**Java 공부 정리**

전채현

**java**

- class (c언어의 구조체 struct) : 서로 관련있는 데이터들을 하나의 단위로 묶어 편하게 관리

하나의 파일에 하나의 클래스

클래스는 결국 하나의 타입이다. ex) int, string, float,

사용자가 새롭게 정의하기 때문에 "사용자 정의 타입"

// new가 만들어내는 것을 객체라고 한다.

// new 명령어로 객체를 생성하는 것이다.

===========================================

**\*\*\*int, double형 타입과 클래스의 차이점**

int 변수 안에 정수값이 저장

Person1 first = new Person1();

first.name = "john";

first.number = "02323";

클래스의 변수가 만들어지지만 그안에 사람과 번호가 저장되지 않는다.

이름과 번호를 저장할 Person1 객체는 new명령으로 따로 만들어야 하고

변수 first에는 그 객체의 주소(참조)를 저장할 수 있다 // C에서 포인터 개념

참조변수 => 프리미티브 타입 int, double, float형의 변수가 아닌 모든 변수는 참조변수

first안에는 객체의 주소를 저장하는 것이다.

실제 데이터가 저장될 객체는 new 명령으로 따로 만들어야 한다.

Person1 members [] = new Person1[100];

members[0] = first; // members[0]도 first와 같은 타입의 변수이다. 그래서 first의 주소를 가져온다.

// 하나의 객체를 first, second, members가 참조하는 것이다.

배열은 프리미티브타입이 아니다. 프리미티브타입의 배열이라도 배열의 각 원소가 프리미티브타입인 것이지

배열 자체가 프리미티브타입은 아니다. 따라서 배열의 이름 members는 참조 변수이다.

numbers = new int[8];을 수행하는 순간 8칸을 가진 배열이 실제로 만들어지고 그 주소가 참조변수 numbers에 저장된다.

===========================================

**class** : 설계도, 객체를 찍어내기 위한 틀

-> 객체지향 프로그래밍에서 객체를 생성하기 위해 변수와 메서드를 정의해둔 틀입니다.

new 명령어로 객체를 만들어 낸다.

객체는 이름이 없기 때문에 주소를 보관하는 용도로 참조변수( Person a = new Person(); )가 필요하다.

Person자체에 이름과 번호를 저장할 수 없기 때문에 new 명렁어로 객체를 생성해서 객체안에

a.name = "David"; 이렇게 저장해 준다.

===========================================

**메서드와 생성자**

클래스는 서로 관련있는 데이터들을 하나의 단위로 묶어두기 위한 것이다.

하지만 이것이 전부는 아니다.

서로 관련있는 데이터들 뿐 아니라, 그 데이터와 관련이 깊은 메서드도 함께 묶어 둘 수 있다.

why? 서로 관련있는 데이터끼리 묶어두냐??

이렇게 함으로써 코드의 응집도를 높이고 결합도를 낮출 수 있다.

클래스는 관련있는 데이터들과 데이터들과 관련있는 함수까지 같이 묶어주는 역할

============================================

**클래스, 참조변수, 객체**

Term2 t = new Term2();

t => 참조변수

t가 가리키는 객체가 Term2()

클래스는 여전히 설계도일 뿐이다. 실체가 아니다.

우리가 실행하는 것은 클래스가 아니라 각각의 객체 안에 있는 메서드들이다.

각각의 객체 안에 calcTerm과 printTerm 메서드가 존재한다.

**객체란???**

객체지향 프로그래밍에서 객체란 "데이터" + "메서드"이다. 데이터는 객체의 "정적 속성"을 표현하고

메서드는 객체의 "기능(동적 속성)을 표현한다."

**객체지향 프로그래밍이 뭔가요?**

문제를 여러 객체 단위로 나누어서 처리하는 프로그래밍 방식입니다.

부품 객체를 만들고 이것들을 하나씩 조립하면서 프로그래밍하는 방법입니다.

<예>

자전거는 모양, 무게, 크기 등의 정적 속성(데이터)과 달린다, 정지한다, 뒤로 간다 등의 기능을 가진다.

이걸 프로그래밍으로 전환하면

하나의 항은 계수와 차수라는 정적 속성과 x의 값을 주면 자신의 값을 계산해준다, 화면에 출력해 준다.등의

기능을 가진다.

===========================================

**생성자**

=> 클래스 안에 그 클래스와 동일한 이름을 가지며 return 타입이 없는 특별한 메서드를 둘 수 있다.

이것을 생성자라고 부른다.

\*\*\*\*\*\*생성자는 new 명령으로 객체가 생성될 때 자동으로 실행된다.

목적은 객체의 데이터 필드의 값을 초기화하는 것이다.

\*\*\* why??생성자가 없는 Term의 경우 다음과 같이 객체를 생성한 후 따로 데이터 멤버의 값을 초기화하였다

Term2 t = new Term2();

t.coef = 3;

t.expo = 2;

생성자가 있는 경우 다음과 같이 객체의 생성과 초기화를 한번에 할 수 있다.

Term3 t2 = new Term3(3,2); <= \*\*\*\*코드의 가독성이 좋아진다.

생성자가 반드시 매개변수를 받아야하는 것은 아니다.

\*\*\*\*생성자는 객체에게 필요한 초기화 작업을 하기에 적절한 장소이다.

===========================================

The method addVertex(int, int) in the type OrthoPolygon is not applicable for the arguments

-> 구문 오류 ?

===========================================

클래스는 여전히 설계도일 뿐이다. 실체가 아니다.

우리가 실행하는 것은 클래스가 아니라 각각의 객체 안에 있는 메서드들이다.

new 명령으로 해당 클래스 타입의 객체를 만든 후, 그 객체에 데이터를 저장하고, 그 객체의 멤버 메서드를 실행하는 것

여기서 하나의 예외가 존재하는데 그것이 static 멤버이다.

static 멤버는 클래스 안에 실제로 존재하며 객체에는 존재하지 않는다.

static 멤버는 class 멤버이고, non-static 멤버는 object 멤버이다.

===========================================

**Questions??**

왜 main 메서드는 반드시 static 이어야 하는가?

->자바는 클래스들의 집합이다. 즉 클래스 말고는 아무것도 없다.

그냥 클래스로 존재하면 어떤 일도 할 수 없다. 어떤 메서드도 실행할 수 없다.

그래서 main은 static이어야 한다. main은 출발점이니까

main이 static이 아니면 자바프로그램은 통째로 허상이다. 아무것도 시작될 수 없다.

왜 static 메서드에서 같은 클래스의 non-static 멤버를 엑서스 할 수 없는가?

->static 멤버는 class 멤버이고, non-static 멤버는 object 멤버이다.

non-static 멤버를 엑서스하려면 객체를 생성해줘야 한다.

객체를 생성하지 않으면 엑서스할 수 없습니다.

다른 클래스에 속한 static 멤버는 어떻게 엑서스하는가?

->public인 경우 import???

static은 클래스에 속한 멤버이지 객체에 속한 멤버가 아니라서 고정된 어떤 값을 그대로 출력한다.

반면 non-static 객체를 참조하는 참조변수의 값에 따라 바뀐다.

static 메서드/필드의 용도는 ?

->

1. main 메서드

2. 상수 혹은 클래스 당 하나만 유지하고 있으면 되는 값(혹은 객체)

ex) Math.PI, System.out

3. 순수하게 기능만으로 정의되는 메서드, 대표적인 예로는 수학함수들

ex) Math.abs(k), Math.sqrt(n), Math.min(a,b)

static 메서드에서는 this 키워드를 사용할 수 없다.

=>this는 호출 당시 실행 중인 객체를 가리키는 레퍼런스이다.

따라서 객체가 생성되지 않은 상황에서도 클래스 이름을 이용하여 호출이 가능한 static 메서드는 this를 사용할 수 없다.

===========================================

**접근 제어** : public, private, default, protected

public : 클래스 외부에서 접근이 가능하다.

private : 클래스 내부에서만 접근이 가능하다.

왜 private를 사용하는가?

L> 데이터 캡슐화, 데이터의 무결성, 카피본을 가져가 원본은 손 못대

viewer보는 느낌??

모든 데이터 멤버를 private으로 만들고 필요한 경우에 public한 get/set 메서드를 제공한다.

이렇게 하면 객체가 제공해주는 메서드를 통하지 않고 서는 객체 내부의 데이터에 접근할 수 없다.

이것을 data encapsulation 혹은 information hiding이라고 부른다.

데이터를 읽을 수는 있지만 변경을 할 수 없다.

default : 동일 패키지에 있는 다른 클래스에서 접근 가능하다.

데이터를 getter, setter 함수로 만들어서 읽어오고 내보내고 가능하다.

getter가져오다, setter내보내다.

모든 데이터 멤버를 private로 바꾸어 보고, public 멤버를 최소화해보자

protected : 동일 패키지의 다른 클래스와 다른 패키지의 하위클래스에서도 접근 가능하다.

외부 클래스 하위 클래스 동일 패키지 내부 클래스

public O O O O

protected X O O O

default X X O O

private X X X O

===========================================

**상위 클래스로의 묵시적 형 변환(업 캐스팅)**

- 상위 클래스 형으로 변수를 선언하고 하위 클래스 인스턴스를 생성할 수 있다

ex) Customer cs = new VIPCustomer();

여기서 cs가 불러올 수 있는 것은 Customer이다

- 하위 클래스는 상위 클래스의 타입을 내포하고 있으므로 상위 클래스로 묵시적 형 변환이 가능하다.

그래서 다운캐스팅할 때 형변환을 해줘야 한다.

===========================================

**상속과 Generic Programming**

상속(Inheritance)

public class Notebook1 extends Computer1{}

extends Computer1를 하면 Computer1에 있는 모든 함수와 클래스를 가져올 수 있다.

데이터 멤버와 메서드를 가져올 수 있다.

그래서 노트북에 extends를 붙여 컴퓨터를 확장한다.

노트북이 컴퓨터를 상속받게 된다.

즉, 상속은 내가 이전에 만들어둔 클래스와 밀접한 관련이 있는 클래스가 있을 경우 (is-a관계)

A Notebook is a Computer.

Computer is the superclass of Notebook.

Notebook is a subclass of Computer.

컴퓨터를 부모 클래스, 노트북을 자식 클래스

===========================================

**\*\*자바에서 모든 클래스들은 적어도 생성자를 가지고 있어야 한다.**

상속과 생성자

- 생성자가 없을 경우 자동으로 no-parameter(매개변수를 받지 않는 public Term(){} ) 생성자가 만들어진다.

생성자가 하나라도 있을 경우 자동으로 만들어지지 않는다.

- 모든 서브 클래스의 생성자는 먼저 수퍼클래스의 생성자를 호출한다.

1. super(man, proc, ram, disk, ...)를 통해 명시적으로 호출해 주거나.

2. 그렇지 않을 경우에는 자동으로 no-parameter 생성자가 호출된다.

- 흔한 오류

수퍼클래스에 no-parameter 생성자가 없는데,

서브클래스의 생성자에서 super()를 호출을 안해준 경우

===========================================

**Method Overriding**

부모 클래스로 부터 물려 받은 것 중에 내가 원하는게 없을 경우

부모 클래스에 있는 매서드를 가지고와서 덮어 쓰는 것을 말한다.

부모 자식 간에도 부모클래스에 private으로 데이터가 설정되어 있으면

자식 클래스안에서도 그 멤버를 엑서스할 수 없다.

이때 protected는 자식 클래스에서는 멤버를 엑서스할 수 있다.

\*\*\* private으로 데이터가 설정되어 있으면

super.toString으로 문자열을 가지고 올 수 있다. 그러면 private 설정에서도

부모의 데이터를 가지고 올 수 있다.

===========================================

**\*\*\*\*\*\*다형성 : Polymorphism, 가장 중요한 개념이다. 자바의 심장이다.**

다형성을 제대로 활용해야 제대로된 자바프로그래밍이다.

- \*\*\*\*\*\*\*\*\*수퍼클래스 타입의 참조변수가 서브클래스 타입의 객체를 참조할 수 있다.

즉 뭔말이냐면

참조변수를 선언할 때

Notebook1 test = new Notebook1("dell", "i5", 4, 1000, 3.2, 15.6, 1.2);

이렇게 해야 하는데

Computer1 test1 = new Notebook1("dell", "i5", 4, 1000, 3.2);

이렇게 선언해도 문법적으로 틀린 것은 아니다.

수퍼클래스 타입의 참조변수가 서브클래스 타입의 객체를 참조할 수 있다는 것이다.

다만 역은 성립이 안된다.

매서드 오버라이딩이 된 경우 덮어 쓴 경우에 다이나믹 바인딩이 일어난다.

System.out.println(test.toString()); // static binding vs. dynamic binding

이때 test.toString은 Computer1에 있는 toString을 참조하는지(static binding)

Notebook1에 있는 toString을 참조하는지(dynamic binding)????

자바에서는 Notebook1에 있는 toString을 참조하는 것으로 약속하고 있다.

test의 타입은 컴퓨터이지만 참조하고 있는것은 노트북이기 때문에 노트북에 있는 toString을 가져온다.

===========================================

String str = "234.134.414.314.134";

String [] tokens = str.split("\\."); // 정규 표현식 , 이스케이프문자를 사용해야 한다.

String [] tokens = str.split("\\.");

String [] tokens = str.split("\\.");

String [] tokens = str.split("\\.");

String [] tokens = str.split("\\.");

String [] tokens = str.split("\\");

Sr

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

System.out.println(tokens[i]);

}

for(init i =0; i < tokens.length; i++){

System.out.println(token[i]);

}

문자열을 특정 단위로 쪼개는 것.

this is a test string 을 쪼깨고 싶을 때는 String [] tokens = str.split("[]+"); 이렇게 사용한다.

===========================================

배열 재할당 : array reallrocate

private void reallocate() {

Event [] tmp = new Event [capacity \*2];

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

tmp[i] = events[i];

events = tmp;

capacity \*= 2;

}

}

===========================================

**class Object**

-클래스 Object는 java에서 모든 클래스의 superclass이다.

-class Object의 멤버 메서드

1. boolean equals (Object obj)

1. boolean equals (Object obj)

2. int hashCode()

3. String toString()

4. Class<?> getClass()

L>자바의 모든 클래스들은 이 4개의 메서드들을 가지고 있는 것이다.

-java의 모든 클래스는 내가 만들어 주지 않아도 이미 equals와 toString 메서드를 가지고 있다.

- 다만 내 의도대로 작동하지 않을 것이다.

toString()

java의 모든 클래스는 내가 만들어 주지 않아도 이미 equals와 toString 메서드를 가지고 있다.

만약 toString 메서드를 따로 만들어주지 않은 클래스의 객체에 대해서 toString()메서드를 호출하면 다음과 같은

String이 반환 된다. -> section3.Test@15db9742

**\*\*\*\*\*equals(Object) vs. ==**

==연산자와 String클래스의 equals()메소드의 가장 큰 차이점은

== 연산자는 비교하고자 하는 두개의 대상의 주소값을 비교하는데 반해

String클래스의 equals 메소드는 비교하고자 하는 두개의 대상의 값 자체를 비교합니다.

== 은 오직 프리미티브타입 데이터만 동일성을 검사할 수 있다.

- Object 클래스의 equals 메서드의 매개변수는 Object 타입이다.

public boolean equals(Object other){}

- 매개변수로 제공된 객체와 자기 자신의 동일성을 검사한다.

- 이 메서드를 의도대로 사용하려면 override해야 한다.

===========================================

**Class**

- 모든 클래스는 하나의 Class 객체를 가진다.

- 이 객체는 각각의 클래스에 대해서 유일하다.

- 메서드 getClass()는 Object 클래스가 제공하는 메서드이며,

이 유일한 Class 객체 반환한다.

- if (this.getClass() == obj.getClass())가 true라면

우리는 비교 대상인 두 객체(this 와 obj)가 동일한 클래스의 객체임을 알 수 있다.

===========================================

**\*\*\* Wrapper class**

- java에서 프리미티브 타입 데이터와 non-primitive type 데이터는 근본적으로 다르게 처리된다.

- Object 타입의 배열에는 \*\*\*\*다형성의 원리에 의해서 모든 종류의 객체를 저장할 수 있다.

하지만, int, double, char 등의 primitive type 데이터는 저장할 수 없다. 객체가 아니므로..

- primitive type 데이터를 객체로 만들어야 할 경우가 있다.

이럴때 Integer, Double, Character 등의 wrapper class를 이용한다.

wrapper class

- 기본 타입의 데이터를 하나의 객체로 포장해주는 클래스

L> Integer, Double, Character Boolean 등

ex) int a = 10;

Integer age = new Integer(a);

Integer time = new Integer(a);

Integer time = new Integer(a);

Integer time = new Integer(a);

Integer time = new Integer(a);

Integer time = new Integer(a);

Integer time = new Integer(a);

int b = age.intValue();

int b = age.intValue();

- 데이터 타입간의 변환 기능을 제공

ex) String str = "1234";

int d = Integer.parseInt(str);

int d = Integer.parseInt(str);

int d = Integer.parseInt(str);

int d = Integer.parseInt(str);

int d = Integer.parseInt(str);

int d = Integer.parseInt(str);

int d = Integer.parseInt(str);

int d = Integer.parseInt(str);

int d = Integer.parseInt(str);

Object [] array = new Object[100];

int a = 10;

Integer age = new Integer(a); // wrapping

Object [] array = new Object[100];

int a = 10;

Integer age = new Integer(a);

array[0] = a;

int b = age.intValue(); // unwrapping

int b = age.intValue();

int b = age.intValue();

int b = age.intValue();

int b = age.intValue();

System.out.println(b);

===========================================

**추상클래스**

- 추상(abstract)메서드는 선언만 있고 구현이 없는 메서드

- 상속 받는 곳에서 구현해야 한다.!!!!

- 추상 메서드를 포함한 클래스는 추상클래스

- 추상 메서드와 추상 클래스는 키워드 abstract로 표시

- 추상 클래스는 객체를 만들 수 없다. 따라서 서브 클래스를 만드는 용도로 사용됨

추상클래스를 확장(extends)을 해서 사용

즉. 추상클래스는 다른 클래스가 extends하는 용도로만 사용가능하다.

즉, 동일한 부모를 가지는 클래스를 묶는 개념으로 상속을 받아서 기능을 확장시키는 것이 목적이다.

\* 클래스의 인스턴스(new명령어로 만든 것)를 객체라고 부른다.ok

===========================================

**인터페이스** -> 추상클래스의 극단적인 표현 interface를 상속받을 경우 implements 키워드를 사용합니다.

- 오직 추상 메서드와 상수만을 멤버로 가질 수 있으며 그 외의 다른 어떠한 요소도 허용하지 않는다

- 인터페이스는 구현된 것은 아무것도 없고 밑그림만 그려져 있는 기본 설계도라고 할 수 있다.

- 추상 메서드만을 가진 순수한 추상클래스

- static final 데이터 멤버를 가질 수 있음

=> static final사용목적은 인터페이스 변수는 스스로 초기화 될 권한이 없기 때문이다.

===========================================

**java.lang.NullPointerException**

\*\*\*객체를 정의한 뒤, 생성자를 이용해 생성(인스턴스 생성)하지 않고 그 객체를 사용 하려고 했기 때문에 에러 발생

Test a;

a.test();

=> Test a = new Test();

a.test();

===========================================

**Interface vs. Abstract Class**

추상 메서드로만 구성된 추상 클래스는 인터페이스와 완전히 동일한가??

- 차이점이 있다. 바로 인터페이스는 다중 상속이 가능하다는 것이다.

다중상속(Multiple inheritance)

- java에서는 다중 상속을 허용하지 않는다.

- 하지만 하나의 클래스가 여러 개의 interface를 implement하는 것은 가능하다.

===========================================

**자바에서 배열**

자바에서 배열이나 리스트를 정렬하려고 한다면 \*\*\*\*\*java.util.Arrays 클래스의 sort() 메서드를 사용하시면

따로 정렬 로직을 짜지 않아도 한번의 메소드 호출로 간편하게 배열이나 리스트를 정렬할 수 있습니다.

Arrays클래스는 배열의 복사, 항목 정렬, 검색과 같은 배열을 조작 기능을 가지고 있습니다.

내림차순 정렬

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

배열 일부분만 정렬

Arrays.sort(arr, 0, 4); // 0,1,2,3 요소만 정렬

===========================================

**Generic Programming** -> 데이터 타입이 픽스되어 있지 않은 것

L> 다형성을 이용한 것이다.

제너릭 프로그래밍은 데이터 형식에 의존하지 않고, 하나의 값이 여러 다른 데이터 타입들을 가질 수 있는

기술에 중점을 두어 재사용성을 높일 수 있는 프로그래밍 방식이다.

재사용성이 높은 프로그래밍 방식입니다.

제너릭 프로그래밍은 데이터 형식에 의존하지 않고, 하나의 값이 여러 다른 데이터 타입들을 가질 수 있는

기술에 중점을 두어 재사용성을 높일 수 있는 프로그래밍 방식을 말합니다.

Generic한 변수/ 자료구조

Event ev;

Event [] events = new Event [capacity];

Object obj;

Generic한 알고리즘(Method) ->> Arrays.sort(shapes, 0, n);

Generic 클래스 ->> Generics

클래스의 객체를 생성할 때 타입을 생성하는 것을 Generics라고 한다.

Generics

public class Box<T>{ <<- T라는 가상의 타입에 의해서 parameterized된 클래스, 두개이상의 타입을 가질 수 있다.

private T t;

public void set(T t) {this.t = t;}

public T get() {return t;}

Box<Integer>integerBox = new Box<Integer>(); <<- 객체를 생성하는 시점에 가상의 타입 T를 실제하는 타입으로 지정

Box <Integer> integerBox = new Box <Integer> (); <<- 객체를 생성하는 시점에 가상의 타입 T를 실제하는 타입으로 지정한다

Box <Integer> integerBox = new Box <Integer> ();

===========================================

**Generic한 리스트 클래스를 만들어 보자!!**

리스트는 여러개의 데이터를 저장하고, 임의의 위치에 새로운 데이터를 추가하고, 삭제가 가능하고

임의의 위치의 데이터를 읽을 수 있고, 용량에 제한이 없고,,

MyArrayList<String> mylist = new MyArrayList<String>();

mylist.add("wjs"); <<- 맨끝에 추가

mylist.set(2,"aff") <<- 덮어쓰기

mylist.remove(1) <<- 삭제해줘

String dwarf = mylist.get(2); <<- 2번 인덱스값을 가져와줘

int index = mylist.indexOf("sneezy"); <<- 배열에 sneezy가 있으면 번호를 호출해줘 ex) 2

int index2 = mylist.indexOf("jumpy"); <<- 배열에 jumpy가 있으면 번호 호출. 없으면 -1을 리턴해줘!

이렇게 만들거야!

===========================================

new가 객체를 생성할 때 실제로 존재하지 않는 타입명이 들어가면 안된다.

theData = (T[]) new Object [INIT\_CAPACITY]; <<- 이런식으로 [ ] 안에 T가 들어가면 자바는 실행이 안됨

theData = (T [] )new Object [INIT\_CAPACITY]; <<- 이런식으로 [] 안에 T가 들어가면 자바는 실행이 안됨

즉, T타입으로 객체를 만드는 것은 불가능하다. T타입의 배열도 생성 불가

그런데 T를 type parameter로 가지는 배열을 만드는 것은 불가능하다. 왜?? 금지시켰을까

\*\*\*\*Node<T> newNode = new Node<T>(item); // T : type parameter << 이건 가능

T를 type parameter로 가지는 다른 클래스의 객체를 생성하는 것은 가능하다.

Node<T> [ ] arr = new Node<T> [100]; << 불가능

다만 T타입으로 변환해주면 사용이 가능하다.

==-> Method, Wildcard 다양함

===========================================

for(Event ev : events) // enhanced for loop <<- events가 ArrayList인데 ArrayList 안에 있는 모든 Event ev에 대해 불러와!

L> 배열이 꽉 차 있을 경우에만 사용하지 다 안차있으면 사용하면 별로다

L> ArrayList를 사용할 경우 사용하면 유용하다.

System.out.println(ev.toString()); // dynamic binding

===========================================

**Vetor 와 ArrayList**

java API java.util 은 Vector와 ArrayList라는 두 가지 유사한 기능의 클래스를 제공한다.

Vector보다 ArrayList가 좀 더 효율적으로 사용됨

ArrayList와 달리 Vector 클래스는 synchronized됨, 즉 다수의 thread가 충돌없이 Vector 객체를 엑서스 할 수 있다.

===========================================

**연결리스트(Linked List)**

리스트 - 기본적인 연산(insert, remove, search..), 리스트를 구현하는 대표적인 방법은 배열과 연결리스트가 있다.

배열은 단점

1. 크기가 고정되어 있어서 reallocation이 필요한다.

2. 리스트의 중간에 원소를 삽입하거나 삭제할 경우 다수의 데이터를 옮겨야 한다.

배열의 장점은 어떤 위치에 접근을 하든 접근시간이 조금의 차이는 있겠지만 거의 비슷하다.

만약 배열의 10번째 위치를 가려면 바로 갈 수 있다.

배열의 크기가 동일하기 때문에 주소를 바로 계산하여 접근이 가능하다.

하지만 연결리스트는 첫번째 값부터 하나씩 순서대로 읽어야 하기 때문에 시간이 오래걸린다.

연결리스트

- 배열의 단점을 보완했지만,단점도 있지!

- 다른 데이터의 이동없이 중간에 삽입이나 삭제가 가능하다.

- 길의 제한이 없다.

- 하지만, 랜덤 엑서스가 불가능하다.

연결리스트 내 데이터와 내 다음 데이터의 주소를 같이 저장한다.

===========================================

**노드**

- 각각의 노드는 "데이터 필드"와 하나 혹은 그 이상의 "링크 필드"로 구성된다

- 링크 필드는 다음 노드를 참조한다

- 첫 번째 노드의 주소는 따로 저장해야한다.

\*\*\* 연결 리스트를 다루는 프로그램에서 "가장 주의할 점"은 내가 작성한 코드가 일반적인 경우만이

아니라 특수한 혹은 극단적인 경우에도 문제없이 작동하는지 철저히 확인하는 것이다. 이 경우에는

기존의 연결 리스트의 크기가 0인 경우, 즉, head가 null인 경우에도 문제가 없는지 확인해야 한다.

장소 = 값;

public void addAfter(Node<T> before, T item) {

Node <T> temp = new Node<T>(item);

temp.next = before.next;

before.next = temp;

size++;

}

\*\*\*\*단순 연결리스트에 새로운 노드를 삽인할 때 삽입할 위치의 바로 앞 노드의 주소가 필요하다.

즉, 어떤 노드의 뒤에 삽입하느 것은 간단하지만, 반대로 어떤 노드의 앞에 삽입하는 것은 간단하지 않다.

(addAfter는 간단하지만 addbefore는 간단하지 않다)

public T removeAfter(Node<T> before) {

if(before.next == null)

return null;

T temp = before.next.data;

before.next = before.next.next;

return temp;

}

\*\*\* 단순연결리스트에 어떤 노드를 삭제할 때는 삭제할 노드의 바로 앞 노드의 주소가 필요하다.

삭제할 노드의 주소만으로는 삭제할 수 없다.

===========================================

**순회**

연결리스트의 노드들을 처음부터 순서대로 방문하는 것을 순회(traverse)한다고 말한다.

indexOf 메서드는 입력된 데이터 item과 동일한 데이터를 저장한 노드를 찾아서 그 노드번호(index)를 반환한다.

그것을 위해서 연결리스트를 순회한다.

연결리스트를 만들때는 예외를 항상 생각해야 한다. 예외적인 상황 극단적인 상황

===========================================

**객체지향 프로그래밍**

- Information Hiding

- Data Encapsulation

- Abstract Data Type

객체지향 프로그래밍에서 인터페이스(Interface)와 구현(implementation)의 분리

- 연결리스트는 리스트라는 추상적인 데이터 타입을 구현하는 한가지 방법일 뿐이다.

- 사용자는 리스트에 데이터 삽입, 삭제, 검색할 수 있으면 되는거다.

그것의 구현에 대해서 세부적으로 알 필요가 없다.

(사용자가 필요로 하는 이런 기능들을 public method들로 제공한다.

이 public method들은 가능한 한 내부 구현과 독립적이어야 한다)

- 인터페이스와 구현을 분리하면 코드의 모듈성이 증가하고, 코드의 유지/보수, 코드의 재사용이 용이해진다.

===========================================

inner class 클래스 안에 다른 클래스를 정의하는 것

이때 private static 으로 만들어 준다. 이렇게 해서 외부에서 inner class에 접근하지 못한다.

private static class 와 private class의 차이점

private static class는 큰 클래스 내부에서만 사용가능 밖에 있는거 엑서스할 수 없다. class멤버이고

private class는 생성자에서 큰 클래스의 non-static 멤버를 엑서스할 수 없다. object멤버이기 때문이다.

**연결리스트의 순회 : Iterator의 필요성**

Iterator

- 노드의 주소를 직접 사용자에게 제공하는 대신 그것을 private 멤버로 wrapping하고 있으면서

사용자가 필요로 하는 일을 (노드에 저장된 데이터를 엑세스하거나 한 칸 전진하는 일..) 해주는

public method를 가진 Iterator객체를 제공한다.

Iterator<E> 인터페이스

-hasNext(), next(), remove()를 제공한다.

연결리스트에서의 Iterator

- Iterator는 개념적으로는 연결리스트의 노드와 노드 사이를 가리킨다.

- 초기상태의 Iterator는 첫 번째 노드의 앞 위치를 가리킨다.

- next()메서드는 한 칸 전진하면서 방금 지나친 노드의 데이터를 반환한다.

- hasNext() 메서드는 다음 노드가 존재하면 true, 그렇지 않으면 false를 반환한다.

- remove() 메서드는 가장 최근에 next()메서드로 반환한 노드를 삭제한다.

\*\*\* 인터페이스와 다형성이 결합된 기능이 iterator

===========================================

**이중 연결리스트**

단방향 연결리스트의 한계

- 단방향의 순회만이 가능하다

- 어떤 노드의 앞에 새로운 노드를 삽입하기 어렵다

- 삭제의 경우에 항상 삭제할 노드의 앞노드가 필요하다

이중연결리스트

- 각각의 노드가 다음노드와 이전노드의 주소를 가지는 연결 리스트

- 양방향의 순회가 가능하다

- LinkedList 클래스는 첫 번째 노드와 마지막 노드를 참조하는 head와 tail 필드를 가진다.

노드삽입 \*\*\*\* 순서 중요 안그러면 앞의 노드주소를 잃어버릴 수 있기 때문이다.

Node<E> newNode = new Node<E>("Sharon");

1. newNode.next = aNode;

2. newNode.prev = aNode.prev;

3. aNode.prev.next = newNode;

4. aNode.prev = newNode;

노드삭제

1. theNode.prev.next = theNode.next;

2. theNode.next.prev = theNode.prev;

===========================================

**Iterartor는 단방향 리스트에 알맞다. 이중연결리스트에는 살짝 안어울린다.**

ListIterator <E> 인터페이스

1. Iterator의 한계

- 단방향으로만 순회할 수 있다.

- remove()메서드는 지원하지만 add()메서드를 지원하지 않는다.

- 항상 리스트의 처음에서 시작한다

2. ListIterator는 Iterator를 확장한다.

3. Iterator처럼 ListIterator 역시 개념적으로는 노드와 노드 사이를 가리킨다.

4. ListIterator의 위치는 0에서 size까지의 인덱스로 표현한다.

ListIterator 인터페이스

void remove() -> history dependent 앞에서 왔는지 뒤에서 왔는지가 중요하다.

void set() - > 덮어쓰는 것. 여기도 앞에서 왔는지 뒤에서 왔는지가 중요하다.

void add() -> 최근에 삽입된 add 뒤에 삽입

\*\*\*\*\* Iterator 커서 느낌!

===========================================

**listiterator() 메서드**

- LIST<E> 인터페이스는 iterator()메서드와 함께 ListIterator를 만들어 주는 listIterator()메서드를 요구한다.

원형 리스트

- 마지막 노드의 다음 노드가 첫번째 노드가 되고 첫 노드의 이전 노드가 마지막 노드가 됨

===========================================

**foreach문 ! for(String key : keys){}**

-> 순서가 없을 때 사용한다. 즉, 인덱스가 없을 때 사용한다.

foreach문은 따로 반복회수를 명시적으로 주는 것이 불가능하고,

1스탭씩 순차적으로 반복될때만 사용가능하다는 제약이 있다.

===========================================

**무조건 StringBuffer를 사용하는 것이 좋을까?**

그건 상황에 따라 다르다. StringBuffer 자료형은 String 자료형보다 무거운 편에 속한다.

new StringBuffer() 로 객체를 생성하는 것은 일반 String을 사용하는 것보다 메모리 사용량도 많고 속도도 느리다.

따라서 문자열 추가나 변경등의 작업이 많을 경우에는 StringBuffer를,

문자열 변경 작업이 거의 없는 경우에는 그냥 String을 사용하는 것이 유리하다.

===========================================

Map<String,String[]> map = request.getParameterMap();

Set<String> keys = map.keySet(); // key값만 모두 뽑아서 Set으로 반환한다.

out.println("<ol>");

for(String key : keys){

out.println("<li>" + key);

// get(key) : key로 map에서 value를 가져옴

String [] values = map.get(key);

for(String value : values){

out.println(value + " ");

}

out.println("</li>");

}

out.println("<ol>");

Enumeration <String> em = request.getParameterNames();

while(em.hasMoreElements()){

String name = em.nextElement();

String [] values = request.getParameterValues(name);

out.println(name);

for(String value : values){

out.println(value);

}

}

===========================================

**자료형이 필요한 이유** : 변수를 선언할 때는 변수가 사용할 메모리 크기와 타입을 구분하기 위해 자료형을 사용한다.

지역변수 자료형 추론 : 변수에 대입되는 값을 보고 컴파일러가 추론하는 것을 말한다

var num = 10; -> int num = 10; var dNum = 10.0; -> double dNum = 10.0; << jre - 10부터 지원

리터럴 : 프로그램에서 사용하는 모든 숫자, 값, 논리값을 말한다(100, A, 3.14)

리터럴에 해당되는 값은 특정 메모리 공간인 상수풀에 있다. 필요한 경우 상수 풀에서 가져와서 사용한다.

명시적 형변환 과 묵시적 형변환

명시적 형변환 : double d = 3.14;

int ad = (int)d;

이렇게 하면 3이 출력된다.

묵시적 형변환 : double ac = 10; 작은수에서 큰수로 가는 것은 자연스럽게 이러난다.

===========================================

**Array(배열) : 같은 형의 데이터 타입을 메모리에 저장하는 선형적 자료구조이다.**

논리적 구조와 물리적 구조가 동일하다. 정해진 길이가 있는거죠~ 또한 인덱스 연산도 가능해요!

배열과 LinkedList의 차이는 원하는 원소를 찾는데 시간소요가 차이가 생깁니다. 링크드리스트는 처음부터 읽어와야해서 오래걸림

다만 배열을 삽입 삭제시 데이터의 이동이 많아진다.

Stack -> LIFO형식, push()와 pop() 바둑게임에서 사용// peek() -> 스택의 맨 위에 있는 원소를 반환 한다. 일종의 get()

queue -> FIFO형식, 선착순 온 순서대로 나가는 것, 대기열

Tree -> Binary Search Tree 검색 !! Binary Search Tree -> 중복허용안함, 이진탐색트리는 왼쪽에 작은 값 오른쪽에 큰값

트리로 만들면 절반으로 줄어들기 때문에 검색의 속도가 빨라진다. 복잡도가 logn

Binary Search Tree-> 유일한 키 값을 가지고, 루트 노드의 키 값이 기준이 된다.

hashing -> 검색을 위한 것. 산술 연산을 이용한 검색 방식,

===========================================

**메서드 오버라이딩과 메서드 오버로딩의 차이점**

개발자의 실수를 방지하기 위해 @Override(annotation) 쓰는 것을

권장 상속의 상속의 경우 이걸쓰는 것이 효율적

메서드 오버로딩

- 동일 이름의 메서드를 매개변수만 다르게 하여 여러 개 정의할 수 있는 기능

- 메서드의 이름이 같아야 한다.

- 매개변수 개수가 달라야 한다.

- 매개변수 개수가 같은 경우에는 데이터 타입이 달라야 한다

- 반환형은 같거나 달라도 된다.

메서드 오버라이딩

- 하위 클래스에서 상위 클래스의 메서드를 재정의할 수 있는 기능

- 오버라이드하고자 하는 메서드가 상위클래스에 존재해야 한다.

- 메서드의 이름이 동일해야 한다.

- 메서드의 매개변수, 타입이 동일해야 한다.

- 메서드의 반환형이 동일해야 한다.

===========================================

if(id.equals("test") && pw.equals("1")){

out.print(id +"님 환영합니다");

}

else if(id.equals("itbank") && pw.equals("it")){

out.print(id +"님 환영합니다");

}

else if(id.equals("root") && pw.equals("qwe@123")){

out.print(id +"님 환영합니다");

}

else if(id.equals("admin") && pw.equals("ad")){

out.print(id +"님 환영합니다");

}

아이디 비교 아이디어

String [] ids = new String[] {"test", "itbank", "root", "admin"};

String [] pws = new String[] {"1", "it", "qwe@123", "ad"};

public String loginPro(String userid, String userpw){

// 매개변수로 전달된 id와 pw를 받아서, 배열에 일치하는 계정정보가 있으면 해당 계정의 id를 반환

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

if(ids[i].equals(userid) && pws[i].equals(userpw)){

return ids[i];

}

}

return null;

}

이런식으로 축소

===========================================

**자바 빈즈 에서 중요한거**

// Java Beans : 웹에서 사용할 데이터를 클래스로 정의하고, 입출력이나 데이터 전송에 사용되는 클래스

// Beans 클래스는 작성 규칙이 있다.

// 1. private 멤버 변수

// 2. public 메서드

// 3. 자동으로 생성되는 생성자나, 혹은 직접 작성한 기본 생성자

// 4. 생성자는 오버로딩을 해서 여러개를 작성할 수 있지만 기본 생성자는 무조건 있어야 한다.

// 5. input태그의 name 이름과 beans의 멤버 변수 이름은 가급적 맞추도록 한다.

//public Member(String id, String pw, String name, int age, String gender) {

//this.id = id;

//this.pw = pw;

//this.name = name;

//}

// 기본생성자가 필수로 필요하다. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

===========================================

자바에서는 객체들의 데이터를 비교하려면 Object 클래스에서

hashCode(), equals()를 오버로딩(재정의)해야 한다.

===========================================

Account[] accs = new Account[]{

new Account("test","1","텍스트"),

new Account("root","qwe@123","관리자"),

new Account("itbank","it","아이티뱅크")

};

===========================================

**객체지향 프로그래밍**

- 객체를 정의

- 객체의 기능을 구현

- 객체 사이의 협력 구현

객체의 속성 - 객체의 특성, 속성, 멤버 변수(학번, 이름, 학년)

객체의 기능 - 객체가 하는 기능들을 메서드로 구현(수강신청, 수업듣기)

패키지는 소스의 묶음 소스코드를 어떤 계층구조로 관리할지가 중요하다.

=============================================

메서드는 함수의 일종입니다. 객체의 기능을제고아기 위해 클래스 내부에 구현되는 함수입니다.

함수는 하나의 기능을 수행하는 일련의 코드를 말합니다.

함수와 스택메모리??

함수가 호출될때 스택메모리를 사용한다. 함수가 호출 되면 메인함수 위에 스택이 쌓인다. 그리고 종료되면 없어진다.

스택이 지역변수가 있는 곳이다.

함수를 왜 쓰나요??

코드를 효율적으로 구현하기 위해서 사용합니다. 필요한 기능별로 구현하면 효율적으로 작업을 할 수 있기 때문입니다.

함수는 하나의 이름을 정하고 함수 이름에 맞게 기능을 하면 된다.

===========================================

\* 클래스의 인스턴스(new명령어로 만든 것)를 객체라고 부른다.ok

클래스의 인스턴스를 객체

클래스의 인스턴스를 객체

클래스형 변수이름() = new 생성자();

변수이름에서 참조변수를 불러서 초기화 해준다.

===========================================

**인스턴스와 힙(heap)메모리**

- 하나의 클래스 코드로 부터 여러 개의 인스턴스를 생성할 수 있다.

- 인스턴스는 힙메모리에 생성된다.

- 각각의 인스턴스는 다른 메모리에 다른 값을 가진다.

- 동적으로 생성된 메모리 할당 가능

- new 명령으로 힙메모리에 생성됨, 가비지 콜랙터가 사용 후 없애준다.

- 힙에 하나의 인스턴스가 생성되면 독립적인 메모리 공간이 할당됨

- 할당된 메모리는 가비지 콜랙터로 사용하지 않는 인스턴스를 수거한다.

===========================================

객체 - 객체 지향 프로그램의 대상이고 생성된 인스턴스를 말한다.

클래스 - 객체를 프로그래밍하기 위해 코드로 만든 상태

인스턴스 - 클래스가 메모리에 생성된 상태

멤버 필드 - 클래스의 속성, 특성을 말하는 것입니다.

멤버 메서드 - 멤버 필드를 이용하여 클래스의 기능을 구현하는 것을 말합니다.

참조 변수 - 메모리에 생성된 인스턴스를 가리키는 변수를 말합니다.

참조 값 - 생성된 인스턴스의 메모리 주소 값을 말합니다. (=> chapter.Student@15db9742)

============================================

생성자가 하나도 없는 경우 자바가 컴파일러 하기 전에 default생성자를 만들어 줍니다. 컴파일러 하기 전에 ok

생성자는 객체가 생성될 때 만들고 싶은 것을 작성한다.

생성자를 두개 만들어도 된다. -> 이걸 생성자 오버로딩이라고 한다.

생성자

- 생성자는 인스턴스를 초기화할 때의 명령어 집합을 말한다.

- 생성자의 이름은 그 클래스의 이름과 같다.

- 생성자는 메서드가 아니다. 상속되지 않으며 리턴값이 없다.

변수의 자료형에 기본 자료형(int, double)과 참조 자료형(String, student...)이 있다

===========================================

**this가 하는 일**

- 자신의 메모리를 가리킨다

public Person(String name, int age){

this.name = name;

this.age = age;

}

-> 위 코드에서 this를 생략하게 되면 name이나 age는 파라미터로 사용되는 name과 age로 인식이 됩니다.

- 생성된 인스턴스 스스로를 가리키는 예약어

- 생성된 메모리의 주소를 가리킨다.

- 생성자에서 다른 생성자를 호출 가능함

public Person(){

this("no",1); <<- 이렇게 하면 앞에 다른 변수를 지정하지 못한다.

}

public Person(String name, int age){

this.name = name;

this.age = age;

}

-> this를 이용하여 다른 생성자를 호출할 때는 그 이전에 어떠한 statement도 사용할 수 없다.

-> 위와 같이 생성자가 여러 개 이고 파라미터만 다른 경우는 constructor overloading이라고 한다.

===========================================

**static 변수는 자바에서 클래스 변수라고 합니다.**

static 변수는 인스턴스가 생성될 때 마다 다른 메모리를 가지는 것이 아니라 프로그램이 메모리에 적재될 때

데이터 영역의 메모리에 생성이 됩니다.

따라서 인스턴스의 생성과 관계없이 클래스 이름으로 직접 참조합니다.

멤버 변수는 다른 말로 인스턴스 변수라고 합니다.

static 메서드는 클래스 메서드라고 합니다.

메서드에 static 키워드를 사용하여 구현합니다. 주로 static 변수를 위한 기능을 제공합니다.

static 메서드도 인스턴스의 생성과 관계없이 클래스 이름으로 직접 메서드해서 호출합니다.

인스턴스의 변수의 경우 곡 인스턴스가 먼저 생성되어야 하므로 static 메서드에서는

생성이 불확실한 인스턴스 변수를 사용할 수 없습니다.

static 변수는 여러 인스턴스가 하나의 메모리 값을 공유할 때 필요하다.

학생이 생성될 때 마다 학번이 증가하는 경우

기준이 되는 static 값이 있어야 한다.

===========================================

Object 클래스가 업캐스팅이다!!!

===========================================

**singleton 패턴 -> 인스턴스 1개만 !!**

싱글턴 패턴은 인스턴스가 오직 1개만 생성되야 하는 경우를 말한다.

객체가 여러개 생성되면 안되는 경우 인스턴스가 여러개 생성되면 안될 때

날짜같은 경우,

싱글턴 패턴은 전 시스템에 단 하나의 인스턴스만이 존재하도록 구현 하는 방식이다.

static 변수를 사용 -> 생성자를 private으로 만듬 -> public 으로 선언된 static 메서드를 제공한다.

객체를 생성하지 않고 메서드를 부르는 방법이 static으로 선언하는 것

-> DTO를 싱글턴 패턴으로 인스턴스해서 사용하는 경우도 있다.

===========================================

char [] ch = new char[26];

char c = 'A';

for(int i = 0; i <ch.length; i++, c++) { << - 증감식 두개 사용할 때

ch[i] = c;

}

for(char chs : ch) {

System.out.println(chs);

}

===========================================

**객체 배열**

-> Login [ ] logins = new Login [ ] {};

Login [] logins = new Login[] {

new Login ("DFA","DF","FA"),

new Login ("DFA","DF","FA")

};

===========================================

**배열이 다 찾을 때 배열의 재할당**

배열 재할당 : array reallrocate

private void reallocate() {

Event [] tmp = new Event [capacity \*2];

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

tmp[i] = events[i];

events = tmp;

capacity \*= 2;

}

}

private void rellocate(){

Event [] tmp = new Event [capacity \*2];

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

tmp[i] = envent[i];

event = tmp;

capacity \*= 2;

}

}

===========================================

iterator 사용할 때

Iterator<String> iter = list.iterator();

===========================================

개발은 책임감이다!! 꼼꼼함

===========================================

class Mystack{

private ArrayList<String> arrayStack = new ArrayList<String>();

public void push(String data){

arrayStack.add(data);

}

public String pop() {

int len = arrayStack.size();

if(len == 0) {

System.out.println("스택이 비었습니다.");

return null;

}

String data = arrayStack.remove(arrayStack.size() -1);

return data;

}

}

이렇게 stack을 만들 수 있다.

===========================================

class MyQueue{

private ArrayList <String> arrayQueue = new ArrayList<String>();

public void enQueue (String data) {

arrayQueue.add(data);

}

public String deQueue() {

int len = arrayQueue.size();

if(len == 0) {

System.out.println("큐가 비었습니다.");

return null;

}

return arrayQueue.remove(0);

}

}

이렇게 queue를 만들 수 있다

===========================================

iterator -> 순서가 없는 것을 사용할 때 좋다

Iterator<String> iter = list.iterator();

while(iter.hasNext()) { // 다음 값이 있는지 확인하고 없으면 널값을 반환해서 반복을 빠져나오는 것

String list2 = iter.next();

}

return;

===========================================

**가상 메서드**

-> 프로그램에서 어떤 객체의 변수나 메서드의 참조는 그 타입에 따라 이루어 진다.

-> 가상 메서드의 경우는 타입과 상관없이 실제 생성된 인스턴스의 메서드가 호출되는 원리이다.

Customer cs = new VIPCustomer();

cs.calcPrice(100);

cs의 타입은 Customer이지만 실제 생성된 인스턴스의 VIPCustomer클래스의 calcPrice()메서드가 호출된다.

다형성 ->

-> 하나의 코드가 여러가지 자료형으로 구현되어 실행되는 것을 말한다.

-> 정보 은닉, 상속과 더불어 객체지향 프로그래밍의 가장 큰 특징이다.

-> 객체지향 프로그래밍의 유연성, 재활용성, 유지보수성에 기본이 되는 특징이다.

public static void main(String[] args) {

// TODO Auto-generated method stub

AnimalTest test = new AnimalTest();

test.moveAnimal(new Human());

test.moveAnimal(new Eagle());

test.moveAnimal(new Tiger());

}

public void moveAnimal(Animal animal) {

animal.move();

}

}

다형성을 구현하기

동일한 moveAnimal()코드에 대해 각기 다른 구현 실행

===========================================

**다운캐스팅**

Animal hAnimal = new human();

If(animal instanceof Human){

Human human = (Human)animal;

}

이런식으로 instanceof 예약어를 사용해서 다운캐스팅할 수 있다.

===========================================

**오류?**

컴파일오류 : 프로그램 코드 작성 중 발생하는 문법적 오류

실행 오류 : 실행 중인 프로그램이 의도 하지 않은 동작을 하거나 프로그램이 중지되는 오류

실행 오류 시 비정상 종료는 서비스 운영에 치명적이다

오류가 발생할 수 있는 경우에 로그를 남겨 추후 이를 분석하여 원인을 찾아야 한다.

자바는 예외처리를 통하여 프로그램의 비정상 종료를 막고 로그를 남길 수 있다.

**오류와 예외 클래스**

시스템 오류 : 가상머신에서 발생함, 프로그래머가 처리할 수 없다. 동적 메모리가 없는 경우, 스택 오버플로우

예외 : 프로그램에서 제어할 수 있는 오류, 읽어 들이려는 파일이 존재하지 않는 경우, 네트워크 연결이 끊어진 경우

===========================================

int [] arr = {1,2,3,4,5}; // 정수형 배열을 만들어

try {

// try 구문으로 for문을 실행할 때 예외가 발생할 가능성이 있는 구문을 넣어주는 역할

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

// i가 arr.length까지 돌 때 배열을 순서대로 출력해줘

System.out.println(arr[i]); // 배열을 순서대로 출력해줘

}

}catch(ArrayIndexOutOfBoundsException e) {

// catch구문에서 이렇게 에러가 발생할 수 있다는 걸 알려주는 거죠

System.out.println(e.toString()); // 어느 부분에서 에러가 발생했는지 알려줘

return; // return은 finally구문이 실행된 다음에 실행이 된다.

}finally {

System.out.println("finally"); // finally는 try가 수행되면 무조건 수행되는 구문이라서 그래 그래서 return이 있으면 end가 출력이 안됨

}

System.out.println("end"); // 프로그램이 죽지 않았다는 것

}

===========================================

**try-with-resources문**

-> finally를 부를 필요가 없는 구문

-> close()를 명시적으로 호출하지 않아도 try{} 블록에서 열린 리소스는 정상적인 경우, 예외 발생한 경우 모두 자동 해체된다.

-> 해당 리소스가 AutoCloseable을 구현해야 해!

===========================================

throw 구문은 예외를 발생시키는 구문

throw new Exception(); 이런 식으로 발생시킨다.

throws 구문은 예외 처리를 미루는 구문이다.

- 예외가 발생한 메서드에서 예외를 처리하지 않고,

이 메서드를 호출한 곳에서 예외 처리를 하겠다는 의미이다.

- main()에서 throws를 사용하면 가상머신에서 처리가 된다.

throws는 메서드 옆에 써야한다.

아래처럼 사용한다.

public Class loadClass(String fileName, String className) throws FileNotFoundException, ClassNotFoundException{

// throws를 통해서 FileNotFoundException, ClassNotFoundException으로 던져줘라는 의미이다.

// public한 클래스를 만들었다 매개변수는 String fileName과 String className을 받는 클래스이다

FileInputStream fis = new FileInputStream(fileName);

// FileInputStream 타입의 fis 변수를 선언하고 FileInputStream의 객체를 생성한다. 이때 fileName을 매개 변수로 받는다.

Class c = Class.forName(className);

// 기본 클래스이다.

return c;

}

===========================================

**java.lang.class 클래스**

-> 클래스의 정보를 얻어오는 클래스입니다.

java.lang.class클래스 안에 forName()메서드는 물리적인 클래스 파일명을 인자로 넣어주면 이에 해당하는 클래스를 반환해 준다.

===========================================

public static void main(String[] args) {

// TODO Auto-generated method stub

ThrowsException ex = new ThrowsException();

try {

ex.loadClass("b.txt", "java.lang.string");

}

catch (FileNotFoundException e) {

System.out.println(e);

} catch (ClassNotFoundException e) {

System.out.println(e);

} catch(Exception e) {//Exception e는 최상위 클래스이기 때문에 catch구문 맨 마지막에 해서 걸려줘야 하는 것이다.

System.out.println(e);

}

// throws로 이쪽으로 미루었기 때문에 여기서 처리해줘야 한다.

System.out.println("end");

}

}

이런식으로 다중 예외 처리를 할 경우에는 Exception을 맨 마지막에 위치시킨다.

그래야지 작은거부터 걸러지고 큰게 마지막에 걸러지는 역할 처음부터 큰게 걸러지면 나머지는 작동을 안하니까 그러는 거다

===========================================

실무에 가면 자바에 없는 예외가 있을 수 있다. 그래서 사용자 정의 예외처리를 만들어야 한다.

public class IDFormatException extends Exception{

public IDFormatException(String message) { // 생성자를 만들어 준다. 컨스트럭쳐 왜 message라는 매개변수를 받기 위해서 이다.

super(message);// Exception에서 제공되는 message가 있다. Exception의 생성자 중에서 하나를 가져다 쓴 것이다.

}

}

즉, 사용자 정의 예외처리를 만들려면 예외처리를 받을 클래스를 하나 만들어서 throw를 통해 미리 만들어 둔 클래스로 보내버리는 것입니다.

이런식으로 Exception을 상속받아서 IDFornatException을 만들어 둔다.

그리고 생성자를 만들어서 String message를 매개변수로 받는다.

public class IDFormatTest {

private String id;

public String getId() {

return id;

}

public void setId(String id) throws IDFormatException {

if(id == null) {

throw new IDFormatException("아이디는 null일 수 없습니다.");

}

else if (id.length() < 8 || id.length() > 20) {

throw new IDFormatException("아이디는 8자 이상 20자 이하로 쓰세요");

}

this.id = id;

}

이런식으로 IDFormatTest 클래스를 만들어 id를 private로 묶었으니까 getter와 setter를 생성해준다. 이때

setter에서 조건식을 준다. 왜 setter에서 주냐 getter는 가져오는 거니까 상관이 없는다.

setter는 받아오는거니까 조건이 있어야 한다.

그래서 if (id == null)이라면 throw new IDFormatException으로 예외처리를 해준다.

else if(id.length() > 8 || id.length() > 20) 일 경우에는

throw new IDFormatException으로 예외처리를 해준다.

public static void main(String[] args) {

// TODO Auto-generated method stub

IDFormatTest idtest = new IDFormatTest();

String id = null;

try {

idtest.setId(id);

} catch (IDFormatException e) {

System.out.println(e);

}

id = "1231";

try {

idtest.setId(id);

} catch (IDFormatException e) {

System.out.println(e);

}

}

}

이렇게 해서 메인 메서드에 IDFormatTest 타입의 idtest 변수를 선언하고 new 명령어로 IDFormatTest객체를 생성한다.

이제 여기서 id의 값을 String id =null; null값으로 준 경우 try -catch문을 사용해서 예외를 IDFormatException으로 처리한다.

===========================================

// 추상메서드는 하위 클래스가 구현해야 하는 것 구현하는 메서드는 공통으로 쓰는 메서드

// 변수의 타입을 따라서 메서드가 따라간다.

===========================================

**java.lang 패키지**

-> 프로그래밍시 import하지 않아도 자동으로 import됨

-> String, Integer, System

Object클래스

모든 클래스의 최상위 클래스이다.

모든 클래스는 Object클래스에서 상속 받음

모든 클래스는 Object클래스의 메서드를 사용할 수 있다(toString, equals, hashCode, getClass)

컴파일러에서 extends Object해준다.

-> final로 선언된 메서드는 재정의 할 수 없다.

hashCode()

-> hash : 정보를 저장, 검색하기 위해 사용하는 자료구조이다.

-> 자료의 특정 값(key 값)에 대해 저장 위치를 반환해주는 해시함수를 사용한다.

-> 힙 메모리에 인스턴스가 저장되는 방식이 hash

-> 논리적으로 동일함을 위해 equals() 메서드를 재정의 하였다면

hashCode()메서드로 재정의 하여 동일한 값이 반환 되도록해야 한다.

-> 그래서 Sources에 보면 hashCode와 equals를 같이 재정의 하는 것이다.

clone()메서드

-> 객체의 원본 복제하는데 사용하는 메서드이다.

-> 원본을 유지해 놓고 복사본을 사용할 때

-> 기본 틀(prototype)을 두고 복잡한 생성과정을 반복하지 않고 복제한다.

-> clone()메서드를 사용하면 객체의 정보가 같은 인스턴스가 생성되는 것이므로

객체 지향 프로그램의 정보은닉, 객체 보호의 관점에서 위배될 수 있다.

-> 객체의 clone()메서드 사용을 허용한다는 의미로 cloneable 인터페이스를 명시해 준다.

class Circle implements Cloneable{

class Circle implements Cloneable

Point point;

private int radius;

public Circle(int x, int y, int radius) {

point = new Point(x,y);

this.radius= radius;

}

public String toString() {

return "원점은 " + point + "이고, 반지름은 " + radius + "입니다";

}

@Override

protected Object clone() throws CloneNotSupportedException {

// TODO Auto-generated method stub

return super.clone();

}

}

===========================================

**String을 선언하는 두 가지 방법**

String str1 = new String("abc");

-> 생성자의 매개변수로 문자열 생성(힙 메모리에 저장)

-> 힙 메모리에 인스턴스로 생성되는 경우

new 연산자는 메모리 상에 새로운 공간을 할당받아서 내용을 만든다

String str2 = "test";

-> 문자열 상수를 가리키는 방식

-> 상수 풀에 있는 주소를 참조하는 방법

String 클래스로 문자열 연결

String sr = new String("java");

System.out.println(System.identityHashCode(sr));

String sr1 = new String("android");

sr = sr.concat(sr1);

// 두개가 붙는 것이 아니라 새로운 한개가 생성되는 것이다.

// 이렇게 하면 새로운 메모리를 생성하는 것이다.

System.out.println(sr);

System.out.println(System.identityHashCode(sr));

sr = sr.concat(sr1);

sr = sr.concat(sr1);

sr = sr.concat(sr1);

이렇게 연결한다

-> 한번 생성된 String값은 불변이다.

-> 두개의 문자열을 연결하면 새로운 인스턴스가 생성된다.

-> 문자열 연결을 계속하면 메모리에 gabage가 생긴다.

그래서 StringBuilder와 StringBuffer를 사용한다.

-> 문자열을 여러번 연결하거나 변경할 때 사용하면 유용하다.

StringBuffer는 멀티 쓰레드 프로그래밍에서 동기화(sybchronization)을 보장

-> 여러개의 쓰레드가 돌아가는 프로그램이 있을 때 쓰레드의 순서를 정해주는 것이 동기화라고 한다.

-> 아무도 못쓰게 암호를 거는 거라고 보면 된다.

단일 쓰레드 프로그램에서는 StringBuilder를 사용하기를 권장

public class StringBuilderTest {

public static void main(String[] args) {

String str1 = new String("java");

System.out.println(System.identityHashCode(str1));

StringBuilder buffer = new StringBuilder(str1);

// StringBuilder타입의 버퍼를 생성해서 그 안에 계속 append를 하는 것이다.

System.out.println(System.identityHashCode(str1));

buffer.append("and ");

buffer.append(" android");

System.out.println(System.identityHashCode(str1));

String str2 = buffer.toString();

System.out.println(str2);

}

}

===========================================

**Class 클래스**

자바에서 jdbc를 사용하기 위해서 제일 처음 해야하는 것은 ojdbc8.jar를 라이브러리에 추가해주고

Class.forName()메서드를 통해서 오라클 드라이버를 가져오는 것이다.

그럼 Class.forName()은 무슨 역할을 할까?

자바의 모든 클래스와 인터페이스는 컴파일 후 class 파일이 생성된다.

class파일에는 멤버필드, 멤버메서드, 생성자 등 객체의 정보가 담겨 있다.

Class 클래스는 이 class파일에서 객체의 정보를 가지고 올 수 있는 역할을 한다.

Class.forName()을 통해서 가져오는 방식을 동적 로딩이라고 한다.

동적 로딩이라고 하는 이유는 다른 class 파일을 불러올 떄는 컴파일 시 static에 class파일이 바인딩 되지만

forName()으로 class 파일을 불러오면 컴파일에서 바인딩이 되지않고 동적바인딩이 되기 때문이다.

실행시 불러서 사용할 수 있기 때문에 동적 로딩이라고 한다

Class.forName("oracle.jdbc.driver.OracleDriver");

-> class파일에는 객체의 정보가 포함되어 있다.

class.forName() <- 동적로딩

동적로딩은??

실행 중에 데이터 타입을 알고 binding되는 방식을 말합니다.

그래서 데이터베이스에서 사용이 됩니다.

데이터베이스는 실행중에 데이터베이스의 타입을 알고 바인딩해야 하기 때문입니다.

reflection 프로그래밍

-> Class 클래스를 이용하여 클래스의 정보를 가져오고 이를 활용하며 인스턴스를 생성하고,

메서드를 호출하느 프로그래밍 방식이다.

-> 로컬메모리에 객체가 없어서 객체의 데이터 타입을 직접 알 수 없는 경우 객체 정보만을 이용하여 프로그래밍할 수 있다.

===========================================

**collection** - list - ArrayList, Vector, LinkedList (순서상관있다. 중복허용한다.)

L>Set - HashSet, TreeSet (집합을 생각하면 됨 중복허용안함, 순서없다)

collection은 단일 객체들을 어떻게 관리하느냐

Map - HashMap

L> TreeMap

L> HashTable - Properties

Map은 쌍으로 된 데이터를 관리하는 것.

컬랙션프레임워크

자바에서 제공하는 컬랙션 프레임워크는 프로그래밍할 때 사용하는 자료구조나 알고리즘에 대한 라이브러리가 정리되어 있는것.

직접 ArrayList 나 LinkedList를 구현하지 않아도 사용가능하다는 것이다.

제너릭 프로그래밍

-> 변수의 선언이나 메서드의 매개변수를 하나의 참조 자료형이 아닌 여러 자료형을 변환될 수 있도록 프로그래밍하는 방식

-> 실제 사용되는 참조 자료형으로의 변환은 컴파일러가 검증하므로 안정적인 프로그래밍 방식

자료형 매개변수 T

-> type의 의미로 T를 많이 사용함

-> <T>에서 <>는 다이아몬드 연산자라고 함

-> static 키워드는 T에 사용할 수 없다

===========================================

**TreeSet클래스**

-> 객체의 정렬에 사용되는 클래스

-> 중복을 허용하지 않으면서 오름차순이나 내림차순으로 객체를 정렬함

-> 내부적으로 이진검색트리로 구현

-> 이진검색트리에 자료가 저장될 때 비교하여 저장될 위치를 정함

-> 객체비교를 위해 Comparable이나 Comparator 인터페이스를 구현해야 함

================================================

**Comparable 인터페이스와 Comparator 인터페이스**

-> 정렬 대상이 되는 클래스가 구현해야 하는 인터페이스

-> Comparable은 comparaTo()메서드를 구현, 매개변수와 객체 자신을 비교

-> Comparator는 compare()메서드를 구현, 두개의 매개변수를 비교

TreeSet 생성자에 Comparator가 구현된 객체를 매개변수로 전달

TreeSet<Member>treeSet = new TreeSet<Member>(new Member());

===========================================

**Map 인터페이스**

-> key-value 쌍의 객체를 관리하는데 필요한 메서드가 정의됨

-> key는 중복 불가

-> 검색을 위한 자료구조

-> key를 이용하여 값을저장하거나 검색, 삭제 할때 사용하면 편리함

-> 내부적으로 hash방식으로 구현됨

TreeMap 클래스

-> key객체를 정렬하여 key-value를 쌍으로 관리하는 클래스

-> key에 사용되는 클래스에 Comparable, Comparator 인터페이스를 구현

HashMap 클래스

-> map 인터페이스를 구현한 클래스 중 가장 일반적으로 사용하는 클래스

===========================================

public boolean removeMember(int memberid) {

if(hashMap.containsKey(memberid)) {// containsKey로 memberid에 key가 있으면 가지고 온다.

hashMap.remove(memberid);// hashMap에서 remove메서드로 지워준다, 가지고 오려면 get()을 사용

// get (Object key) 라서 get은 검색을 위한 알고리즘이다.

// keySet()으로 모든 key를 가지고 와줘

return true;

}

System.out.println(memberid + "가 존재하지 않습니다.");

return false;

}

===========================================

public void showAll() {

Iterator <Integer> it = hashMap.keySet().iterator();

while(it.hasNext()) { //다음 key가 있으면

int key = it.next();// key값을 가지고 와서

Member member = hashMap.get(key);// key로 부터 value를 가지고 온다.

System.out.println(member);

}

}

===========================================

**내부클래스**

클래스안에 또 클래스를 만든 것

위젯의 이벤트를 핸들링할 때 사용된다.

내부클래스는 inner class라고 부른다

inner class 클래스 안에 다른 클래스를 정의하는 것

이때 private static 으로 만들어 준다. 이렇게 해서 외부에서 inner class에 접근하지 못한다.

// 인스턴스 inner class

class OutClass{

class OutClass{

private int num = 10;

private int num = 10;

private static int snum = 20;

private static int snum = 20;

private InClass inClass;

private InClass inClass;

public OutClass() {

inClass = new InClass();

inClass = new InClass();

}

class InClass {

class InClass {

int inNum = 10;

// inner class에서는 static을 사용할 수 없다, static 메서드와 static 변수는 사용이 불가능하다.

// The field sInNum cannot be declared static in a non-static inner type,

// static int sInNum =10;

void inTest() {

void inTest() {

System.out.println(num);

System.out.println(snum);

}

// The method sTest cannot be declared static;

// static void sTest() {}

}

// public void로 밖으로 정보를 내보낼 수 있다.

public void usingInTest() {

public void usingInTest() {

inClass.inTest();

}

static class InStaticClass{

int inum = 100;

static int sinnum = 200;

void inTest() {

snum += 10;

System.out.println(snum);

System.out.println(inum);

System.out.println(sinnum);

}

// 이때 정적변수들만 호출할 수 있다.

static void sTest() {

System.out.println(snum);

System.out.println(sinnum);

}

}

}

public class InnerTest {

public static void main(String[] args) {

// TODO Auto-generated method stub

OutClass oc = new OutClass();

// 여기서 private로 묶어놔서 다른 메서드들이 안보는 것이다.

// oc.inClass.inTest();

OutClass.InClass ic = oc.new InClass();

OutClass.InClass ic = oc.new InClass();

ic.inTest();

oc.usingInTest();

// OutClass안에 InStaticClass의 메서드를 가지고 오기 위한 방식이다.

OutClass.InStaticClass sInClass = new OutClass.InStaticClass();

sInClass.inTest();

}

}

===========================================

**지역 내부 클래스**

class Outer{

int outNum = 10;

int outNum = 10;

int outNum = 10;

static int sNum = 200;

static int sNum = 200;

public Runnable getRnunnable(final int i) {

public Runnable getRnunnable(final int i) {

// 메서드가 호출되면 스택에 지역변수는 자리를 잡는다.

// 메서드가 호출되면 스택에 지역변수는 자리를 잡는다. 메서드의 호출이 없어지면 사라진다.

// 메서드의 호출이 없어지면 사라진다.

int localnum =100;

//이것이 지역내부 클래스이다.

class MyRunnable implements Runnable{

@Override

public void run() {

// TODO Auto-generated method stub

// 오류가 남 why?? 다른 작업을 하다가 다시 호출하면 이미 스택에서 사라질 수 있기 때문이다.

// localNum += 100;

// 그래서 지역내부 클래스를 사용할 경우에는 final 키워드를 앞에 붙여준다

// 그러면 상수가 저장되는 곳에 자리를 잡기 때문에 변할 수 없다.

// 지역내부 클래스에서 이름을 없앤 것이 익명 내부 클래스이다.

System.out.println(outNum);

System.out.println(sNum);

System.out.println(localnum);

System.out.println(i);

}

}

return new MyRunnable();

===========================================

**익명 내부 클래스**

class Outer{

int outNum = 10;

static int sNum = 200;

// 외부에서 쓰레드를 사용할 때 사용한다. 익명의 내부 클래스의 경우,

// 하나의 추상클래스나 하나의 인터페이스만 구현가능

// 외부에서 쓰레드를 사용할 떄 사용한다.

// 익명 내부 클래스의 경우

// 하나의 추상클래스나 하나의 인터페이스만 구현할 수 있다.

Runnable runnable = new Runnable() {

@Override

public void run() {

// TODO Auto-generated method stub

System.out.println("runnable");

}

};

public Runnable getRnunnable(final int i) {

// 메서드가 호출되면 스택에 지역변수는 자리를 잡는다.

// 메서드의 호출이 없어지면 사라진다.

int localnum =100;

//이것이 지역내부 클래스이다.

return new Runnable() {

@Override

public void run() {

// TODO Auto-generated method stub

System.out.println(outNum);

System.out.println(sNum);

System.out.println(localnum);

System.out.println(i);

}

};

===========================================

**private static class 와 private class의 차이점**

why??

private static class는 큰 클래스 내부에서만 사용가능 밖에 있는거 엑서스할 수 없다. class멤버이고

private class는 생성자에서 큰 클래스의 non-static 멤버를 엑서스할 수 없다. object멤버이기 때문이다.

===========================================

**람다식??**

자바에서 함수형 프로그래밍을 구현하는 방식

클래스를 생성하지 않고 함수의 호출만으로 기능을 수행

함수형 프로그래밍??

- 순수 함수를 구현하고 호출함으로써

외부 자료에 부수적인 영향을 주지 않고 매개 변수만을 사용하도록 만든 함수

- 입력 받은 자료를 기반으로 수행되고 외부에 영향을 미치지 않으므로 병렬처리 가능

- 안정적인 확장성이 있는 프로그래밍

람다식 구현하기

- 익명 함수 만들기

int add(int x, int y){return adds}

람다식 장점

1. 코드를 간결하게 만들 수 있습니다.

2. 코드가 간결하고 식에 개발자의 의도가 명확히 드러나므로 가독성이 향상됩니다.

3. 함수를 만드는 과정없이 한번에 처리할 수 있기에 코딩하는 시간이 줄어듭니다.

4. 병렬프로그래밍이 용이합니다.

람다식 단점

1. 람다를 사용하면서 만드는 무명함수는 재사용이 불가능합니다.

2. 디버깅이 다소 까다롭습니다.

3. 람다를 남발하면 코드가 지저분해질 수 있습니다. (비슷한 함수를 계속 중복생성할 가능성이 높음)

4. 재귀로 만들경우에는 다소 부적합한면이 있습니다.

람다식 사용 X

Person rin = new Person();

rin.hi(new Say() {

public int someting(int a, int b) {

System.out.println("My Name is Coding-Factory");

System.out.println("Nice to meet you");

System.out.println("parameter number is "+a+","+b);

return 7;

}

})

람다식 사용 O

Person rin = new Person();

Person rin = new Person();

rin.hi((a,b) -> {

rin.hi((a,b) ->{

System.out.println("This is Coding-Factory!");

System.out.println("Tank you Lamda");

System.out.println("parameter number is "+a+","+b);

return 7;

});

============================================

**스트림 (Stream)**

- 자료의 대상과 관계 없이 동일한 연산을 수행

(배열, 컬랙션을 대상으로 동일한 연산을 수행함, 일관성 있는 연산으로 자료의 처리를 쉽고 간단하게 함)

- 한번 생성하고 사용한 스트림은 재사용 불가능

(다른 연산을 위해서는 새로운 스트림 생성)

- 스트림 연산은 기존 자료를 변경하지 않음

(자료에 대한 스트림을 생성하면 별도의 메모리 공간을 사용)

- 스트림 연산은 중간 연산과 최종 연산으로 구분됨

(최종연산은 마지막에 한번만 적용)

int [] arr= {1,2,3,4,5};

IntStream stream = Arrays.stream(arr);

int Asum = stream.sum();

System.out.println(Asum);

int counts = (int) Arrays.stream(arr).count();

System.out.println(counts);

// 이런식으로 이미 스트림이 사용되어 버리면 다시 재사용이 불가능하다.

// 다시 Arrays.stream을 지정해줘야 한다.

// 스트림은 람다식의 형태로 사용된다.

List<String> slist = new ArrayList<String>();

slist.add("aaa");

slist.add("bbb");

slist.add("ccc");

Stream<String> streams = slist.stream();

streams.forEach(s -> System.out.println(s));

// stream도 커서느낌이다.

for(String s : slist) {

System.out.println(s);

}

// sorted()가 중간 연산자이고 foreach가 최종연산자이다.

slist.stream().sorted().forEach(s -> System.out.println(s));

스트림의 reduce()연산

- 정의된 연산이 아닌 프로그래머가 직접 지정하는 연산을 적용

- 최종 연산으로 스트림의 요소를 소모하며 연산을 수행

Arrays.stream(arr).reduce(0, (a, b), -> a+b));

-> 배열에 여러 문자열이 있을 때 길이가 가장 긴 문자열 찾을 때 사용한다

import java.util.Arrays;

import java.util.function.BinaryOperator;

class CompareString implements BinaryOperator<String>{

@Override

public String apply(String s1, String s2) {

// TODO Auto-generated method stub

if(s1.getBytes().length >= s2.getBytes().length)

return s1;

else return s2;

}

}

public class ReduceTest {

public static void main(String[] args) {

// TODO Auto-generated method stub

String [] greeting = {"aaa","bbb","afffff","adfaffaff"};

// 람다식으로 구현

System.out.println(Arrays.stream(greeting).reduce("", (s1, s2) -> {

if(s1.getBytes().length >= s2.getBytes().length)

return s1;

else return s2;

}

));

String st = Arrays.stream(greeting).reduce(new CompareString()).get();

System.out.println(st);

}

이런식으로 구현한다. DB랑 비슷한듯

===========================================

**스트림의 구분**

대상 기준

(입력-대상으로부터 자료를 읽어 들이는 스트림/출력-대상으로 자료를 출력하는 스트림)

-> 입력과 출력 스트림을 동시에 사용하지 않는다.

자료의 종류

(바이트- 동영상, 음악 파일 등을 읽고 쓸 때 사용/

문자- 바이트 단위로 자료를 처리하면 문자는 깨짐, 2바이트 단위로 처리하도록 구현된 스트림)

기능(기반- 대상에 직접 자료를 읽고 쓰는 기능의 스트림/

보조- 직접 읽고 쓰는 기능은 없이 추가적인 기능을 더해주는 스트림)

-> 보조 스트림은 직접 읽고 쓰는 기능은 없으므로 항상 기반 스트림이나 또 다른 보조 스트림을 생성자 매개변수로 포함한다.

표준 입출력

- System클래스의 표준 입출력 멤버

public class System{

public static PrintStream out;

public static InputStream in;

public static PrintStream err;

}

- System.out

표준 출력 스트림

System.out.println("에러 메시지");

- System.in

표준 입력 스트림

int d = System.in.read();// 한 바이트 읽어내기

- System.err

표준 에러 출력 스트림

System.err.println("데이터");

===========================================

System.out.println("엔터를 누르세요");

int t = 0;

try {

while(( t = System.in.read()) != '\n') {

System.out.print((char)t);

}

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

===========================================

FileReader fis = null;

int i = 0;

try {

fis = new FileReader("input.txt");

while((i = fis.read()) != -1) {

System.out.print((char)i);

}

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}finally {

try {

fis.close();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

System.out.println("end");

}

-> 파일에서 글자 읽어오기 char를 사용해서

===========================================

try (FileInputStream fis = new FileInputStream("input2.txt")){

byte [] bs = new byte[10];

int i;

while((i = fis.read(bs)) != -1) {

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

System.out.print((char)bs[j]);

}

}

} catch (FileNotFoundException e) {

System.out.println("구문 예외 : " + e.getMessage());

}catch(IOException e) {

System.out.println(e);

}

byte 단위로 읽어오기 10개 단위로

===========================================

-> 파일을 만들어서 내보내기

try(FileOutputStream fos = new FileOutputStream("output.txt")){

fos.write(65);

fos.write(44);

fos.write(66);

}catch(IOException e) {

System.out.println(e);

}

System.out.println("end");

}

fos의 write메서드를 이용해서 파일안에 내용을 채울 수 있다.

===========================================

byte [] bs = new byte[26];

byte data = 65;

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

bs[i] = data++;

}

try(FileOutputStream fos = new FileOutputStream("output.txt", true)){

// 이어서 작성하고 싶으면 true를 적어주면 된다.

fos.write(65);

fos.write(44);

fos.write(66);

System.out.println();

fos.write(bs);

}catch(IOException e) {

System.out.println(e);

}

System.out.println("end");

}

-> 이런식으로 알파벳을 넣을 수 있다.

===========================================

try (FileWriter fw = new FileWriter("wrtier.txt")){

fw.write("a");

char [ ] cbuf = {'c', 'b', 'a'};

fw.write(cbuf);

fw.write("hi");

}catch(IOException e) {

System.out.println(e);

}

System.out.println("end");

}

-> 문자 배열도 이렇게 넣을 수 있다.

===========================================

**보조 스트림**

-> 실제 읽고 쓰는 스트림이 아닌 보조적인 기능을 추가하는 스트림

-> 데코레이터 패턴

-> FilterInputStream과 FilterOutputStream이 보조스트림의 상위 클래스

try(InputStreamReader irs = new InputStreamReader(new FileInputStream("reader.txt") )) {

// 다른 스트림을 받아서 바이트면 문자로 전환해 준다.

int i = 0;

while((i = irs.read()) != -1){

System.out.print((char)i);

}

} catch (FileNotFoundException e) {

System.out.println(e);

}catch(IOException e) {

System.out.println(e);

===========================================

long milliseconds = 0;

int len = 0;

try( FileInputStream fis = new FileInputStream("a.exe");

FileOutputStream fos = new FileOutputStream("b.exe")){

milliseconds = System.currentTimeMillis();

int i ;

while((i = fis.read()) != -1 ) {

fos.write(i);

len++;

}

milliseconds = System.currentTimeMillis() - milliseconds;

}catch(IOException e) {

System.out.println(e);

}

System.out.println(len);

System.out.println(milliseconds);

}

}

-> exe파일을 복사하는데 걸리는 시간을 측정

===========================================

**직렬화 - Serialization**

Serializable 인터페이스

-> 직렬화는 인터페이스 내용이 외부로 유출되는 것이므로 프로그래머가 직렬화 의도를 표시해야한다.

class Person implements Serialzable{ <-- implements Serialzble이 직렬화하겠다는 의도로 표시

String name;

String job;

}

===========================================

콘솔 창 에러 났을 경우

Window - Perspective - ResetPerspective

===========================================

**JAVA - AWT - ActionListener**

ActionListener는 JButton/JTextField/JCheckBox/JRadioButton 등

주로 버튼을 클릭할 경우나 enter키를 눌렀을 경우 발생하는 ActionEvent를 제어하는 역할을 합니다.

이때, 중요한 것은 ActionPerformed(ActionEvent 변수)를 override 해야한다는 것입니다

===========================================

ctrl + shift + w => 모든 창 닫기

===========================================

package controlStatement;

import java.util.Scanner;

public class Ex04 {

public static void main(String[] args) {

// primitive는 == 연산자로 일치여부를 비교할 수 있지만, Reference Type은 그렇지 않다

// 1번과 2번이 같은 이유 : 프로그램이 실행되기 전에 상수 데이터를 먼저 메모리에 준비하는데

// 기존에 준비된 상수가 똑같이 다른 변수에서 사용되면, 하나의 상수 주소를 서로 다른 변수에게 똑같이 전달

// 1번과 3번이 다른 이유 : new 연산자는 메모리 상에 새로운 공간을 할당받아서 내용을 만든다

// malloc()

// 1번과 4번이 다른 이유 : 사용자에게 입력받기 위한 변수는 어떤값이 들어올지 모르니까 공간을 새로 할당받는다

Scanner sc = new Scanner(System.in);

String str1 = "Hello";

String str2 = "Hello";

String str3 = new String("Hello");

String str4;

System.out.print("Hello 라고 입력하세요 : ");

str4 = sc.nextLine();

// 각 참조변수가 같은 객체를 참조하고 있는가(같은 대상을 바라보고 있는가)

System.out.println(str1 == str2);

System.out.println(str1 == str3);

System.out.println(str1 == str4);

// 문자열끼리 내용의 일치여부를 판별하려면

System.out.println(str1.equals(str2)); // 객체(String)간의 내용 비교

System.out.println(str1.contentEquals(str3)); // 문자열과 준 문자열간의 내용 비교

System.out.println(str1.equalsIgnoreCase(str4)); // 문자열끼리 비교(대소문자 구분없이)

sc.close();

}

}

- 모든 switch~case 는 if문으로 대체 가능(python은 switch가 아예 없다)

- case에는 일반적으로 break로 마무리

- 자바의 switch는 문자열 비교도 가능하다

- switch에는 변수 혹으 변수를 포함한 연산식이 들어갈 수 있다.

===========================================

**재귀함수란?**

함수가 직접 또는 간접적으로 자신을 호출하는 프로세스를 재귀함수라고 합니다

ex.피보나치 수열 ,팩토리얼 함수 등등

package exam;

import java.util.Scanner;

public class Ex15 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("정수를 입력하세요" );

int num = sc.nextInt();

// 입력 받은 값인 num까지 반복문을 돌려 fibo함수를 출력하게 하는 것이다

for(int i =0; i < num; i++) {

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

}

sc.close();

}

private static int fibo(int num1) {

// 입력 받은 값이 0부터 1까지는 그대로 출력한다

if (num1 <= 1) {

return num1;

}else {

// 예를 들어 5를 입력하면 0 1 1 2 3 이 나와야 하는데

//1까지는 for문을 돌면서 그대로 출력

//2부터는 0+1 로 1이 만들어 진 것이기 때문에

//0 + 1의 조건을 만들어 줘야 한다

//0은 입력받은 값의 -2 이고 1은 입력받은 값의 -1이다

//이렇게 더해서 반환해주는 것이다

return fibo(num1 -2) + fibo(num1-1);

}

}

}

===========================================

**API(application programming interface)**

- 응용프로그램 함수들의 집합이라고 할 수 있는데 미리 만든 함수를 모아 놓은 곳을 말한다

- 클래스에서 배우게 될 인터페이스는 미리 구현해야 할 메서드들을 미리 정의해 놓은 것입니다

- 우리는 API만 보고도 필요한 함수를 찾아가며 쉽게 개발할 수 있습니다

javadoc로 API 생성하기

===========================================

**new 연산자** : 메모리에 데이트를 저장할 공간을 할당받고 그 공간의 참조값을 반환합니다

===========================================

String name = "ad|af|adf|adf|afadf|adff";

String [] a1 = name.split("\\|");

System.out.println(Arrays.toString(a1));

Arrays.toString : 메서드로 배열로 출력가능, 배열의 내용을 문자열로 반환 (2차원 이상에서는 안됨)

deepToString : 지정된 배열의 내용을 문자열로 자세히 표현하여 반환 (2차원에서도 사용가능)

Arrays.copyOf(object[], original, int newLength) : 배열의 길이만큼만 배열을 복사하여 반환

Arrays.binarySearch() : 정렬된 배열을 이진 검색 알고리즘을 사용하여 지정된 값을 배열에서 찾아서 인덱스를 반환

===========================================

**thread : 자바에서 다중작업을 구현하기 위한 클래스**

실행중인 프로그램 안에서도 하나의 흐름으로 작업을 처리하는 것이 아니라

시간을 카운트하면서 입력을 대기중인 경우

온라인 게임에서 Queue를 찾으면서, 채팅을 진행할 수 있는 경우

A작업을 수행하면서, 동시에 B작업을 처리하는 하나의 프로세스가 존재할 경우

프로세스 내부의 각각의 서로 다른 작업 흐름을 구분하는 단위가 스레드

===========================================

함수형 인터페이스 : 인터페이스가 오직 하나의 추상메서드만 가지는 형태

-> 함수형 인터페이스는 람다식을 사용할 수 있다

인터페이스가 오로지 하나의 추상 메서드만 포함하는 경우, java 1.8부터 지원되는 람다식 객체 생성을 활용

for(int i = 0; i < a.length-1; i++){

for(int j = i+1; j<a.length; j++){

int tmp = a[i]

a[i] = a[j]

a[j] = tmp

}

}

===========================================