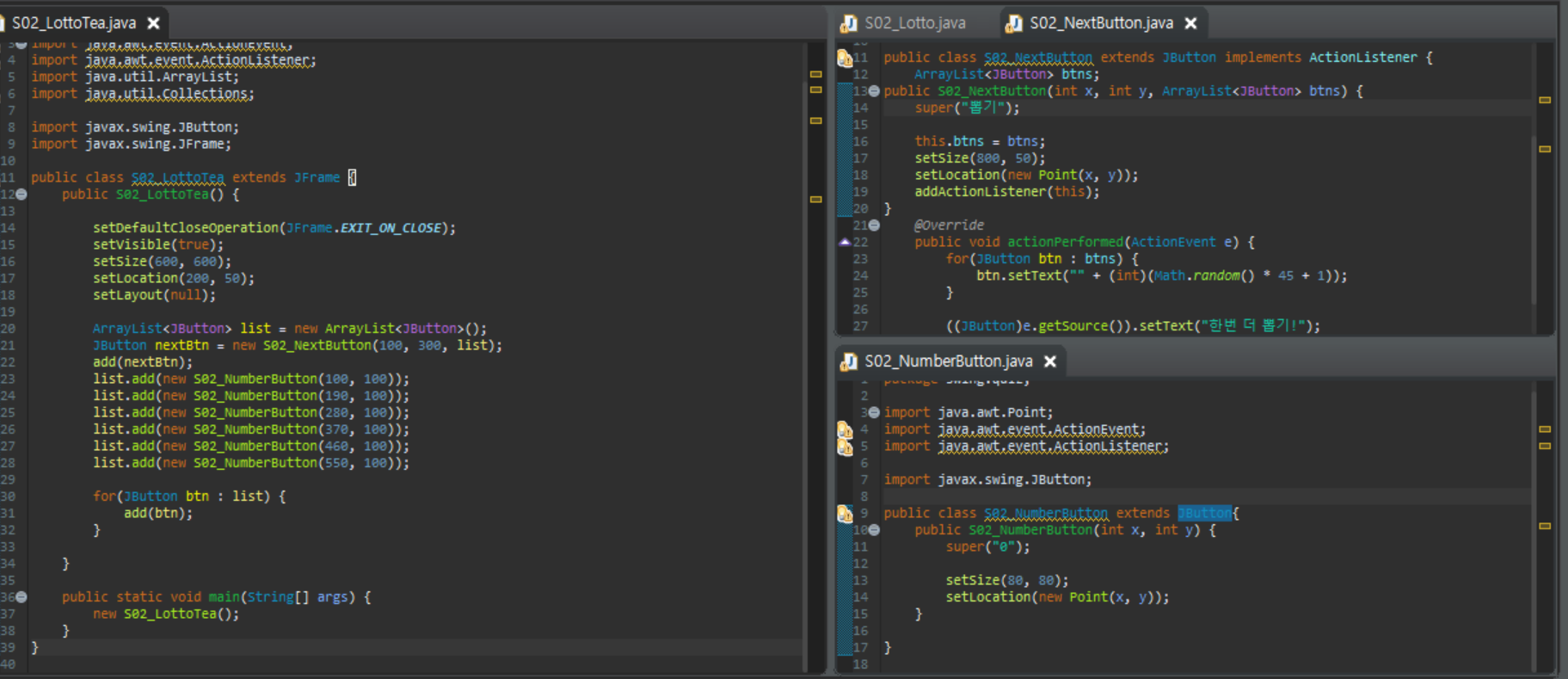


\*e.getSource()에는 액션이 발생한 컴포넌트의 인스턴스가 들어있고 다운캐스팅 하여 사용해야 한다  
((JButton) e.getSource()).setText("눌렸음");

21.06.03  
  
🡪버튼을 누를때마다 버튼 6개에 랜덤 숫자가 적힘



\*ActionListener의 기능에 대해서는 따로 클래스를 만들어서 구현을 하자



위에 대한 설명

설명 1. S02\_NumberButton과 S02\_NextButton은 Jbutton을 상속받았기에 Jbutton nextBtn = new S02\_NumberButton이 가능함

설명 2. S02\_NextButton는 ActionListener라는 인터페이스를 상속받았기에 그 클래스 자체에 클릭하였을 때에 액션 기능이 포함되어 있음

설명 3. addActionListener(this) 의 this는 S02\_NextButton의 생성자를 의미하므로 actionPerformed를 의미한다 즉   
addActionListener(this) 는 를 의미하는 거임

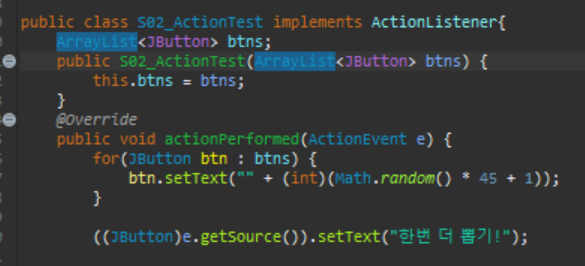


this를 하게되면 자기 자신 주소에 actionPerformed를 알아서 찾게 됨

이와 같은 방법은 NextButton 클래스에 버튼 기능까지 추가하였지만

애초에 ActionListener 인터페이스를 상속받은 클래스를 만들어 거기서 actionperformed기능을 만들고 그 클래스를 nextbutton이 상속받으면 그게 더 괜찮음!

즉  
클래스를 만들고  
| 를 하면 됨



\*.setBound(x, y, whidth, height) : 버튼의 위치와 크기 설정을 한번에 한다

\*.setText() : 해당 컴포넌트의 텍스트를 변경

\*setEnabled(T/F) : 해당 컴포넌트를 활성/비활성화 한다

\*.setBackground(new color(255, 255, 0) : 버튼의 배경색

\*.setIcon(new ImageIcon(경로)) : 해당 컴포넌트의 아이콘을 설정한다

\*BufferedImage의 getSubimage(x, y, width, height);는 이미지를 자름

\*JtextField : 텍스트를 표시하는 곳  
🡪텍스트 입력 후 <ENTER>키를 입력하면 Action 이벤트가 발생하는 거임



21.06.04

JTable table = new JTable(10, 4);



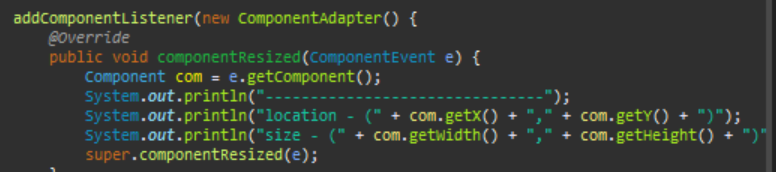
\*ComponentListener에 대해서 new ComponentListener() 을 하면 아래처럼 모든걸 다 설정해 줘야함

하지만 addComponentListener(new ComponentAdapter()을 하게 되면



위 메서드 중 원하는 메서드 한 가지만 구현해도 됨

🡪컴포넌트의 위치 및 크기가 달라질때마다 이벤트가 발생해 출력하게 함



addMouseListener(new MouseAdapter() 도 마찬가지 마우스 움직임에 대해 설정 가능  
(각 메서드에 대한 기능은 구글링 ㄱㄱ)

1\*GridLayout

setLayout(new GridLayout(2, 5, 15, 5)); d  
🡪(줄 개수, 행 개수, 가로 간격, 세로 간격)



**# Jpanel**

* 컨테이너 역할을 하는 컴포넌트
* 프레임 위에 붙여서 사용하는 컨테이너
* 레이아웃도 설정할 수 있다.



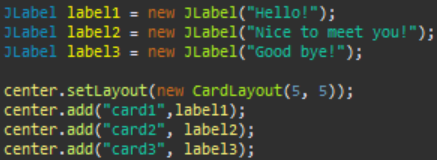
**#CardLayout**

* 컴포넌트를 카드처럼 넘기면서 사용하는 레이아웃

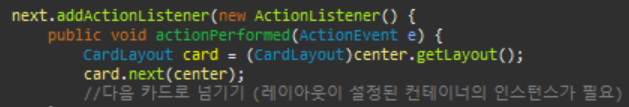
new CardLayout(5, 5)   
🡪(왼쪽 gap, 오른쪽 gap)

-카트 레이아웃에 컴포넌트를 추가할 때는 컴포넌트의 이름을 지정해 놓는 것이 좋다

**#Jlabel : 내용을 출력하는 곳**



🡪next라는 버튼을 누르면 label이 순차적으로 넘어감  
(cardlayout위에 jpanel을 여러 개 추가하면 버튼 누를때마다 panel이 넘어가게 됨)



card.previous : 이전 카드 보기, card.show(parent, card\_name) : 원하는 카드 보기  
card.first, last도 있음

🡪만약에 화면 전체를 넘기고 싶으면 JFrame안에 Jpanel을 만들어서 거기에다가 레이아웃을 정하고 분리하고 해야함!  
(Frame은 못넘김)

\*BorderLayout에서 각각의 상, 하, 좌, 우의 크기를 조절 하려면 bottom.setPreferredSize(new Dimension(1200, 300)); 이다

\*JLabel에서 배경색을 주려면 label5\_1.setOpaque(true);을 선언 하고 색을 label5\_1.setBackground(Color.***blue***);주어야 한다.

\*JLable에서 텍스트 좌.가,우 맞춤은 setHorizontalAlignment(JLabel.***RIGHT***);

\*글자 색 setForeground(Color.***RED***);

.setBorder(BorderFactory.*createEmptyBorder*(10, 90, 10, 90));  
🡪내부 공백을 주는 코드

\*Jbutton 테두리 라운드는 <https://leirbag.tistory.com/15>를 참고!

**\*싱글톤 패턴**-이미 선언된 다른 패키지에 있는 변수 및 Jlabel 등등을 쓸 수 있는 디자인 패턴

1. 싱글톤을 사용한 변수에 대해서  
private static 변수유형 변수명을 선언  
private static JLabel *memberNameValue*;

2.

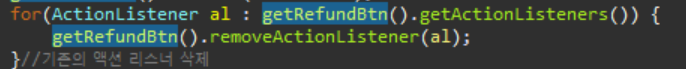


이렇게 하면 다른 패키지에 대해서도  
EastPayPanel.*getMemberNameValue*().setText(total\_payment.getText());

**을 가져와 쓸 수 있음!(이거 존나 중요)**

\*각종 기능들을 오버라이딩하여 액션 리스너를 바꾸는 방법

1. 기존의 액션리스너를 삭제한다  
  
🡪이때 리스너를 삭제하려는 개체에 대해서 return을 해주는 get메서드가 필요함



2. 바꾸려는 리스너를 넣어준다  
getRefundBtn().addActionListener(new ReceiptAction(this));

\*팝업

JOptionPane.*showMessageDialog*(null, "배달하려면 3만원 이상과 마트 회원이여야 합니다.");

JLabel.isVisible()  
🡪현제 컴포넌트가 보여져있는지 bollean형태로 반환하는 메서드

\*테이블 만드는 법

\*Jtable을 클릭했을 때 클릭한 행(row)의 번호를 알려주는 코드  
((JTable) e.getSource()).getSelectedRow();

\* TopTable.*getDefaultModel*().setNumRows(0);//테이블 초기화  
🡪테이블 초기화