181105

//

x = sc. nextInt();

switch (x)

{

case 1 : System.out.println("콜");

case 2 : System.out.println("콜");

case 3 : System.out.println("콜");

case 4 : System.out.println("콜");

case 0 : System.out.println("맛점 하세용");

}

이거 그거 알지. switch 하고 각 case마다 끝날 때 break 넣어줘야 하는 거. 근데, 방금 실수로 break 다 빼고 해봤다? 그랬더니 해당 case 이후로 있는 연산들은 다 수행해버리더라. 그러니까 case 1에 해당하면, case 1, 2, 3, 4, 0 다 수행하고, case 3에 해당하면, case 3, 4, 0 수행하고, case 0이면 case 0만 수행하고. 그런 식으로. 그래서, 각 case 마다 break를 걸어 주는 거네. 말하자면, switch의 기능은, 소괄호 안에 있는 변수(여기서는 x)에 대해, 괄호 안에 있는 연산 중, x가 해당하는 경우로 바로 점프하는 거네.

//메서드 이름, 특수문자는 달라, 언더바만 가능, but 웬만하면 쓰지 말긔

숫자는 맨 앞 아니면 가능, class랑 비슷

메서드는 독립적으로 성립 못해, 그 상위 개념인 class가 선행 되어야 함.

어떤 class에든지 종속돼 있어야 함.

같은 class 안에 있는 애들은 그냥 부르면 되고,

다른 class 안에 있는 애들은, class까지 포함해서 불러 줘야 하고.

메서드 나눠서 서술하면, 특정 작업 단위를 덩어리로 묶어서 한 단위로 묶어서

움직여다닐 수가 있으니까 편함. 이제 이게 엄청 복잡해지고 거미줄처럼

왔다 갔다 하고, 그렇게 되는 게 진짜 프로들의 세계겠지.

//x = sc. nextInt();

switch (x)

{

case 1 : verse1(); break;

case 2 : verse2(); break;

case 3 : verse3(); break;

case 4 : verse4(); break;

case 0 : break;

}

예를 들어 이런 경우, 그러니까 사용자가 1번, 2번, 3번, 4번 고를 때 실행할 연산이 모두 다를 때, 이게 그냥 뭐 간단하게 한 줄 출력하거나 하면 메서드를 따로 나누는 게 오히려 더

복잡해지겠지마는, 뭐 게임처럼, 1번 선택했을 때 스토리가 바뀌고, 2번 선택했을 때 얻는 아이템이 바뀌고, 이런 식이면 각 상황에 따라 아예 다시 짜서, 하나로 묶어서 움직이는 게 낫지.

x = sc. nextInt(); 이것도 메서드야. nextInt(); 이것도. 누군가가 만들어 놓은 메서드

//return은 메서드를 끝내는 명령. return 돌아가다. 자기가 원래 하던 메서드로 돌아가는 거야. 무슨 말이냐면, 내가 원래 하던 메서드 안에서 연산 쭉 수행하다가 갑자기 다른 메서드 호출해서 수행하래. 그런 우선 지가 하던 작업 손에서 놓고, 호출된 메서드부터 수행해. 그러다가 해당 메서드에서 return을 만나면, 기존의 메서드 위치로 돌아가는 (return) 것이지. 그래서 이름이 return이야. 여러 개의 메서드를 한 번에 나가려면 ‘return 메서드이름;’ 이렇게. return g; 처럼. continue랑 마찬가지. continue g; 하면 g라는 이름 붙여 놓은 루프로 돌아가서 이어가듯이.

//public static void 메서드이름. 여기서 void 자리는 해당 메서드를 끝내고 돌아올 때 return 할 때, 가지고 오는 것. 그 자리에 void 쓰면, 그냥 빈손으로 돌아온다, 아무것도 실행하지 않는다. int라고 쓰면, 정수 하나를 받아온다. return 값은 항상 하나다. 하나만 가져올 수 있다.

//파라미터 전달인자 매개변수값 등등, 여러 이름으로 쓰는데, 다른 메서드 호출하면서 그 메서드한테 보내는 값이야. 갈 때 가지고 가라고. 그 값도 여러 개를 줄 수도 있어.

메서드 이름 옆에 소괄호 안에는 필요한 재료들이 뭐가 있나 명시해 놓는 거야. 예를 들어

public static int sum(int m, int n)

의 경우, sum이라는 이름의 메서드고, int 형식의 값을 가져갈 거다, 그리고 int m과 int n을 가지고 온다.

그 메서드는 해당 재료들을 모두 받았을 때에만 호출할 수 있는 거여.

가져다 준다는 값 있으면, 꼭 가져다 줘야 해. return 값을 줘야 해. int 준대놓고 배열 주면 안 되고, 배열 준다 해놓고 int 주거나 하면 안 되고. char, double, 다 마찬가지.

//배열 선언할 때 예시

int arr[];

arr = {1, 2, 3, 4} //위의 두 줄 쌍으로, 또는 아래 중 하나.

int[] arr = new int[5]; 값은 없고 배열의 크기만 선언한 경우

int[] arr = {1,2,3,4}; 선언시에 크기와 값을 가진 경우

//return ar; 의 의미는, ar이라는 이름을 가지는 배열 자체를 의미하는 거고,

return ar[]; 의 의미는, ar이라는 배열 안에 있는 값들 중에서 특정한 값을 의미

int arr[]; 이거는, 선언할 때, 얘가 무슨 종류의 애인지 알려줘야 하니까 []를 붙이는 거고,

일단 선언하고 난 후에는, 얘가 지금 무슨 종류 애인지 알고 있으니까 그냥 변수명으로만

사용해도 됨.

181106

//똑같은 메서드 이름인데, 여러 개가 존재해. 자세히 보면 뭐가 달라. 자료형이 달라. 이렇게 실질적 똑같은 일을 하지만 여러 일을 하는 것처럼 만드는 것은 ‘오버로딩’이라는 이름

객체 지향형 언어가 가지는 ‘다형성’이라는 특징 중 하나

//new 는 heap 에 무언가를 저장할 때 쓰는 말

// stack에 넣는 애들은, 정의를 해주지 않으면 쓰지 못해. 컴파일에서 오류나.

heap에 넣는 애들은 자동으로 0 초기화 된대.

배열에서는 데이터 번호가 항상 0번 부터 시작해.

인덱스 첨자 맨 앞이 0 번, 그 다음 1번, .... 4번 이런 식으로 총 5개겠지.

100개 방이면 0부터 세니까 99번 방까지 있겠지

//a[2]의 뜻은, a 어레이의 2번 방, a[i]의 뜻은 a 배열의 i번 데이터.

만약 어레이가 4까지밖에 없는데, 5까지 출력하래. 실행할 때 에러 떠.

이제는 실행하다가 에러 떠. 컴파일은 잘 돼. 이제 더 어렵.

그러면 '그 어레이의 개수'를 나타내는 명령어, 'a.length; 가 있음.

int x;

x = 1;

이걸 줄여서

int x = 1;

(뒤에서 다시 int x = ~~ 하면, 이미 쓴 변수 이라서 선언이 안 돼.)

마찬가지로 어레이 선언과 정의를 줄여서 하는 방법

int a[];

a = new int[5]; //5개짜리 어레이 a를 heap에 생성해주세요

a[0] = 11;

a[1] = 22;

....

이걸 줄여서

int []a = {11, 22, 4, 65, 23);

이렇게.

어레이의 기능 중 하나 -

nextInt의 경우 버퍼에 저장하므로, 다음 숫자 받으면 받아 놓은 숫자가

사라지는데, 어레이에 저장한다고 하면, 나머지 애들도 다 저장해놓을 수 있어

//array.length 라고 하면, 해당 어레이에 10개의 데이터를 저장해 놓았다고 했을 때,

'번호'만 놓고 보자면 0 ~ 9번까지밖에 없는 것이여. 긍께 array[array.length]하면

array[10]이 되는 거지. 따라서 array[array.length]는 없는 데이터지.

181107

//stack이 포인터, heap이 메모리, heap 에 저장된 정보가 '객체'

// 이미 배열로 정의된 ar에 대해서, int c = ar; 이러면 가능? ㄴㄴ 자료형이 달라.

string을 int에 저장하려는 것과 같아. 자료형 맞춰서, c 또한 배열로 정의해 줘야 해.

//

int c[] = ar; 이러면 ar과 같은 배열이 c라는 거에 하나 더 생길까? c라는 식별자에 인스턴스를 하나 더 만들까? 그렇지 않다. 인스턴스는 하나고, 그걸 가리키는 포인터만 두 개가 생기는 것. 즉, 집은 하나인데, 그걸 가리키는 주소만 두 개 생성. 우리집 주소가 도로명 주소도 있고 구주소도 있는 것처럼. 따라서, 같은 배열을 하나 더 만들려면, 하나하나 다시 해줘야 해. 기존의 배열과 똑같은 크기의 배열을 new int[] 를 통해서 heap에(인스턴스 저장하는 곳) 생성해 놓고, 기존 첫 번째 데이터 = 새로 첫 번째 데이터; 기존 두 번째 데이터 = 새로 두 번째 데이터; 일일이 반복해 줘야 함.

//

선언, 정의, 자료형 호환, 알고리즘, 변수 이름, 루프 범위, 조건문 조건,

181108

// int[] ar // int ar[] 등등의 차이는?

// ar = randomNumber(ar); 이라고 해야 하는지, 그냥

randomNumber(ar); 이라고 해야 하는지,

주는 값에 대하여, return을 받아야 하는 연산인지,

지 혼자 처리하고 나면 끝나는 연산인 건지,

선언하고 크기까지 정해서 줬으면, 그 안에 채우는 연산만 해 주는 거지, 그러니까,

내가 선언해주고 크기까지 정해주면 저절로 0 초기화 돼서 그 안에 다 정의가 되잖아.

이미 정의 된 배열을 가지고, 그 안에 내용을 바꿔주는 연산(메서드), 기능인지,

아니면 기존의 값들을 가지고 들어가서 특정 연산을 수행한 끝에,

어떤 특정 값을 return 해주는 연산(메서드), 기능인지.

그러면 그냥 return 받는 자료형의 특정 데이터라고 생각해도 돼.

ar = ~~ 메서드. 넣어주면, ar = 3, ar = 'A', ar = false 등, 그냥 특정 값이 되는 거.

//정렬 같은 경우는, 이미 거의 다 돼 있는데, 뭐 하나 조금 바껴서 다시 정리하는 경우가 많단 말야. 그러니까 하나하나 일일이 연산하는 것들 보다는, bubble sort같이, 이미 괜찮으면 그냥 지나갈 수 있게끔 짜주는 코드가 더 효율적이래.

//flag 초기값 true 설정해 놓고, 트로이 목마처럼, 알고 싶은 연산 위치에 false로 숨겨 놓기. 연산 다 끝나고 나면, 해당 위치 지나갔는지, 아닌지 알 수 있음.

181109

//이중 배열, 2차원 배열

int ar[][]; //선언

ar = new int[4][3]; //heap 생성

ar[0][0]=56; //정의 시작

ar[0][1]=78;

ar[0][2]=63;

....

or

ar[][] = { {1,1,1,1}, {2,2,2,2}, {3,3,3,3}, {4,4,4,4} } //정의 원소나열법으로

ar[][] = { {1,1,1,1}, //정의 원소나열법 - 보기 좋게 줄바꿈

{2,2,2,2},

{3,3,3,3}

{4,4,4,4,} }

// 배열을 선언하면서 크기를 정의해버리면, 값이 모두 각각 0초기화 돼 있는

값으로 채워진, 해당 크기의 배열이 생성돼. 게다가, 이미 특정 값이 채워진 배열은 나중에

{1,2,3,4,5} 이런 식으로 중괄호로 묶어서는 값을 다시 못 넣어. 하나하나 해줘야 해.

즉, 정리하자면, 크기를 생성하는 순간, 이미 값까지 완성이 돼버리는 거야.

따라서, 크기를 생성하기 전에, 값을 알려줘야 해. 이게 근데 동시에 이루어져.

선언만 해놓고, 그 각각의 값들을 중괄호로 묶어서 {1,2,3,4,5,6} 이런 식으로 정의해 주면,

그 해당 크기만큼을 가진 배열을 알아서 생서해 주는 거야. 그러니까 {~~~~} 이런 정의에는

heap 에 넣어달라는 의미가 포함돼 있고, 그 크기도 포함돼 있고, 그 각각의 값들도

포함돼 있는 수식인 거지. 명령인 거지.

그럼 자료형만 우선 선언해 주고, 그 안에 들어갈 값은 그냥 나열만 해주는 것으로 충분.

int arr[];

arr = {1,2,3,4,5,6,7,8};

이러면 1)heap 에다가, 2)8개의 크기를 가지는, 3)각각 1~8 값을 가지는, 배열을 생성함.

이건, 크기가 4인 x축과 크기가 3인 y축으로 생각하는 게 편함. 즉,

가로 4줄, 세로 3줄로 이루어진 바둑판 식으로 생각하면 편함. 그럼

표가 생기겠지.

“이렇게 이중 배열 쓰면 더 성가실 것 같은데, 장점이 뭔가요?”

응, 더 성가실 수도 있고, 더 간단할 수도 있어. 그건, 이걸 어디에 어떻게 쓰느냐에

따라 달려 있음. 예를 들어,

경우1) 1명의 학생이 12과목의 시험을 봤어.

국어 영어 수학 사회 과학 음악 미술....

그러면 그냥 12개의 크기를 가진 1차원 배열을 만드는 게 더 낫지. 그런데.

경우2) 만약에 4명의 학생이 각각 3과목씩의 시험을 봤어.

학생1도 {과목1, 과목2, 과목3}; 학생2도 {과목1, 과목2, 과목3}....

이러면 성적[1][1] 성적[1][2] 성적[1][3]; 성적[2][1] 성적[2][2], 성적[2][3]; ....

이런 식으로 만들어야 한다면, 이중배열, 이차원 배열이 낫겠지.

//주의할 점 - 메서드 선언할 때, 매개변수에 자료형 선언도 같이 해줘야 함.

public static void printDoubleArr(int arr[][])

- 끝에 int 쓰고, arr[][] 왜냐면 매개변수 받아오는 건 식별자

지 맘대로야, 어차피 매개변수 받은 메서드 안에서만 쓸 거라서.

근데, 이차배열이 무슨 자료형의 배열인지 모르잖아.

문자형으로 채워진 이차배열일 수도 있고, 더블형으로 채워진 이차배열일

수도 있고... 모르니까. 매개변수 받아올 때, 자료형 선언도 같이 해서

자료형을 받아와야지. 말하자면, 매개변수 받아오면서,

“이제부터 이런 애를 받아왔다고 치고, 그 변수 이용해서

해당 메서드를 짤게!” 라고 하는 거니까. 선언도 같이 해줘야지.

뭔 자료형의 뭔 변수명인지 모르니까. 각 메서드는 독립적이니까.

매개 주는 메서드랑은 변수명을 꼭 같게 맞춰줘야 하는 건 아니니까.

주는 애랑 받는 애랑 변수명을 다르게 해서 주고받을 수 있으니까. 선언도 같이 해줘야지.

181112

//1일의 출력위치를 요일만큼 띄우기

for (i = 0; i < d; i++) System.out.print("\t");

//해당월 달력 출력

for (i = 1; i <= month[m - 1]; i++)

{

System.out.print("\t" + i);

if (d++ % 7 == 6)

{ //토요일까지 출력후 줄바꿈

System.out.println();

}

}

예를 들어 나머지가 3이었으면, 3개만큼 띄고, 4개 찍고, 5개 찍고, 6개 찍고, 7개 찍고, 그 담에 줄 바꾸고, 그럼 나머지라는 애 데리고 와서 1씩 더해서 카운트로 쓰면 될 거를

/캘린더 만들기

시작 : 일 수 구하기 - 그로부터 7로 나눠서 시작 요일 구하기

중간 : 7일 마다 줄 바꿈 - 첫 줄은 시작 요일에 따라, 그 전 요일은 모두 공백 처리.

(첫 줄에서는- 줄에도 카운트 넣어서 해보자 - 나눈 몫이 1보다 작으면 or 나머지가

어쩌구...)

끝 : 해당 년, 해당 월에 따라 끝나는 일 수 결정

(월마다 몇까지 있는지, 배열로 생성해 놓음)

“년도/ 월” 출력

월 화 수 목 금 토 일

1 2 3 4

5 .......... 이런 식으로

1. 년/월 입력 받으면 그 전 월까지의 일수 구하는 메서드

2. 리턴 받은 값을 7로 나눠서 나머지에 따라 시작하는 요일 구하기

3. 첫 줄은 7-나머지 갯수 만큼 출력 후에 줄 바꿈

4. 둘째 줄 부터는 7개마다 줄 바꿈

cld[5][6] = 전체 달력의 틀(가로 6칸 세로 7칸)

cld 이중 배열에 값을 일일이 넣는 코딩

int date;

date =0;

for( i = 0~5)

{

for( j = 0~6)

{

cld[i][j] = date;

date = date + 1;

if(date가 해당 월이 가지고 있는 일 수를 초과하냐?)

{

for구문 나가기 (아예 바깥쪽으로 나가야 함. 라벨링-break 이용)

}

}

}

안에 값들이 일일이 채워진 이중 배열인 cld를 출력하는 구문

remain = 1년1월1일부터 입력받은 월까지 존재하는 모든 일 수를 7로 나눈 나머지

for( i = 0~5)

{

for( j = 0~6)

{

if( ‘cld[i][j] - 나머지’가 음수냐?)

{ //음수면 그냥 공백을 출력해라

System.out.print( “ ”);

}

else if ( ‘cld[i][j] - 나머지’가 양수냐?)

{//그럼 그냥 출력해라

System.out.print( cld[i][j] - remain);

date = date + 1;

if(date가 해당 월이 가지고 있는 일 수를 초과하냐?)

{

for구문 나가기 (아예 바깥쪽으로 나가야 함. 라벨링-break 이용)

//이거는 해당 월이 가진 최대 일수까지 출력을 완료 했으면,

작업을 멈추도록 만들기 위해

}

}

}

}

for 0~5 도는 구문 안에,

for 0~6 도는 구문 안에,

if를 통해서 “cld[i][j] - remain”이 음수인 경우와 양수인 경우로 나누고,

음수인 경우에는 공백을 출력하라,

양수인 경우에는 원래 값인 “cld[i][j] - remain”을 출력해라.

대신 매번 양수인 경우에는 최대 일 수까지 왔는지 확인해보고

끝까지 나왔으면 break로 나가라.