JAVA & Eclipse & Spring

블럭설정 후 ctrl shift F = 들여쓰기

블럭설정 후 ctrl shift / = 주석처리

블럭설정 후 alt 위아래방향키 = 줄바꾸기

위에서 Scanner 변수명 = new Scanner(System.in); 선언 후 아래서 변수 선언(정수는 int x = 변수명.nextInt(); / 문자는 char x = 변수명.next().charAt(0); / 문자열은 String x = 변수명.next();)하면 변수 선언 자리에 콘솔에서 키보드 입력 가능

System.out.printf(“/n”); / System.out.println(“”); 둘다 가능

2진법으로 표현된 숫자를 3개씩 끊어 뒤에부터 계산하면 8진법이고, 4개씩 끊어 계산하면 16진법이다. 그 반대는 반대로 계산하면 된다.

실수의 진법 변환 방법은 48페이지

2의 보수 = 1의 보수 + 1(2진법에서 마지막 자리만 빼고 전부 반대로 바꾸기 = 전부 반대로 바꾸고 마지막에 1만 더하기)

비트에서 맨 앞비트는 MSB(Most Significant Bit)라고 부른다.(0이면 양수, 1이면 음수)

부동소수점 = 실수 (부호 / 지수(정수부) / 가수(소수부))

(int)(Math.random() \* 100) + 1은 0과 1 사이의 무수한 더블형 소수를 임의로 배정하고, 100을 곱한 후(int)형으로 소숫점을 삭제하여 정수만 남긴다. 0부터 시작하기때문에 마지막에 + 1을 해준다.

변수 선언을 하면서 바로 값을 대입하는걸 초기화라 하고, 선언 후에 값을 따로 대입하는 것을 대입이라 한다.

(조건)?참일 때:거짓일 때 -> 3항연산자(if ~ else문과 같은 의미지만 문자에 의한 처리가 아니라 기호에 의한 처리이다)

0b: 2진법

0: 8진법

0x: 16진법

null은 문자열의 끝을 의미하고, 문자열의 첫글자가 대문자면 초기화를 null로 해줘야 한다.

String name2 = name1 + 8.0; // 문자열 + 다른 자료형 = 문자열로 자동캐스팅됨.

System.***out***.printf("finger = [%5d]%n", finger); // 5자리를 만들고 마지막부터 값을 채움(우측정렬)

System.***out***.printf("finger = [%-5d]%n", finger); // 5자리를 만들고 처음부터 값을 채움(좌측정렬)

System.***out***.printf("finger = [%05d]%n", finger); // 5자리를 만들고 마지막부터 값을 채우되, 공백 부분은 0으로 채움

System.***out***.printf("hex = %#x%n", hex); // x가 소문자면 f도 소문자로

System.***out***.printf("hex = %#X%n", hex); // X가 대문자면 F도 대문자로

System.***out***.printf("octNum = %o, %d%n", octNum, octNum);

System.***out***.printf("hexNum = %x, %d%n", hexNum, hexNum);

System.***out***.printf("binNum = %s, %d%n",Integer.*toBinaryString*(binNum), binNum); // 해당 메서드를 사용해 10진수를 2진수로 변환하면, 문자열로 변환되므로 %s를 사용했다

System.***out***.printf("d = %f\n", d); // 기본적으로 소수점 7째 자리에서 반올림해서 6째 자리까지 표현

System.***out***.printf("d = %14.10f\n", d); // .을 포함하여 총 14자리를 만들고 그중 소수점을 10자리까지 표현

String x = sc.next(); // 공백이 포함되어 있으면 안됨

String y = sc.nextLine(); // 공백이 포함되어 있어도 가능

**long** a = 1\_000\_000 \* 1\_000\_000; // 계산값(int)형이 이미 오버플로우 되었고 그 값을 long에 대입해봤자 에러값임

**long** b = 1\_000\_000 \* 1\_000\_000L; // 앞쪽 백만이 int형이였다가 뒤쪽 백만에 의해 자동으로 long형으로 캐스팅 된 후 계산되기 때문에 정상값임

char c2 = c1 + 1; // c1 + 1는 int타입이고 이를 char타입인 c2에 대입하려고 해서 에러발생

**char** c2 = 'a' + 1; // 위의 c1은 변수지만 'a'자체는 상수(리터럴)이기 때문에 컴파일러가 알아서 계산이가능하다.

**float** shortPi = (**int**)(pi \* 1000) / 1000f; // pi \* 1000 = 3141.592f; -> (int)형에 의해 소수점이 떨어지고 3141이 됨 -> 1000f로 나누면 float형으로 자동 캐스팅되면서 3.141f이 된다.

**double** pi = 3.141592;

**double** shortPi = Math.*round*(pi \* 1000) / 1000.0; // 0.5를 안더해줘도 알아서 반올림하는 함수

System.***out***.printf("몫은 %d이고 나머지는 %d입니다%n", x / y, x % y);

// %연산 할때 피연산자 두개 중 앞쪽에 의해 부호가 결정된다. 즉 뒤쪽 피연산자의 부호는 무시된다. ex) -10 % 8 = -2, -10 % -8 = -2

**if**(order.equals("yes") || order.equals("YES")) // order.equals()함수는 order와 "yes"를 비교해서 참이면 true, 거짓이면 false를 출력한다. 여기서 if안의 조건문은 order.equalsIgnoreCase("yes")와 같은 의미이다.

//연산자 계산 우선순위: 대입<논리<비교<산술(이항<단항)

// <<(좌 쉬프트), >>(우 쉬프트)는 산술 연산자지만, +-\*/보다 우선순위가 낮다.

System.***out***.println(x << 2 + 1); // x의 값 5를 2 + 1비트만큼 즉, 3비트만큼 를 좌측으로 옮겨라. 계산결과는 5를 2진수로 표현한 101(2)를 전체적으로 좌측으로 3비트 옮겨서 101000(2) => 십진수로 40이 된다.

System.***out***.println(x < -1 || x > 3 && x < 6); // And &&이 Or ||보다 우선순위가 높다.

// ~는 2진수 1과 0을 전부 1의 보수를 취하는 기호이다. !는 반대의 의미이다.

j = i++; // 대입연산자 =를 먼저 계산하고 단항연산자++이 계산됨.

j = ++i; // 단항연산자 ++이 먼저 계산되고 대입연산자 =가 계산됨.

// 다른 수식이나 메서드와 같이 쓰이지 않고 단독으로 쓰면 선증가나 후증가나 결과는 같다.

**byte** a = 10, b = 20, c = (**byte**)(a \* b); // a + b에서 \*가 int형이므로 byte타입에 대입하려면 강제캐스팅 해줘야함 하지만 결과가 200, 즉 바이트의 범위를 넘어가기 때문에 값 손실이 난다.

String tmp = sc.nextLine(); // 스캐너로 입력받은 문자열 내용을 문자열 tmp에 저장

input = Integer.*parseInt*(tmp); // 문자열 tmp를 int형으로 변환

//input = sc.nextInt(); 해도 똑같음.

String regNo = sc.nextLine();

**char** gender = regNo.charAt(7); // regNo의 8번째 값을 문자형 gender에 저장함.

**for**(**int** k = 1; k <= 3; k++)

{

System.***out***.println(""+i+j+k);

// ""를 붙여주는 이유는 String형이 int형보다 큰 크기라서 String으로 자동캐스팅되어 i j k가 숫자형태로 더해지지 않고 문자형태로 붙어서 출력됨.(문자열 + 다른 타입 = 문자열)

}

aaa : **while**(**true**) // 반복문에 aaa라는 라벨을 지정. c는 불가능하지만 java에선 가능하다. 후에 break aaa; 해주면 aaa반복문을 종료하라는 뜻.

aaa: **for**(**char** c = 'A'; c <= 'C'; c++) // 반복문에 aaa라는 라벨을 지정. 후에 continue aaa; 해주면 aaa반복문으로 이동하라는 뜻.

**case** 1:

System.***out***.println("result = " + num \* num); // 제곱 구하기

**break**;

**case** 2:

System.***out***.println("result = " + Math.*sqrt*(num)); // 제곱근(4는 2의 제곱, 2는 4의 제곱근) 구하기

**break**;

**case** 3:

System.***out***.println("result = " + Math.*log*(num)); // 로그 값 구하기

**break**;

**int**[] arr = {10, 20, 30, 40, 50}; // int arr[]이랑 같은 의미임.

**for**(**int** i = 0; i < arr.length; i++) // 기존 배열을 for문에 적용하는 방법

**for**(**int** tmp : arr) // 향상된 배열을 for문에 적용하는 방법

answer = (**int**)(Math.*random*() \* 100) + 1; // Math함수의 결과는 문자열이기 때문에 (int)로 강제캐스팅 해준다.

Loop1: **for**(**int** i = 2; i <= 9; i++) // 외부 for문에 Loop1이라는 라벨을 붙임.

int arrayNumber[] = new int[3]; // 배열 선언, 객체 생성을 합쳐서 이렇게도 쓸 수 있다.

String city[] = {"서울", "대구", "부산"}; // stack공간만 있고 heap공간은 없는 배열 초기화

String city2[] = **new** String[]{"서울", "대구", "부산"}; // stack, heap 공간 모두 생성한 배열 초기화

여러 변수의 값이 같을 때 stack공간에서는 새로운 공간을 만들지 않고, heap공간에서는 새로운 공간을 만든다.

**int** array1[] = **new** **int**[3]; // 배열의 선언과 객체 생성

**for**(**int** i = 0; i < array1.length; i++) // array1의 배열 길이(3개)만큼 i를 반복하라.

**public** **static** **void** printInfo(String...infos) // ...의 의미는 String타입의 infos라는 배열을 선언함과 동시에 연속 생성하는 것. 나중에 printInfo라는 함수를 사용함으로써 배열이 자동 생성된다. 배열을 여러개 생성하기 귀찮기 때문에 이런식으로 연속 생성을 한다.

**public** **static** **void** printInfo(String[] infos) /// String...infos와 다르게 infos라는 배열을 선언만 해놓고 main함수에서 printInfo함수를 사용할 때 new String으로 배열를 직접 생성하여 값을 초기화하여야 한다.

String str[][] = **new** String[2][2]; // 2행 2열로 이해하면 편함. [0][0], [0][1], [1][0], [1][1] 네가지 공간 생성

ArrayList haksaList = **new** ArrayList(); // arraylist는 1차원 배열만 가능. 배열에 어떤 타입이던 섞어서 넣을 수 있어 편하지만 나중에 어떤타입인지 몰라 찾기가 힘들다.

ArrayList<String> haksaList = new ArrayList<String>(); // 그러한 문제 때문에 지네릭 <자료타입>으로 미리 배열에 들어갈 수 있는 타입을 지정해 놓는다.

**for**(**int** i = 0; i < haksaList.size(); i++) // 여기선 length가 아니라 size 메소드를 사용한다.

{

System.***out***.println(haksaList.get(i)); // 대괄호가 아니라 get함수로 배열 내의 값을 끌어온다.

}

HashMap<String, String> haksaHash = **new** HashMap<String, String>(); // hashmap은 다차원 배열 가능. 지네릭<>내부에 들어가는 타입 중 첫번째 값은 이름(key)이고, 두번째 값은 값(value)이다.

System.***out***.println(haksaHash.get("age")); // ArrayList와 마찬가지로 대괄호가 아니라 get함수로 배열 안의 값을 끌어온다.

ArrayList<HashMap<String, String>> haksaList = **new** ArrayList<HashMap<String, String>>(); // 1차원 배열 안에 2차원 배열을 넣은 유형

**int**[] arr = **new** **int**[0]; // 배열의 길이가 0인 배열도 만들 수 있다.

**int**[] arr2 = **new** **int**[] {}; // 배열의 길이가 0인 배열도 만들 수 있다.

System.***out***.println(Arrays.*toString*(iArr)); // Arrays.toString(배열이름) 사용 시 배열 내부의 값(엘리먼트)들을 []안에 문자열 타입으로 출력한다.

System.***out***.println(iArr); // 결과의 I는 1차원 int타입 배열이란 뜻이고, @ 뒤에 붙은 16진수는 배열의 주소이다.

System.***out***.println(chArr); // 예외적으로 char타입 배열은 컴파일러가 자동으로 동작하여 배열의 주소가 아닌 배열 내부의 엘리먼트들이 출력된다.

System.*arraycopy*(abc, 0, result, 0, abc.length); // abc배열의 abc[0]부터의 값들을 result배열의 result[0]부터 abc.length배열의 개수(길이) 만큼 복사한다.

avg = sum / (**double**)score.length; // 총점 / 배열의 길이(갯수)를 해주면 평균이 나온다. 이 때 double형으로 강제캐스팅을 안해주면, int형이 되니까 소수점 이하는 버려진다.

tmp = ball[i];

ball[i] = ball[j];

ball[j] = tmp;

//ball[i]와 ball[j]를 서로 바꿔주는 작업임

numArr[i] = (**int**)(Math.*random*() \* 10);

System.***out***.print(numArr[i]);

// System.out.print(numArr[i] = (int)(Math.random() \* 10);과 같은 의미이다.

**static** **boolean** *logic*; // main함수 밖에서 boolean변수를 선언하면 기본 초기화는 false로 지정된다.

**static** String *str*; // main함수 밖에서 string변수를 선언하면 기본 초기화는 null로 지정된다.

String titleContent = **new** String("제목|내용"); // 01234 번지에 각각의 문자가 들어간다

**int** indexI = titleContent.indexOf("|"); // titleContent배열에 |위치(2)를 찾아 그 인덱스를 indexI변수에 저장한다

String title = titleContent.substring(0, indexI); // substring에 의해 titleContent배열의 0번지부터 indexI(2)-1번지 까지를 출력.

String content = titleContent.substring(indexI + 1); // substring에 의해 titleContent배열의 index + 1번지부터 마지막 번지(5)까지를 출력.

String객체는 읽을 수만 잇을 뿐 기존 내용을 변경할 수 없다. 그렇기 때문에 새로운 공간을 만들거나 기존 존재하던 공간을 가리키는 일을 할 뿐이다. 변경 가능한 문자열 배열을 만들려면, StringBuffer객체를 사용해야 한다.

String str = **new** String(chArr); // char배열인 chArr를 String배열로 변환

**char**[] tmp = str.toCharArray(); // String배열인 str배열을 char배열로 변환해서 tmp배열에 저장

**char** ch = src.charAt(i); // src배열에서 i번지 주소를 가져와서 문자열 변수 ch에 저장하라.

String y = str.substring(0, 3); // str배열의 0번지 값 "A"부터 3번지 값"D"바로 직전까지 가져와서 변수 y에 저장한다.

**int**[][] score = **new** **int**[][] {{100, 100, 100}, {20, 20, 20}, {30, 30, 30}, {40, 40, 40}, {50, 50, 50}}; // 5행 3열의 2차원 배열

System.***out***.println(score.length); // 배열의 길이는 행의 길이다

System.***out***.println(score[0].length); // 배열 속 배열의 길이는 열의 길이다

**int**[][] score = **new** **int**[2][]; // 행만 지정하고 열은 지정하지 않는다

score[0] = **new** **int**[1]; // 0번째 행의 열은 1개로 지정

score[1] = **new** **int**[2]; // 1번째 행의 열은 2개로 지정

//score배열의 총 개수는 3개가 된다

개발단계 – 오버로드(메서드명 중복기술)

유지보수단계 – 오버라이드

Ctrl Shift O: 필요한 임포트가 자동으로 되고, 쓸모없는 임포트는 사라진다

새로운 라이브러리 추가하기: 프로젝트 우클릭-build path-labraries-add external jars

부모가 2개인 다중상속은 불가.

/\*\*

\* 도움말주석. Javadoc으로 API생성시 포함되는 주석임. 그때그때 작업하면서 써놓는게 좋다. 한글 적용하려면 javadoc utf-8로 인코딩을 바꿔야 함

\*

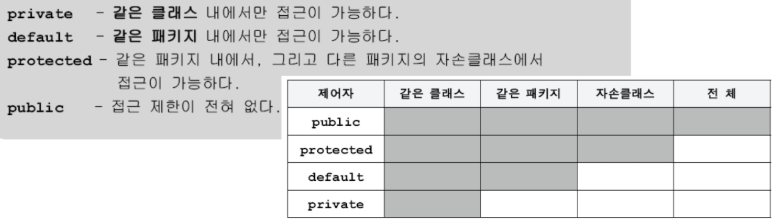
\*\*/

String타입의 변수명.contains(“a”) -> 변수 내에 a라는문자열을 포함하면 true를 리턴, 아니면 false를 리턴

final이 클래스 앞에 붙으면 다른 클래스에서 상속금지,

final이 메서드 앞에 붙으면 자식(상속한) 클래스에서 오버라이딩 금지,

final이 변수 앞에 붙으면 변경 금지.



부모타입에 자식타입을 대입하고 부모를 호출하면, 변수는 부모게 호출되고 메서드는 자식게 호출된다

a instanceof B: a를 B에 대입할 수 있는지 물어봄. 가능하면 true반환, 불가능하면 false반환

회사의 서버 버전과 내 컴퓨터의 jdk버전을 맞춰주기

인터페이스 클래스: 파이널스태틱 상수와 추상메서드만으로 이루어진 클래스.

클래스명 앞에 class대신 interface를 적는다.

Object같은 최고조상이 없고, 인터페이스 클래스는 인터페이스 클래스들을 다중상속 가능하다 extends대신 implements를 적어 구현(상속)한다

컴파일된 파일을 보면 외부클래스명$내부클래스명 의 형태로 되어 있다.

예외 발생시, try → catch → finally의 순서로 실행되고

예외 미발생시, try → finally의 순서로 실행된다.

2진수->8진수 : 2진수 3자리씩 끊기

2진수->16진수 : 2진수 4자리씩 끊기

메이븐으로 만들지 않은 다이나믹웹페이지에서 라이브러리를 쓰려면 jar파일을 다운받아 WebContent/WEB-INF/lib파일 안에 넣으면 된다

1. page 디렉티브의 errorPage 속성에서 지정한 에러 페이지를 보여준다.
2. JSP 페이지에서 발생한 예외 타입이 web.xml 파일의 <exception-type>에서 지정한 예외 타입과 동일한 경우 지정한 에러 페이지를 보여준다.
3. JSP 페이지에서 발생한 에러 코드가 web.xml 파일의 <error-code>에서 지정한 에러 코드와 동일한 경우 지정한 에러 페이지를 보여준다.
4. 아무것도 해당되지 않을 경우 웹 컨테이너가 제공하는 기본 에러 페이지를 보여준다.

/\*

최대공약수 공식(유클리드 호제법)

a, b : 최대공약수를 구하고자 하는 두 수

r : a를 b로 나눈 나머지 = ( a % b ) = ( a mod b )

식 : gcd(a,b) = gcd(b,r)

\*/

**private** **long** **gcd**(**long** a, **long** b) {

**long** **r**;

**while** (b != 0) {

r = a % b;

a = b;

b = r;

}

**return** a;

}

zipFileException 에러 발생시 이클립스 끄고 c드라이브-사용자-.m2에서 모두 지우고 이클립스 다시 킴, 이걸 수행했을 때는 cmd에서 오라클 메이븐인스톨세팅도 같이 해줘야함

spring.io홈페이지 projects – spring tools 4

System.***out***.println(Arrays.*toString*("qwertyqwerty".split("(?<=\\G.{1})")));

System.***out***.println(Arrays.*toString*("qwertyqwerty".split("(?<=\\G.{2})")));

System.***out***.println(Arrays.*toString*("qwertyqwerty".split("(?<=\\G.{3})")));

System.***out***.println(Arrays.*toString*("qwertyqwerty".split("(?<=\\G.{4})")));

System.***out***.println(Arrays.*toString*("qwertyqwerty".split("(?<=\\G.{5})")));

System.***out***.println(Arrays.*toString*("qwertyqwerty".split("(?<=\\G.{6})")));

Emmet 깔고 windows preference emmet에서 jsp확장자 추가, variables ko, ko-KR변경

Properties editor 설치

Tern eclipse 설치

Windows preference web에서 html,css,js UTF-8로 변경

Dbeaver 깔고 windows preference editors-file assosications에서 xml파일 디폴트 스프링컨피그에디터로 변경, xml파일 열고 Namespaces에서 context 체크, windows perference에서 keys에서 delete line 검색 후 단축키 변경

Lombok 설치시 sts종료하고, Lombok.jar파일이 있는 디렉토리에서 씨엠디 켠 후 java –jav lomb + tab키 누르고 sts있는 디렉토리 설정, 그다음 sts실행 후 디펜던시 추가

@Autowired는 세터주입이건, 생성자주입이건 필드명 위에 붙이면 된다

]

스프링에서 레거시로 mvc파일 만들면 폼에서 자바1.8로 바꿔주고 스프링버전을4.3.22로 변경

@Repository – Used to mark a bean as DAO Component on persistence layer  
@Service – Used to mark a bean as Service Component on business layer  
@Controller – Used to mark a bean as Controller Component on Presentation layer  
@Configuration – Used to mark a bean as Configuration Component.  
@Component – General purpose annotation, can be used as a replacement for above annotations.