인프런 JAVA 공부

* **리팩토링**
* **패키지 및 접근 제한**

패키지: 많은 파일을 효율적으로 관리하기 위해 JAVA에서는 ‘패키지’라는 구조를 사용합니다. 클래스를 모아놓은 폴더.

api들을 잘 활용하면 수많은 기능들을 활용할 수 있다. Jre -> Api -> index.html 에 들어가면 패키지들의 정보가 있음.

* 패키지의 일반적인 설계 방식

주로 패키지 이름은 유니크한 도메인 주소를 역방향으로 하여 만듭니다.

Ex). 회사 도메인이 [www.javalec.com](http://www.javalec.com) 이라면 ‘com.javalec.’라고 하고 뒤에 기능별 또는 개발팀에서 정한 규칙을 따름.

* Import

다른 패키지의 클래스를 가져올 때, 사용.

* 접근제어

: 클래스의 데이터(인스턴스) 또는 메소드에 대해서 다른 클래스에서의 접근을 제한하는 것.

Public: 접근 제한 하지 않음

Private: 자신의 클래스에서만 사용 가능, 외부에서는 사용 불가능

Protected: 해당 클래스와 동일한 패키지에 있거나 추후에 배울 상속받은 클래스에서만 사용 가능

Default: 해당 클래스와 동일한 패키지에 있을 때만 사용 가능.

* Static

객체를 ‘공유’ 한다고 생각, static 키워드를 공유라고 생각하자. (지갑 클래스와 돈을 나눠주는 예시)

객체를 생성하기 전 클래스의 데이터를 공유하는 것!

Static을 이용하면 객체를 생성하지 않고도 클래스명을 이용해 바로 static 변수에 접근 가능

* **상속/추상화**

: 어떤 객체가 다른 객체로부터 객체의 데이터(속성)와 메소드를 상속받을 수 있음. 이럴 경우 상속받은 객체의 데이터 및 메소드를 이용할 수 있고 또는 변경할 수도 있습니다.

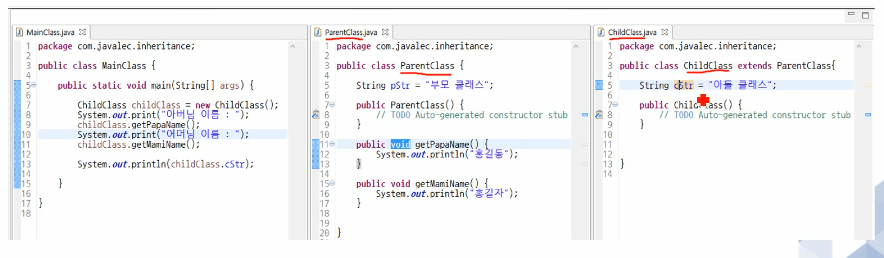
Ex) 상속을 통해서 4발 자전거를 만드는 것

4발 자전거는 바퀴가 4개가 있어야함, 그리고 새로 클래스를 만든다면 시간도 많이 쓰게 됨.

상속이 필요한 이유

1). 기존의 소스를(검증된) 이용할 수 있음

2). 다양한 객체(타입)을 상속을 통해서 하나의 객체(타입)으로 묶을 수 있음 (추상화)



상속받는 클래스는 클래스 이름 뒤에 **extends** 예약어를 쓰고 그 뒤에 상속 받을 클래스를 씀.

* 오버라이드(재정의)

상속 받은 클래스의 메소드를 이름을 똑같이 하면서 내용을 바꿀 수 있음 (오버라이드, 재정의)

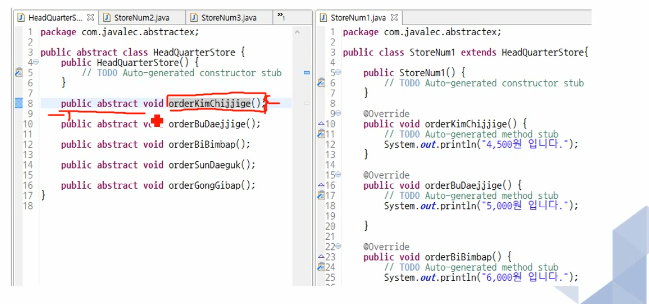
* Super 키워드(오버라이드의 장점과 단점의 이해)

오버라이드의 장점은 잘 만들어진 클래스를 상속받아 일부 메소드를 수정(재정의)할 수 있다는 것, 하지만 자식클래스에서 부모클래스를 오버라이드하면 부모클래스의 메소드는 사용할 수가 없습니다. 이것이 오버라이드의 단점. 자식클래스에서 부모클래스의 메소드를 사용하고자 할 때 **super** 키워드를 사용함.

* **추상클래스**

**“추상적으로 정의할 테니, 사용자가 꼭 재정의 해라” (강제적으로 구현해야만 함)**

오버라이드를 해야한다는 것, 그런데 오버라이드를 안하면 문제가 될 수도 있음.

  
왼쪽과 같은 방법으로 abstract 키워드를 사용, 메소드 이름만 선언. 만약 메소드들이 abstract 선언이 되있으면 그 클래스를 상속받는 옵션을 체크하고 클래스를 생성하면 자동으로 오버라이드를 유도하게 (재정의하기 쉽게) 만들어줌. 추상클래스에서는 정의만함, 구현은 안함.

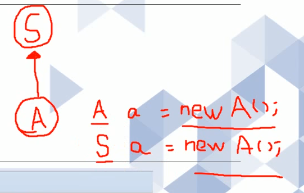
* 추상클래스 상속과 일반 상속과의 차이: 차이는 있어도 어느 쪽이 더 좋다고는 못함.

15\_1\_ex1\_launchpay 예제에서 PriceTable과 같이 상수를 이용하는 법, 그리고 추상화 클래스를 어떻게 이용하는지에 대해서 알 수 있음.

* **인터페이스**

: 객체와 객체의 소통 수단, 작업명세서 – “앞으로 이렇게 만들어요.”

**다형성** - JAVA에서 다형성을 가능하게 함.



사용법은 어렵지 않지만, 실제 개발에 적용시키기는 쉽지 않음

디자인 패턴이나 프레임웤을 통해 습득하는 것이 최선

객체를 부속품화 – 다양한 객체를 제품의 부속품처럼 개발자 마음대로 변경할 수 있음(정품>비정품)

* 인터페이스 문법

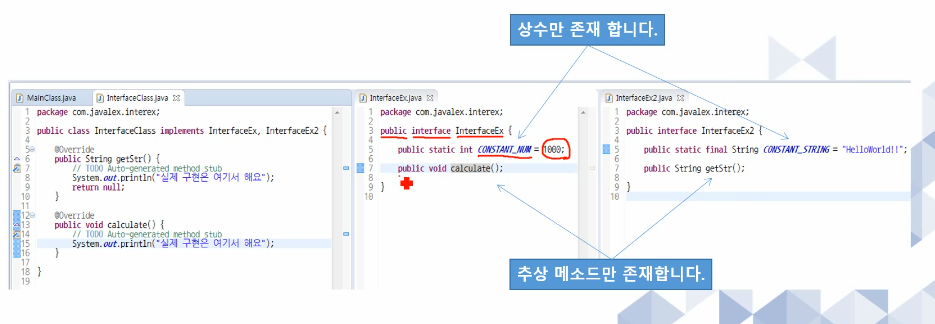
실제 기능이 없음 – 구현된 기능은 없음, 추상메소드와 상수만이 존재

Private는 안됨. – 상수를 만들 때 private 접근 제한자는 안됨.

추상화 – 메소드는 무조건 추상 메소드만 존재

객체타입 – 인터페이스는 객체는 아님, 다만 객체 타입으로만 사용

구현은 실행되는 객체의 메소드에서 함.



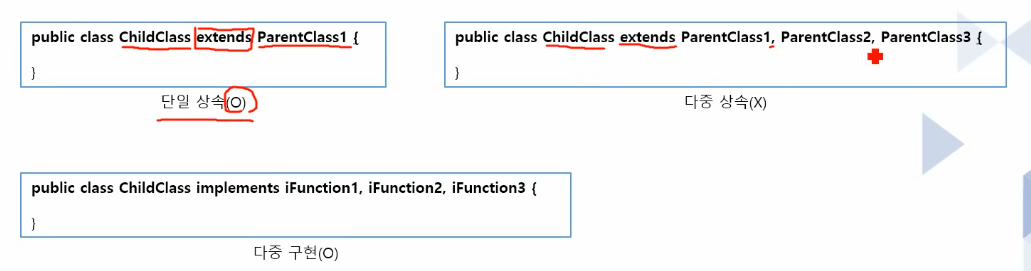
Implements , interface 키워드

Implements 뒤에 인터페이스의 개수는 몇 개이든 상관없음

Interface를 데이터타입 이라고 생각해도 됨. (대신 상속받은 클래스에서 구현은 해야함)

* 인터페이스와 다중상속

자바는 다중상속이 불가함. java에서는 interface를 통해 여러 가지 타입으로 객체를 선언할 수 있는데 이것은 다중상속이 아닌 ‘다형성’이 존재한다고 이해해야 함.



인터페이스를 상속받을 때는 ‘빈 화살표’로 표기할 수 있음.



Toy는 아무런 내용이 없지만 장난감들은 Toy 라는 하나의 묶인 타입을 쓸 수 있음.

* 인터페이스와 추상클래스

공통점

1. 추상메소드를 가지고 있다 – 추상메소드를 가지고 있어 하위 클래스에서 구현해야 함.
2. 데이터 타입이 목적 – 객체생성이 목적이 아닌 데이터 타입을 정의하는 것이 목적
3. 객체 생성은 anonymous 를 이용해야함.

차이점

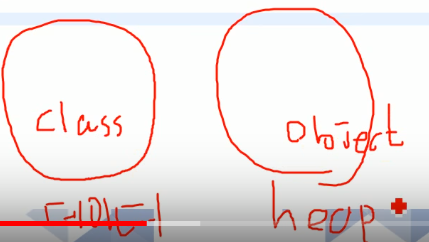
1. 상속, 구현 – 추상메소드는 상속을 통한 사용이고, 인터페이스는 구현을 통한 사용
2. 구성요소 차이 – 추상클래스는 일반 클래스와 동일하게 변수, 메소드의 모든 기능을 사용할 수 있지만, 인터페이스는 상수와 추상메소드만이 존재
3. 단일상속, 다중구현 – 추상클래스는 상속이므로 단일 상속만 지원, 인터페이스는 다중구현이 가능.

* 패턴

객체지향 언어의 상속, 인터페이스, 추상화 등의 기능을 이용해서 시간과 인력 투입의 효율을 극대화 하기 위한 방식이 정립되기 시작함. 이러한 방식을 어떠한 틀로 정해 놓은 것이 바로 **디자인 패턴.** 객체지향 언어의 장점들을 모아 가장 효율적으로 개발을 할 수 있게 만들어 놓은 틀.

* 싱글턴 패턴(singleton)

어떤 클래스의 인스턴스는 오직 하나임을 보장하며, 이 인스턴스에 접근할 수 있는 전역적인 접촉점을 제공하는 패턴. 전역의 개념으로 객체를 사용.



* API(Application Programming Interface)

기능들을 미리 정의해둔 클래스들을 말함. API는 누군가가 만들어 놓은 기능들이라고 생각하면 됨.

필수 API가 있음.

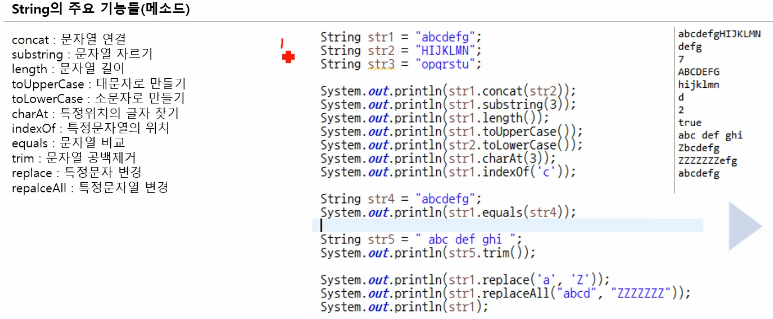
* String

객체 자료형, 대문자로 시작함, 생성자(new)를 쓰지 않고도 만들 수 있게 해놓음.

String string = “Helloworld”;

String string = new String(“HelloWorld”);

위의 두 줄은 똑같은 의미.



String의 문제점: String은 메모리를 과소비함. 스트링은 처음 생성된 데이터에 변화가 생기면 기존 것을 활용하기 보다는 새로운 것을 찾음.

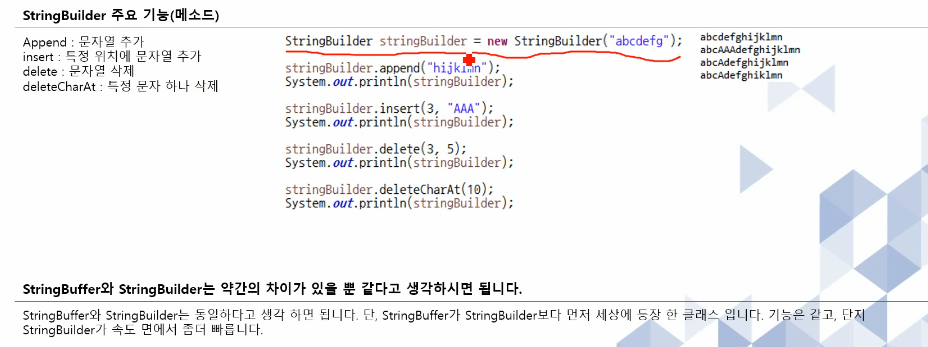
* StringBuffer와 StringBuilder의 등장

String클래스의 새로운 메모리를 생성하는 문제로 인해 속도가 느려지는 현상이 있음.

속도적인 측면에서 더욱 개선된 StringBuffer와 StringBuilder가 등장

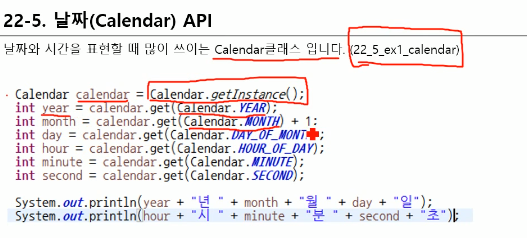
일단 String이 아니기 때문에 일반적인 객체 생성과 같이 해야함.

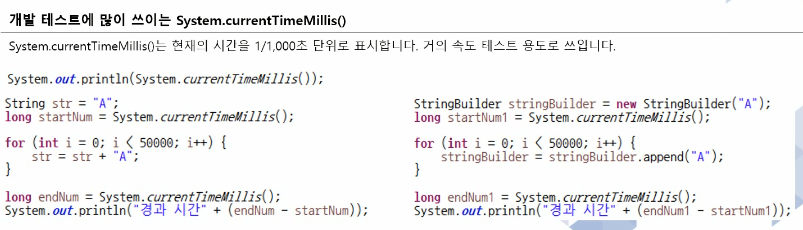
StringBuilder stringBuilder = new StringBuilder(“abcdef”);



* 날짜(Calendar API)

Calendar calendar = Calendar.getInstance();





* Random 클래스

Math.random()은 과거에 쓰임 double타입의 임의의 수를 발생

Double d = Math.random()

Random클래스 사용법

Random random = new Random(); random.nextInt(100);

직접 객체를 생성, 정수형의 난수 생성, nextInt로 bound 값을 줄 수 있음.

* Scanner

: 무엇인가를 얻어 올 때 사용, 키보드에서 타이핑하는 문자열 또는 입출력 프로그래밍에서 문자열을 읽어올 때 사용

Scanner scanner = new Scanner(System.in);

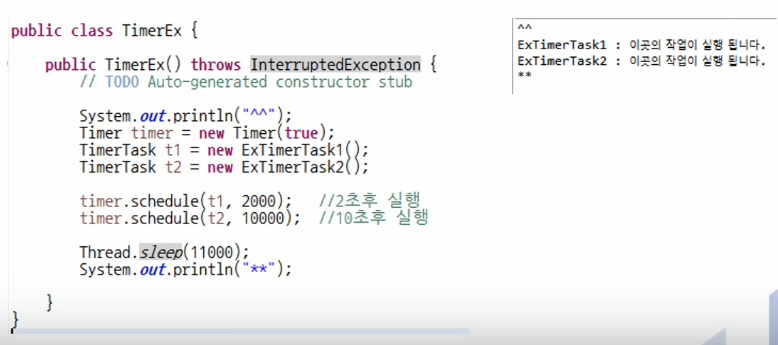
Int i = scanner.nextInt(); // 정수형

String str0 = scanner.next(); // 문자열

String str1 = scanner.nextLine(); //문자열(처음 개행 문자를 만나면 거기까지 받고 나머지 버림)

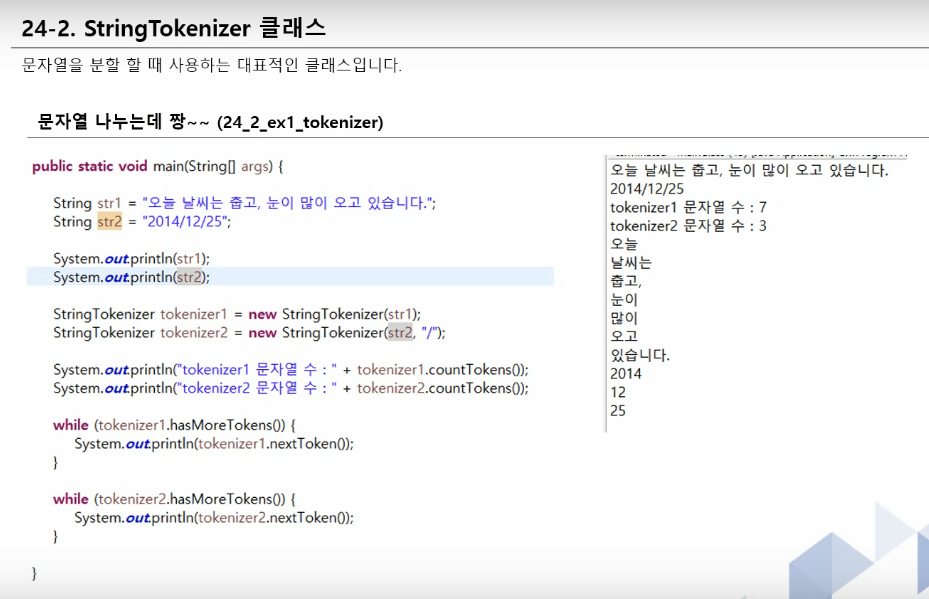
* Timer 클래스

Timer 객체는 일정한 시간이 되면, TimerTask객체가 작동 됨. 단 TimerTask는 추상클래스. 따라서 TimerTask 클래스를 상속받는 클래스를 만들어서 써야함.



* 프로젝트 실습한거 다시 보기.
* StringTokenizer 클래스

: 문자열 나누는데 좋음, 문자열 안의 구분자를 통해 나눌 수 있음.(, / ; 등등)



정해진 값이 없을 때는 ‘공백’으로 문자열을 나눠줌.

.countToken() = 토큰의 수만큼(분리된 수만큼) 출력

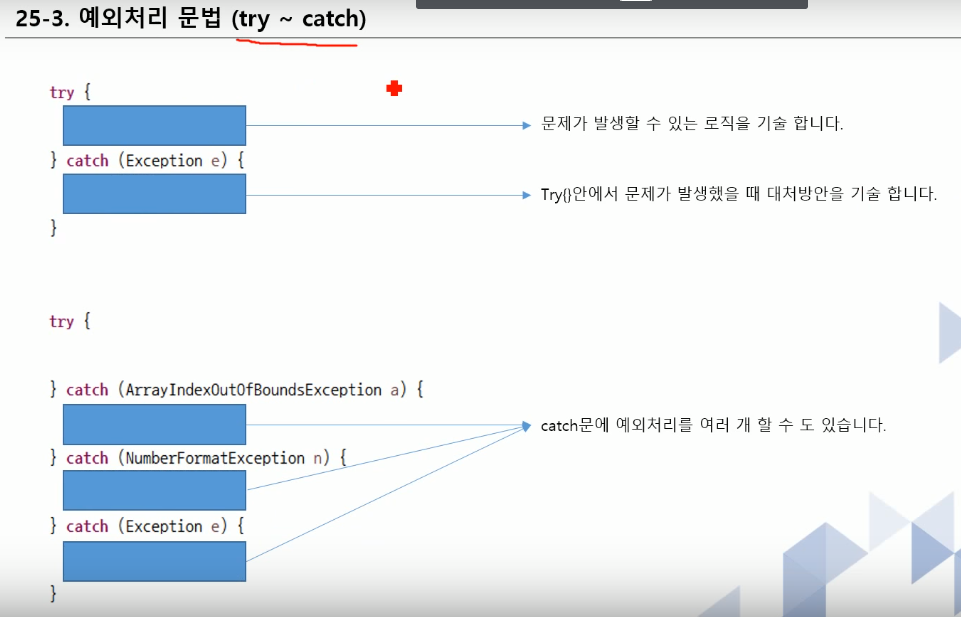
.hasMoreTokens() = 다음 토큰이 있을 때 값이 true (즉, 반복문으로 토큰이 존재할때까지 반복문을 돌릴 수 있음).

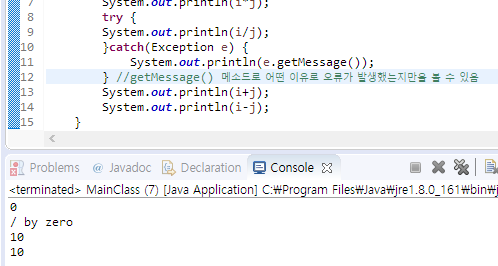
.nextToken() = 다음 토큰.

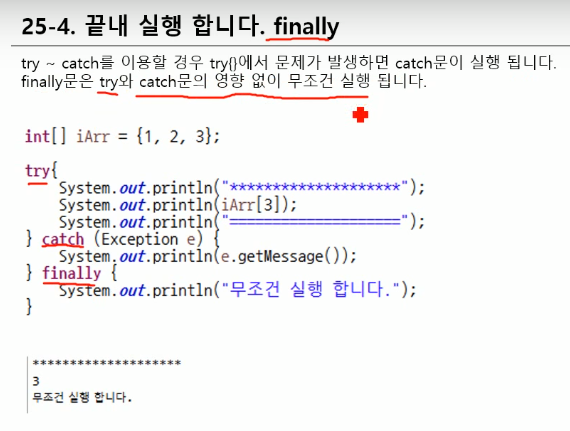
* 예외처라

: 프로그램에서 문제가 발생될 만한 곳을 가상해 문제가 발생하면 이렇게 처리해라 라고 프로그래밍하는 것.

예외의 필요성은 어느 한 부분에서 예외가 발생하더라도 계속해서 프로그램이 동작되도록 하는데 목적이 있음. (ex: 어떤 숫자를 0으로 나누는 경우)







Finally는 예외가 발생했거나 안했거나 무조건 발생함.

* Throws 문법

: try \_ catch 예외 처리 방법은 예외가 발생했을 때 자체적으로 catch문을 이용해서 해결, throw의 경우네는 예외를 발생시킨 (호출) 쪽으로 예외를 던져버리는 방식.

* 일반적으로 많이 보게 되는 예외들

Exception의 종류는 아주 많음. Exception 클래스는 많은 예외들(자식클래스)를 가지고 있음.

1. ArrayIndexOutOfBoundsException: 배열을 사용시 존재하지 않는 index 값을 호출하면 발생.
2. NullPointerException: 존재하지 않는 객체를 가리킬 때 발생.
3. NumberFormatException: 문자를 숫자로 처리할 때 발생 (Interger.parseInt(str)……)
4. ClassNotFoundException: 드라이브 이름을 찾지 못했을 때
5. SQLException: db, url, id, pw가 올바르지 않을 때

* **Collection (자료구조)**

컬렉션이란 자료구조이다. (배열)

JAVA도 결국 효과적인 데이터를 처리하기 위한 수단:

JAVA에서는 다양한 자료구조형을 제공하고 있음. 다양한 자료구조형이 제공되는 이유는 데이터의 성질에 따라 데이터를 관리(정리)해야 하는 방식이 다르기 때문. 자료구조형안에서는 객체의 레퍼런스만을 관리함.

* **List 계열 컬렉션 클래스**

**1). ArrayList:** 배열과 매우 비슷, 인덱스가 존재, 데이터가 중복되어도 상관없음, 크기를 가변적으로 줄이고 키울 수 있음.

. arrayList.add(“str1”); : 원소 추가하기.

. arrayList.get(3) : 인덱스값이 3에 해당하는 값 가져옴.

. arrayList.set(2,”str222”) : 인덱스값이 2에 해당하는 값을 “str222”로 바꿈

. arrayList.toString() : arrayList를 순서대로 출력.

. arrayList.remove(2) : 인덱스값이 2에 해당하는 값을 삭제함.

. arrayList.clear() : arrayList를 삭제함(모두 다).

* **LinkedList**

LinkedList<String> linkedList = new LinekedList<String>();

. linkedList.add(2, “str”) : 인덱스가 2인 값이 있는 곳에 “str”을 넣고 나머지는 뒤로 하나씩 밀림

. linkedList.add(“str5”) : 리스트의 제일 끝에 “str5”를 추가

. linkedList.size() : 리스트의 size를 반환

나머지는 arrayList의 메소드와 같음.

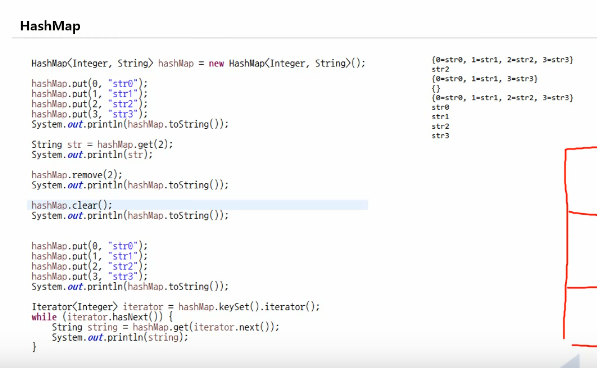
* **Vector**

: 어레이리스트와 비슷하지만 속도가 떨어지고, 멀티쓰레드 환경에서는 보다 안전함.

* **Map 계열 컬렉션 클래스**

물품보관함을 생각하면 됨. 해당 물품보관함의 키가 중요. 키만 있으면 키에 해당하는 물품보관함을 이용할 수 있음. List계열과 달리 인덱스가 없고, 키와 값만 있음. 키는 유니크해야함. 우리가 값을 관리하고자 한다면 키를 이용해서 값을 관리할 수 있음.

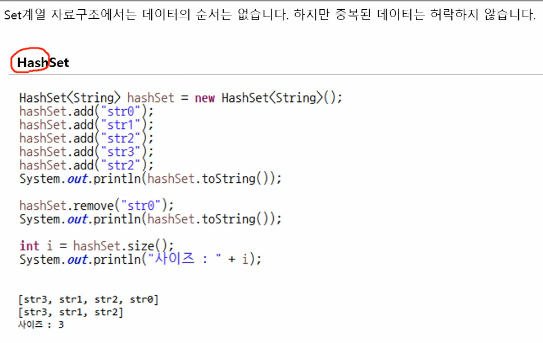
* HashMap



Iterator: 반복자 (인덱스 값을 차례로 얻어오는 방법)

while문에서 .hasNext() 메소드로 다음의 인덱스의 값이 있는가를 확인하고 계속 값이 있을 때만 반복.

* HashSet



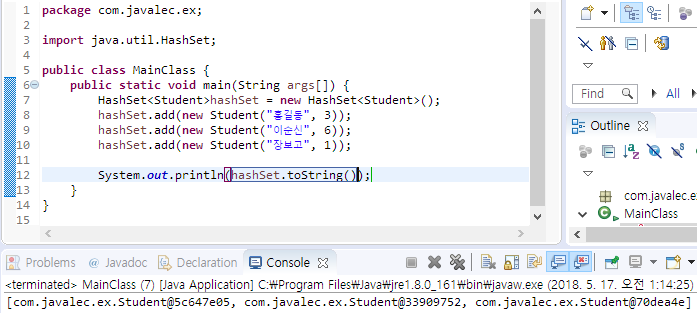
: 인덱스가 없고 중복된 데이터 허용하지 않음.

. hashSet.remove(“str0”) : Set에서 “str0”의 값을 지움(인덱스 값하고는 상관없음).

. hashSet.add(“str1”) : Set에 “str1”의 값을 추가.



: toString을 overriding 함



: 결과창은 객체들의 주소값이 찍힌 것.



: 왜 데이터가 없어지지 않고 남아있는가????

해당 기초데이터는 삭제하지만 Student를 사용자가 직접 만든 객체이므로 만들 때 메모리하고 .remove할 때의 메모리하고 차이가 있기 때문에 그대로 나옴.

* 해결하는 방법: .hashCode()

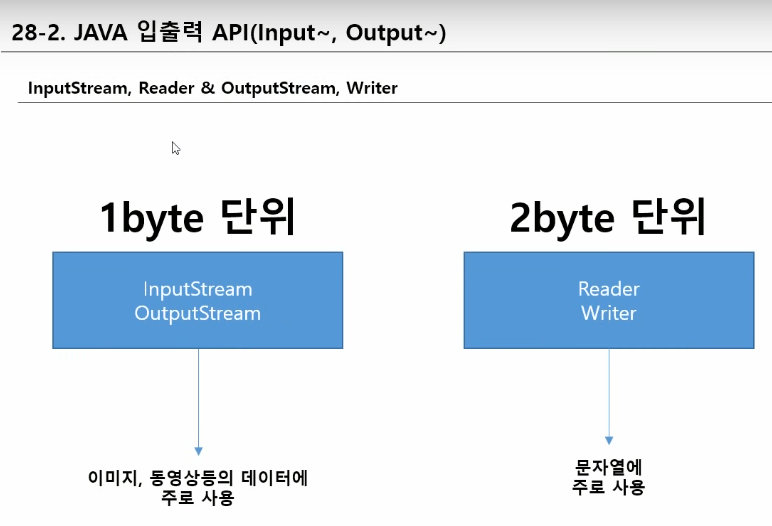


* Iterator 자료구조

: 많은 자료구조형이 존재하는 것은 결국은 데이터를 잘 관리하기 위해서임. 잘 관리한다는 것은 저장하고, 정렬하고, 검색하는 것. 이중에서 데이터의 검색은 무엇보다도 중요. 그래서 JAVA에서는 Iterator라는 인터페이스가 있음. 의미는 ‘반복자’ 라는 의미로 데이터를 반복적으로 검색하는데 아주 유용한 인터페이스임. 모든 자료구조형은 iterator() 메소드를 지원함.

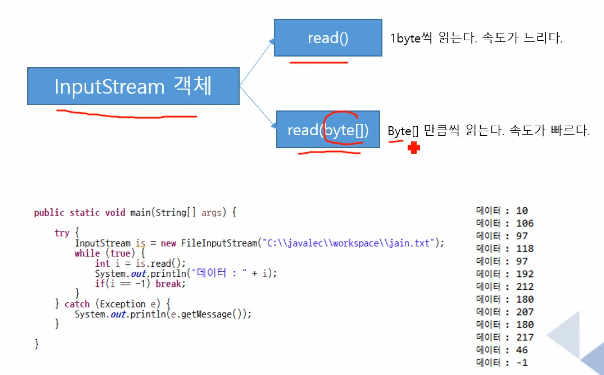
* JAVA I/O

입력과 출력을 뜻함.



InputStream, OutputStream: 1바이트, 이미지/동영상등의 데이터에 주로 사용

Reader, Writer: 2바이트, 문자열에 주로 사용.



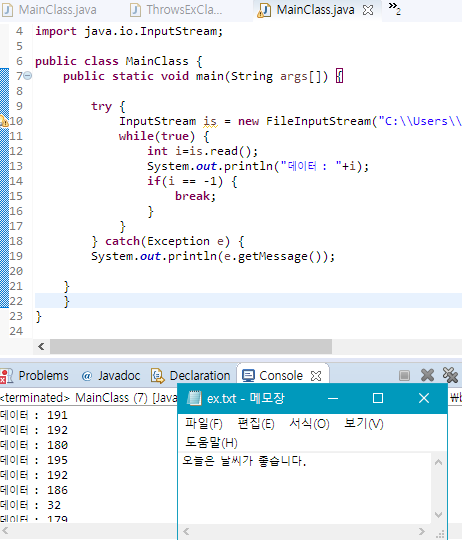
* InputStream 사용법:

1). InputStream (추상)클래스를 이요해서 객체를 만듬. 또는 다른 클래스의 메소드에서 반환(리턴)되는 타입 객체를 얻는다.

2). read() 메소드를 이용해서 데이터를 읽으면 됨.

3). Read(), read(byte[]) 두 개의 메소드를 이용할 수 있음.

I/O와 관련된 것은 IOException을 이용해서 예외처리를 해야함(프로그램 자체가 강요함).



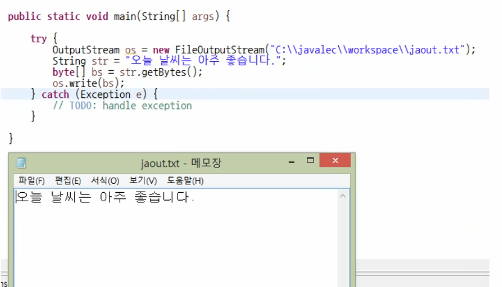
* 단순히 데이터를 읽어왔을 때
* OutputStream 사용법

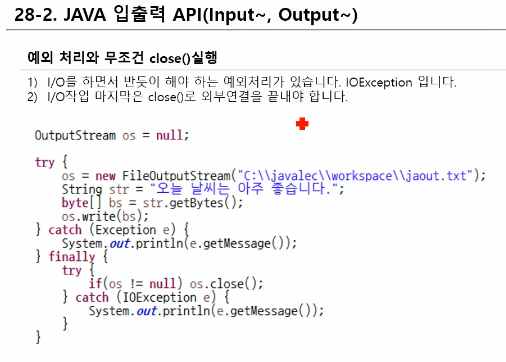
1). OutputStream (추상)클래스를 이용하여 객체를 만듬. 또는 다른 클래스의 메소드에서 반환(리턴)되는 타입 객체를 얻는다.

2). Write()메소드를 이용해서 데이터를 읽으면 됨.

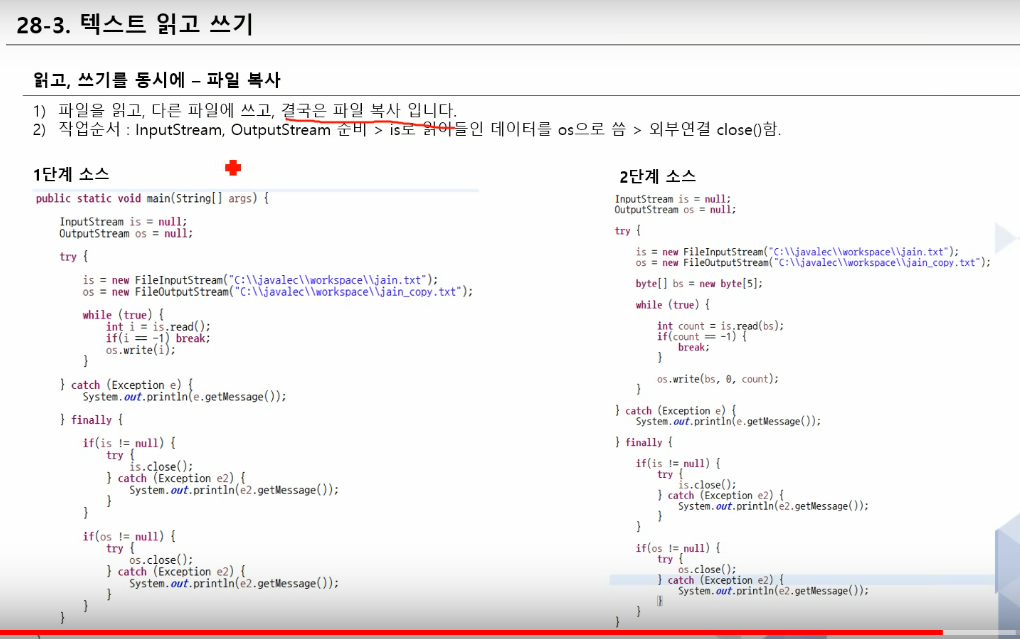
3). Write(), write(byte[]), write(byte[], int, int) 세개의 메소드를 이용할 수 있음.

4). Write(byte[], int, int)는 데이터를 원하는 위치에서 원하는 숫자만큼 쓸 수 있음.

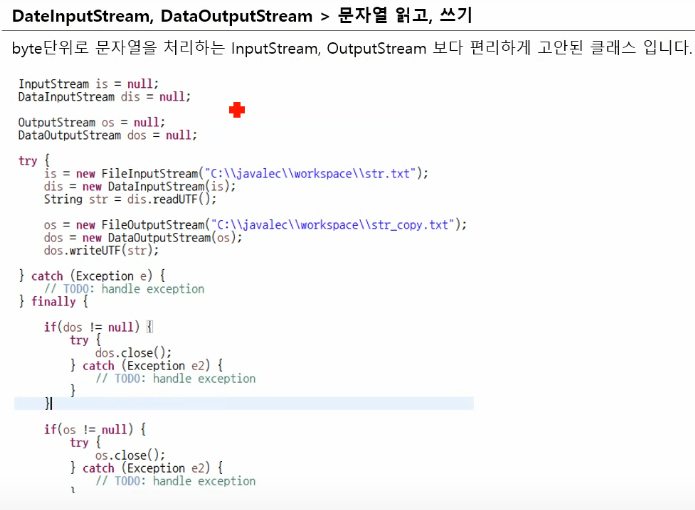




Finally 문은 반드시 실행 되기 때문에 이 안에 close() 를 넣음.



1단계와 2단계는 같은 과정이지만 2단계는 바이트 수를 늘려서 write함.

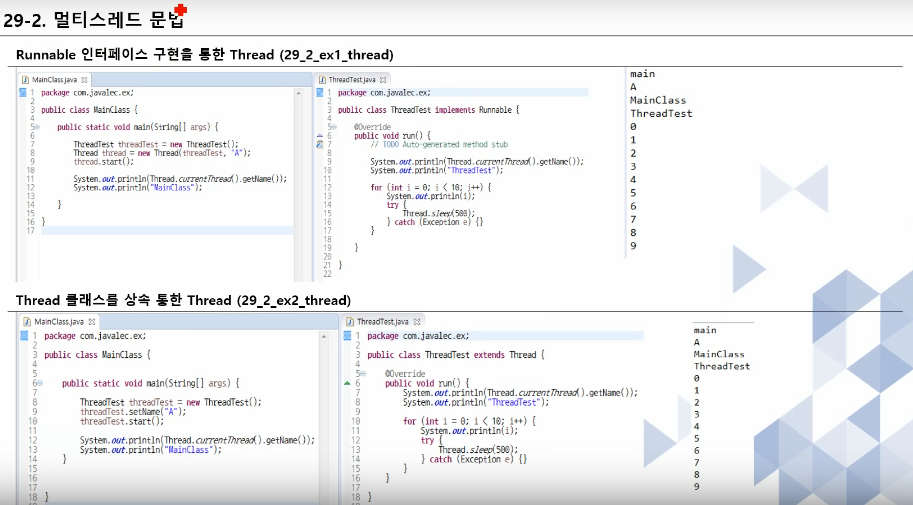


* JAVA 쓰레드

멀티프로세스는 컴퓨터가 여러 가지 일을 동시에 하고 있는 것을 뜻함. 쓰레드는 전에 말한 하나의 프로세스에서 다시 여러 가지 일을 하는 것을 뜻함. Ex). 파일을 전송하면서 채팅을 하고 있음. 동일한 채팅프로그램에서 파일전송과 채팅을 동시에 하고 있는 것. JAVA는 기본적으로 멀티스레드를 지원함. 두 가지 형태의 멀티스레드, 하나는 객체 하나를 n개의 스레드가 공유하는 것, 또 하나는 객체 하나당 하나의 스레드가 존재하는 방식.



* 멀티스레드 문법



Runnable을 상속받으면 run() 메소드를 구현해야함.

처음 찍히는 main은 main 쓰레드의 이름

그리고 사용자가 “A”라는 이름의 쓰레드를 생성시켰기 때문에 “A” 쓰레드, ThreadTest가 찍힘.

辛宇英