자바 기초 프로그래밍 강좌 (Java Programming Tutorial) – 동빈나 2019.05.29 ~

1. Hello World!

* 자바 언어 소개: 현재 사용되는 대부분의 시스템 개발의 기본이 되는 프로그래밍 언어. C++과 같이 대표적인 객체 지향 프로그래밍 언어.
* 자바를 설치하고 자바 개발을 도와주는 대표적인 개발 환경인 이클립스 ide 설치해보자
* 첫 번째 프로젝트 만들어보기

public class Main {

public static void main(String[] args) {

System.out.println("Hello world!");

}

}

* Hello world!

“public static void main(String[] args) {“

자동으로 메인함수를 만들어 줌. 하나의 프로젝트에서 제일 먼저 실행되는 함수

1. 변수(variable)

* 변수 VS 상수

변수는 프로그램이 실행되는 동안에 언제든지 저장된 값이 변경될 수 있는 공간

상수는 한번 정해지면 값을 변경할 필요가 없는 데이터

* 변수 설정

정수는 int 변수 이름 = 값;

실수는 double 변수 이름 = 값;

문자형은 String 변수 이름 = “값”;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

**int** intType = 100;

**double** doubleType = 150.5;

String stringType = "이정화"; //변수 선언 및 값 설정하여 초기화

System.***out***.println(intType);

//값 출력 후 한칸 줄바꿈

System.***out***.println(doubleType);

System.***out***.println(stringType);

}

}

* 상수 정의해보기

**public** **class** Main {

**final** **static** **double** ***PI*** = 3.141592;

// 상수는 메인함수 바깥 쪽에 정의

**public** **static** **void** main(String[] args) {

**int** r = 30;

System.***out***.println(r \* r \* ***PI***);

}

}

* Overflow 개념

**public** **class** Main {

**final** **static** **int** ***INT\_MAX*** = 2147483647; // int 최댓값

**public** **static** **void** main(String[] args) {

**int** a = ***INT\_MAX***;

// 상수를 a라는 변수에 넣어줌

System.***out***.println(a + 1);

// -2147483647 이 나옴:int최솟값. int의 범위를 넘었기 때문에 오류로 최솟값이 나옴: 순환구조

}

}

* 사칙연산 프로그램 작성

**public** **class** Main {

**public** **static** **void** main(String[] args) {

**int** a = 1;

**int** b = 2;

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

}

}

* a +b = 3

1. 자료형(Data Type)

* 변수 쓰임에 대한 팁

1. 자바에서는 변수 선언 후 값 대입(초기화)을 안 하면 사용 못 함
2. 정수 타입에 byte, short, int, long

각각 차지하는 메모리크기 (즉 표현가능 범위)가 다름

Int는 4 byte, -21억 ~ 21억 정도까지. 값의 범위를 넘어갈 경우 long 쓰면 됨

1. 정수 변수 안에 실수 넣으면 정수 부분만 저장됨 int a = (int) 0.5; -> 0
2. 실 수 값 반올림하기: 변수에 0.5 더한 뒤에 정수형으로 형 변환

(int)(실수 + 0.5);

Double b = 0.5;

Int a = (int) ( b + 0.5); -> 1

* 원시 자료형: 자바에서 기본적으로 사용이 되는 자료형. Boolean, char, byte, short, int, long..
* 비원시 자료형: 다른 자료형들을 엮어서 만든 것. String, array 등..
* 아스키코드 기반의 char형을 사용해 a부터 z까지 출력해보기 (반복문 사용)

**public** **class** Main {

**public** **static** **void** main(String[] args) {

**for** (**char** i = 'a'; i <= 'z' ; i++) {

System.***out***.print(i + " ");

}

}

}

* 10진수를 8진수 혹은 16진수로 바꿔보기

**public** **class** Main {

**public** **static** **void** main(String[] args) {

**int** a = 200;

System.***out***.println("10진수 : " + a);

System.***out***.format("8진수 : %o\n", a); // 콤마 뒤에 값을 쓰면 a가 % 뒷 자리에 들어감 System.***out***.format("16진수 : %x\n", a); // \n으로 줄 바꿈

}

}

* Substring 함수 써보기

**public** **class** Main {

**public** **static** **void** main(String[] args) {

String name = "John Doe";

System.***out***.println(name);

System.***out***.println(name.substring(0, 1)); // 0부터 시작해서 마지막 자리 숫자까지

System.***out***.println(name.substring(3, 6)); // 3번째 숫자부터 6번째 자리까지 -> n D

}

}

* String은 클래스 기반의 비원시 자료형:substring가 같은 내부적인 함수가 존재하기 때문
* 정수를 나타내는 자료형이 많은 이유는 각 자료형이 차지하는 메모리 공간의 크기가 다르기 때문
* 자바에서 string은 내부적으로 char의 배열

1. 연산자(operator)

* 가장 많이 사용되는 연산자는 +, -, \*, /(몫만), %(modular, 나머지 값)
* 초를 입력 받아 몇 분 몇 초인지 계산하는 프로그램 작성해보기

**public** **class** Main {

**final** **static** **int** ***SECOND*** = 1000;

**public** **static** **void** main(String[] args) {

**int** minute = ***SECOND*** / 60;

**int** second = ***SECOND*** % 60; // 나머지값

System.***out***.println(minute + "분" + second + "초");

}

} -> 16분 40초

* ++와 - -연산의 개념을 바르게 이해하고 프로그램을 작성해보자 (증감 연산자의 활용)

**public** **class** Main {

**public** **static** **void** main(String[] args) {

**int** a = 10;

System.***out***.println("현재의 a는 " + a + "입니다.");

a++;

System.***out***.println("현재의 a는 " + a + "입니다."); // a는 11

System.***out***.println("현재의 a는 " + ++a + "입니다.");

// a는 12 (a에 먼저 1을 더해준 후 출력)

System.***out***.println("현재의 a는 " + a++ + "입니다.");

// a는 12 (출력이 된 이후에 빠져나가면서 a에 1을 더해줌)

System.***out***.println("현재의 a는 " + a + "입니다."); // a는 13

}

}

* %(모듈러)연산자의 사용법 배워보기

**public** **class** Main {

**public** **static** **void** main(String[] args) {

System.***out***.println(1 % 3); // -> 1 나눈 후 나머지값 반환

System.***out***.println(2 % 3); // -> 2

System.***out***.println(3 % 3); // -> 0

}

}

* ==, >, <, &&(and), ||(or), ! 연산
* ‘조건? 참:거짓’ (삼항 연산자) 연산의 형태를 숙지: (조건식)? 참일 때 값 : 거짓일 때 값

**public** **class** Main {

**public** **static** **void** main(String[] args) {

**int** x = 50;

**int** y = 60;

System.***out***.println("최댓값은 " + *max*(x,y) + "입니다.");

}

//함수 형태: 반환형, 함수이름(매개 변수)

**static** **int** max(**int** a, **int** b) { // 매개변수 a와 b 중 더 큰 값을 리턴

**int** result = ( a > b) ? a : b; // 참값이면 a가 결과값에 들어감, 거짓이면 b가 들어감

**return** result;

}

}

* 최댓값은 60입니다.
* Pow()를 이용한 거듭제곱 연산 프로그램 작성해보기: 함수의 기능에 가까움

**public** **class** Main {

**public** **static** **void** main(String[] args) {

**double** a = Math.*pow*(3.0, 20.0); // Math class의 pow함수(거듭제곱)

System.***out***.println("3의 20제곱은 " + (**int**) a + "입니다.");

}

}

* 3의 20제곱은 2147483647입니다.

1. 조건문 & 반복문 (1)

* 논리의 흐름을 정하는 가장 기본적인 기술
* If문을 이용해서 문자열이 특정 문자열 포함하는지 확인할 수 있는 프로그램 작성해보기

**public** **class** Main {

**public** **static** **void** main(String[] args) {

String a = "I Love You.";

**if** (a.contains("Love")) {

//포함하는 경우 실행

System.***out***.println("Me too");

} **else** {

System.***out***.println("I hate you");

}

}

}

* Me too.

1. 조건문 & 반복문 (2)

* 점수에 따라서 다른 메시지를 출력하는 프로그램 작성해보기
* 기본적이만 자주 쓰는 형태

**public** **class** Main {

**public** **static** **void** main(String[] args) {

**int** score = 95;

**if** (score >= 90) {

System.***out***.println("A+입니다.");

}

**else** **if** (score >= 80) {

System.***out***.println("B+입니다.");

}

**else** **if** (score >= 70) {

System.***out***.println("C+입니다.");

}

**else** {

System.***out***.println("F입니다.");

}

}

}

* 문자열과 정수형을 각각 조건문을 이용해 활용해보고 그 차이점을 알아보자

**public** **class** Main {

**public** **static** **void** main(String[] args) {

String a = "Man";

**int** b = 0;

// 자바는 String을 비교할 때 equal()을 이용.

**if** (a.equals("Man")) {

System.***out***.println("남자입니다");

}

**else** {

System.***out***.println("남자가 아닙니다");

}

**if** (b == 3) {

System.***out***.println("b는 3입니다");

}

**else** {

System.***out***.println("b는 3이 아닙니다");

}

**if** (a.equalsIgnoreCase("man") && b == 0) //대소문자를 무시하고 같은지 판단

{

System.***out***.println("참입니다");

}

**else** {

System.***out***.println("거짓입니다");

}

}

}

* 남자입니다.

3이 아닙니다.

참입니다.

* while문을 이용하여 1부터 1000까지의 합을 출력하는 프로그램을 작성해보자

**public** **class** Main {

**public** **static** **void** main(String[] args) {

**int** i = 1, sum = 0; // 한 번에 변수 두 개 설정 가능

**while**(i <= 1000) {

sum += i++; // i에 1이 더해지며 sum에 더해짐

}

System.***out***.println("1부터 1000까지의 합은 " + sum);

}

}

* 1부터 1000까지의 합은 500500
* for문을 이용하여 삼각형을 출력하는 프로그램을 작성해보자 (이중 for문)

**public** **class** Main {

**final** **static** **int** ***N*** = 30;

**public** **static** **void** main(String[] args) {

//for문 : 초기화 부분, 조건 부분, 연산 부분

**for** (**int** i = ***N***; i > 0; i--) {

**for** (**int** j = i; j > 0; j--) {

System.***out***.print("\*"); // 처음 별을 삼십개 연달아 출력

}

System.***out***.println(); //위쪽 반복이 끝날 때 마다 줄바꿈

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*

\*\*\*

\*\*

\*

- 사용빈도는 적지만 조건이 많으면 switch문 사용하면 좋음

- 주의할 것은 switch 조건으로 사용할 수 있는 데이터 타입은 byte, short, char, int, enum, String, Character, Byte, Short, Integer.

**public** **class** Main {

**public** **static** **void** main(String[] args) {

System.***out***.println("switch(1)");

**switch**(2) { //괄호 안에는 어떤 값

**case** 1: //switch문 뒤의 값과 일치하면 하기 코드 실행

System.***out***.println("one");

**break**; // 조건문에 해당되는 실행문이 종료되면 나가기

**case** 2:

System.***out***.println("two");

**break**;

**case** 3:

System.***out***.println("three");

**break**;

**default**: // 상기 조건에 맞지 않는 값이 들어오면 디폴트 실행문

System.***out***.println("default");

}

}}

* For(;;){}, while(true){} 는 무한 루프로 동작 -> 특정 조건 달성시 break를 써서 조건문 종료시키는 방식으로 사용 가능
* 공백 넣기

1) 탭

**public** **class** Main2 {

**public** **static** **void** main(String[] args) {

System.***out***.print("나나나\t니나노"); //탭 공백주기

}

}

2) \r

**public** **class** Main2 {

**public** **static** **void** main(String[] args) {

System.***out***.print("나나나\r니나노"); // 뒤따라오는 문자열이 다음 줄 첫 줄로 옴

주로 \n이랑 혼용해서 씀

}

}

* 나나나

니나노

1. 기본입출력

* 자바에서는 대표적으로 Scanner 클래스를 이용하여 사용자와 상호작용: 일반적으로 Scanner sc = new Scanner(System.in);으로 클래스 객체를 생성한 뒤에 sc.nextint();와 같은 방법으로 int형을 입력받음
* 특정한 정수를 입력 받아서 그대로 출력하는 프로그램 작성해보기

**import** java.util.Scanner; // java에서 제공해주는 라이브러리를 사용하기 위해 import

**public** **class** main {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***); // class형태로 내부적으로 함수를 가질 수 있음

System.***out***.print("정수를 입력하세요 : ");

**int** i = sc.nextInt(); // sc라는 입력 함수가 사용자가 입력하는 정수를 i에 넣어줌

System.***out***.println("입력한 정수는" + i + "입니다.");

sc.close();

}

}

* 파일에 입력된 정수에 100을 곱해 출력하는 프로그램 작성해보기

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.util.Scanner;

**public** **class** main {

**public** **static** **void** main(String[] args) {

File file = **new** File("input.txt"); // 텍스트파일을 불러와서 file 변수에 저장

**try** {

Scanner sc = **new** Scanner(file); // 입력받는 변수는 file을 통해서 받음

**while** (sc.hasNextInt()) { // sc가 읽어오고 있는 파일에서 다음의 정수가 있나?

System.***out***.println(sc.nextInt() \* 100); // 정수가 있다면 코드 실행

}

sc.close();

} **catch** (FileNotFoundException e) { //surround with try catch. 예외 경우 실행될 코드

System.***out***.println("파일을 읽어오는 도중에 오류가 발생했습니다.");

}

}

}

* 파일에 입력되어있는 정수에 각각 100씩 곱한 결과 출력
* Scanner로 문자열을 입력받고 싶을 때에는 next() 함수와 nextLine()을 사용

1. 배운 내용 정리하기
2. 사용자 정의 함수 (1)

* 객체지향: 객체는 일반적으로 말하는 물건을 의미하며 여기서 물건은 단순한 데이터가 아니고 그 데이터의조작 방법에 대한 정보 또한 포함하고 있어 그것을 대상을 다루는 수법을 객체지향이라고 함: 객체는 기능별로 연관되어있는 변수와 메소드를 그룹핑한 것
* 사용자 정의 함수: 정해진 특정 기능을 수행하는 모듈을 의미. 함수를 적절히 활용하면 하나의 문제를 잘게 분해할 수 있음. 함수는 각각의 모듈로서 쉽게 수정되서 보완될 수 있다는 장점이 있음
* 3개의 수 최대 공약수를 찾는 프로그램 작성해보기

//최대 공약수 찾기

**public** **class** Main {

// 반환형, 함수명, 매개변수

**public** **static** **int** function(**int** a, **int** b, **int** c) {

**int** min;

**if** (a > b) {

**if** (b > c) {

min = c;

}

**else** {

min = b;

}

} **else** {

**if** (a > c) {

min = c;

}

**else** {

min = a;

}

}

**for** (**int** i = min; i >0; i--) { // 세 개의 수에서 가장 작은 수부터 시작

**if** ( a % i == 0 && b % i == 0 && c % i == 0) { // 모든 수에서 i값이 나눠떨어질 때

**return** i;

}

}

**return** -1; // 최대공약수가 없는 경우

}

**public** **static** **void** main(String[] args) {

System.***out***.println("(400, 300, 750)의 최대 공약수: " + *function*(400, 300, 750));

}

}

1. 사용자 정의 함수 (2)

* 약수 중 k번째로 작은 수를 찾는 프로그램 작성해보기

**package** tutorial10.method2;

**public** **class** main {

**public** **static** **int** function(**int** number, **int** k) {

**for**(**int** i =1; i <= number; i++) {

**if** (number % i == 0) {

k--;

**if** (k ==0) { //k번째 약수를 찾았을때

**return** i;

}

}

}

**return** -1; //함수가 비정상적으로 종료되었음을 알려줌

}

**public** **static** **void** main(String[] args) { // void는 반환형이 없다는 뜻

**int** result = *function*(3050, 10);

**if** (result == -1) {

System.***out***.println("3050의 10번째 약수는 없습니다.");

} **else** {

System.***out***.println("3050의 10번째 약수는 " + result + "입니다.");

}

}

}

* 문자열에서 마지막 단어를 반환하는 함수를 작성해보자

**public** **class** main {

**public** **static** **char** function(String input) {

**return** input.charAt(input.length() -1);

// 문자열의 길이에서 마지막에 위치한 문자 하나를 반환.

charAt은 특정한 단어를 0부터 시작해서 지정한 인덱스에 있는 단어를 하나 가져옴

}

**public** **static** **void** main(String[] args) { // void는 반환형이 없다는 뜻

System.***out***.println("Hello world의 마지막 단어는 " + *function*("Hello world"));

}

}

* Max()를 이용하여 최대값을 저장하는 프로그램 작성해보기

**package** tutorial10.method2;

**public** **class** main {

**public** **static** **int** max(**int** a, **int** b) {

**return** ( a > b) ? a:b; //삼항연산자

}

**public** **static** **int** function(**int** a, **int** b, **int** c) { //총 세개의 변수 중에 최대값 찾기

**int** result = *max*(a, b); // a,b 비교

result = *max*(result, c); // 위에서 나온 값과 c를 다시 비교

**return** result; // 그 결과를 리턴

}

**public** **static** **void** main(String[] args) {

System.***out***.println("(345, 567, 789) 중에서 가장 큰 값은" + *function*(345, 567, 789));

}

}

* 계산기 실습

**public** **class** Calculator {

**public** **static** **void** sum(**int** a, **int** b) {

String result = a + "와" + b + "를 더한 값은: " + (a + b);

System.***out***.println(result);

}

**public** **static** **void** main(String[] args) {

**int** a = 3424;

**int** b = 2342;

*sum*(a, b);

System.***out***.println();

}

}

1. 배열

* 다량의 데이터를 저장하고 처리하기 위해 사용하는 자료형
* 정수형 배열 **int [] array(배열의 이름) = new int[정수의 개수];**
* 인덱스는 0부터 시작
* 원하는 개수만큼의 배열 생성 및 최갯값 구하는 프로그램 작성해보기

**public** **class** Main {

**public** **static** **int** max(**int** a, **int** b) {

**return** (a > b) ? a : b;

}

**public** **static** **void** main(String[] args) {

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("생성할 배열의 크기를 입력하세요 : ");

**int** number = scanner.nextInt();

**int** [] array = **new** **int**[number];

**for** (**int** i = 0; i < number; i++) {

System.***out***.print("배열에 입력할 정수를 하나씩 입력하세요(양수):" );

array[i] = scanner.nextInt();

}

**int** result = -1; // 모든 값은 -1보다 큼

**for**(**int** i = 0; i < number; i++) {

result = *max*(result, array[i]);

}

System.***out***.println("입력한 모든 정수 중에서 가장 큰 값은: " + result);

}

}

* 100개의 랜덤 정수의 평균을 구하는 프로그램 작성해보기

**public** **class** Main {

**public** **static** **void** main(String[] args) {

**int**[] array = **new** **int** [100];

**for** (**int** i = 0; i < 100; i++) {

array[i] = (**int**) (Math.*random*( ) \* 100 + 1);

// Math.rondom -> 0 ~ 0.xxx의 실수 가져옴. 100곱해주면 1 ~ 100사이의 정수 가져옴

}

**int** sum = 0;

**for** (**int** i = 0; i < 100; i++) {

sum += array[i];

}

System.***out***.println("100개의 랜덤 정수의 평균 값은 " + sum / 100 + "입니다.");

}

}

1. Collections framework - ArrayList

* Array 사용할 경우 한번 정해진 배열의 크기를 변경할 수 없었음

String[] arrayObj = new String[2];

arrayObj[0] = "one";

arrayObj[1] = "two";

// arrayObj[2] = "three"; 오류가 발생한다.

* 대안으로 “Arraylist” 컬렉션즈 프레임워크에 포함된 기능 중 하나
* Arraylist 생성할 때는 몇 개의 값을 담을 수 있는지 지정할 필요 없음

Import java.util.ArrayList;

ArrayList **al** = new ArrayList(); // array 생성 방법

**al**.add(“one”); // 데이터 추가 방법, 인자의 데이터형식은 object이여야 함: 어떤 형식도 담을수 있도록

al.add(“two”);

al.add(“three”);

for (int i= 0; I < al.size(); i++) // 데이터 내용 열거 방법

{

String value = (String) al.get(i);

// get을 통해 i를 받을 때 이것은 object 타입으로 저장되어 있으므로,

이것을 String 타입의 value 변수에 담기 위해서는 string으로 형 변환을 해주어야 함

System.out.prinln(value);

}

* 주의할 점: ArrayList의 데이터 형식을 특정 데이터 형식으로 리턴해주기 위해서는 먼저 형변환을 해줘야함
* 최근에는 하기와 같이 제네릭을 사용

ArrayList<String> **al** = new ArrayList<String>() 을 해줄 경우 입력되는 데이터 타입을 명시했기때문에 get을 통해 데이터를 가져올 때 string 타입으로 가져 옴

1. Collections framework – 전체적인 그림

* Collection
  1. Set [HashSet, LinkedHashSet, TreeSet] : 중복되지 않는 값들을 담을 때
  2. List [ArrayList, Vector, LinkedList] : 중복을 허용
  3. queue
* Map : key-value 형식으로 값들을 담을 때

1. Collections framework – List와 Set

* Set-HashSet의 예

Import java.util.HashSet;

Public class SetDemo {

Public static void main(String [] args) {

HashSet<Integer> A = new HashSet<Integer>();

A.add(1); // A 의 HaseSet에는 1, 2, 3이 담김

A.add(2);

A.add(2); // 이렇게 값이 중복되어도 중복된 값이 제외되고 출력됨

A.add(3);

Iterator hi = (Iterator) A.iterator(); //위 코드를 출력해줌

While (hi.hasNext()) {

// hasNext: 반복할 데이터가 더 있으면 true, 더 이상 반복할 데이터가 없다면 false를 리턴

System.out.println(hi.next());

//next: hasNext가 true라는 것은 next가 리턴할 데이터가 존재한다는 의미

}

HashSet<Integer> B = new HashSet<Integer>();

B.add(3);

B.add(4);

B.add(5);

}

}

* List는 입력된 모든 값이 저장되고, set은 입력된 값 중 고유한 값만 저장됨

1. Collections framework –Set

Import java.util.HashSet;

Public class SetDemo {

Public static void main(String [] args) {

HashSet<Integer> A = new HashSet<Integer>(); //set은 집합을 의미

A.add(1);

A.add(2);

A.add(3);

HashSet<Integer> B = new HashSet<Integer>();

B.add(3);

B.add(4);

B.add(5);

HashSet<Integer> C = new HashSet<Integer>();

C.add(1);

C.add(2);

System.out.println(A.containsAll(B)); //false “B는 A의 부분집합이 아니다”

System.out.println(A.containsAll(C)); // true “C는 A의 부분집합(subset)”

A.addAll(B); // a집합에 b집합을 합쳐서 a로 만듦

A.retainAll(B); // a집합과 b집합에 공통되는 값을 a에 닮음 -> a에는 3만 남음(교집합)

A.removeAll(B); // a집합에서 b집합에 있는 값을 다 뺌 -> a에는 1, 2만 남음(차집합)

Iterator hi = (Iterator) A.iterator();

While (hi.hasNext()) {

System.out.println(hi.next());

}

}

}

* Set은 값의 순서를 보장하지 않음: set을 통해 저장된 데이터들은 인덱스 사용 불가

1. Collections framework – Collection interface

* Collection, set, list는 인터페이스. 리스트 카테고리에 속해있는 클래스들은 리스트 메소드(get(int index), )를 사용: index와 관련된 api가 구현 되어 잇음
* Hashset, linkedhashset, ArrayLIst 등은 클래스: collection의 메소드(add, remove 등)를 쓸 수 있음

1. Collections framework – Iterator (반복자): 컨테이너에 담긴 값들을 하나하나 꺼내서 하나씩 처리

HashSet<Integer> A = new HashSet<Integer>();

// iterator라는 데이터타입을 가진 인스턴스를 hi변수에 담음

A.add(1);

A.add(2);

A.add(3);

Iterator hi = (Iterator) A.iterator(); // iterator라는 데이터타입을 가진 데이터를 hi변수에 담음

While (hi.hasNext()) {

// 가상의 컨테이너 hi는 a의 값들을 가지게 되고, 값이 존재하면 true가 됨

System.out.println(hi.next());

// hi 값 중의 하나를 리턴하고 그 값을 가상의 컨테이너에서 없앰.

모든 값을 리턴하게 되면 더 이상 남아있는 값이 없으므로 hasNext조건문은 false가 되고 반복문 종료

* iterator값은 단지 참조값이므로 원본 데이터에 영향을 주지 않음
* arrayList나 hashset이나 둘 다 컬렉션 인터페이스를 구현하고 있고, 컬렉션인터페이스를 구한하고 있는모든 클래스들은 iterator 메소드를 쓸수 있음
* 위의 예에서 arrayList나 hashset을 Collection으로 바꿔도 그대로 기능함

Collection<Interger> A= new Collection<Interger>();

1. Collection framework - Map의 기본 사용법

* List는 데이터를 add할 경우 위에서부터 순차적으로 데이터가 들어 감 -> index 사용가능 get(int index)
* Set은 집합이므로 데이터가 순서와 상관없이 들어 감
* Map은 key가 저장되어있는 공간과 value가 저장되어있는 공간이 따로 있음
* key값은 중복될 수 없고, value값은 중복 허용: 중복된 key값을 줄 경우 기존에 key와 연결되어있던 value값은 새로 들어온 value값으로 대체됨

import java.util.Collection;

public class MapDemo {

public static void main(String[] args) {

HashMap<String, Integer> a = new HashMap<String, Integer> ();

// 두 가지 제너릭을 사용. Key의 데이터타입과 value의 데이터타입을 각각 지정해주기 위해서

a.put(“one”, 1); // put메소드는 collection에서는 없고 map에서만 있음

a.put(“two”, 2); // two는 key, 2는 value가 됨

a.put(“three”, 3);

a.put(“four”, 4);

System.out.println(a.get(“one”)); // 1. 값을 가져올 때는 key값을 인자로 전달: 키가 중복되면 안되는 이유

}

}

* map에 저장된 값들을 하나씩 열거해서 반복적으로 처리하기: 두가지 방식- iteratorUsingForEach, iteratorUsingIterator

public class MapDemo {

public static void main(String[] args) {

Map<String, Integer> a = new HashMap<String, Integer> ();

a.put(“one”, 1);

a.put(“two”, 2);

a.put(“three”, 3);

a.put(“four”, 4);

System.out.println(a.get(“one”));

iteratorUsingForEach(a);

iteratorUsingIterator(a);

}

}

// 첫번째 방법. map 자체는 iteration기능이 없기 때문에 map에 있는 데이터를 가지고있는 set을 만들어서,

Set안에 있는 각각의 Map.entry들의 key값은 getkey(), value는 getvalue()로 알아내는 것

static void iteratorUsingForEach(HashMap map) { // map은 인자로 전달된 a

Set<Map.Entry<String, Integer>> entries = map.entrySet();

// Map인터페이스가 갖고 있는 entry 데이터타입이 entries에 담기게 됨. Entry에는 두가지 중요한 메소드있음: getkey(), getvalue()

For (Map.Entry<String, Integer> entry : entries) { // entries는 set형

System.out.println(entry.getKey() + “ : “ + entry.getValue());

}

}

// 두번째 방법. entrySet을 이용해서 map에 대응하는 set을 만들고, 그것을 entries에 담아서 iterator를 사용해서

반복하여 i에 담음. i.hasNext() 이용하여 조회할 데이터가 있으면 반복문 실행 i.next()를 사용해 하나씩 꺼내진 값을 Entry에 담음

Static void iteratorUsingIterator(HashMap map) {

Set<Map.Entry<String, Integer>> entries = map.entrySet();

Iterator<Map.Entry<String, Integer>> i = entries.iterator();

While (i.hasNext()) {

Map.Entry<String, Integer> entry = i.next();

System.out.println(entry.getKey() + “ : “ + entry.getValue());

}

}

1. Collection framework – Collection의 사용

* Collection 프레임워크를 사용해서 데이터를 정렬하는 방법을 알아보기

Class Computer implements Comparable {

// Comprable: 시리얼넘버에 따라 정렬을 하고자 사용하는 인터페이스. Collections클래스의 sort 메소드 쓰려면 필요

Int serial;

String owner;

Computer(int serial, string owner) {

This.serial = serial;

This.owner = owner;

}

Public int compareTo(Object o) {

Return this.serial – ((Computer)o).serial; // 리턴값이 양수냐, 음수 따라 어떤 것이 큰지를 비교.

}

Public String toString() {

Return serial + “ “ +owner;

}

Puclilc class CollectionsDemo {

Public static void main(String[] args) {

List<Computer> computers = new ArraryLIst<Computer> ();

computers.add(new Computer(500, “egoing”));

computers.add(new Computer(200, “leezche”));

computers.add(new Computer(3233, “graphittie”));

Iterator I = computers.iterator();

System.out.println(“before”);

While (i.hasNext()) {

System.out.println(i.next()); //arrayList는 순서가 있기 때문에 추가한 순서대로 나옴

} // 하지만 시리얼넘버에 따라 오름차순으로 정렬하고 싶다면? 밑에 코드

Collections.sort(computers);

// Collections클래스에 있는 sort메소드를 사용. 인자로 위에서 정의한 computers array를 넘겨줌

Collections클래스의 메소드들은 모두 static이므로 사용할 때 인스턴스화할 필요 없음

System.out.println(“\nafter”); // 한 줄 띄우고 나오기 \n

I = computers.iterator();

While (i.hasNext()) {

System.out.println(i.next());

} //-> 200 ~~, 500 ~~, 3233 ~~ 으로 정렬되서 나옴

}

}