**<예제 모음>**

[**4강 – 예제**](#_4강)

[**5강 – 예제**](#_5강)

[**6강 – 예제**](#_6강)

[**7강 – 예제**](#_7강)

[**8강 – 예제**](#_8강)

[**9강 – 예제**](#_9강)

**<연습문제 모음>**

[**4강 – 연습문제**](#_4강_1)

[**5강 – 연습문제**](#_5강_1)

[**6강 – 연습문제**](#_6강_1)

[**7강 – 연습문제**](#_7강_1)

[**8강 – 연습문제**](#_8강_1)

[**9강 – 연습문제**](#_9강_1)

**예제 모음**

# **4강**

**예제 4-1 UseAssign**

**package** Unit4;

**public** **class** UseAssign {

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

// **TODO** Auto-generated method stub

**int** num1 =7;

**long** num2 = num1 +10;

**double** num3 = 1.234\_5678\_912;

**float** num4 = (**float**)num3; // num3의 값을 float형으로 변환

System.***out***.println("num1= "+num1);

System.***out***.println("num2= "+num2);

System.***out***.println("num3= "+num3);

System.***out***.println("num4= "+num4);

}

}

**예제 4-2 UseCompound**

**package** Unit4;

**public** **class** UseCompound {

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

// **TODO** Auto-generated method stub

**int** num =10;

{ **int** num1 = 20;

System.***out***.println("첫번째 블록 num1 = "+num1);

}

{

**int** num1 = 30;

System.***out***.println("두번째 블록 num1 = "+num1);

}

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

}

}

**예제 4-3 UseOperator**

**package** Unit4;

**public** **class** UseOperator {

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

**int** num1 = 2\*7;

**int** num2 = num1/3;

**int** num3 = -num2;

**double** num4 = num1/ 3.0;

System.***out***.println("num1 = "+num1);

System.***out***.println("num2 = "+num2);

System.***out***.println("num3 = "+num3 + ", num4 = "+num4);

num1 = num1%4; // 앞서 int num1 선언을 했으나, 뒤에서 num1 을 변경할 경우 이와 같이 정의 값이 수정된다.

System.***out***.println("나머지 연산 후 num1 = "+num1);

}

}

**예제 4-4 UseRelOperator1**

**package** Unit4;

**public** **class** UseRelOperator {

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

// **TODO** Auto-generated method stub

**int** num1 = 4, num2 = 6, num3= 10;

**boolean** bl1, bl2, bl3;

bl1 = num1 > num2;

bl2 = num1 <= num3;

bl3 = num1 == num2;

System.***out***.println("num1 = "+num1);

System.***out***.println("num2 = "+num2);

System.***out***.println("num3 = "+num3);

System.***out***.println("num1 > num2 ? "+bl1);

System.***out***.println("num1 <= num3 ? "+bl2);

System.***out***.println("num1과 num2는 같은 가 ? "+bl3);

}

}

**예제 4-5 UseRelOperator2**

**package** Unit4;

**public** **class** UseRelOperator2\_4\_5 {

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

// **TODO** Auto-generated method stub

**int** num1 = 5, num2 = 10;

**boolean** bl1, bl2, bl3, bl4;

bl1 = num1 < num2;

bl2 = num1 <= 5;

bl3 = bl1 && bl2;

bl4 = !bl3;

System.***out***.println("num1 = "+ num1);

System.***out***.println("num2 = "+ num2);

System.***out***.println("num1 <= num2? "+bl1);

System.***out***.println("num1 <= 5 ?" +bl2);

System.***out***.println("num1< num2 && num1 <= 5 ?"+bl3);

System.***out***.println("!(num1< num2 && num1 <= 5) ?"+bl4);

}

}

**예제 4-6 UseOperatorIncDec**

**package** Unit4;

**public** **class** UseOperatorIncDec\_4\_6 {

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

// **TODO** Auto-generated method stub

**int** num1 = 1, num2= 1, num3, num4;

++num1; ++num2;

System.***out***.println("num1 = "+num1+", num2 = "+num2);

--num1; --num2;

System.***out***.println("num1 = "+num1+", nnum2 = "+num2);

num3 = ++num1 +10;

num4 = num1++ +10;

System.***out***.println("num3 = "+ num3+" , num4 = "+num4);

}

}

**예제 4-7 UseOperatprBit**

**package** Unit4;

**public** **class** UseOperatorBit\_4\_7 {

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

// **TODO** Auto-generated method stub

**int** num1 = 9, num2 = 7;

System.***out***.println("num1 = "+num1);

System.***out***.println("num2 = "+num2);

System.***out***.println("num1 & num2 = "+(num1 & num2));

System.***out***.println("num1 ^ num2 = "+(num1 ^ num2));

System.***out***.println("num1 | num2 = "+(num1 | num2));

}

}

**예제 4-8 UseOperatorShift**

**package** Unit4;

**public** **class** UseOperatorShift\_4\_8 {

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

// **TODO** Auto-generated method stub

**int** num1 =14, num2 = 2;

System.***out***.println("num1 = "+num1);

System.***out***.println("num2 = "+num2);

System.***out***.println("num1 << num2 = "+(num1 << num2));

System.***out***.println("num1 >> num2 = "+(num1 >> num2));

System.***out***.println("num1 >>> num2 = "+(num1 >>> num2));

}

}

**예제 4-9 UseOperatorCond**

**package** Unit4;

**import** java.util.Scanner;

**public** **class** UseOperatorCond {

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

// **TODO** Auto-generated method stub

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

System.***out***.println("두 정수 입력 -->");

**int** num1 =s.nextInt();

**int** num2 =s.nextInt();

**int** max = (num1 > num2) ? num1 : num2; // 엑셀에서 if 함수와 동일한 형태

System.***out***.println("num1 = "+num1);

System.***out***.println("num2 = "+num2);

System.***out***.println("num1과 num2 중 큰 수: "+max);

s.close();

}

}

**예제 4-10 OperatorAssign**

**package** Unit4;

**public** **class** OperatorAssign\_4\_10 {

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

// **TODO** Auto-generated method stub

**int** num1 = 8, num2 =6, num3= 18, num4 = 20;

num1 += num2;

num2\*= 4;

num3&= num1;

num4 >>=3;

System.***out***.println("num1 = "+num1);

System.***out***.println("num2 = "+num2);

System.***out***.println("num3 = "+num3);

System.***out***.println("num4 = "+num4);

}

}

# **5강**

**예제 5-1 UseIfStatement**

**package** Unit5;

**import** java.util.Scanner;

**public** **class** UseIfStatement\_5\_1 {

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

// **TODO** Auto-generated method stub

{Scanner s = **new** Scanner(System.***in***);

System.***out***.print("두개의 수 입력: ");

**int** num1 = s.nextInt();

**int** num2 = s.nextInt();

**int** max;

**if**(num1> num2) max = num1; // if 값을 통해 산출된 결과가 참일 경우 max 실행

**else** max = num2; // if 내의 값이 참이 아니면 else 실행

System.***out***.println("num1 ="+ num1);

System.***out***.println("num2 ="+ num2);

System.***out***.println("maxinum number ="+max);

s.close();

}

}

}

**예제 5-2 UseNestedIfStatement**

**package** Unit5;

**import** java.util.Scanner;

**public** **class** UseNestedIfStatement {

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

// **TODO** Auto-generated method stub

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

System.***out***.print("하나의 문자나 수 입력: ");

String str =scanner.next();

**char** ch = str.charAt(0); // str의 첫번쨰 문자를 ch 으로 지정

**if**(ch >= '0' && ch <='9')

System.***out***.println("입력된 것은 숫자 입니다.");

**else** **if**(ch>= 'A' && ch <='Z')

System.***out***.println("입력된 것은 영문 대문자 입니다.");

**else** **if**(ch>= 'a' && ch <= 'z')

System.***out***.println("입력된 것은 영문 소문자 입니다.");

**else** **if**(ch>= '가'&& ch<= '힝')

System.***out***.println("입력된 것은 한글입니다.");

**else**

System.***out***.println("입력된 것은 기타 특수문자입니다.");

scanner.close();

}

}

**예제 5-3 NestedIfStatement2**

**package** Unit5;

**import** java.util.Scanner;

**public** **class** NestedIfStatement2\_5\_3 {

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

// **TODO** Auto-generated method stub

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

System.***out***.print("1-3까지 수 입력: ");

**int** no = scanner.nextInt();

**if**(no ==1 )

System.***out***.println("컴퓨터공학");

**else** **if** (no ==2)

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

**else** **if** (no ==3)

System.***out***.println("전자공학");

**else**

System.***out***.println("기타");

scanner.close();

}

}

**예제 5-4 SwichCalculator**

**package** Unit5;

**import** java.util.Scanner;

**public** **class** SwitchCalculator5\_4 {

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

// **TODO** Auto-generated method stub

**int** result;

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

System.***out***.print("수식입력 -- (ex) 3 \* 7 --> ");

**int** num1 = scanner.nextInt();

String str = scanner.next(); //수식을 입력받는 매개변수

**int** num2 = scanner.nextInt();

**switch**(str) // str 에 대한 switch를 시행하라는 의미 str에 수식을 입력하게 되어 있다.

{ **case** "+" : result = num1 +num2;

System.***out***.println(num1+" + "+num2+" = "+result);

**break**;

**case** "-" : result = num1 - num2;

System.***out***.println(num1+" - "+num2+" = "+result);

**break**;

**case** "\*" : result = num1 \* num2;

System.***out***.println(num1+" \* "+num2+" = "+result);

**break**;

**case** "/" : result = num1 / num2;

System.***out***.println(num1+" / "+num2+" = "+result);

**break**;

**case** "%" : result = num1 / num2;

System.***out***.println(num1+" / "+num2+" = "+result);

**break**;

**default**: System.***out***.println("잘못된 연산자 입니다."); // case에 해당x인 값이 str에 인식되는 경우

}

scanner.close();

}

}

**예제 5-5 ForFactorial**

**package** Unit5;

**import** java.util.Scanner;

**public** **class** ForFactorial\_5\_5 {

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

// **TODO** Auto-generated method stub

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

System.***out***.print("factorial 을 곱할 수 입력: ");

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

**int** factorial =1; // for 내부에서 사용할 factorial 값을 미리 지정

**for**(**int** i =2; i<=n; i++) // 2\*1= 2, 3\*2= 6, 4\*6= 24. 5\*24= 120 굳이 1\*1을 넣을 필요가 없다(곱셈이므로)

factorial\*= i;// factorial= factorial\*i

System.***out***.println("1부터 "+n+"까지의 곱: "+factorial);

scanner.close();

}

}

**예제 5-6 GugudanEx**

**package** Unit5;

**public** **class** GugudanEx\_5\_6 {

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

// **TODO** Auto-generated method stub

**for**(**int** i=0; i<=9; i++)

{

**for**(**int** j=0; j<=9; j++)

System.***out***.printf("%d \* %d = %2d ",i,j,i\*j);

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

}

}

}

**예제 5-7 ForeachEx enum .values()**

**package** Unit5;

**import** java.util.Arrays;

**enum** Day { ***MONDAY***, ***TUESDAY***, ***WEDNESDAY***, ***THURSDAY***, ***FRIDAY***, ***SATURADY***, ***SUNDAY***;}

//반드시 class 선언 위치 혹은 메인메소드 바깥에서 선언(package import 직후가 가장 적합)

**public** **class** ForeachEx\_5\_7 {

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

// **TODO** Auto-generated method stub

**int** [] num = { 10, 20, 30, 40, 50};

**int** sum = 0;

**for**(**int** n : num) { // for구문(for 내부에서 순환할 값 : 배열) -> 배열 내의 모든 원소를 나열하라는 의미

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

sum= sum + n;

}

System.***out***.println("합은 "+sum); // 마지막 sum 값이 저장된다.

System.***out***.println("num 배열 : "+Arrays.*toString*(num));

String f[] = {"망고", "사과", "체리", "딸기", "바나나"};

System.***out***.print("과일 이름: ");

**for**(String s : f) // for구문(for 내부에서 순환할 값 : 배열) -> 배열 내의 모든 원소를 나열하라는 의미

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

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

System.***out***.println("과일 배열 f : "+Arrays.*toString*(f));

Day today = Day.***MONDAY***; // Day라는 데이터 타입의 today(변수 이름) 선언 , 이때 Day는 enum 으로 선언한 배열의 데이터 타입을 의미, today= MONDAAY

System.***out***.println("오늘은 "+today+" 입니다.");

System.***out***.print("요일 이름: ");

**for**(Day d : Day.*values*()) // 배열.values()의 의미는? == enum 클래스 내의 메소드 중 하나로서, 정의된 배열의 모든 원소를 반환하는 메소드

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

}

}

**예제 5-8 WhileStatement**

**package** Unit5;

**import** java.util.Scanner;

**public** **class** WhileStatement\_5\_8 {

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

// **TODO** Auto-generated method stub

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

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

**int** i = 1, sum =0;

**while**(i<=n)

sum+= i++; // sum = sum+i ; i++; 를 하나의 문장으로 표현

System.***out***.println("1부터 "+n+"까지의 while 문 합: "+sum);

sum =0; //sum의 값을 새롭게 0으로 초기화

**for**(i =1; i<= n;i++) { // 주의 for, if와 같이 입력값에 따라 참/거짓이 나뉘고, 참/거짓 여부에 의해 실행이 결정되는 메소드의 경우 ; 를 무분별하게 사용할 경우 출력값에 오류 발생 가능 (주의)

sum+=i;

}

System.***out***.println("1부터 "+n+"까지의 for문 합: "+sum);

scanner.close();

}

}

**예제 5-9 DoWhileStatement do while 구문(실행 후 조건 확인)**

**package** Unit5;

**import** java.util.Scanner;

**public** **class** DoWhileStatement {

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

// **TODO** Auto-generated method stub

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

System.***out***.print("합을 구할 수 입력: ");

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

**int** i =2, sum =0;

**do** {

sum+=i; // sum = sum+i;

i+=2;// i= i+2; i의 값이 한번 순환할 때마다 2씩 증가

/\*

\* do 뒤에 sum = sum+1과 i+=2가 선언되어 우선 sum=2와 i=4 은 값이 실행된 뒤에 while문으로 넘어가서 조건이 참인지 확인하는 절차를 가지게 된다.

\*/

}**while**(i<=n);// i가 n과 같거나 작다는 조건이 참일 경우 실행

System.***out***.println("1부터 "+n+"까지 짝수의 합: "+sum);

scanner.close();

}

}

**예제 5-10 BreakStatement break 활용**

**package** Unit5;

**import** java.util.Scanner;

**public** **class** BreakStatement\_5\_10 {

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

// **TODO** Auto-generated method stub

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

System.***out***.print("합을 구할 수 입력 :");

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

**int** i = 1, sum =0;

**while**(**true**) { // 조건식을 true로 지정하여 참의 값이 나올 경우 실행한다는 조건

// 이때 while문은 if(i>n) break 까지 포함하기 때문에 결과값이 true / false로 구분 된다. 따라서 만약에 true가 아닌 false 값이 발생할 경우, 마지막의 break 구문을 이용하여 for을 탈출

sum= sum+i ;

i++;

**if**(i>n) **break**;

}

System.***out***.println("1부터"+n+"까지의 합: "+ sum); // while 구문을 거치면서 sum의 최종값은 while을 통과한 뒤 마지막 결과값이 된다.

scanner.close();

}

}

**예제 5-11 ContinueStatement**

**package** Unit5;

**import** java.util.Scanner;

**public** **class** ContinueStatement\_5\_11 {

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

// **TODO** Auto-generated method stub

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

System.***out***.print("합을 구할 수 입력 :");

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

**int** i = 0, sum =0;

**while** (i<n) {

i++;

**if**(i%3 == 0) // 3의 배수가 아닌 것 = 3을 나누었을 때의 나머지가 0이면 순환시켜서 i의 값이 +1 되도록 설정 , if에 ;를 넣게되면 뒤의 수식들이 if 내부에 포함되는 것인지 별개인지 모호해진다.

**continue**;

sum + = i; // sum = sum + i;

}

System.***out***.println("1부터 "+n+"까지 3의 배수를 제외한 합: "+sum);

scanner.close();

}

}

**예제 5- 12 ReturnStatement**

**int 형 메소드의 return 반환 , computeSum( ) : ( )안의 수까지 합계 (시그마 역할 수행)**

**int 형 메소드에 system.구문은 넣을 수 없다.**

**package** Unit5;

**import** java.util.Scanner;

**public** **class** ReturnStatement5\_12 {

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

// **TODO** Auto-generated method stub

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

System.***out***.print("합을 구할 수 입력: ");

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

**int** sum = *computeSum*(n); // sum에 합계를 대입

System.***out***.println("1부터 "+n+"까지의 합: "+sum);

scanner.close();

}

**public** **static** **int** computeSum(**int** n) {

**int** i = 1, hap =0;

**while**(i < n)

hap += i++;

**return** hap;// while문을 거친 hap의 최종값을 computeSum(int n)의 값으로 반환한다는 의미 -> 메인메소드에서 타 메소드 자체의 값을 출력하고 싶을 떄 return을 이용해서 해당 메소드의 값을 도출해내게 만들 수 있다.

}

}

# **6강**

**예제 6-1 ExtendsClass System. 등등 구문은 반드시 메소드 내부에서 실행 가능!**

**package** Unit6;

**class** SuperClass {

**int** a= 1;

**int** b= 2;

}

**class** SubClass **extends** SuperClass {

**int** sum;

**int** a = 10;

**int** b = 20;

**void** Display() {

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

System.***out***.println("슈퍼 클래스 a = "+**super**.a+" b = "+**super**.b); // 실제 슈퍼클래스인 SuperClass의 멤버 a와 b의 호출

}

**public** **int** computeSum() {

sum = a+b;

**return** sum;

}

**public** **int** computeSup() {

sum = **super**.a+**super**.b;

**return** sum;

}

}

**public** **class** ExtendsClass\_6\_1 {

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

**int** sum1, sum2;

SubClass sub = **new** SubClass();

sub.Display();

sum1 = sub.computeSum();

sum2 = sub.computeSup();

System.***out***.println("서브 클래스 a + b = "+sum1);

System.***out***.println("슈퍼 클래스 a + b = "+sum2);

}

}

**예제 6-2 ConstructorClass 생성자 : 클래스 호출 시 자동 실행**

**package** Unit6;

**class** SuperClass2{

**int** a =1;

SuperClass2(){

System.***out***.println("슈퍼클래스 생성자 호출 a = "+a);

}

}

**class** SubClass2 **extends** SuperClass2{

**int** b =2;

SubClass2(){

System.***out***.println("서브 클래스 생성자 호출 b = "+b);

}

}

**public** **class** ConstructorClass {

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

// **TODO** Auto-generated method stub

SubClass2 sub = **new** SubClass2(); // 클래스를 호출함과 동시에 subclass2의 생성자와 superclass2의 생성자가 동시에 선언(순서는 슈퍼 – 서브)

System.***out***.println("main 메소드 출력문");

}

}

**예제 6-3 ConstructorSuper overloading**

**package** Unit6;

**class** SuperClass3{

**int** a =1; // 초기의 int a 값이 어떤 정수를 갖더라도 큰 의미 x -> 아래의 int 수식에서 int x 의 값으로 바뀌기 때문

SuperClass3(){

System.***out***.println("슈퍼 클래스 생성자 호출1 a = "+a);

}

SuperClass3(**int** x){ // a =1 로 선언한 int a 값을 x로 변환하는 작업

a=x; // this.a = x ; 선언할 수도 있다.

System.***out***.println("슈퍼클래스 생성자 호출2 a = "+a);

}

}

**class** SubClass3 **extends** SuperClass3{

**int** b =2;

SubClass3(){

**super**(100); //super에 100을 넣은 값이 SuperClass3의 int x 값에 100이 들어가는 결과를 만든다.

System.***out***.println("서브 클래스 생성자 호출 b = "+b);

}

}

**public** **class** ConstructorSuper\_6\_3 {

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

// **TODO** Auto-generated method stub

SubClass3 sub = **new** SubClass3(); // 생성자인 SubClass3를 호출한다. 이때 SubClass3에 SuperClass3가 상속되어 있기때문에 SuperClass3의 매개변수도 함께 포함하여 출력 이때 *슈퍼 클래스 생성자 호출1 a =*이 뜨지 않는 이유는 super(100) 때문이다.

만약에 super( )이었다면 *슈퍼 클래스 생성자 호출2 a =* 가 안떴을 것이다.

System.***out***.println("main 메소드 출력문");

}

}

**예제 6-4 MethodTest**

**package** Unit6;

**class** SuperClass4{

**int** a =1;

**public** **void** display() {

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

}

}

**class** SubClass4 **extends** SuperClass4{

**int** b =2;

**public** **void** display() {

System.***out***.println("서브 클래스 출력문 b = "+b);

**super**.display(); //super class의 display를 호출하라는 의미

}

}

**public** **class** MethodTest\_6\_4 {

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

// **TODO** Auto-generated method stub

SubClass4 sub = **new** SubClass4();

System.***out***.println("서브 클래스 메소드에서 슈퍼 클래스 메소드 호출");

sub.display();// sub라고 이름 붙인 SubClass4의 메소드에서 display를 호출. 이때 sub의 display안에 int b 값과 super.display 값이 모두 포함되어 있으므로 함께 호출된다.

}

}

**예제 6-5 OverridingOverloading**

**package** Unit6;

**class** SuperClass5{

**int** a =1;

**public** **void** display() {

System.***out***.println("슈퍼 클래스 출력문 a = "+a); // SubClass5에 동일한 생성자 display가 선언되어 있으므로 SuperClass5의 메소드는 덮힌 상태가 된다.

}

}

**class** SubClass5 **extends** SuperClass5{

**int** b =2 ;

**public** **void** display() {

System.***out***.println("서브 클래스 출력문 1 b = "+b);

}

**public** **void** display(**int** x) {

b =x ;

System.***out***.println("서브 클래스 출력문 2 b = "+b);

}

}

**public** **class** OverridingOverloading\_6\_5 {

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

// **TODO** Auto-generated method stub

SubClass5 sub = **new** SubClass5();

sub.display(); // ()안에 값이 정의되지 않은 메소드의 경우 sub class 내의 display() 메소드가 실행된다. 이때 SuperClass5의 메소드 display는 SubClass5의 메소드로 덮힌 상태라서 super.display를 하지 않는 이상 출력되지 않는다.

sub.display(20);// ()안의 값이 int형태의 데이터 = 20 (정수) 이므로 자동으로 int x를 매개변수로 갖는 메소드 display로 실행된다.

}

}

**예제 6-6 AbstractClassTest 추상클래스 -> 상속 시 메소드 overring을 강제한다.**

**package** Unit6;

**abstract** **class** AbstractClass{ **// 1개의 메소드라도 추상이면 추상 클래스**

**public** **abstract** **void** display1(); **// 추상클래스 선언의 형태**

**public** **void** display2() {

System.***out***.println("추상 클래스에서의 출력문");

}

}

**class** AdsSubClass **extends** AbstractClass{

**int** a =1;

**public** **void** display1() { // 왜 public void display()가 호출되기 전에서는 AdsSubClass값에 오류 표시가 되었을까?

// -> abstract class는 자신의 객체를 바로 생성 불가능 -> display는 추상메소드로 선언되어 있으므로 AbsSubClass(서브 추상 클래스)를 서브 클래스로 정의하여 추상메소드를 실행한다.

System.***out***.println("서브 클래스 출력문 a = "+a);

}

}

**public** **class** AbstractClassTest {

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

// **TODO** Auto-generated method stub

AdsSubClass sub = **new** AdsSubClass();

sub.display1();

sub.display2();

}

}

# **7강**

**예제 7-1 BookOutput 와 InterfaceTest**

**인터페이스 선언 시 { } 미포함**

**인터페이스 구현할 경우 public 을 앞에 붙이기**

**\*BookOutput**

**package** Unit7;

**public** **interface** BookOutput {

**int** ***CODE***[] = {1001,1002,1003};

**void** print(); // 메소드 void print를 선언

**void** getCode(**int** code); // 메소드 void getCode를 선언 이때 int값을 받음

}

**\*InterfaceTest**

**package** Unit7;

**public** **class** InterfaceTest\_7\_1 **implements** BookOutput{

**public** **void** print() { // 반드시 public 을 붙여야 한다.

System.***out***.print("세 CODE: ");

**for**(**int** i =0 ; i < ***CODE***.length; i++)

System.***out***.print(***CODE***[i]+" ");

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

} // print 메소드 실행

**public** **void** getCode(**int** code) {

System.***out***.println("취득한 code = "+code);

}// gerCode 메소드 실행

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

// **TODO** Auto-generated method stub

InterfaceTest\_7\_1 obj = **new** InterfaceTest\_7\_1();

obj.print(); // 3가지 원소를 출력하는 메소드

obj.getCode(1007);// 대입한 값(정수 값 = int)를 출력하는 메소드 int안넣으면 오류 발생

}

}

**예제 7-2 BookOutput / SubBook / Interfaceextends**

**인터페이스 간의 상속에는 overriding 이 필요 없다.**

**\*BookOutput**

**package** Unit7;

**public** **interface** BookOutput {

**int** ***CODE***[] = {1001,1002,1003};

**void** print();

**void** getCode(**int** code);

}

**\*SubBook**

**package** Unit7;

**public** **interface** SubBook **extends** BookOutput { // 인터페이스인 BookOutput\_7\_1을 상속 받음. 이미 SubBook은 BookOutput\_7\_1의 서브 인터페이스임.

**int** ***CODE***[] = {1004, 1005};

**public** **void** printbook(**int** code);

}

**\*InterfaceExtends**

**package** Unit7;

**public** **class** InterfaceExtends\_7\_2 **implements** SubBook{ // interface인 SubBook에 이미 BookOutput\_7\_1이 상속되어 있으므로, SubBook만 상속받아도 동시레 BookOutput\_7\_1도 이용가능하다.

**public** **void** print() { // BookOutput\_7\_1의 print 메소드

System.***out***.print("다섯 CODE: ");

**for**(**int** i =0; i <BookOutput.***CODE***.length; i++) // 추가적으로 BookOutput\_7\_1의 객체 정의 필요 없이 바로 호출 가능

System.***out***.print(BookOutput.***CODE***[i]+" ");

**for**(**int** i =0; i<***CODE***.length; i++)

System.***out***.print(***CODE***[i]+" ");

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

}

**public** **void** getCode(**int** code) { // BookOutput\_7\_1의 getCode 메소드

System.***out***.println("취득한 code = "+code);

}

**public** **void** printbook(**int** code) { // SubBook의 printbook 메소드

String name = "";

**boolean** flag = **true**;

**switch** (code){

**case** 1001: name ="자바"; **break**;

**case** 1002: name ="C++"; **break**;

**case** 1003: name ="ASP"; **break**;

**case** 1004: name ="JSP"; **break**;

**case** 1005: name ="C"; **break**;

**default**: System.***out***.println("잘못된 코드를 입력하였습니다.");

flag = **false**; // case에 해당하지 않는 값을 입력 받았을 경우, flag의 값이 false가 되도록하여 오류 값 dafault 를 출력하도록 설정

}

**if**(flag) // 만약 flag 값이 "true"이면 코드 실행

System.***out***.println("code "+code+"책 이름 = "+name);

}

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

// **TODO** Auto-generated method stub

InterfaceExtends\_7\_2 obj = **new** InterfaceExtends\_7\_2();

obj.print();

obj.getCode(***CODE***[1]); // CODE의 두번째 원소 (= 1) / 컴퓨터는 0부터 첫번째로 인식하기 때문에 1이면 두번째 원소를 지칭

obj.printbook(BookOutput.***CODE***[0]); // 객체.메소드(인터페이스.변수)

}

}

**예제 7-3 BookOutput1 와 ClassExtxtends**

**\*BookOutput1**

**package** Unit7;

**public** **interface** BookOutput1 {

**int** ***CODE***[] = {1001,1002,1003};

**void** print();

**void** getCode(**int** code);

}

**\*ClassExtends**

**package** Unit7;

**class** BookClass{ // int형태의 code를 요구하는 클래스

**public** **void** getCode(**int** code) {

System.***out***.println("code = "+code);

}

}

**public** **class** ClassExtends **extends** BookClass **implements** BookOutput{ // 슈퍼클래스 BookClass를 상속, 인터페이스 BookOutput을 상속

**public** **void** print() { // 인터페이스의 print( )

System.***out***.print("세 CODE: ");

**for**(**int** i =0; i<***CODE***.length; i++) // 인터페에스 BookOutput의 매개변수(배열)

System.***out***.print(***CODE***[i]+ " ");

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

}

**public** **void** getCode(**int** code) { // 인터페이스의 getCode 선언

System.***out***.println("취득한 code = "+ code);

**super**.getCode(code); // 이때 int 형태의 code 값은 슈퍼클래스 BookClass의 getClass를 호출

}

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

ClassExtends obj = **new** ClassExtends();

obj.print();

obj.getCode(***CODE***[2]); // 인터페이스 BookOutput의 매개변수CODE 배열 내의 3번째 원소 출력 -> 1003

}

}

**예제 7-4 DefaultBookOutput**

**인터페이스 내부에서 값을 지정해줄 경우, static 이나 default 를 지정해주어야 한다.**

**\*DefaultBookOutput**

**package** Unit7;

**public** **interface** DefaultBookOutput {

**int** ***CODE***[] = {1001, 1002, 1003};

**int** ***CODE2***[] = {2001, 2002, 2003};

**static** **void** print() {

System.***out***.println("\* static Method 출력 ");

System.***out***.print("세 CODE: ");

**for**(**int** i = 0; i<***CODE***.length; i++)

System.***out***.print(***CODE***[i]+" ");

System.***out***.println("\n");

}

**default** **void** print2() {

System.***out***.println("\* Default Method 출력 ");

System.***out***.print("세 CODE: ");

**for**(**int** i=0; i<***CODE2***.length; i++)

System.***out***.print(***CODE2***[i]+" ");

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

}

}

**\*DefaultEx**

**package** Unit7;

**public** **class** DefaultEx {

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

// **TODO** Auto-generated method stub

DefaultBookOutput.*print*(); // static 인경우, interface 역시 재정의 없이 바로 호출 가능(static = 1개 뿐 최상위의 메소드라는 의미)

DefaultBookOutput db = **new** DefaultBookOutput() {}; // default 로 정의 된 경우, 클래스 재정의 후 호출하듯 재정의 해주고 사용할 메소드를 호출해야 한다.

db.print2(); // default 의 print2 메소드 호출

}

}

**예제 7-5 LambdaEx**

**함수형 인터페이스 인터페이스를 따로 형성x class 안에 함께 선언한다.**

**@FunctionalInterface**

**package** Unit7;

**import** java.util.Scanner;

@FunctionalInterface // 함수형 인터페이스를 사용한다는 의미 내포

**interface** Compute{

**int** max(**int** n1, **int** n2); // 인터페이스 Compute에서 max 메소드 선언

}

**public** **class** LambdaEx {

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

// **TODO** Auto-generated method stub

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

System.***out***.print("두 수 입력 --> ");

**int** n1 = S.nextInt();

**int** n2 = S.nextInt();

System.***out***.println("\*\* 람다식 활용 \*\*");

Compute maxNum = (x, y) -> x>y? x : y;

// main 메소드에 인터페이스 Compute의 객체 maxNum 을 생성

//int x, y 가 x >y 값이 참이면 x 거짓이면 y 를 maxNum에 저장

System.***out***.print(n1+" 과(와) "+n2+" 중 큰 수: ");

System.***out***.println(maxNum.max(n1, n2));

/\* main 메소드에 선언된 maxNum의 객체 안에 scanner로 입력받은 n1과 n2 삽입

\* 이때 삽입된 n1과 n2 는 int max(n1 과 n2)로 저장되며 n1 =x , n2 =y로 대응된다.

/\*

/\*

\* 결과적으로 maxNum >> int x, y; if(x>y) return = x; default = y;를 대신한 람다식

\*/

S.close();

}

}

**예제 7-6 LambdaFunEx**

**객체 = ( ) -> { body내용 } ; →객체에서 실행할 명령문을 담는다는 의미**

**package** Unit7;

**import** java.util.Scanner;

@FunctionalInterface

**interface** FunctionInterface{

**public** **void** methodFunc();

}

**public** **class** LambdaFunEx {

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

// **TODO** Auto-generated method stub

FunctionInterface fi; // 함수형 인터페이스 fi객체 선언

System.***out***.println("\*\* 람다식 활용 \*\*");

fi = () -> { //이 부분

String str = "람다식 활용";

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

};

fi.methodFunc();

fi= () ->{ //이 부분

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

System.***out***.print("두 수 입력: ");

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

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

**int** sum = x +y;

System.***out***.println(x+" + "+y+" = "+sum);

scanner.close();

};

fi.methodFunc();

}

}

# **8강**

**예제 8-1 TestString**

**random 클래스 메소드 :**

int형객체명 = (**int**)(Math.*random*()\*100+1); //1부터 100까지의 난수 생성

**package** Unit8;

**import** java.util.Random;

**public** **class** TestString {

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

// **TODO** Auto-generated method stub

String str1 = **new** String(" java\t"); // java 뒤에 한 탭 만큼 간격 두기.

String str2 = **new** String(", C++, C ");

String str3 = "java";

System.***out***.println("str1: "+str1+" , str2: "+str2+" , str3: "+str3);

str1 = str1**.trim();** // 공백문자와 탭을 사용한 공백문자 삭제

System.***out***.println("trim 후 str1:\""+str1+"\"");// \ -> "은"을 출력

System.***out***.print("str1 과 str3이 같은 문자열 인가? ");

System.***out***.println(str1**.contentEquals(**str3**)**); // 똑같이 가지고 있으면 true 반환

System.***out***.println("str1 안에 v 문자의 유무:"+str1.contains("v"));// v를 포함하고 있으면 true, 아니면 false 반환

str1 = str1**.concat(**str2**)**;// str1 객체에 str2 객체를 연결

System.***out***.println("str1과 str2 연결: "+str1); // 바로 위에 선언한 str1 을 불러오기 떄문에, str1.concat의 값을 불러옴

String s[] = str1**.split(**","**)**; // str1의 원소 내 문자열 분리

**for**(**int** i=0; i<s.length;i++)

System.***out***.println("분리 된 문자열"+i +":"+ s[i]);

str1 = str1**.replace(**"java","C#"**)**; // str1의 값을 java와 C#으로 변경

System.***out***.println("java를 C#으로 replace:"+str1);

str1 = str1**.substring(**4**)**; // 인덱스 4부터 끝까지 서브 스트링 반롼(substring의 값 출력)

System.***out***.println("substring 후 str1:"+str1);

**char** ch = str1**.charAt(**0**)**;// str1의 0번째 문자

System.***out***.println("str1의 index 0 문자 :"+ch);

StringBuffer sb = **new** StringBuffer("Hello"); // StringBuffer 클래스 선언하여 호출

System.***out***.println("sb: "+sb);

sb**.append(**"world"**)**;// 문자열 끝에 "world" 추가

System.***out***.println("추가 후 sb: "+sb);

sb**.insert(**5, "java"**)**;// sb의 index 5에 "java" 삽입

System.***out***.println("삽입 후 sb: "+sb);

sb**.replace(**6, 10, "my"**)**;// sb의 index 6부터 9까지 "my"로 변경

System.***out***.println("replace 후 sb: "+sb);

}

}

**예제 8-2 TestArrays**

**package** Unit8;

**import** java.util.Arrays;

**public** **class** TestArrays {

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

// **TODO** Auto-generated method stub

**int** [] ar = {12, 75, 28, 83, 36};

Arrays.*parallelSort*(ar);

System.***out***.print("배열 ar : ");

**for**(**int** n : ar) // 배열 n 의 모든 원소를 지칭

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

/\*

\* for(int n : ar) 은 ar안의 각 원소를 n으로 간주하고 찍어내라는 의미

\*/

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

System.***out***.println("배열 ar : "+Arrays.*toString*(ar));// [ ] 안에 원소들 나열

**int** inx = **Arrays.*binarySearch*(**ar, 75**)**; // 몇 번 째 순서인지 검증

System.***out***.println("75의 첨자 : "+inx);

**Arrays.*fill*(**ar, 15**)**;// ar의 모든 원소 값을 15로 변경

System.***out***.println("배열 ar : "+Arrays.*toString*(ar));

**int** [] ar2 = **Arrays.*copyOf*(**ar, ar.length**)**; // 배멸 ar의 원소 모두를 복사하여 반환 , length의 경우 ar의 원소 수를 의미하므로 ar.length로 입력

System.***out***.println("배열 ar2 : "+**Arrays.*toString*(**ar2**)**);

System.***out***.println("두 배멸이 같은 가? "+**Arrays.*equals*(**ar, ar2**)**); // 두 배열이 같은 지 비교 true / false 값이 출력된다.

}

}

**예제 8-3 TestCalendar**

**package** Unit8;

**import** java.util.Calendar;

**public** **class** TestCalender {

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

// **TODO** Auto-generated method stub

Calendar cal = Calendar.*getInstance*(); // 현재 일자를 반환

**int** year = cal.get(Calendar.***YEAR***);

**int** month = cal.get(Calendar.***MONTH***);

**int** day = cal.get(Calendar.***DATE***); // \*\*\*DAY가 아니라, DATE라고 입력해야한다.\*\*\*

**int** hour = cal.get(Calendar.***HOUR\_OF\_DAY***);

**int** min = cal.get(Calendar.***MINUTE***);

System.***out***.println("\*\*\* 현재 시간 \*\*\*");

System.***out***.println(year+ "년 "+ month +"월 "+ day+"일 ");

System.***out***.print(hour+"시 "+min+"분 ");

System.***out***.println(cal.get(Calendar.***SECOND***)+"초 "); // 따로 int 데이터 형태 정의 없이 바로 초(second) 불러오기

}

}

# 9강

예제 9-1 DevideZeroException **do – while 구문에 try / catch를 삽입한 경우**

**package** Unit9;

**import** java.util.Scanner;

**public** **class** DevideZeroException {

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

// **TODO** Auto-generated method stub

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

**int** num;

**int** divisor; // 나눗셈 상수

System.***out***.print("수를 입력하시오: ");

num = scanner.nextInt();

**do** {

System.***out***.print("제수(상수)를 입력하시오: ");

divisor = scanner.nextInt();// 나눗셈 상수 입력

**try** {

System.***out***.println(num+"를 "+divisor+"로 나눈 몫: "+num/divisor);

**break**;

}

**catch**(ArithmeticException e) { // 0으로 나누게 될 경우의 예외처리

System.***out***.println("0으로 나눌 수 없어요. 다시 입력하세요.");

**continue**;

}

}**while**(**true**);

scanner.close();

}

}

예제 9-2 ExecuteThread **Thread라는 클래스가 자바 내에 이미 존재한다.**

**package** Unit9;

**import** java.util.Calendar;

**class** TestThread **extends** Thread // Thread 클래스 상속

{**public** TestThread(String name) {

**super**(name);

}

**public** **void** run() {

**try** {

Calendar cal = Calendar.*getInstance*();

**for**(**int** i =1; i<=3; i++) {

**int** year = cal.get(Calendar.***YEAR***);

**int** month = cal.get(Calendar.***MONTH***)+1;

**int** day = cal.get(Calendar.***DAY\_OF\_MONTH***);

**int** hour = cal.get(Calendar.***HOUR\_OF\_DAY***);

**int** min = cal.get(Calendar.***MINUTE***);

System.***out***.print("\*현재시간 : ");

System.***out***.print(year+"년 "+month+"월 "+day+"일 ");

System.***out***.print(hour+"시 "+min+"분");

System.***out***.println(cal.get(Calendar.***SECOND***)+"초");

System.***out***.println("\* 실행중인 스레드 : "+getName()+" i = "+i);

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

Thread.*sleep*(1000);// 1초 정지

}

}**catch**(InterruptedException e) {

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

}

}

}

**public** **class** ExecuteThread {

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

TestThread thr = **new** TestThread("현재시간 표시 스레드");

thr**.start();** // 스레드 동작 명령

}

}

예제 9-3 Excute3ThreadRun **스레드의 join 메소드**

**package** Unit9;

**import** java.util.Calendar;

**class** TestThread **extends** Thread{

**public** TestThread(String name) {

**super**(name);

}

**public** **void** run() {

**try** {

Calendar cal = Calendar.*getInstance*();

**int** year = cal.get(Calendar.***YEAR***);

**int** month = cal.get(Calendar.***MONTH***)+1;

**int** day = cal.get(Calendar.***DAY\_OF\_MONTH***);

**int** hour = cal.get(Calendar.***HOUR\_OF\_DAY***);

**int** min = cal.get(Calendar.***MINUTE***);

System.***out***.print("\* 현재시간 : ");

System.***out***.print(year+"년 "+month+"시 "+day+"일 ");

System.***out***.print(hour+"시 "+min+"분 ");

System.***out***.println(cal.get(Calendar.***SECOND***)+"초 ");

System.***out***.println("\* 실행중인 스레드 : "+getName());

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

Thread.*sleep*(1000);

}**catch**(InterruptedException e){

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

}

}

}

**public** **class** Excute3ThreadRun {

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

// **TODO** Auto-generated method stub

TestThread thr;

**for**(**int** i=1; i<= 3; i++) {

thr = **new** TestThread("현재시간 표시 스레드"+i);

thr.start();

thr**.join();**// 스레드의 처리가 끝날 때까지 기다림

}

}

}

예제 9-4 RunnableThread **스레드 구현의 다른 방법 Runnabe 인터페이스 를 구현**

**package** Unit9;

**import** java.util.Calendar;

**class** exThread **implements** Runnable{ **// Runnable 인터페이스 구현**

**public** **void** run() {

**try** {

Calendar cal = Calendar.*getInstance*();

**for**(**int** i =1 ; i <= 3; i++) {

**int** year = cal.get(Calendar.***YEAR***);

**int** month = cal.get(Calendar.***MONTH***)+1;

**int** day = cal.get(Calendar.***DAY\_OF\_MONTH***);

**int** hour = cal.get(Calendar.***HOUR\_OF\_DAY***);

**int** min = cal.get(Calendar.***MINUTE***);

System.***out***.print("\* 현재시간 : ");

System.***out***.print(year + "년 "+month+"달 "+day+"일 ");

System.***out***.println(hour+ "시"+min+"분");

System.***out***.println("\* 실행 중인 스레드 : "+"ㅑ = "+i);

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

Thread.*sleep*(1000);

}

}**catch**(InterruptedException e) {

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

}

}

}

**public** **class** RunnableThread {

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

// **TODO** Auto-generated method stub

Thread thr = **new** Thread(**new** exThread());

thr.start(); // 동작 방법은 스레드와 동일하다.

}

}

예제 9-5 ExLambdaThread **입력, 실행을 람다식으로 처리 = ( ) -> { body내용 };**

**package** Unit9;

**import** java.util.Calendar;

**public** **class** ExLambdaThread {

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

// **TODO** Auto-generated method stub

**Runnable irthr = () -> {** // 람다식 시작 인터페이스의 명령어를 입력

**try** {

Calendar cal = Calendar.*getInstance*();

**for**(**int** i =1 ; i <= 3; i++) {

**int** year = cal.get(Calendar.***YEAR***);

**int** month = cal.get(Calendar.***MONTH***)+1;

**int** day = cal.get(Calendar.***DAY\_OF\_MONTH***);

**int** hour = cal.get(Calendar.***HOUR\_OF\_DAY***);

**int** min = cal.get(Calendar.***MINUTE***);

System.***out***.print("\* 현재시간 : ");

System.***out***.print(year + "년 "+month+"달 "+day+"일 ");

System.***out***.println(hour+ "시"+min+"분");

System.***out***.println("\* 실행 중인 스레드 : "+"i = "+i);

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

Thread.*sleep*(1000);

}

}**catch**(InterruptedException e) {

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

}

**};**// 람다식 끝

Thread thr = **new** Thread(irthr);

thr.start();

//new THread(irthr).start(); 위의 두 문장을 한 문장으로 작성

}

}

**연습문제 모음**

# **4강**

**4-1 자바에서 사용가능한 연산자를 종류별로 구분하여 설명하시오.**

연산자는 크게 8가지로 구분할 수 있다.

**산술연산자:** + - \* / % 단항+ 단항- 로 산술연산을 하는 기능

**관계연산자**: > >= < <= == != 양쪽 항의 관계를 표현하는 기능

**논리 연산자:** && || ! 양쪽 항의 true, false 값 간의 관계를 표현하는 기능

**증가 감소 연산자:** ++ -- 일정한 크기로 증/감이 가능하게 하는 기능

**비트 연산자:** & | ^ ~ << >> >>> 2진수인 비트 값에 대한 빠른 계산이 가능

**조건 연산자:** ? : if와 같은 기능을 하며 조건에 따른 참/거짓 값의 구분하는 기능

**배정 연산자:** += \_= \*= /= %= &= |= ^= <<= >>= >>>= 양측 항의 계산에 있어 동일한 항이 존재할 때 생략하고 입력을 편하게 하는 기능

**캐스트 연산자 :** (자료형) 임의로 데이터의 “자료형” 형태를 바꾸는 기능

**4-2 다음 수식을 자바 산술식으로 나타내시오.**

1. X + (y\*z)
2. X – (y\*(z\*z))
3. X \*(3/5)
4. (x-y)/x

4-3 **자바에서 사용하는 연산자들의 우선순위를 비교하여 나열하시오.**

증가 감소 연산자 > 산술연산자 > 비트 연산자 > 관계연산자 > 논리 연산자> 조건 연산자 > 배정 연산자 > 캐스트 연산자

4-4 **다음 수식을 계산 한 후 결과 값을 쓰시오.(단 x=10, y= 20이라 가정)**

1. 20
2. 7
3. true
4. true
5. 4 (2진법== 100)

**4-5 메소드를 사용해 두 정수를 입력받아 산술연산(+ - \* / %)과 비트연산(& | ^)을 수행한 결과를 출력하는 프로그램을 작성하시오.**

**package** 과제;

**public** **class** Calculation1 {

**static** **void** Calculation11(**int** x, **int** y){

**int** a= x+y;

**int** b= x-y;

**int** c= x\*y;

**double** d= (**double**) x/y;

**int** e= x%y;

**int** f= x&y;

**int** g= x|y;

**int** h= x^y;

System.***out***.println("두 정수 입력 -->"+x+" "+y);

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

System.***out***.println(x+" - "+y+" = "+b);

System.***out***.println(x+" \* "+y+" = "+c);

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

System.***out***.println(x+" % "+y+" = "+e);

System.***out***.println(x+" & "+y+" = "+f);

System.***out***.println(x+" | "+y+" = "+g);

System.***out***.println(x+" ^ "+y+" = "+h);

}

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

// **TODO** Auto-generated method stub

Calculation1 CAL =**new** Calculation1();

**int** x =11, y=3;

CAL.*Calculation11*(x, y);

}

}

**4-6 세 수 중 가장 큰 수를 구하는 계산식을 하나의 조건 연산자를 사용하여 작성하시오.**

**package** 과제;

**public** **class** ChooseMax3 {

**static** **void** Read(**int** a, **int** b, **int** c) {

System.***out***.println("세 정수 입력 --> "+a+" "+b+" "+c);

**if**((a > b))

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

**else** **if**((b>c))

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

**else** **if**((c>a))

System.***out***.println("max = "+c);

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

}

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

// **TODO** Auto-generated method stub

ChooseMax3 MAX = **new** ChooseMax3();

MAX.*Read*(98, 75, 67);

MAX.*Read*(12, 34, 62);

MAX.*Read*(45, 91, 75);

}

}

# **5강**

**5-1 if 문과 switch문을 비교하시오**

**If 문:** 1개의 조건이 만족되면 실행하고 만족되지 않을 경우 else if 또는 else의 구문으로 이동하여 맞는 조건을 실행하는 구문

**Switch 문**: 여러 개의 조건 중 부합하는 하나의 조건을 찾을 경우 각 조건들을 case로 분류하여 각 구문화 하고, case에서 벗어나기 위해 break; 을 이용한다.

**5-2 반복문을 사용하는 목적이 무엇이며 사용 예를 각각의 반복문별로 예시를 드시오.**

**For 구문:** 정해진 횟수만큼 명령문들을 반복하는 기능을 수행

**For -each 구문:** 배열이나 enum클래스(열거 enumeration)의 원소 수 만큼 루프를 반복해서 실행하고 각 원소릐 값을 순서대로 접근해 사용할 경우 유용한 반복문

형식1 : for(자료형 변수이름 : 배열이름)

문장;

**int**[] num= {10,20,30,40,50};

**int** sum =0;

**for**(**int** n : num)

sum +=n;

형식2 : for(enum클래스 이름 객체 이름 : enum클래스 이름.values() )

문장;

Enum 클래스의 경우, 클래스 외부나 클래스의 멤버를 선언하는 위치에서 정의(main method / method에서는 사용 불가능)

**While 문 :** 주어진 조건식의 조건이 참일 때 명령문을 반복해서 실행. 만약 조건이 거짓이면 while 문 안의 명령문을 한번도 실행x

형식: (앞에 do 가 오면 조건에 관계 없이 실행한다는 의미를 갖는다)

초기식;

while(조건식){

문장;

증감식;

}

**5-3 분기문을 종류대로 분류하고 각각을 사용하는 경우를 예를 들어서 설명하시오.**

**break 문 :**

문장을 실행한 후 switch문을 벗어나거나 for문 while문의 반복 루프에서 빠져나오기 위해 사용

break;

break 레이블; -> 이동시킬 레이블(구문)을 입력

**continue문 :**

반복문에서 문장을 실행하다가 다음 반복을 시작하는 곳으로 제어를 옮기는 기능을 수행

다음 문장을 실행하지 않고 반복을 시작할 곳으로 제어를 이동

continue;

continue 레이블;

**return 문:**

메소드의 실행을 종료하고 자신을 호출한 곳으로 제어를 옮겨 이동하는 문장

return;

return 식;

**5-4 (특정)구구단을 출력하는 프로그램 작성**

**package** 연습문제;

**import** java.util.Arrays;

**public** **class** Gugudan\_5\_4 {

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

// **TODO** Auto-generated method stub

System.***out***.println("\t\*\*\*\*\*\*\*\*\*\*\*\* 구구단 \*\*\*\*\*\*\*\*\*\*\*\*" );

**int** Dan [][] = **new** **int**[12][12];

**for**(**int** i =1; i<11; i+=2) {

**for**(**int** j=3; j<11; j+=2) {

Dan[i][j] = i \* j;

System.***out***.print( +i+" \* "+j+" = "+Dan[i][j]+"\t");

}

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

}

}

}

# **6강**

**6-1 슈퍼클래스와 서브클래스를 비교하여 설명하시오**

**super class :**

기존의 클래스로서 상속을 해주는 대상이 되는 원본 클래스의 형태

**sub class :**

슈퍼 클래스로부터 상속을 받는 대상으로서 슈퍼클래스 내의 멤버도 모두 상속받아 사용할 수 있게 된다.

단 이때 주의 사항은 class 간의 상속은 단 1개 씩 일대일 대응의 상속만 가능하다.

**6-2 추상 클래스란 무엇인가? 추상 클래스에서 객체를 생성하는 방법을 설명하시오**

**Abstract (추상)클래스/ 메소드:**

**abstract 클래스를 상속받은 클래스가 강제로 abstract 클래스 내의 abstract 메소드를 overrinding 하도록 강제하는 기능(일종의 지침서 이자 형태를 변형하여 사용하는 내용)**

직접적으로 사용(호출) 불가능하다. (private을 외부 클래스에서 사용할 때 간접적으로 사용한 것과 유사)상속하여 사용하도록 강제된 형태

특징:

1) abstract가 사용된 메소드에는 본문이 없다. 따라서 { } 도 없이 바로 ( ) ; 로 구문이 끝나게된다.

2) 클래스 내부에 1개의 메소드라도 abstract가 선언되어 있는 경우에 class 앞에도 반드시 abstract를 선언해야 한다.

**package** 실험실;

**abstract** **class** Example{

**public** **abstract** **void** Set(); // 이처럼 메소드 내부에 body가 없다.

}

**class** Test **extends** Example{

**int** a,b;

**public** **void** Set() { // overriding 시에 반드시 abstract method의 접근제어자와 동일한 제어자 설정 필요

**int** b =1;

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

}

}

**public** **class** AbstractClassEX {

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

// **TODO** Auto-generated method stub

Test TT = **new** Test();

TT.Set();

}

**6-3 슈퍼 클래스와 서브 클래스에 동시에 정의한 멤버 변수를 구분하여 사용하는 방법을 설명하시오**

동시에 정의한 멤버 변수의 경우 일반적으로 컴퓨터는 가장 최근에 추가된 멤버를 변수로 인지한다. 따라서 sub class 의 멤버 변수는 별다른 조작 없이 바로 멤버변수를 호출하면 된다.

그러나 super class의 멤버 변수를 호출할 경우, super.”멤버 변수 이름” 으로 호출한다.(클래스 나 객체 내의 메소드/ 멤버를 호출 할 때 처럼)

**6-4 슈퍼 클래스에 선언한 메소드를 서브 클래스에서 재정의 했을 경우 두 메소드를 구분하여 사용하는 방법을 예를 들어 설명하시오.**

동시에 정의한다 = overriding 을 의미

일반적으로 overriding 된 경우 서브 클래스에서 새로 정의된 메소드를 사용하게 된다.

이때 슈퍼 클래스의 멤버 변수로 사용할 때엔 super.”메소드 이름”( ) ; 으로 지정해주어야 한다.

**class** SuperClass4{

**int** a =1;

**public** **void** display() {

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

}

}

**class** SubClass4 **extends** SuperClass4{

**int** b =2;

**public** **void** display() {

System.***out***.println("서브 클래스 출력문 b = "+b);

**super**.display();

}

}

**6-5 세 정수와 세 실수를 읽고 서브 클래스를 생성하여 세 정수의 합과 세 실수의 합을 구해 출력하는 과정을 메소드를 사용하여 프로그램을 작성하시오.**

**package** 연습문제;

**import** java.util.Scanner;

**class** Read{

**int** aa, bb, cc ;

**double** AA, BB, CC;

**public** **void** Read2() {

System.***out***.print("세 정수와 세 실수 입력 --> ");

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

**int** aa = SC.nextInt();

**double** AA =SC.nextDouble();

**int** bb = SC.nextInt();

**double** BB =SC.nextDouble();

**int** cc = SC.nextInt();

**double** CC =SC.nextDouble();

**this**.aa=aa;

**this**.bb=bb;

**this**.cc=cc;

**this**.AA= AA;

**this**.BB=BB;

**this**.CC=CC;

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

}

}

**class** Run **extends** Read {

**public** **void** run(){

**super**.Read2();{

System.***out***.println("\*\* 세 정수와 세 실수의 합 \*\* ");

**int** sum1 = aa+bb+cc;

**double** sum2 = AA+BB+CC;

System.***out***.println(aa+" + "+bb+" + "+cc+" = "+sum1);

System.***out***.println(AA+" + "+BB+" + "+CC+" + "+" = "+sum2);

}

}

}

**public** **class** 세정수와실수연습문제{

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

// **TODO** Auto-generated method stub

Run RR= **new** Run();

RR.run();

}

}

# **7강**

**7-1 인터페이스란? 클래스와 공통점, 차이점을 기술하시오.**

**Interface : 인터페이스(규제 中 일부로서 제작자와 사용자의 원활한 소통을 위한 구조)**

Interface를 참조한 객체는 interface가 선언한 메소드만을 가져야 한다.(일종의 지침서/설명서)

특징:

1) 반드시 인터페이스에 선언되어 있는 메소드의 몸체(body)를 overriding 해야 한다.

2) interface에 선언되어 있는 모든 메소드는 반드시 “public”이어야 한다.

Interface 내의 메소드는 본문이 없다. { } 없이 일반 메소드를 실행하는 것처럼 ( ) ; 로 끝

3) 1개의 class가 여러개의 interface를 구현할 수 있다.

4) 구현할 때는 implements 를 사용한다.

5) 인터페이스 끼리도 상속(구현)이 가능하다.

6) 1개의 인터페이스 파일에는 1개의 public inerface만 위치할 수 있다.

클래스처럼 여러 개의 public 인터페이스 선언 불가능

Publlic이 아닌, default interface는 여러개 선언 가능

**클래스와의 차이점:**

1) 클래스는 내부에 내용을 가지고 있을 수 있다. 추상 클래스 마저도 내용이 있는 일부의 메소드, 매개변수를 가질 수 있다.

2) 클래스를 상속받을 경우, 슈퍼클래스의 모든 메소드를 서브클래스에 재정의할 필요가 없다.(자동 상속)

3) 클래스 간에는 단 1개의 클래스만 서로 상속 가능하다.

4) 클래스의 상속에는 extends를 이용한다.

**7-2 자바에서 클래스의 다중상속을 흉내 낼 수 있는 방법을 설명하시오.**

implements 뒤에 여러 개의 인터페이스를 , 로 나열하여 삽입 가능하다.

**package** 실험실;

**public** **class** InterfaceEx **implements** I4, I5 {

**public** **void** x() {

}

**public** **void** y() {

}

**public** **void** z() {

}

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

// **TODO** Auto-generated method stub

}

}

**7-3 인터페이스에서 슈퍼 인터페이스를 서브 인터페이스로 확장하여 정의하는 방법을 예를 들어 설명하시오.**

**package** 실험실;

**public** **interface** I3 {

**public** **void** x();

}

**interface** I4 **extends** I3{

**public** **void** y();

}

**interface** I5 **extends** I4{

**public** **void** z();

}

위의 예시처럼 슈퍼 인터페이스를 서브 인터페이스로 상속할 수 있다.

**7-4 인터페이스에서 다중 상속이 이뤄지는 경우를 예를 들어 설명하시오.**

**package** 실험실;

**public** **class** InterfaceEx **implements** I4, I5 {

**public** **void** x() {

System.***out***.println("x를 출력");

}

**public** **void** y() {

}

**public** **void** z() {

}

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

// **TODO** Auto-generated method stub

InterfaceEx IE = **new** InterfaceEx();

IE.x();

}

}

위의 예시처럼 InterfaceEx클래스가 인터페이스 I4와 I5를 다중 상속 받는다.

**7-5 슈퍼 인터페이스와 서브 인터페이스에 동시에 선언한 상수를 서로 구분하여 사용하는 방법을 설명하시오.**

**package** Unit7;

**class** BookClass{ // int형태의 code를 요구하는 클래스

**public** **void** getCode(**int** code) {

System.***out***.println("code = "+code);

}

}

**public** **class** ClassExtends **extends** BookClass **implements** BookOutput // 슈퍼클래스 BookClass를 상속, 인터페이스 BookOutput을 상속

{**public** **void** print() {

System.***out***.print("세 CODE: ");

**for**(**int** i =0; i<***CODE***.length; i++) // 인터페에스 BookOutput의 매개변수(배열)

System.***out***.print(***CODE***[i]+ " ");

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

}

**public** **void** getCode(**int** code) { // getCode 선언

System.***out***.println("취득한 code = "+ code);

**super**.getCode(code); // 이때 int 형태의 code 값은 슈퍼클래스 BookClass의 getClass를 호출

}

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

ClassExtends obj = **new** ClassExtends();

obj.print();

obj.getCode(***CODE***[2]); // 인터페이스 BookOutput의 매개변수CODE 배열 내의 3번째 원소 출력 -> 1003

}

}

에서 **super.getCode(code);** 처럼 “ super.멤버”로 상위 인터페이스를 선언할 수 있다.

**7-6 인터페이스에서 dafault 메소드와 static 메소드를 사용하는 방법을 비교하여 설명하시오.**

**default 메소드 호출:**

**인터페이스이름 객체이름 = new 인터페이스이름() { };**

**이때 뒤에 { } ;를 반드시 추가해주어야 한다.**

**DefaultBookOutput db = new DefaultBookOutput() {};**

**static 메소드 호출:**

인터페이스이름.static 메소드이름( );

**DefaultBookOutput db = new DefaultBookOutput();**

**7-7 두 수의 합을 구하는 add(x,y) 와 1 부터 n까지의 짝수의 합을 구하는 add(n) 메소드를 갖는 인터페이스를 선언하고, 인터페이스를 구현하는 클래스를 만들어 두 메소드의 실행 결과를 출력하는 프로그램을 작성하시오.**

**인터페이스1:**

**package** 연습문제;

**public** **interface** 인터페이스연습문제1 {

**default** **void** add1()

{

**int** a = 38;

**int** b = 15;

**int** sum = a+b;

}

}

**인터페이스2:**

**package** 연습문제;

**public** **interface** 인터페이스연습문제2{

**default** **void** add2(){

**int** A = 12;

}

}

**클래스**

**package** 연습문제;

**public** **class** 인터페이스연습문제 **implements** 인터페이스연습문제1, 인터페이스연습문제2 {

**public** **void** add1() {

**int** a = 38;

**int** b = 15;

**int** sum = a+b;

System.***out***.println("두 정수 입력 --> "+a+" "+b);

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

}

**public** **void** add2() {

**int** A = 13;

**int** i=1;

**int** sum =0;

**while**(i++<A) {

**if**(i%2==0) {

sum +=i;

}

}

System.***out***.println("1부터 12까지의 짝수의 합 : "+sum);

}

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

// **TODO** Auto-generated method stub

인터페이스연습문제 IT = **new** 인터페이스연습문제();

IT.add1();

IT.add2();

}

}

# **8강**

**8-1 자바에서 지원하는 패키지들의 종류와 기능을 설명하시오**

**기본패키지: java.lang java.util java.io java.awt javax.swing java.applet**

**java.lang** 에는 Arrays, Date, Random, Scanner

**java.awt** 추상 윈도우 툴킷 클래스들을 포함하며 GUI응용프로그램을 만들 수 있는 패키지

**javax.swing** 스윙 컴포넌트 클래스를 포함하며 GUI응용프로그램을 만들 수 있는 패키지

**java.awt.image** 이미지 처리와 관련된 클래스를 포함

**java.io** 입출력 관련된 클래스들을 포함

**java.net** 네트워크 연결과 관련된 클래스들을 포함

**java.applet** 애클릿을 만ㄷㄹ 수 있게 해주는 패키지

등...

**8-2 가장 기본이 되는 패키지는 무엇이고 어떤 기능을 포함하고 있는 가?**

**java.lang** 에는 Arrays, Date, Random, Scanner 등 다양한 클래스를 포함하며 import 문을 사용하지 않아도 자바 컴파일러에서 자동으로 포함시키는 패키지

**8-3 새로운 패키지를 선언하는 방법과 기존에 만들어진 패키지를 프로그램에 포함시켜 사용하는 방법을 설명하시오**

패키지를 생성하는 방법: file – new 에서 package 를 설정

생성된 패키지를 이용하는 방법: import 문을 사용한다.

**import 패키지이름. 클래스이름;**

**8-4 래퍼 클래스란? 래퍼 클래스에는 어떤 클래스들이 있는 가?**

래퍼클래스(Wrapper class) : 자바에서 지원하는 8개의 기본자료형 데이터에 대한 클래스를 지원하며 객체로 처리할 수 있도록 제공

ex) int 형 → Integar , Char형 → Charactor, double형 → Double

넘버클래스(Number class) : 8개의 기본자료형 데이터의 추상클래스로서 Byte, Short, Integar, Long, Float, Double 클래스릐 슈퍼클래스로 기능

**8-5 integar 클래스에서 사용하는 메소드의 종류와 기능을 설명하시오.**



**8-6 String 클래스에서 선언한 메소드들의 기능을 설명하시오.**



**8-7 1부터 100까지의 값 중 10개의 정수를 난수로 생성한 후 두 개의 배열 a1, a2에 각각 저장하고 각 배열에서 가장 큰 값을 갖는 첨자를 찾아 각 각 출력하고 두 배열을 정렬한 후 두 배열이 같은 지 비교해 출력하시오.**

**package** 연습문제;

**import** java.util.Arrays;

**import** java.util.Random;

**class** 난수{

**static** **void** Run(){

**int** n[]= **new** **int**[11];

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

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

}

System.***out***.println("a1: "+Arrays.*toString*(n));

**int** max = n[0];

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

**if**(max<n[i]) { // max(n의 i 번째 인자인 max)가 n 배열 내의 i+1보다 작은지 확인

max = n[i]; // max에 새로운 n의 i+1번째 인자를 넣어주는 구문

}

}

System.***out***.println("a1 원소 중 maxinum = "+max);

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

**int** n2[]= **new** **int**[11];

**for**(**int** j=0; j<=10;j++) {

n2[j] = (**int**)(Math.*random*()\*100+1); // 난수 지정

}

System.***out***.println("a2: "+Arrays.*toString*(n2)); // 배열출력

**int** max2 = n2[0];

**for**(**int** i=0; i<n2.length; i++) {

**if**(max2<n2[i]) { // max(n의 i 번째 인자인 max)가 n 배열 내의 i+1보다 작은지 확인

max2 = n2[i]; // max에 새로운 n의 i+1번째 인자를 넣어주는 구문

}

} //배열 내의 가장 큰 수를 max2에 담기

System.***out***.println("a2 원소 중 maxinum = "+max2);

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

System.***out***.println("a1 과 a2는 같은 배열인가? "+Arrays.*equals*(n, n2));

// 배열비교

}

}

**public** **class** 난수지정 **extends** 난수 {

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

난수 NN = **new** 난수();

NN.*Run*();

}

}

# **9강**

**9-1 자바에서 발생하는 예외에는 어떤 것이 있는 가?**

**ArithmeticException** : 0으로 나눌 경우

**ArrayIndexOutofBoundsException**  : 배열의 첨자가 음수 값이거나 범위를 벗어날 경우

**InterruptedException** : 부적절한 자료형으로 변환하거나 자료형에 맞지 않게 입출력할 때 인터럽트가 발생하는 경우

**9-2 예외를 발생하고 처리하는 구문을 예를 들어 설명하시오.**

1) **getMessage()** : 축약정보 (오류)

2) **toString()** : 자세한 정보 (오류, 원인)

3) **printStackTrace()** : 모든 에러사항을 출력(오류, 원인, 어떤 위치의 로직)

**import** java.util.Calendar;

**class** TestThread **extends** Thread

{**public** TestThread(String name) {

**super**(name);

}

**public** **void** run() {

**try** {

Calendar cal = Calendar.*getInstance*();

**for**(**int** i =1; i<=3; i++) {

**int** year = cal.get(Calendar.***YEAR***);

**int** month = cal.get(Calendar.***MONTH***)+1;

**int** day = cal.get(Calendar.***DAY\_OF\_MONTH***);

**int** hour = cal.get(Calendar.***HOUR\_OF\_DAY***);

**int** min = cal.get(Calendar.***MINUTE***);

System.***out***.print("\*현재시간 : ");

System.***out***.print(year+"년 "+month+"월 "+day+"일 ");

System.***out***.print(hour+"시 "+min+"분");

System.***out***.println(cal.get(Calendar.***SECOND***)+"초");

System.***out***.println("\* 실행중인 스레드 : "+getName()+" i = "+i);

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

Thread.*sleep*(1000);// 1초 정지

}

}**catch**(InterruptedException e) {

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

}

**9-3 스레드란 무엇인가? 프로세스와의 차이점을 기술하시오.**

**thread(스레드)** 는 순차 프로그램과 유사하게 실행의 흐름을 나타내는 것으로 시작, 실행, 종료의 실행 순서를 갖는다. 스레드는 단일 프로그램이 아니므로 독립적으로 실행할 수 없으며 프로그램 내에서만 실행이 가능하다.(프로세스 내에서 하나의 연속적인 실행 흐름)

**9-4 멀티스레드란 어떤 경우 사용하는 가?**

여러 개의 스레드가 존재하며 여러 작업을 실행하는 것 = **multithreading(멀티스레딩)**

**9-5 스레드 클래스를 작성하는 방법을 설명하시오.**

**1)스레드를 생성할 경우:**

**Thread 객체이름 = new Thread( );** 의 형식으로 사용한다.

**객체이름.start( );** → 스레드 객체를 실행하라는 명령

**2)서브스레드 클래스에서 Thread 클래스를 상속받을 경우:**

Thread 클래스의 “**run ( )** “ 메소드를 상속받는데, 이때 run 메소드 안에서 스레드에 수행할 문장들을 재정의하려 스레드에서 필요한 작업을 수행할 수 있다.

**스레드클래스이름 객체이름 = new 스레드클래스이름( );**

**객체이름.start( );**

**9-6 스레드를 동작하는 방법을 설명하시오.**

**스레드객체이름.start( );**

**9-7 Calendar 클래스를 사용해 날짜와 시간과 초를 표시하는 5개의 스레드를 만들어 서로 다른 초 값을 출력하도록 멀티스레드를 사용해 프로그램을 작성하시오.**