// 생성자 호출 순서 예제

public class BlockTest {

// [클래스 초기화 블럭] 초기화 블럭 앞에 static을 붙여서 클래스 초기화 블럭으로 만든다.

static {

System.out.println("static { }");

}

// [인스턴스 초기화 블럭] 초기화 블럭 앞에 아무것도 없으면 인스턴스 초기화 블럭이 된다.

// 인스턴스 블럭은 생성자보다 먼저 수행되며, 생성자들에서 공통되는 부분을 추려서 만든다.

{ System.out.println("instance { }"); }

public BlockTest() { System.out.println("생성자"); }

public BlockTest(String str) { System.out.println(str + "생성자"); }

public static void main(String[] args) {

System.out.println("BlockTest bt=new BlockTest();");

BlockTest bt = new BlockTest();

System.out.println();

System.out.println("BlockTest bt2=new BlockTest();");

BlockTest bt2 = new BlockTest();

System.out.println();

System.out.println("BlockTest bt2=new BlockTest();");

BlockTest bt3 = new BlockTest("String parameter");

} // end of main

} // end of class BlockTest

public class FactorialTest {

public static void main(String[] args) {

int result = factorial(4);

System.out.println(result);

} // end of main

// static 메서드이므로 클래스가 생성될 때 메모리에 올라간다.

// 그리고 인스턴스(객체)를 생성하지 않고, 직접 호출이 가능하다.

// 또한 main 메서드와 같은 클래스 내에 소속되어 있기에 클래스 이름을 생략하는 것도 가능하다.

// 그래서 FactorialTest.factorial(4) 대신 factorial(4)로 사용이 가능한 것이다.

static int factorial(int n) {

int result = 0;

if (n == 1) {

result = 1;

} else {

result = n \* factorial(n - 1);

}

return result;

} // end of factorial

} // end of class FactorialTest

import java.util.Scanner;

public class flowEx34 {

public static void main(String[] args) {

int menu = 0, num = 0;

Scanner scanner = new Scanner(System.in);

// outer로 반복문에 이름을 붙여주었다. 이름은 for, while에 붙여줄 수 있다.

outer: while (true) {

System.out.println("(1) square");

System.out.println("(2) square root");

System.out.println("(3) log");

System.out.println("원하는 메뉴(1~3)를 선택하세요. (종료:0)>");

String tmp = scanner.nextLine();

menu = Integer.parseInt(tmp);

if (menu == 0) {

System.out.println("프로그램을 종료합니다.");

break;

} else if (!(1 <= menu && menu <= 3)) {

System.out.println("메뉴를 잘못 선택하셨습니다. (종료는 0)");

// continue는 반복문의 끝으로 이동하여 다음 반복으로 넘어간다.

// 즉, 특정 조건을 만족하는 경우를 제외하고자 할 때 유용하며 break문과 달리 반복문을 벗어나지 않는다.

continue;

}

// 무한 반복문으로 아래와 같이 while (true)로 써도 된다.

for (;;) {

// while (true) {

System.out.println("계산할 값을 입력하세요. (계산 종료:0, 전체 종료:99)>");

tmp = scanner.nextLine();

num = Integer.parseInt(tmp);

if (num == 0)

break;

if (num == 99)

// 여기가 핵심 포인트이다.

// outer는 while과 for 전체를 아우르는 이름이므로, 이렇게 빠져나가면 된다.

break outer;

switch (menu) {

case 1:

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

break;

case 2:

System.out.println("result=" + Math.sqrt(num));

break;

case 3:

System.out.println("result=" + Math.log(num));

break;

}

}

}

scanner.close();

}

}

class MyMath2 {

long a, b;

// 인스턴스 변수 a, b만을 이용해서 작업하므로 매개변수가 필요없다. 즉, 아래는 인스턴스 메서드이다.

long add() { return a + b; }

long subtract() { return a - b; }

long multiply() { return a \* b; }

long divide() { return a / b; }

// 인스턴스 변수와 관계없이 매개변수만으로 작업이 가능하다.

// 아래는 클래스 메서드이며, 이 메서드 안에는 인스턴스 변수를 넣을 수 없다.

// 이 경우, static으로 선언하여 클래스 메서드로 만드는 것이 낫다.

// 왜냐하면 클래스 메서드로 만들면 클래스가 생성될 때 바로 메모리에 올라가기 때문에 성능이 향상되기 때문이다.

// 단, 메모리 소비는 인스턴스 메서드보다는 크다.

static long add(long a, long b) { return a + b; }

static long subtract(long a, long b) { return a - b; }

static long multiply(long a, long b) { return a \* b; }

static double divide(double a, double b) { return a / b; }

} // end of class MyMath2

public class MyMathTest2 {

public static void main(String[] args) {

// 아래의 클래스 메서드는 객체 생성없이 바로 호출이 가능하다.

System.out.println(MyMath2.add(200L, 100L));

System.out.println(MyMath2.subtract(200L, 100L));

System.out.println(MyMath2.multiply(200L, 100L));

System.out.println(MyMath2.divide(200.0, 100.0));

// 아래의 인스턴스 메서드는 객체를 생성해야 호출이 가능하다.

MyMath2 mm = new MyMath2();

mm.a = 200L;

mm.b = 100L;

System.out.println(mm.add());

System.out.println(mm.subtract());

System.out.println(mm.multiply());

System.out.println(mm.divide());

} // end of main

} // end of class MyMathTest2

// 클래스(static) 변수의 활용 예제

class Product {

// 여기서 static으로 선언된 count는 모든 클래스의 공통 변수가 된다.

// 따라서 클래스를 new로 생성하면 초기화 블럭이 호출되고,

// 그 초기화 블럭에서 count를 1씩 증가시켜서 serialNo에 저장한다.

// 여기서 클래스 변수는 클래스가 로딩될때 딱 한번만 로딩되므로

// 만약, count를 인스턴스 변수로 선언헸다면 매번 0으로 초기화가 되게된다.

static int count = 0;

int serialNo;

// 생성자가 하나 밖에 없기 때문에 인스턴스 블럭 대신에 생성자에 넣어도 되는 코드이다.

// 하지만, 공통적으로 수행되는 부분이기에 인스턴스 블럭으로 만든 것이다.

{

++count;

serialNo = count;

} // end of instance initialization block

public Product() { }

} // end of class Product

public class ProductTest {

public static void main(String[] args) {

Product p1 = new Product();

Product p2 = new Product();

Product p3 = new Product();

System.out.println("p1의 제품번호(serial no)는 " + p1.serialNo);

System.out.println("p2의 제품번호(serial no)는 " + p2.serialNo);

System.out.println("p3의 제품번호(serial no)는 " + p3.serialNo);

System.out.println("생산된 제품의 수는 모두 " + Product.count + "개 입니다.");

} // end of main

} // end of class ProductTest

class Data3 { int x; }

public class ReferenceReturnEx {

public static void main(String[] args) {

Data3 d = new Data3();

d.x = 10;

// d2에서는 d를 복사한 객체의 값을 가져갈 수 있다.

Data3 d2 = copy(d);

System.out.println("d.x = " + d.x);

System.out.println("d2.x = " + d2.x);

} // end of main

/\*\*

static 메서드이며, Data3 타입으로 리턴을 하는 함수이다. Data3 타입의 객체 d를 파라메터로 받아서,

새로운 Data3 타입의 객체 tmp를 생성하고 생성된 tmp 객체에 파라메터로 받은 객체의 x의 값을 복사하고

tmp 객체를 반환한다. 이 메서드가 종료되면 tmp 객체는 소멸한다.

참고로 copy는 static 메서드이므로 클래스가 생성될때 메모리에 올라가며, 인스턴스를 생성하지 않고 직접 호출이 가능하다.

\*/

static Data3 copy(Data3 d) {

Data3 tmp = new Data3();

tmp.x = d.x;

return tmp;

} // end of copy

} // end of class

// return값을 여러 개 만드는 예제

// parameter로 배열 혹은 class를 넘겨주고, 메서드 안에서 instance 값을 업데이트 하는 형식으로 한다.

public class ReturnTest {

public static void main(String[] args) {

ReturnTest r = new ReturnTest();

int result = r.add(3, 5);

System.out.println(result);

int[] result2 = { 0 };

r.add(3, 5, result2);

System.out.println(result2[0]);

} // end of main

int add(int a, int b) { return a + b; } // end of add

void add(int a, int b, int[] result) {

// return 값을 없애고, 대신 result 배열의 0번 index에 값을 업데이트했다.

result[0] = a + b;

} // end of add

} // end of class ReturnTest

// 다음은 거스름돈을 몇 개의 동전으로 지불할 수 있는지를 계산하는 문제이다. 변수 money의 금액을 동전으로 바꾸었을 때 각각 몇 개의 동전이 필요한지 계산해서 출력하라. 단, 가능한 한 적은 수의 동전으로 거슬러 주어야한다.

// 추가 연습: 동전의 단위가 랜덤으로 주어졌다면?

import java.util.Arrays;

import java.util.Collections;

public class Ex\_05\_06 {

public static void main(String[] args) {

Integer[] coinUnit = { 100, 50, 500, 10 };

int money = 2680;

int tmp = 0;

System.out.println("money=" + money);

// 동전의 단위를 큰 숫자부터 정렬한다. 즉, 내림차순으로 정렬

// java.util.Arrays 클래스의 sort() 메서드를 사용하여 정렬이 가능하다.

// Arrays 클래스는 배열의 복사, 정렬, 검색등 배열의 조작 기능을 가지고 있다.

// 오름차순은 기본형 타입의 배열로 Array.sort(arr)하면 되나,

// 내림차순으로 정렬할 때는 Collections 클래스의 reverseOrder()를 사용해야 한다.

// 이때 배열은 wrapper 클래스로 만들어야 한다. 왜냐하면, 그래야 배열의 값들이 참조값을 가지고 넘겨줄 수 있기

// 때문이다.

Arrays.sort(coinUnit, Collections.reverseOrder());

for (int i = 0; i < coinUnit.length; i++) {

tmp = money / coinUnit[i];

money = money % coinUnit[i];

System.out.println(coinUnit[i] + "원: " + tmp + "개");

} // end of for i

} // end of main

} // end of class

//다음은 배열 answer에 담긴 데이터를 읽고 각 숫자의 개수를 세어서 개수만큼 ‘\*’

//을 찍어서 그래프를 그리는 프로그램이다.

public class Ex\_05\_08 {

public static void main(String[] args) {

int[] answer = { 1, 4, 4, 3, 1, 4, 4, 2, 1, 3, 2, 1 };

int[] counter = new int[4]; // 4인 이유는 1의 갯수, 2의 갯수, 3의 갯수, 4의 갯수 이므로

for (int i = 0; i < answer.length; i++) {

// answer[i]는 answer[1]=1, answer[2]=4, answer[3]=4 ... 이렇게 간다.

// answer[i]-1은 answer[0]-1=0, answer[1]-1=3, answer[2]-1=3 ... 이렇게 간다.

// 즉, counter[answer[0]-1] => counter[0], counter[answer[1]-1] =>

// counter[3] 이 된다.

// 여기서 1을 빼는 이유는 counter 배열의 인덱스가 0부터 시작하기 때문이다.

// 그리고 answer값 자체를 counter의 인덱스 숫자로 할당해서 하나씩 카운트 하는 것이다.

counter[answer[i] - 1]++;

} // end of for i

for (int i = 0; i < counter.length; i++) {

System.out.print(i + 1 + "의 갯수: ");

for (int j = 0; j < counter[i]; j++) {

System.out.print("\*");

} // end of for j

System.out.println();

} // end of for i

} // end of main

} // end of class

//주어진 2차원 배열의 데이터보다 가로와 세로로 1이 더 큰 배열을 생성해서 배열의 행과 열의 마지막 요소에 각 열과 행의 총합을 저장하고 출력하는 프로그램이다.

//

//실행 결과

//100 100 100 300

// ..

//240 240 240 720

public class Ex\_05\_11 {

public static void main(String[] args) {

// 2차원 배열 5x3 선언

int[][] score = { { 100, 100, 100 }, { 20, 20, 20 }, { 30, 30, 30 }, { 40, 40, 40 }, { 50, 50, 50 } };

// 2차원 배열 6x4 선언

int[][] result = new int[score.length + 1][score[0].length + 1];

for (int i = 0; i < score.length; i++) {

for (int j = 0; j < score[i].length; j++) {

// 보다 식을 간단하게 하려면 result.length - 1 = score.length 로 대체 가능

result[i][j] = score[i][j];

result[i][result[i].length - 1] += score[i][j];

result[result.length - 1][j] += score[i][j];

result[result.length - 1][result[i].length - 1] += score[i][j];

} // end of for j

} // end of for i

for (int i = 0; i < result.length; i++) {

for (int j = 0; j < result[i].length; j++) {

System.out.printf("%4d", result[i][j]);

} // end of for j

System.out.println();

} // end of for i

} // end of main

} // end of class

// String은 immutable한 성격을 가지고 있어서 어차피 바뀌지 않는다.

// 따라서 String 클래스에서 + 연산을 하면, 주소를 변경하고 새롭게 변경된 주소에 + 연산의 결과를 저장한다.

// 바꾼 값을 활용하려면 return으로 String을 주는 방법이 있다.

public class Ex\_06\_19 {

static String staticStr = "static";

public static void change(String str) { str += "456"; }

public void changeStr(String str) { str += "789"; }

public static void main(String[] args) {

String str = "ABC123";

System.out.println(str);

change(str);

System.out.println("After change:" + str);

System.out.println();

// staticStr은 static variable이라서 따로 선언할 필요가 없다.

System.out.println(staticStr);

// changeStr 매서드는 인스턴스 메서드라서 이렇게 선언을 해줘야 한다.

Ex\_06\_19 c = new Ex\_06\_19();

c.changeStr(staticStr);

System.out.println("After change:" + staticStr);

}

}

// shuffle로 배열의 값을 섞기

public class Ex\_06\_20 {

static int[] shuffle(int[] arr) {

// 유효성 체크는 반드시 하고 넘어가야 한다. 여기서 유효성 체크는 배열의 길이가 된다.

if (arr == null || arr.length == 0)

return arr;

for (int i = 0; i < arr.length; i++) {

int j = (int) (Math.random() \* arr.length);

int tmp = arr[i];

arr[i] = arr[j];

arr[j] = tmp;

} // end of for i

return arr;

} // end of class shuffle

public static void main(String[] args) {

int[] original = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

System.out.println((java.util.Arrays.toString(original)));

int[] result = shuffle(original);

System.out.println((java.util.Arrays.toString(result)));

} // end of main

} // end of class

// 배열의 최대, 최소값 구하기

public class ex\_06\_23 {

static int max(int[] arr) {

int max = arr[0];

for (int i = 0; i < arr.length; i++) {

if (arr[i] > max) {

max = arr[i];

} // end of if arr

} // end of for i

return max;

} // end of max

static int min(int[] arr) {

int min = arr[0];

for (int i = 0; i < arr.length; i++) {

if (arr[i] < min) {

min = arr[i];

} // end of if arr

} // end of for i

return min;

} // end of min

public static void main(String[] args) {

int[] data = { 3, 2, 9, 4, 7 };

System.out.println(java.util.Arrays.toString(data));

System.out.println("최대값: " + max(data));

System.out.println("최대값: " + max(null));

System.out.println("최대값: " + max(new int[] {}));

System.out.println("최소값: " + min(data));

}

}