2024-03-04 Github 연수

검색에서 gitBash 실행

$ cd Desktop/

$ cd BitComputer2024/

$ cd JAVAEXAM/

$ git add .

$ git commit -m “커밋 내용 입력”

$ git push

Local Repository – 공유되지 않은 소스 코드

Remote Repository – Remote Repository – 공유된 깃 허브

총 3가지 단계

Working Directory(추가/수정/삭제확인)

-> Staging Directory (Working Directory중 확인된 파일에서 무엇을 저장할지 대상을 지정하는 영역), (git소스코드의 저장보다 소스코드의 저장이유와 기능설명 중요)

> Git Directory : Remote Directory 저장하는 대상,

Git Branch - >

Git의 가장 대표적인 기능

소스코드 병합을 할거고, 그때 문제가 생기는지 점검을 하고,

하나의 소스파일을 여러 개의 소스파일처럼 관리할 수 있다.

Branch는 코드에 대한 작업이 완료되면 삭제한다.

Switch –

- Git branch이동하는 기능

- 해당 branch에 추가/수정/삭제된 소스가 있다면 Switch에 제한이 걸릴 수 있다.

명령어

Status -> git의 파일저장상태를 체크한다. 제일 많이 씀.

Add -> commit 대상을 지정하는 기능 / 변경된 소스파일을 저장대상에 넣는다.

Commit -> remote Repository로 보내는 대상이며 택배상자와 비슷한 원리/ 추가,수정,삭제된 소스코드에 대한 사유와 설명을 적는다.

Push -> 해당 Branch에 정의된 Commit 된 내용을 Remote Repository로 보내는 명령어. 하루에 한번은 무조건해야 한다. 하루 넘어가면 팀과 상의 해야 한다. 소스코드 푸쉬 하루라도 밀리면, 사이클 무너진다. 그러면 깃을 사용할 의미가 없다.

Pull -> Remote Repository의 내용을 Local Repository로 보내는 기능. pull하고 push해야 한다. 습관처럼 해야 한다.

Merge – 다른 branch의 내용을 합치는 과정. 반드시 수행을 하지 않아도 된다. 반드시 수행하면 좋으며 안해도 동작은 될 수 있지만 오류 발생되면 제일 후회한다.

Stash – git의 Branch이동하기 전 임시저장 기능

Github 실습

Read Me – 마크다운 배우면 좋지만, 시간이 없으면 노션으로 작성해서 붙이면 된다.

$ pwd 현재위치

$ git config --global user.email sandcocktail@gmail.com

$ git config --global user.name "Ilseob-Han"

$ git init

/

$ git status //상태확인

$ git add . //테스트 디렉토리 전체

$ git commit -m "hello 추가(내용 입력)"

Cd .. -> 윗 폴더 이동

깃동기화 절차

$ git init

$ git status햣

$ git add .

$ git commit -m "hello 추가(내용 입력)"

$ git branch main

$ get remote add origin https://github.com/ilseob-han/gitTest20240304.git

$ git push

ls (list)– 현재 위치 파악 가능하다.

$ git branch /브랜치 확인

$ git branch Test // 브랜치 추가

$ git switch Test //브랜치 이동 -> 초록색으로 변한게 내가 작업하는 브랜치다.이거 확인하는 습관들이면 좋다.

git add src/ //src안에 있는 모든 것이라는 의미.

Commit을 버릇처럼 해야 한다. 매우 중요.

단순하지만 가장 힘든 부분이다. 게으르다 이런게 아니라.

주변환경이 하나의 종결을 만들지 못하고 계속 방해가 된다. ㅊㅇ

// fatal: The current branch Test has no upstream branch.

To push the current branch and set the remote as upstream, use

git push --set-upstream origin Test

To have this happen automatically for branches without a tracking

upstream, see 'push.autoSetupRemote' in 'git help config'. //

$ git branch - d Test

git은 습관이 되어야 한다. 누가 Push하면 한시간내에 pull을 해야 한다.

복사는 좋은 습관이다.

타이핑 안좋다.

푸쉬하면 푸쉬했다고 동료에게 공지를 해야 한다.

알림설정 대부분 안하기 때문에-

강제종료:

1. Esc

2. :q!

git clone https://github.com/ilseob-han/Test20240304.git //

질의 : 왜 ilseob 브랜치 만들어 졌는데, 왜 깃허브에 업데이트가 안되었지?

Push 에서

$ git push -u origin main

$ git branch -M main

를 하면 강제로 merge가 되는건지? 이 -u와 -M이

브랜치에 미치는 영향은 어떻게 되는건지?

폴더지정

Echo “metro” < a.txt

First에 푸쉬하고

Second 브랜치 만들고

Echo “bus” > a.txt

퍼스트 먼저 머지 진행 :

Git 프로젝트 생성 > 세팅 > 콜라보레이터 에서 이메일로 사람 추가

 추가된 사람이 깃홈페이지에서 accept

$ git init

$ pwd

$ cd ..

$ ls

$ git init

2024-03-05 자바 연수

스크린타임 암호 1597

Java(OOP)

OOP – Object – 사람이 정의할 수 잇는 것 모두. 단, 너무 추상적인건 안된다. -> 문법적으로 class로 표현한다. Class = object = instance

OOP의 4가지 특징

– 시험문제 꼭 낼거다.

1. 추상화 – 추상적인 것을 구체화 하는 것을 추상화라고 한다. 이것이 설계 파트다.

2. 캡슐화 – Data의 보호/은닉. 데이터의 흐름이 명확하다. 캡슐은 알갱이를 보호하기 위해서다. 그리고 알갱이는 데이터다. 데이터를 누가 썼는지를 확인하기 쉽다. 캡슐을 제거한 사람이니까.

3. 상속 – 오브젝트를 물려 받는다. 확장성이 좋다. 유지보수성이 좋아진다. 관계설정이 중요하다.

4. 다형성 – 편하려고 쓰는거다. 안쓰면 코드 길어지고, 가독성 떨어진다. 하나로 여러 개를 만족할 수 있다. 오버로딩, 오버라이딩, 동적바인딩

자바 설치

Class path

Jave\_home

Path 설정

자바는 썬사의 것이었다. 처음 만든 목적은 임베디드였다. 사물에 들어가는 프로그램 목적이었다. 자동차, 냉장고 이런거. 처음에는 임베디드라는 시장이 없었다. 그러다 웹페이지 나오면서 – 인터넷뱅킹, // 웹에서 자바가 엄청 떠버렸다. 처음부터 임베디드 목적이므로 OS에서 자유롭다. 자바는 느리지만, PC속도 빨라지며 크게 성장. 자바 무료이고, 임베디드여서 안드로이드가 자바로 만들어졌다. 지금은 자바 세상이다.

클래스는 java가 기본 확장자

자바 접근제어자 4개 -> 외부에서 접근할 때/

Public -> 모두 접근 가능하기 때문에 데이터는 public으로 하는 것을 항상 고민해야 한다.

Private – 외부 접근 불가

Default – 같은 패키지 안에서만 사용할 수 있다.

Package

Protect – 내부접근 가능 / 외부 패키지 중 상속받은 친구만 접근 가능

표준함수 : main

사용자 정의 함수 –

System.out.println("Hello..."); 여기서 .. 이 패키지다. 주소라고 보는 거다.

이름 짓는 규칙 :

1. 클래스 첫글자는 대문자로 한다.

2. 메쏘드, 변수는 소문자로 작성한다. 변수란 상수를 담는 메모리 공간

3. 상수 : 전부 대문자로 쓴다. MAX -> 변하지 않는 값

4. 단어와 단어가 이어질 때 두번째 단어의 첫글자는 대문자로 작성해라. blackWhite, BlackWhite

DataType

1. Datatype

A. 문자형 char 2byte

B. 정수형 byte short int long -> 값의 범위

2. Reference Type : 레퍼런스 변수를 사용한다.

A. 배열

B. 객체

C. enum

증감연산자

A++ / ++A 단독으로 쓰면 차이가 없다.

B= ++A

B= A++ 두개 차이 있으므로 조심.

A= A+4

어떻게 하지? - > A+=4 -> 복합할당 연산자.

A = A/4 A/=4 => 이렇게 표현 가능하다.

A<<1 left shift operator \*2

A>>1 right shift operator /2 가 된다.

2<<1 =>4

2>>1 => 1

구름이라는 사이트가 있다 – 초보자에게 적합함

자바 기본 문법부터 문제가 있다.

프로그래머스 난이도 높다.

비트연산자

Casting

D. 자동형변환- 3+ 4.2 => 이건 에러다. Int + double 에러다. 서로 다른 타입은 연산이 불가하다. 그런데 막상하면 에러가 안난다. 3.0 + 4.2로 casting이 일어나서 7.2가 나온다.

E. 강제형변환 3/4 = 0 => int/int => 0이다. 이 경우 (double)3/(double)4 = 0.75

메모리

Stack Heap 두가지 영역이 있다.

Stack : hello는 여기 잡힌다.

Heap :

/\*

\* 자바는 100% 상속구조

\* 자바는 100% 객체지향 언어이다. -> 무조건 객체를 사용 -> Class

\* 시작과 끝을 담당하는 함수 => main함수

\* 함수 => function => 자바에서는 method라고 사용한다. 실체는 동일하다.

\*/

public class Hello {

public static void main(String []args) {

//프로그램을 시작해주고 끝내준다. main이 없으면 시작과 끝을 할 수 없다.

//프로그램 전체에는 main이 하나만 들어가야 한다.

//static -> 공유를 목적으로 한다. 딱 하나만 할때. 그리고 메모리 생성되는 시점이 다르다. static은 구동하자마자 메모리 먹는다.

//여기서 Hello, String, System -> 대문자로 시작하는 클래스다.

//datatype 변수명 ;

//변수명 = 값; -> 이것을 초기화라고 한다.

//변수 - 지역변수 local variable : method안에서 선언되어지는 변수. !항상 꼭 초기화 해야 한다.

// ; -> 문장의 끝에 무조건 붙인다.

int num;

num = 10;

char ch = 'A'; //"" 문자열

System.out.println(num);

System.out.println((int)ch);

}

}

문장 입력 후 자동완성: System.out.println을 작성하려는 위치에 커서를 놓고 syso라고 입력한 후 Ctrl + Space를 누르면 자동완성 기능이 활성화됩니다.

Day2

Continue; -> 다시 for문 최초로 이동을 한다.

if(a==3) continue; -> 3일때는 동작하기 원치 않다.

Do { } while(조건식);

조건이 True일때 do로 올라가서 수행한다.

Function : Method - 기능

사용자 정의 함수 (기능) -> 분업화가 목적이다. 즉 모듈화.

모듈화 방식

1. call by name

2. call by value - 메쏘드와 메쏘드 간에 값을 주고 받는 것.

3. call by reference -

함수 만드는 방법

리턴타입 method이름 ( ) { }

변수 받는거 여러개 받을 수 있지만,

보내는 건 하나만 보낼 수 있다.

인스턴스 메소드instance method - 객체 생성후 사용할 수 있는 메쏘드

스태틱 메소드 static method- 객체 생성 유무와 상관없이 사용할 수 있음.

자바는 함수 잘게 쪼개서 만드는게 좋다.

자바의 정석:

참조 변수는 1개의 객체 주소만 저장 가능하다.

대부분 인스턴스 메쏘드를 만든다.

Array

=======

String-> 객체이다. 문법적으로 기본 타입처럼 쓸 수 있게 해주었다.

String 특징 - 문자열은 기존의 값을 바꿀수 없다. 넣으면 끝이다.

=======

배열 - Array - int [] arr;

1. \*\*동일한 dataType을 메모리 상에 \*\*순차적으로 확보

2.

대부분 루프와 함께 사용한다.

===

int[][] arr2 = new int[2][3]

==

해외 IT 는 정말 빠르다.

우리나라 매너리즘 빠지는 경우가 많은데, IT는 그러면 안되는데,

그러다 치킨 집을 차릴 수도 있다

자기 계발 꾸준히 하면, 평생 먹고 사는데 지장이 없다.

오프라인 인맥을 잘 쌓아라.

커피한잔이라도 보내고

얘기한번 하고, 밥도 한번 먹고

60살 넘어서 일하는 사람도 이쏙,

시간이 지나면, 포기만 안하면 하게 된다.

IT는 솔직히 개발자로 시작을 꼭 해야 한다.

길게 보면, IT는 좋다. 다른 직원에 비해서.

IT는 결혼하고, 양육하면서도 프리랜서로 하는데 어려움이 없다.

남자는 2job 3job도 많이 한다.

본인이 노력하는게 좋고,

전문가 인맥을 쌓는게 중요하다.

그러면 스카웃 제안이 많이 온다.

대기업 다니는 애들 다 중소기업 갔다 이직도 많이 했다.

경제적으로는 다른 분야보다 좋긴하다.



이 예제를 생각해보자.

일단 Static으로 설정한 값을 다른 클래스에서 변경할 수 있었으며,

이 경우 객체를 지정하고 static변수를 불러와도 다른 클래스에서 변경된 변수가 반영됨.

아울러 c1, c2에서 불럿을 때 동일하게 변경된 것이 보임. 즉 static변수는 어디서든 변경이 가능하며,

변경이 가능하면 그것을 참조하는 모든 객체에도 영향을 미친다는 의미임.

책 248페이지 참고하면 Card.width, c1.width, c2.width는 모두 같은 값을 가짐.

이건 값을 class변수라고 하는데 이 경우 Card.width로 표현하는게 좋음.

===

3일차 클래스Class

1. Field : -> 외부에서 접근
   1. 객체가 생성된 후 지속적으로 사용되어지는 변수
   2. 접근지정자 주로 private으로 지정
   3. Static, Final(읽기 전용) 쓸 수 있다. -> final은 static과 같이 쓴다.
2. Constructor(생성자) :
   1. 메소드인데, 굳이 생성자를 따로 만들었다. 이는 생성과 관련된 것.
   2. 객체 생성시 자동 호출되어지는 함수(딱 1번)
   3. ClassExam ce = new ClaseExam() => 여기 어디선가 자동으로 불러왔다. 5개의 함수만 있어도, 실제로 6개가 있는 것이다.
   4. 생성자 함수를 명시적으로 제공하지 않을 경우 디폴트 생성자를 제공.
   5. 리턴 타입이 존재하지 않고, 함수명은 클래스 명과 동일하다.
   6. 기능 :
      1. 디폴트 기능 : 객체등록
      2. 사용자 기능 : 객체 생성시 필드초기화
   7. 오버로딩이 가능하다.
   8. 접근지정자를 주로 public으로 지정
   9. 디폴트생성자를 명시적으로 만들 경우, 숨어있는 디폴트는 제공하지 않는다.
   10. 명시적으로 생성자를 하나 이상 만들 경우 디폴트는 제공하지 않는다.
   11. 생성자 호출 중에 (끝내기 전에) 생성자를 또 호출할 수 있다.
3. Method
   1. 외부에서 내부에 필드를 접근해서 사용할 수 있게 하기 위해.
   2. 접근지정자를 주로 public으로 지정
   3. Instance 메소드, Static 메소드 : 문법적인 가장 큰 차이점은 this 여부. -> this가 객체를 구분해 준다. instance에서 100% 있다.
   4. 오버로딩이 가능하다. -> 다형성 중의 하나다. 한가지 이름으로 여러 번 쓸 수 있다.
   5. 함수는 함수가 부른다 단 main만 os가 부른다.
   6. Main은 내부이지만 외부라고 생각하고 코딩을 해야 한다. 그래야 외부에서 실제 쓸때도 문제가 안생긴다.

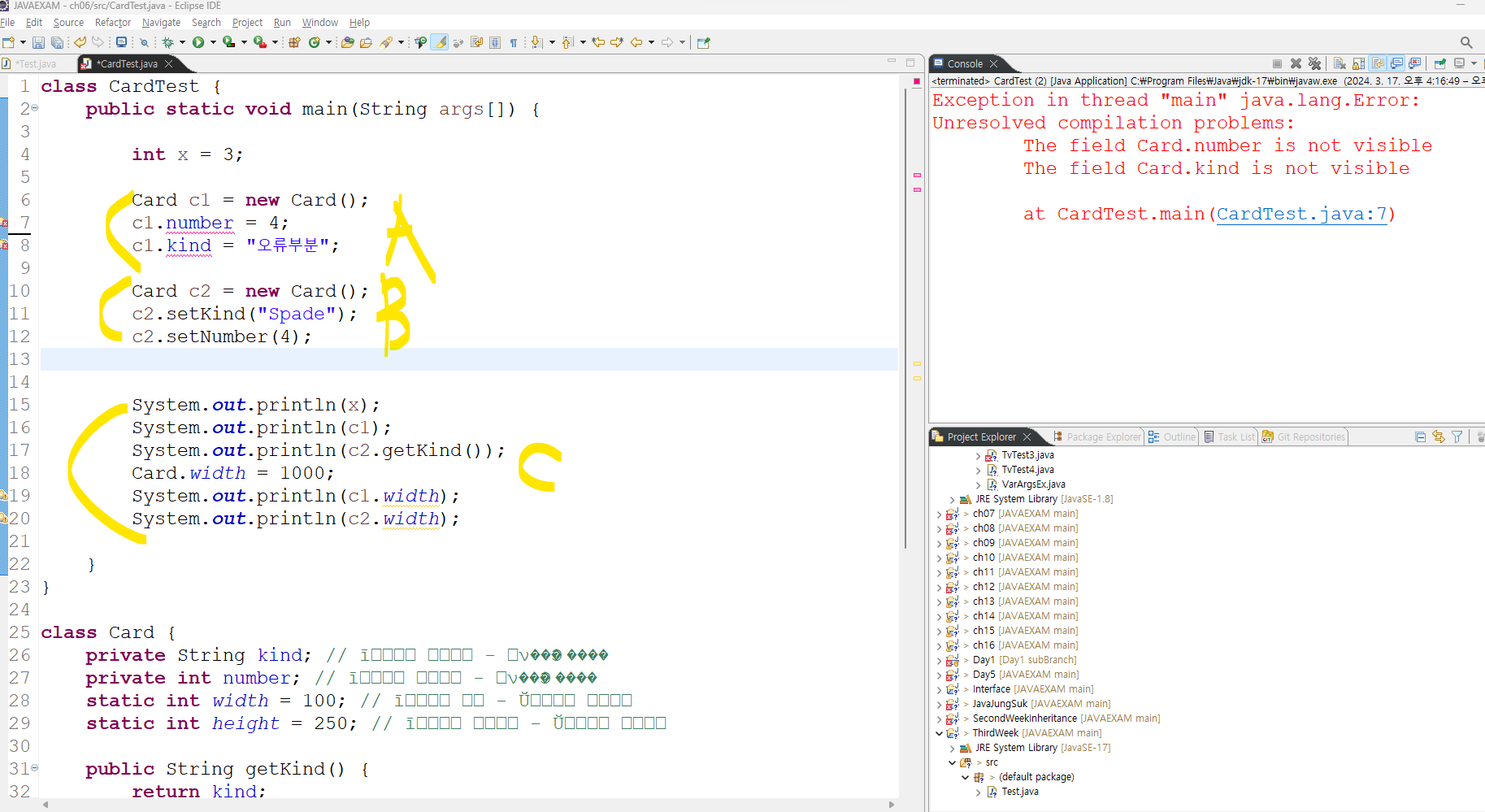
오버로딩:

1. 동일한 함수 명으로 여러 개의 함수를 만들어서 제공하는 것- 이게 오버로딩.
2. 같은 기능을 수행하는 것을 하나의 이름으로 묶어서 제공함으로써 사용자가 편하게 쓸 수 있게 해주는 기능.
3. 같은 성질을 지닌 것끼리 묶어야 한다.
4. 함수명을 동일하게 하면서 매개변수(argument)를 구분할 수 있어야 한다.
   1. 매개변수로 구분하는 방법 2가지.
   2. 매개변수 타입이 다를 경우
   3. 단, 리턴타입이 다를 경우는 안된다.

Method Setter / getter 메소드

Setter : 외부에서 데이터를 입력하는 함수 –

Getter: 외부에서 필드 값을 얻어가는 함수



A 부분 에러난 것은 private instance변수에 접근하려고 했기 때문이다.

이에 B부분처럼 게터와 세터를 사용해야 한다.

C부분을 보면 c1.number로는 원하는 값을 얻을 수 없으며,

Getter를 사용해야만 하는 것을 알 수 있다.

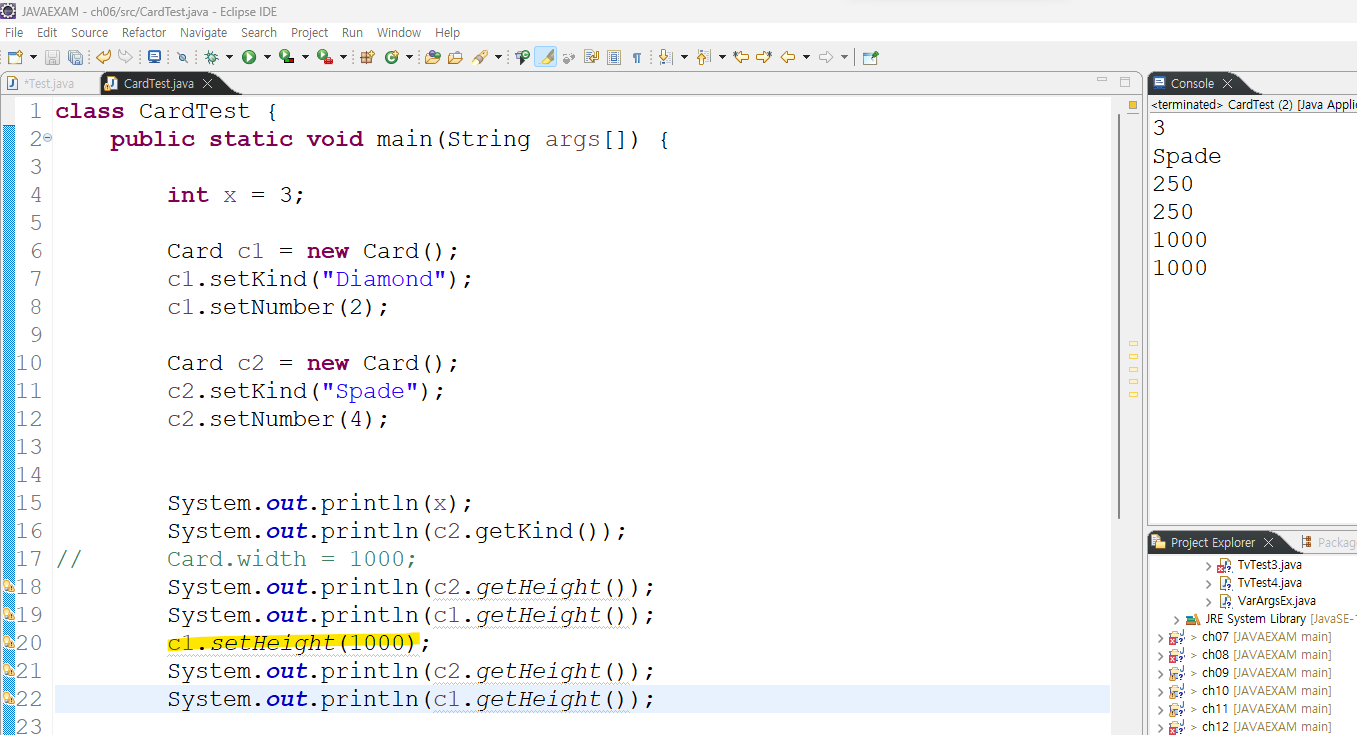
아울러 static 변수는 그냥 이전 방식으로 변경이 가능한데,

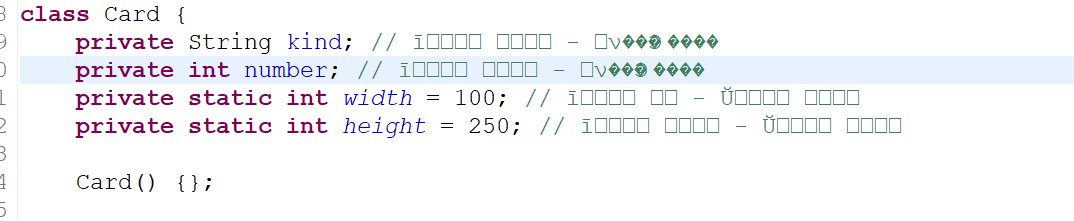
static변수도 private으로 변경이 가능했다. 아래 처럼.

이 경우도 세터게터를 설정해야 했으며,

세터로 변경을 하면 해당클래스 변수자체가 변경되어,

모든 객체의 값이 동일하게 변경이 되는 것을 확인 할 수 있었다.





여기보면 Card() {} 가 있다. 내가 생성자를 만들어 준 것이다. 생성자는 조금 더 공부할 필요가 있다,.

세터가 필요한 이유:

객체 지향 언어인 자바에서는 클래스간 캡슐화가 중요한데,

이는 보안 또는 코드 유지보수를 위해 반드시 필요한 요소이다.

클래스에는 필드를 통해서 멤버 변수를 제공하는데,

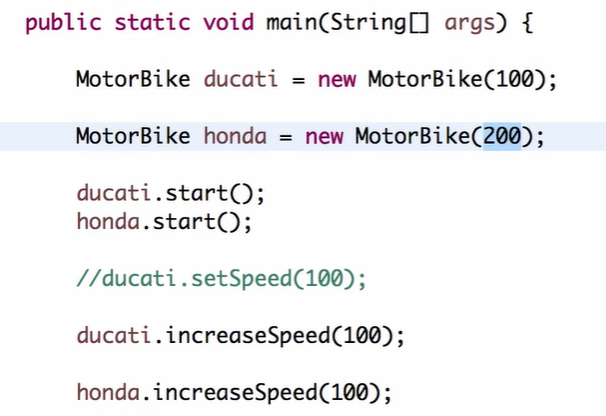
이 멤버변수를 세터를 통해서만 접근함으로써 캡슐화를 유지할 수 있다.

게터가 필요한 이유:

사용자는 게터를 통해 멤버변수를 제공 받을 수 있는데,

문법과 상관은 없지만 이를 통해 데이터의 통제 및

표준적인 방식으로 클래스에서 멤버 변수를 제공 할 수 있다.



This – instance 메소드의 첫번째 매개변수로 항상 존재한다.

Static 메소드에는 this가 없다.

This란 무엇일까? 자기 자신의 객체를 접근하는 Reference 변수다. heap영역에 접근하는 변수다.

인스턴스 메소드의 첫번째 매개 변수로 항상 존재한다. 선언할 수는 없고, 사용만 가능하다.

객체를 구별해주는 역할을 한다. -> 디폴트 기능.

this어떤목적으로 쓰는가? 2가지 목적으로만 쓴다. 예시 ThisExam01

1. 필드명과 지역변수 이름이 같을 경우 필드를 구별하기 위해
2. 인스턴스 메소드에서 자기 자신의 개체를 리턴하고 싶을 때

필드 변수 할 때 가급적 this 붙이는게 좋다.

This() : 생성자 호출하는 것

* + - * 자기 자신의 또다른 생성자를 호출하는 것

예시 ThisCall

==

Object를 문법적으로 표현한 것이 Class다.

명사는 field

동사는 메소드 로하면 90% 맞다.

추상화란 이렇게 자꾸 추출해야 한다.

객체는 항상 누가쓰든 쓸 수 있도록 독립적으로 만들어야 한다.

has관계와 is관계

has관계 – 객체가 객체를 가지는 것,

예를 들어 String Class – 100% has관계로만 쓸 수 있다.

레고 블록처러메 가져다 쓸 수 있다. Oop의 가장 큰 특징.

==

마커를 쓰려면 객체를 꼭 생성해야 한다.

내 안에서 직접 만드는 방법이 있고,

외부에서 만드는 방법이 있다.

객체 주입: DEPENDENCE INJECTION

외부에서 만들어서 내부에 넣는거

자바의 정석

|  |
| --- |
| Ch6-8  객체배열 == 참조변수 배열    참조변수 배열 == 객체배열 |
| Ch6-9  클래스 = 설계도, 데이터+함수, 사용자 정의타입  배열 : 같은 타입만,  구조체 : 다른 타입도 OK  구조체에서 더 발전한게 클래스다.  클래스 데이터와 함수를 같이 묶는다. 함수 어떤 작업, 계산을 한다. 작업은 데이터로 한다. 관계있는 데이터와 함수를 같이 묶는거다.  사용자정의 타입 : 원하는 타입을 직접 만들 수 있다. 예를 들어 시간: 시간은 자바에 기본형 8개에 다. |
| 6-11 선언위치에 따른 변수 종류  Instance variable, : iv 앞에 static이 붙으면 class variable이 된다.  class variable,  local variable  객체는 iv를 묶어 놓은 것. |

상속 – 2024-03-12

Super class -> Sub Class

수퍼클래스는 서브클래스 들의 공통점을 가지고 있다.

Is~a 관계 ; 상속관계 –

Extends : 상속을 표현한다. –

클래스는 단일상속만 된다, 인터페이스는 다중 상속을 허용한다.

객체끼리는 단일 상속만 허용한다.

상속을 받았다고 할지라도 private은 안된다. 경우 setter getter로 접근을 해야 한다.

부모의 내용을 수정/가공해서 제공하고자 할 경우

상속: 코드 반복된걸 줄일 수 있고, 확장이 유용한다.

상속은 있는 그대로만 된다.

//

Super, Super();

1. Super : 부모의 멤버를 사용하고자 할 때 쓴다. 오버라이딩된 멤버를 사용할 때 부모를 명시적으로 적용하기 위해 사용.
2. Super() : 부모의 생성자를 호출하는 것. 항상 sub class의 생성자 첫번째 라인에 항상 존재하고, 명시적으로 사용이 가능하나, 위치는 변경할 수 없다. -> 얘를 언제 쓸까?

오버라이딩 : 다형성의 하나다.

부모의 멤버 (메소드)와 똑 같은 형태로 제공.

부모의 멤버를 가공하고 싶을 때 – 오버라인딩을 쓴다.

오버라이딩은 동적 바인딩의 필수 요건이다.

SuperExam파일 참고

추상화/ 추상클래스

강제성 부여는 추상메서드를 통해서 할 수 있다. - > 수퍼클래스가 강제하는 것.

* + - * 추상메서드를 하나 이상 가진 클래스
      * 추상 클래스. 추상클래스는 객체 생성을 할 수 없고, 상속을 해주는 목적으로 사용.
      * 추상메소드는 서브클래스에 강제성을 부여(오버라이딩)
      * 오버라이딩을 하지 않을 경우 서브 클래스도 추상클래스화 되어서 객체를 생성할 수 없다.

추상클래스 사용 방법

* + - * 상속관계시 항상 공통된 것은 수퍼 클래스에 정의
      * 개념적으로는 수퍼에 있어야 하지만 기능적으로는 수퍼에 정의할 수 없을 때 만든다.
      * 상속받는 것이 하나라도 있어야 한다. 무조건 수퍼여야 하는 것이다.
      * 이나미머스 클래스 사용

동적 바인딩 : 다형성의 하나. 부모의 레퍼런스로 자식에 접근하는 거.

자료구조,

DB- > 급여, 자바랑 연동해서

이야기 하는 동작 다 구현해야 한다.

Final class -> String -> 더 이상 상속하지 않겠다는 클래스.

메소드에도 final을 붙일 수 있다. -> 오버라이딩이 안된다. / 슈퍼클래스 특정기능 바꾸기 원치않을 때- final 메소드

String은 그래서 HasA관계만 가능하다.

API에 있는 클래스 많이 보자. – 클래스 설계의 가장 좋은 예시다.

==

Super 호출: 서브클래스 생성자 첫줄에 항상 있다. 부모의 멤버 접근할 수 있다

오버라이딩 되었을 때 구분이 안되니까, super을 통해 구별한다.

인터페이스 -> 메뉴판 – 명세서와 같다. 이러이러한게 있다. 실제로 구현은 (주방) 즉 자손 클래스에서 한다.

메뉴판에 있는거 다 만들어서 줄 수 있다.

인터페이스는 여러 개 상속시킬 수 있다.

인터페이스도 동적 바인딩도 가능하다.

기술은 가르치면 되지만, 영어는 해서 와야 한다 – 삼성 임원 /

/\*

\* inner class : outer class의 멤버를 내것처럼 사용할 수 있게 해주는 클래스.

\* : 특정복적 : outer class 의 맞는 목적.

\* : 이벤트 처리할때 주로 사용.

\* 1. member inner class

\* 2. static inner class – ex. System.out. => 객체 생성안하고 접근할 수 있다.

\* 3. local inner class - 잘 안씀

\* 4. anonymous inner class - 엄청 많이 씀.

\* outer class

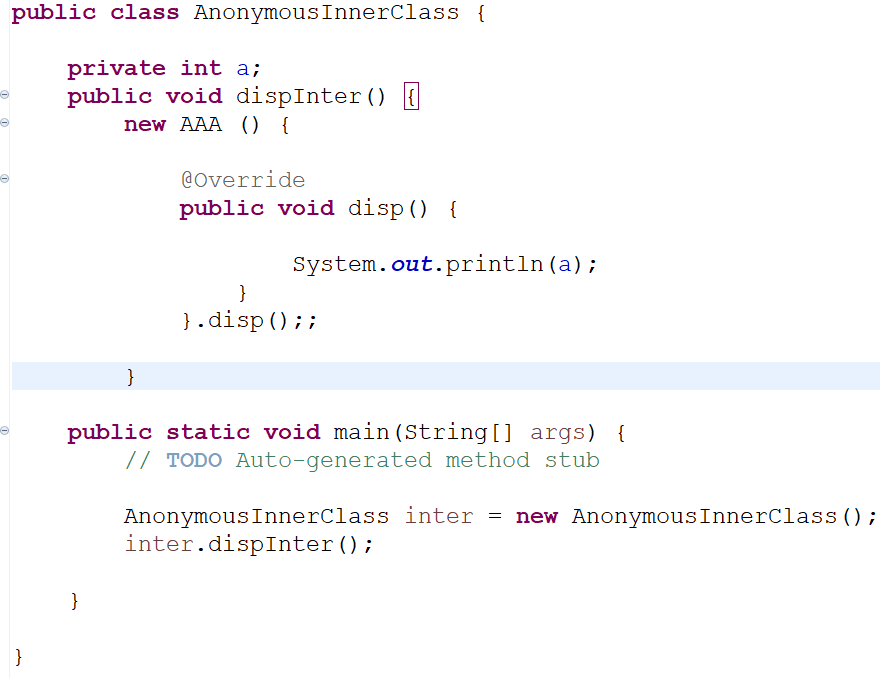
\* inner class

\*/

member inner class



AnonymousInnerClass :



예외 처리:

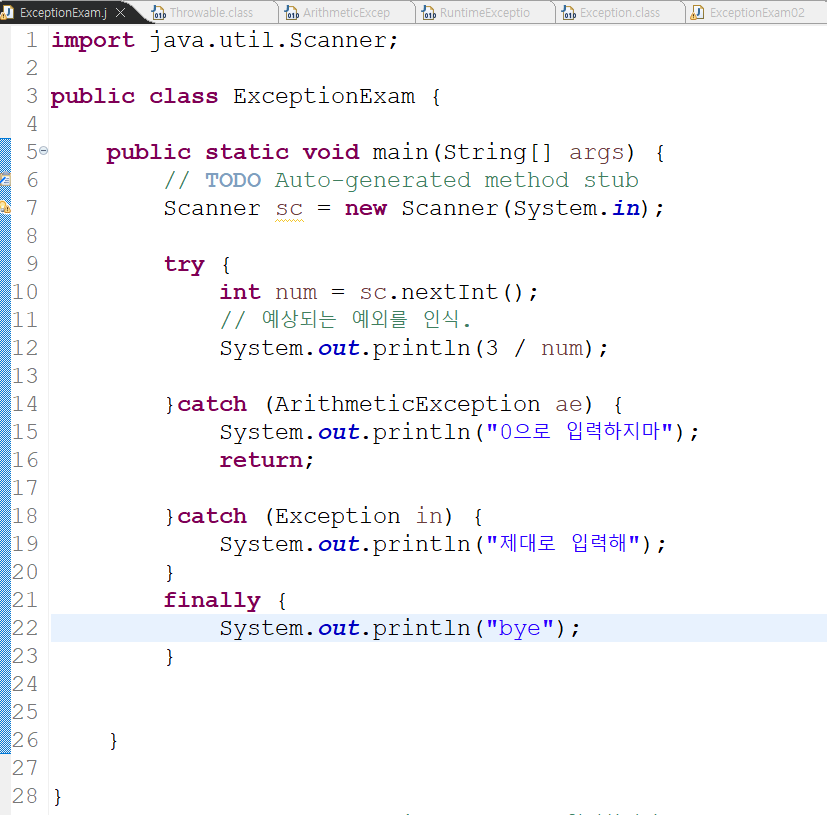
목적:

-정상적인 마무리를 수행하게 해주는 거.

-Try ~ catch문을 사용한다. -> 많이 안쓰는게 좋다.

Exception 의 경우 마지막에 있어야.

0으로 입력하지마 실행하고 더 포괄적인 exception을 입력해야 한다.

****

Finally – return으로 종료되더라도 무조건 실행되는 구문

예외처리로 강제종료되어도, DB 닫는 등 시스템 안정적으로 유지하도록 하는 구문

쓰로우즈 – 이거 필기 놓침.

황정선- 교수 이름

6-11 선언위치에 따른 변수 종류

Instance variable, : iv 앞에 static이 붙으면 class variable이 된다.

class variable,

local variable

객체는 iv를 묶어 놓은 것. - 프로그래밍 관점에서 보면.

클래스 변수: 클래스가 메모리에 올라갈때 만들어진다(하드디스크에서 로딩해서, 메모리에 올라갔을때)- -객체 생성 안해도 된다. 클래스 변수는 객체 생성이 필요 없다. IV는 객체 생성이 필요하다. CV는 아무때나 사용가능하다. 뭐 만들거나 할 필요없이 그냥 쓰면 된다. iv는 객체 생성을 해야 쓸 수 있다.

\*\*객체는 iv변수 묶음이다.\*\* 그냥 외워라

\*\*객체는 iv변수 묶음이다.\*\* 그냥 외워라

\*\*객체는 iv변수 묶음이다.\*\* 그냥 외워라

영역- 클래스 영역 IV, CV(Static붙은거)

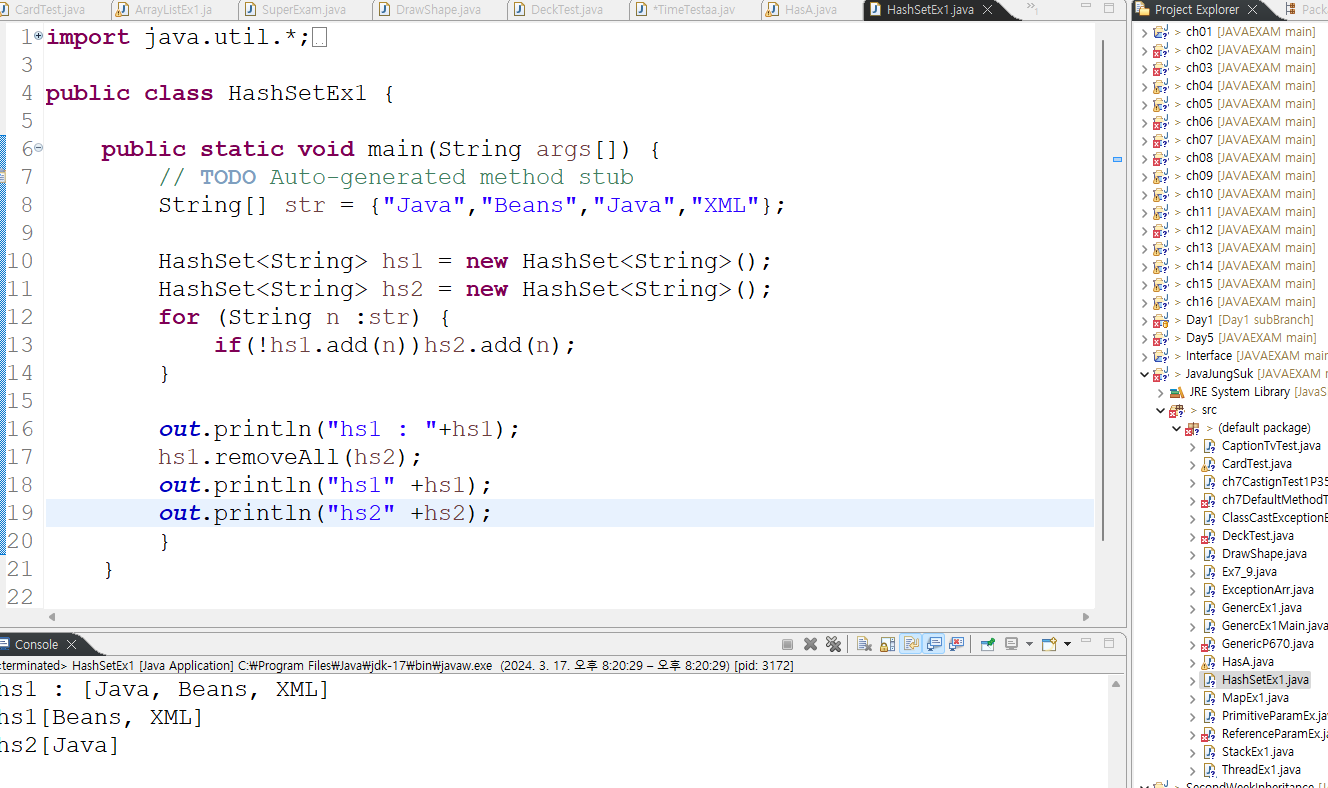
메소드 영역. LV

제네릭 –

실행시에 자료

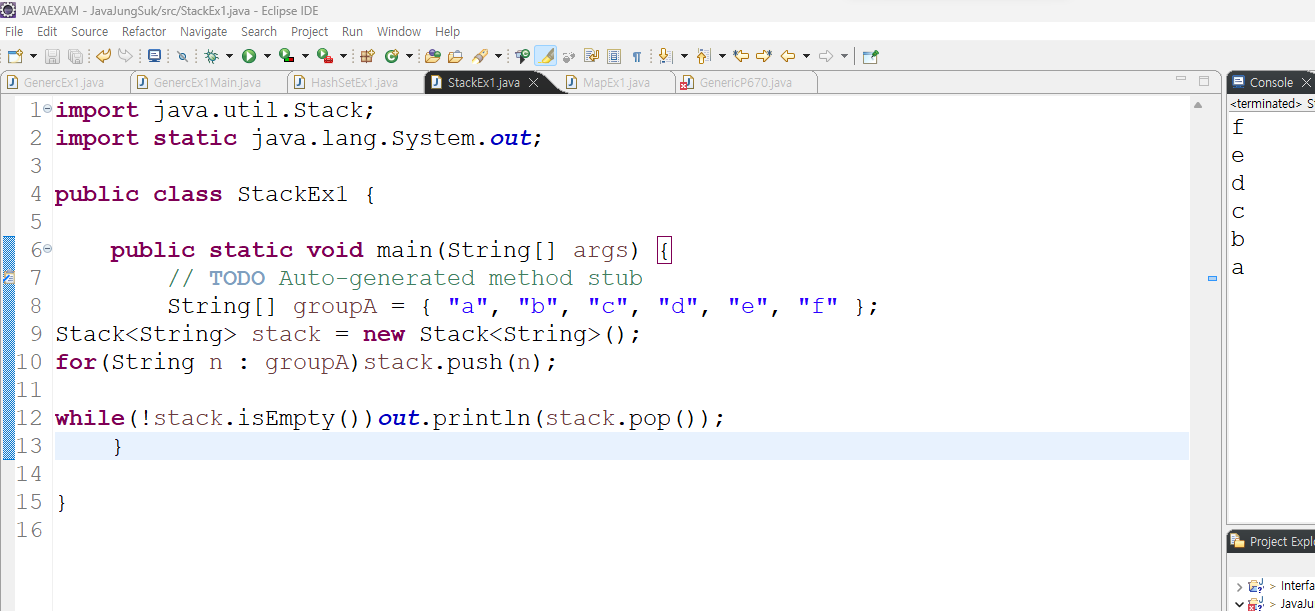
세트SET

* + - * 단점 : 리스트와 맵에 비해 속도가 엄청 느리다.
      * 하지만 중복 안되는 자료 관리할 때 편하다.



자바 컬렉션 프레임 워크 5

1. List –
   1. Stack
   2. Vector – 스레드 가능
   3. arraylist– 스레드 동기화 지원 않음 -> 가장 많이 사용한다.



자바 컬렉션 프레임워크9

1. linked list –

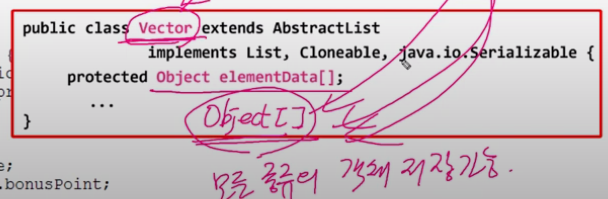
자바 컬렉션 프레임워크12 MAP

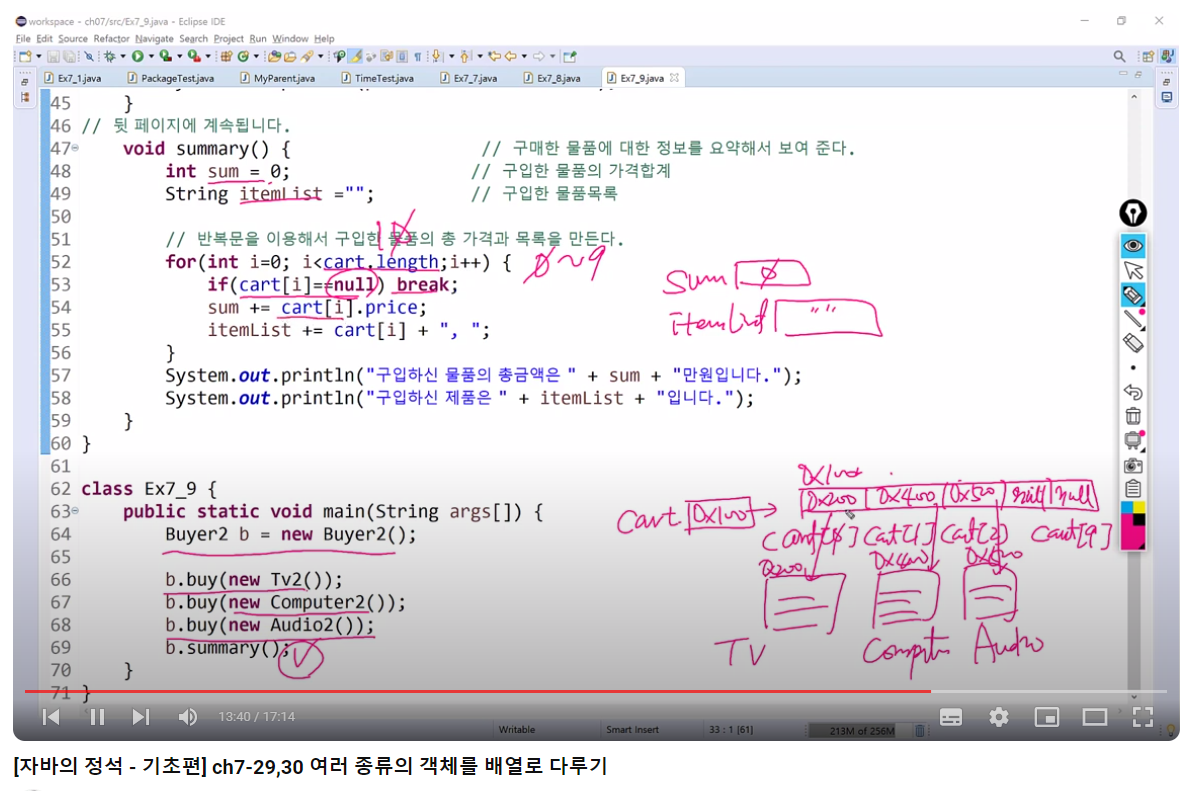
1. MAP – 파이썬 딕/셔너리와 동일. / 검색속도가 빠르다.
2. key값은 절대 중복될 수 없다.
3. 단점:



쓰레드 :

가변배열 모든 종류의 객체 저장 가능 – Vector Class





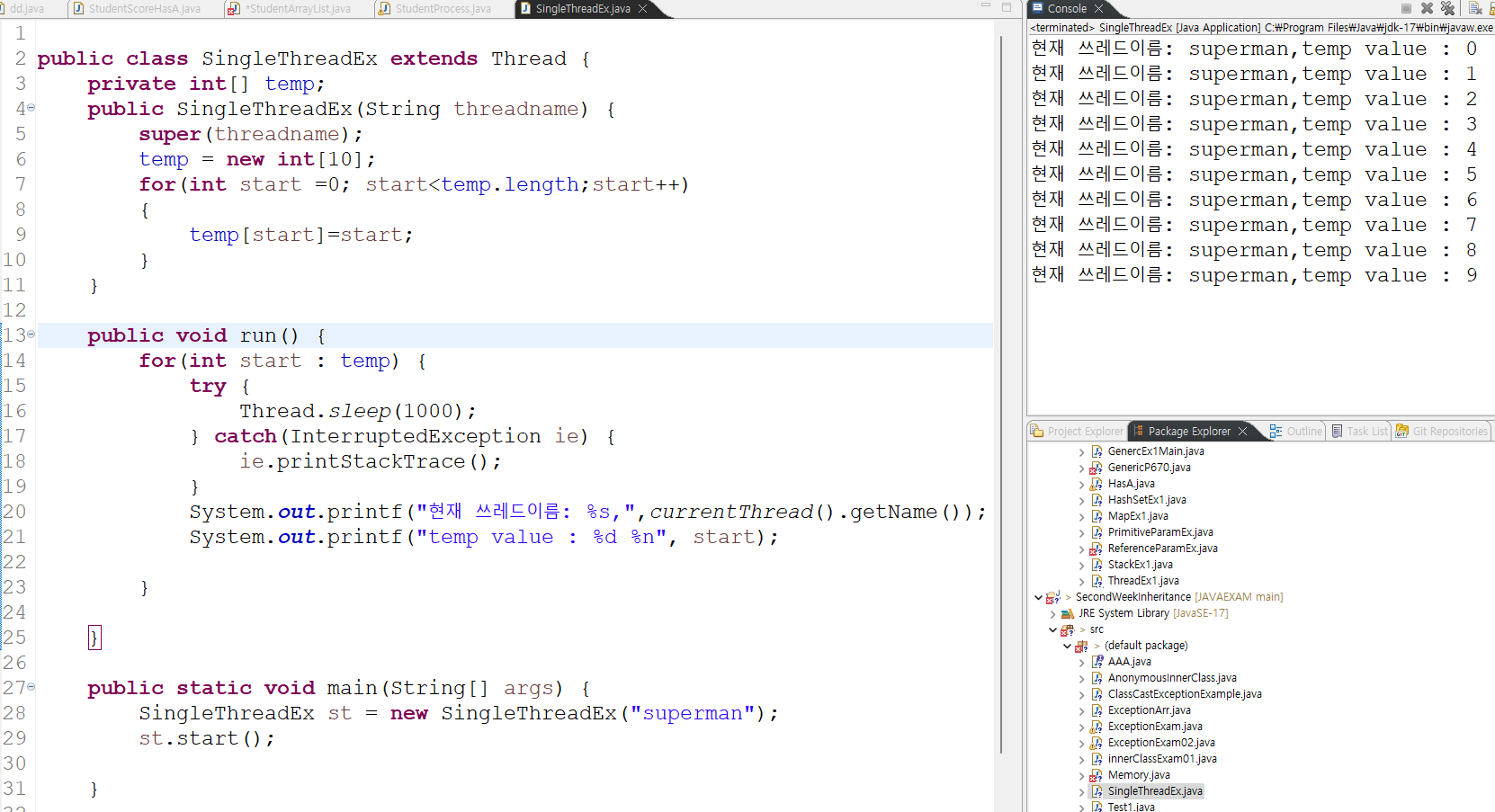
toString –

iterator –

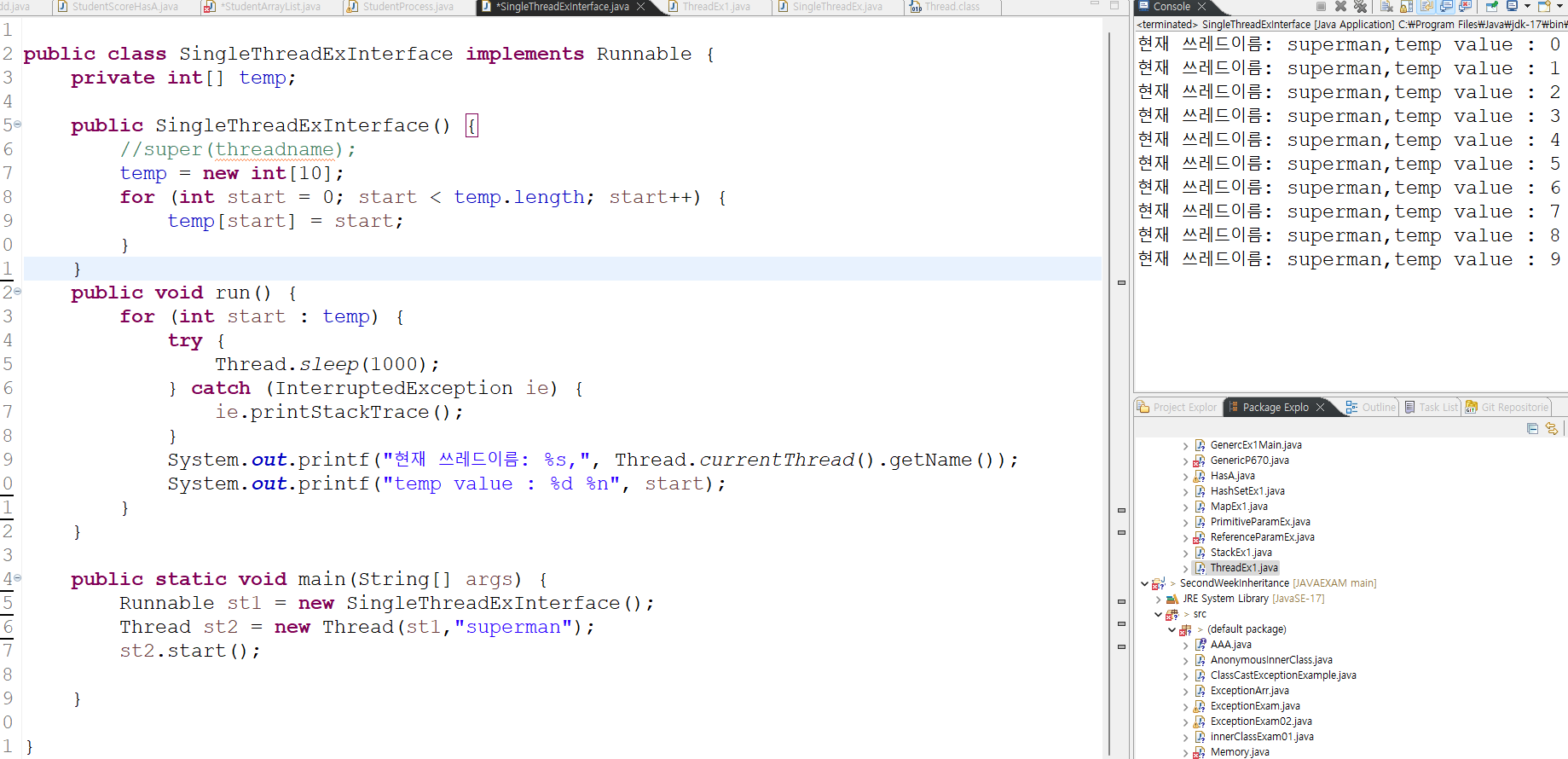
스레드 :

통신에서 가장 많이 쓴다. 예: 채팅 들어오면-

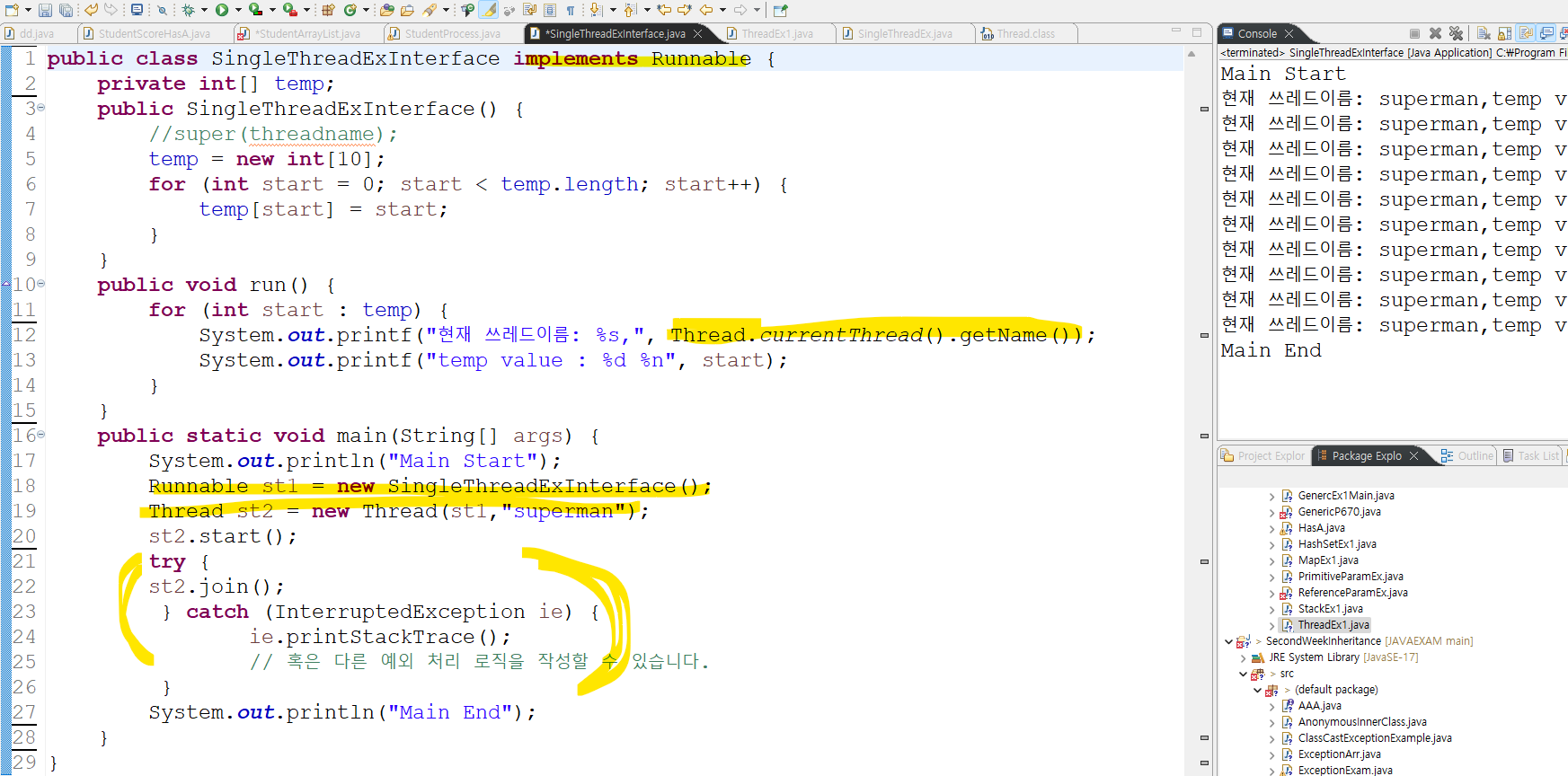
메소드를 활용한 쓰레드



임플러먼트 를 활용한 쓰레드



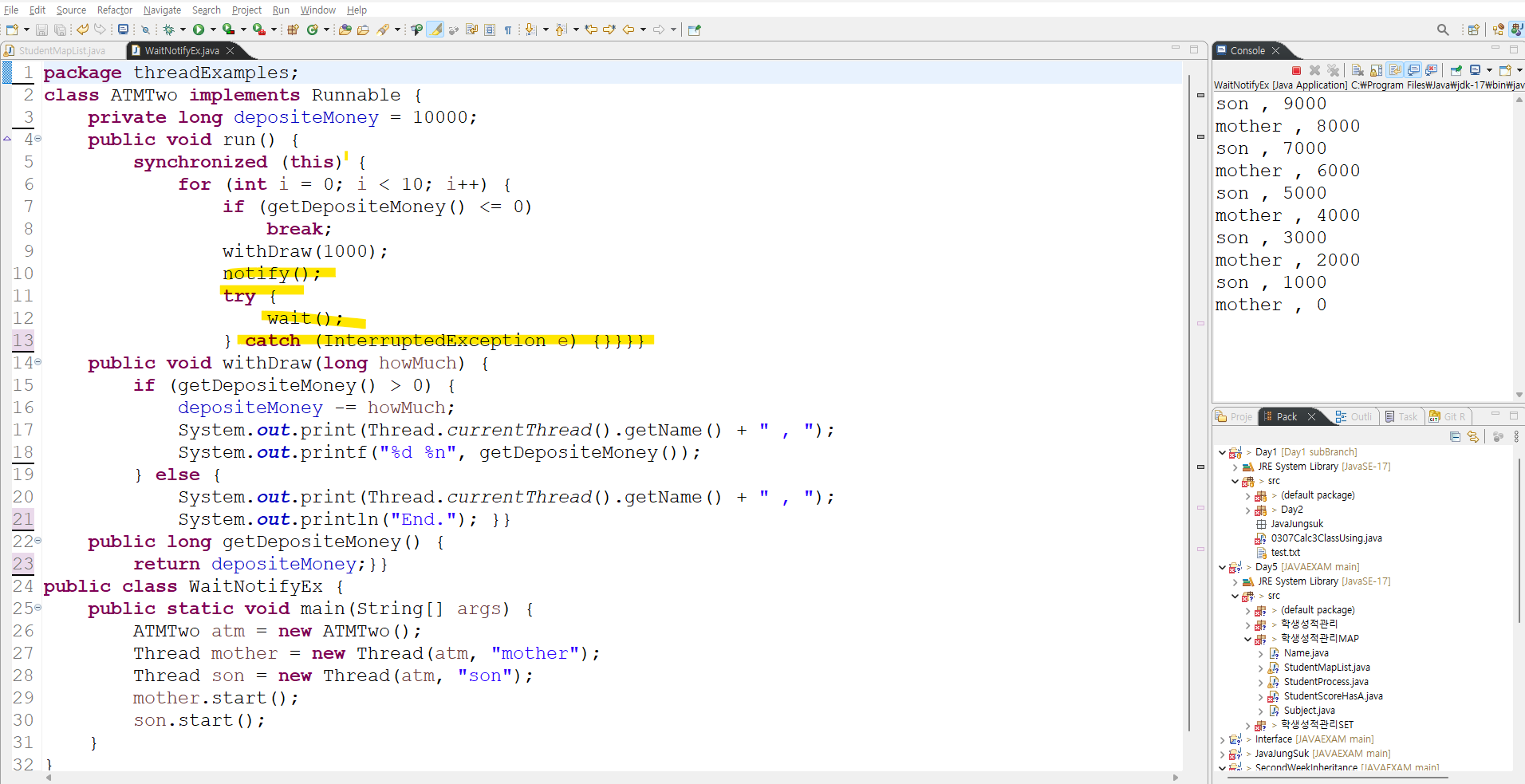
Join을 통한 쓰레드 속도 조절



동기화

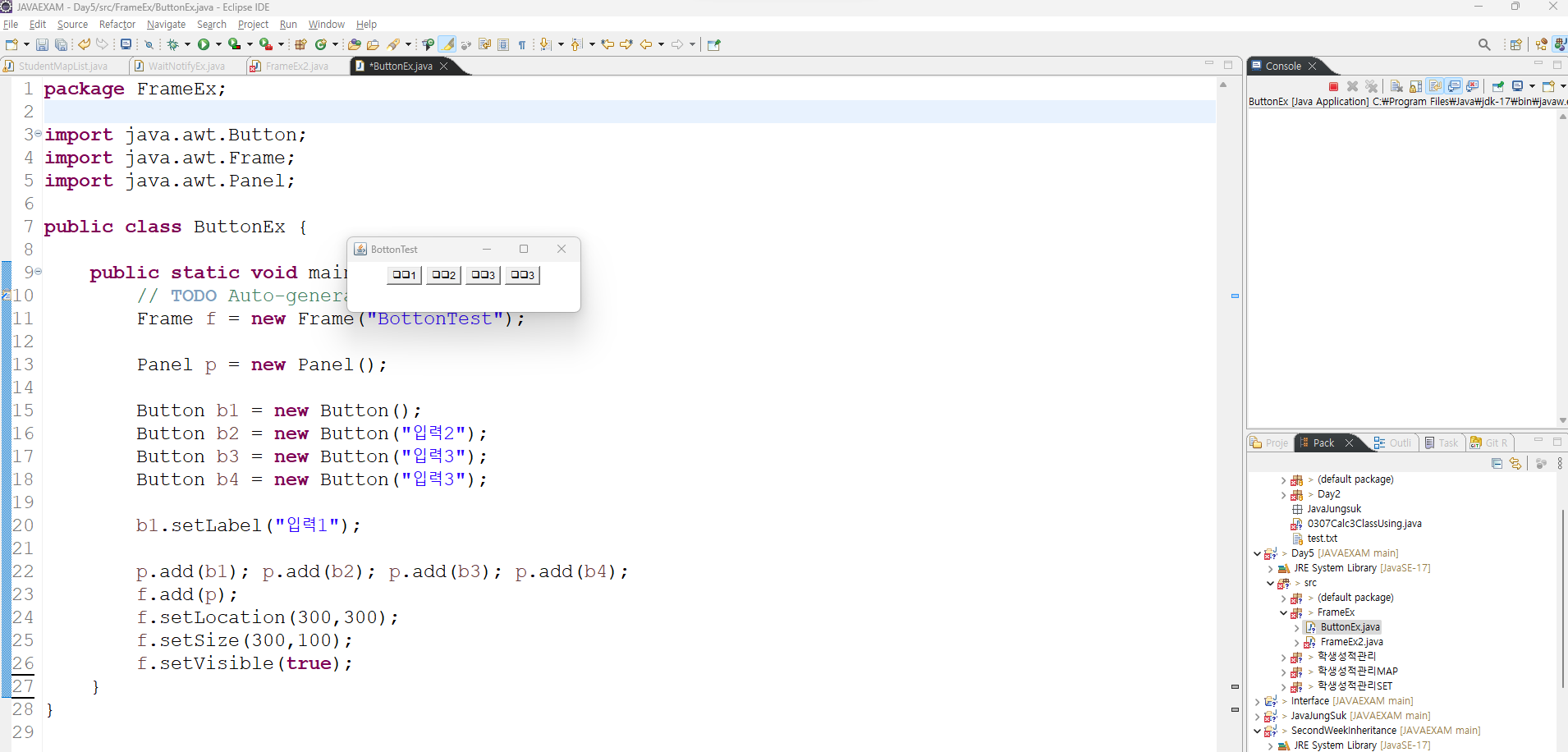
Notify

wait

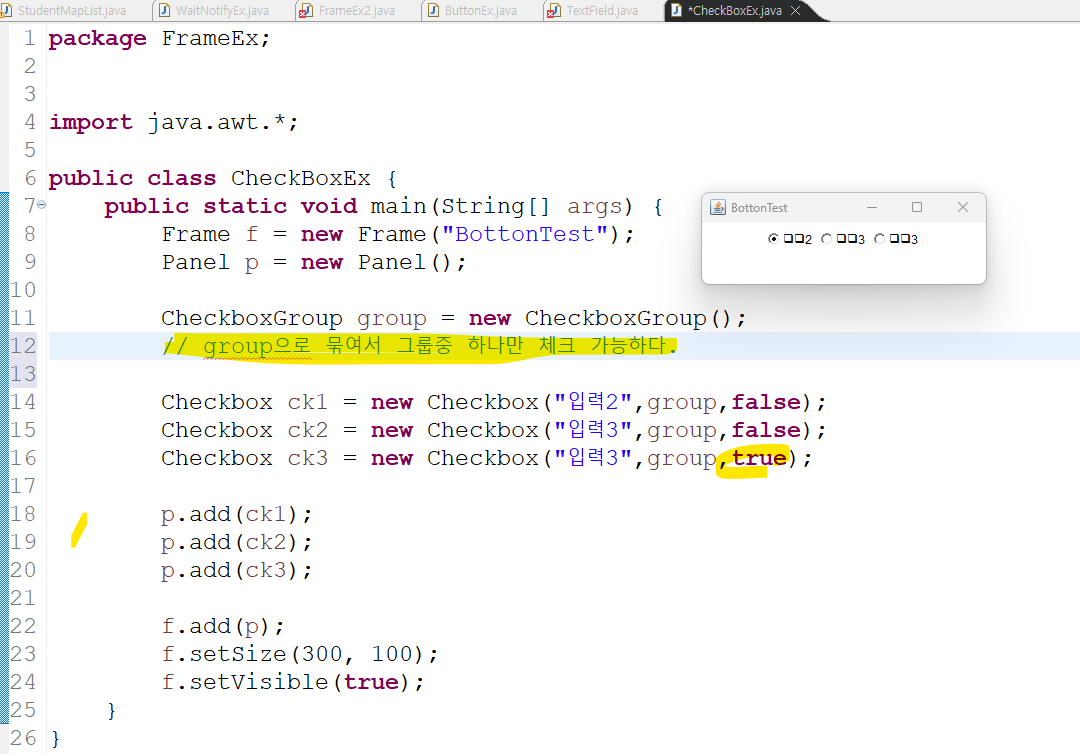


2024-03-15 그래픽

GUI



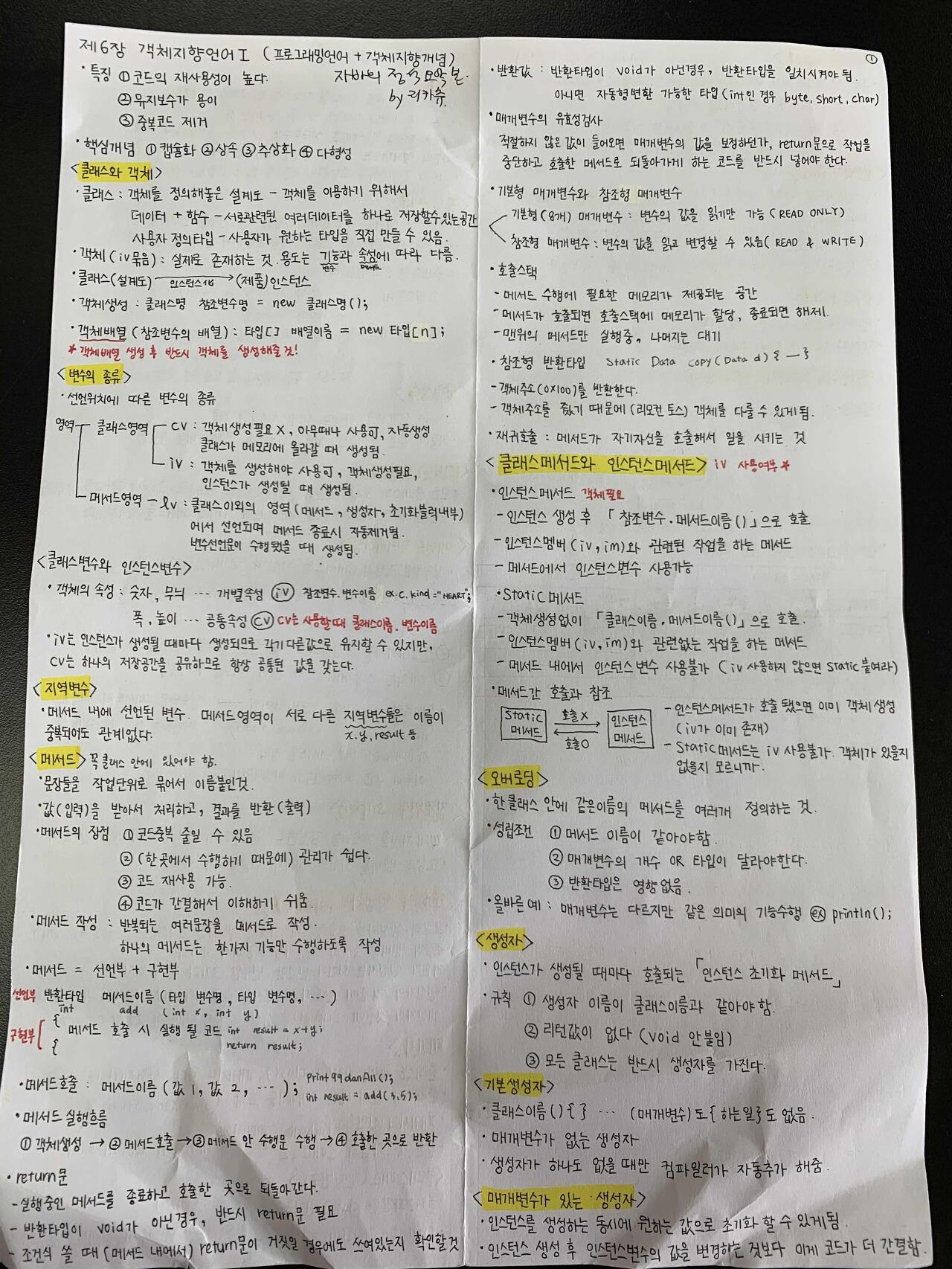
Check Box

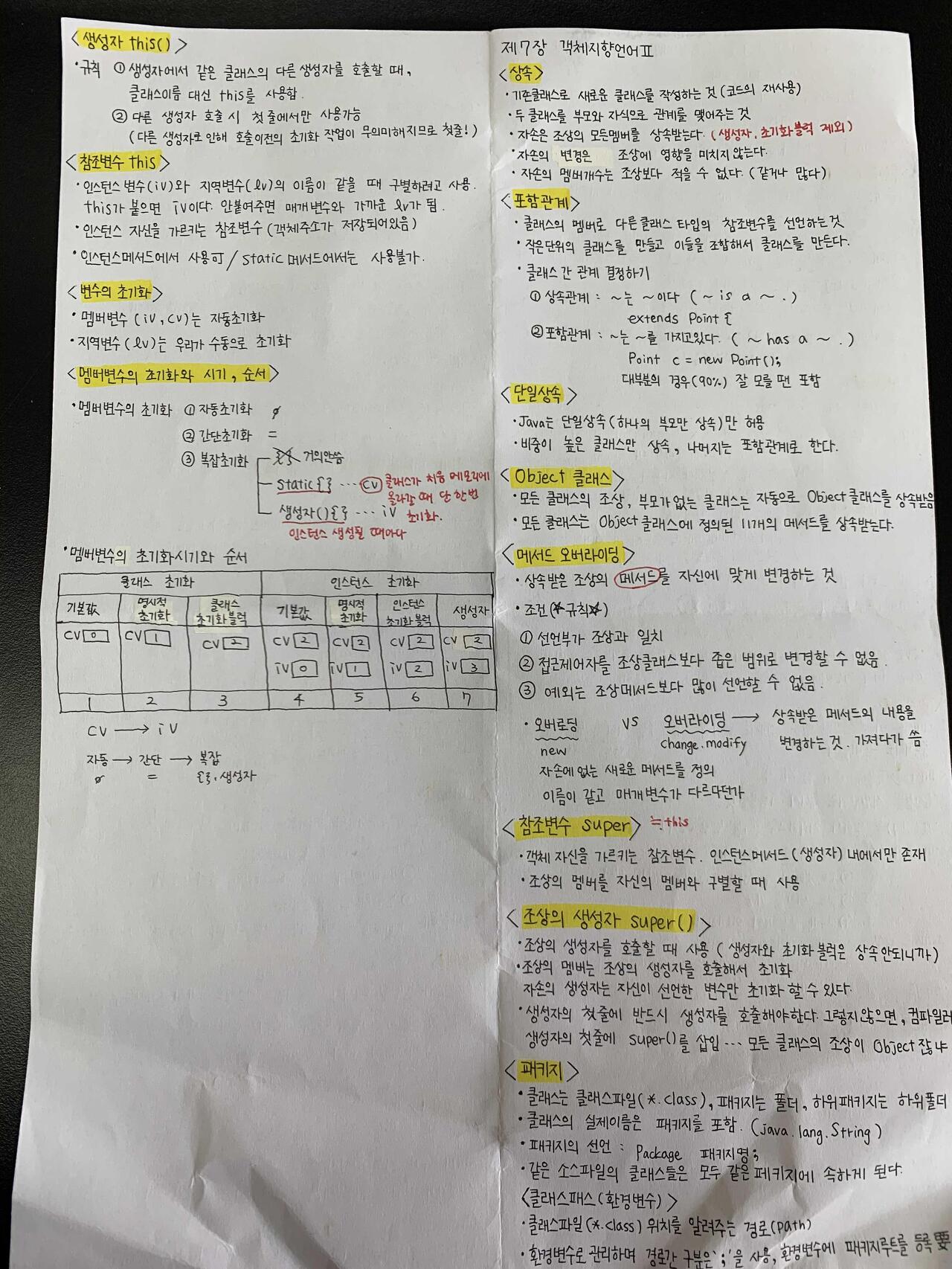


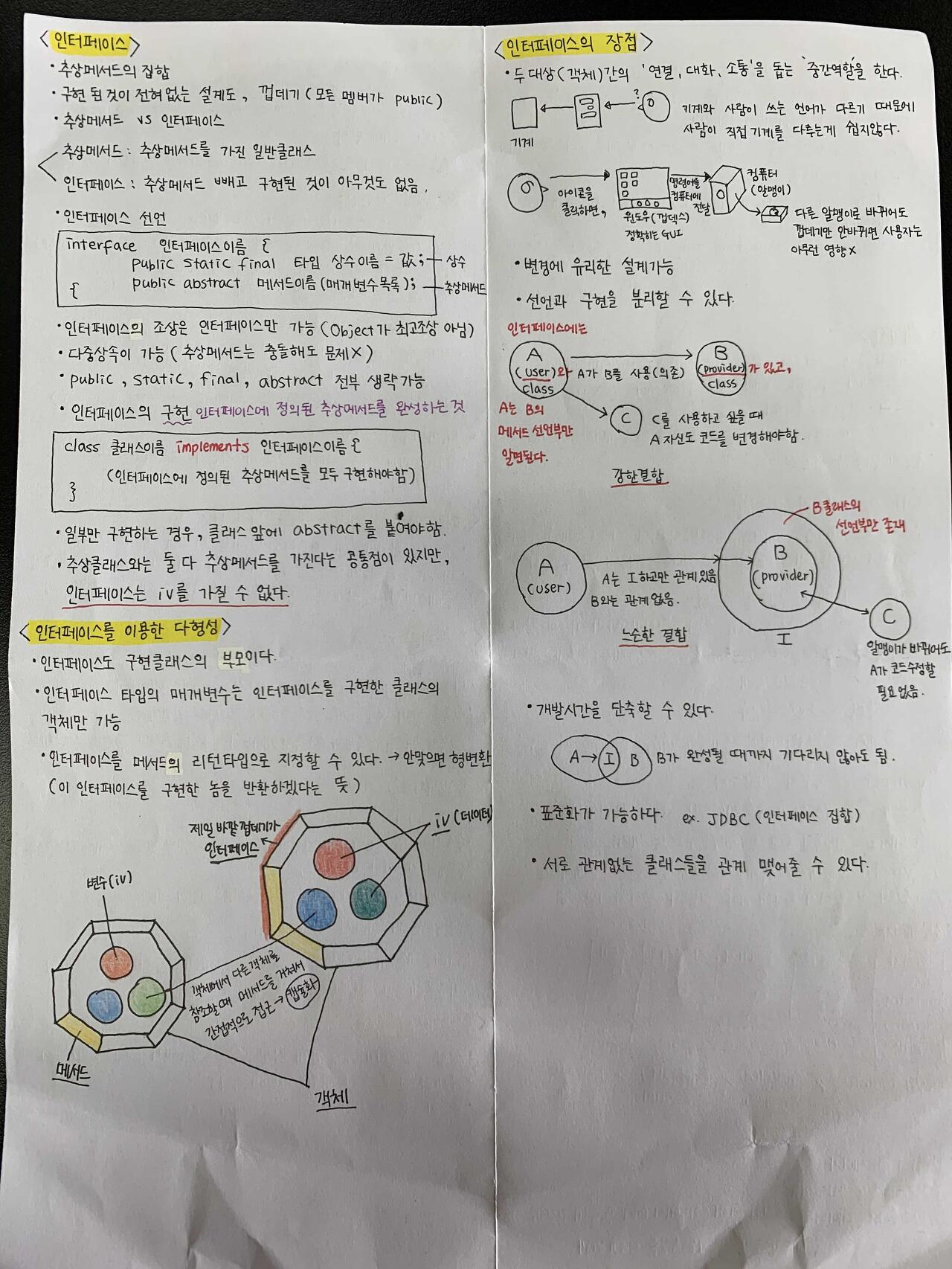
Layout Manager :

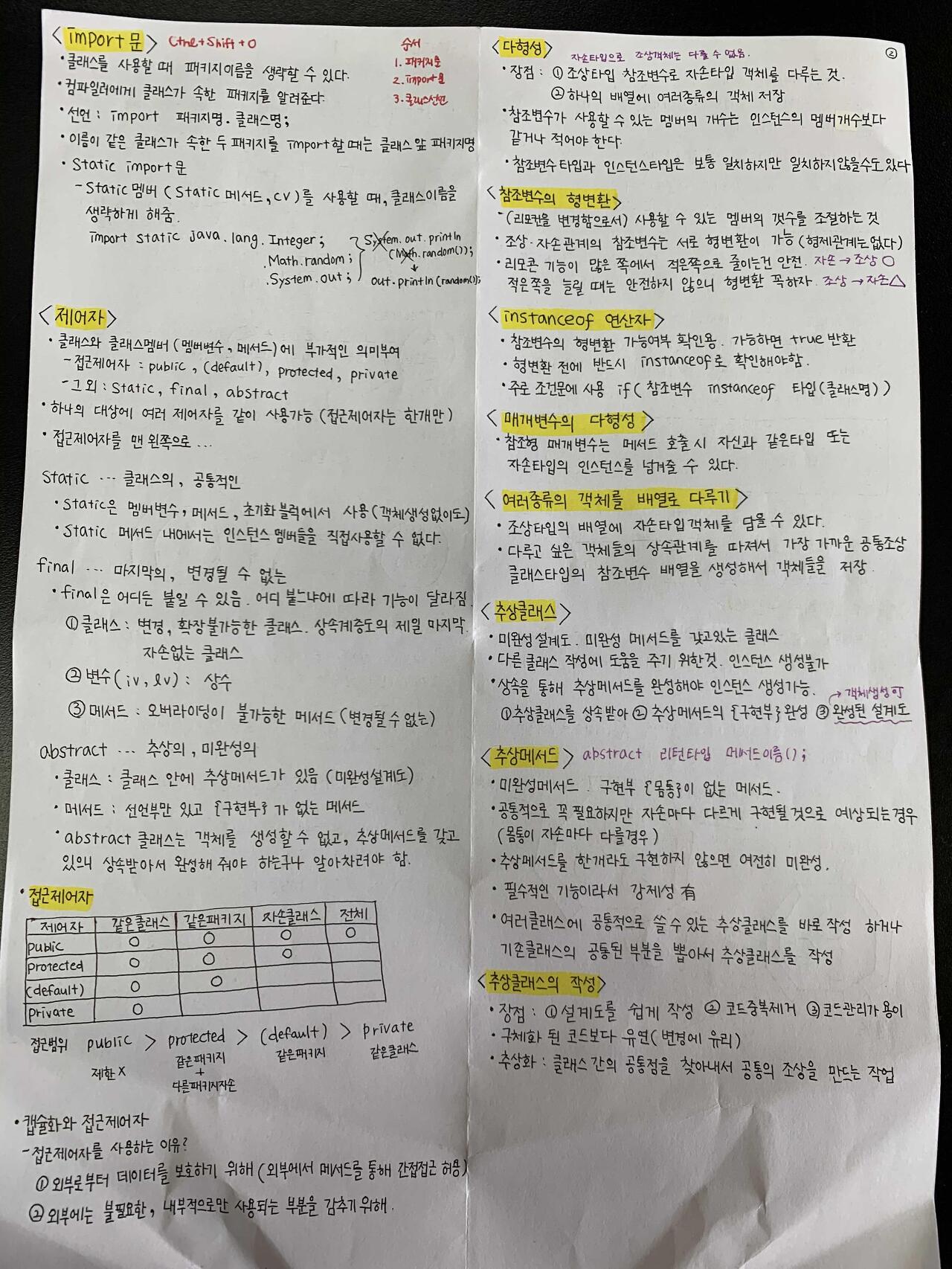
이벤트 핸들러-

이벤트 소스 결정 -> 이벤트 리스너 작성 -> 이벤트 소스와 이벤트 리스너 연결(이게 제일 중요. 빠뜨리는 사람이 많다)









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

[예제를 공부하는 방법]

1. 일단 따라치기 - 에러 없이 결과가 잘나오는지 확인하는 것으로 충분

2. 예제를 약간만 변경하기. - 조금씩 바꿔보고 결과를 예상후 확인. 다양하면서도 가볍게.

3. 예제 조합. - 둘 이상의 예제를 합쳐서 새로운 예제 만들기.

4. 주석 쓰고 코드 지우기 - 예제에 주석을 상세히 달고. 코드를 지운후 코드 작성하기.

5. 예제의 일부를 떼어서 별도로 테스트 하기

6. 하드 코딩 하기 - 변수에 직접 값 대입해보기

7. 그림 그리기

8. 질문하기​

[연습문제 푸는 방법] - 정답을 찾지 말것. 자신의 아이디어로 코딩을 해보는 것이 중요.

0. 풀 수 있는 문제를 찾고, 풀 수 있는 문제가 없으면 책을 보고 비슷한 예제를 찾을 것

1. 하루에 한 문제 풀기. - 하루 종일 틈틈히 고민하고 아이디어 떠오르면 시도

2. 가장 쉬운 방법으로 문제를 풀자. - 그리고 계속 개선. 무턱대고 반복문 적지 말 것.

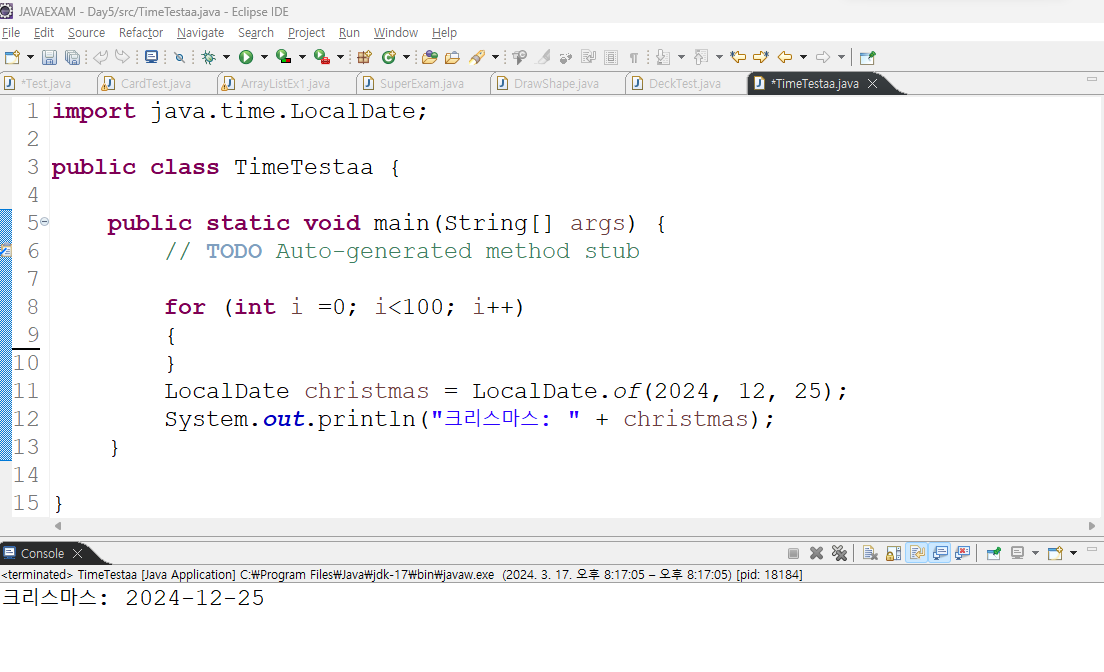
3. 문제를 작은 부분으로 나누고, 제일 쉬운부분 부터 해결

4. 그림 그리기

5. 질문해서 힌트 얻기

스크립트

날짜 넣기



씨언어 공부한 사람이 없어서- 몸값이 높다.

디자인패턴 – 헤드 퍼스트 라는 책이 있어.

소설책처럼 볼 수 있다. 패턴을

디자인패턴 공부를 강추한다.

네트워크

호스트 = 서버



**java.net.InetAddress** 클래스는 네트워크 상에서 호스트에 대한 IP 주소를 나타내는 데 사용됩니다. 이 클래스는 호스트 이름과 IP 주소 간의 매핑 및 호스트 주소의 해석을 위한 기능을 제공합니다. **InetAddress** 객체를 사용하여 네트워크 통신에 필요한 IP 주소 및 호스트 이름 정보를 얻을 수 있습니다.

일반적으로 **InetAddress** 객체는 다음과 같은 방법으로 생성됩니다.

1. **getLocalHost()**: 현재 호스트의 **InetAddress** 객체를 반환합니다.
2. **getByName(String host)**: 지정된 호스트 이름에 해당하는 **InetAddress** 객체를 반환합니다.
3. **getAllByName(String host)**: 지정된 호스트 이름에 대한 모든 가능한 IP 주소의 배열을 반환합니다.
4. **getByAddress(byte[] addr)**: 지정된 IP 주소 바이트 배열에 대한 **InetAddress** 객체를 반환합니다.
5. **getByAddress(String host, byte[] addr)**: 지정된 호스트 이름과 IP 주소 바이트 배열에 대한 **InetAddress** 객체를 반환합니다.

**InetAddress** 객체를 사용하여 호스트 이름과 IP 주소를 얻을 수 있으며, 네트워크 통신 및 서버 클라이언트 응용 프로그램에서 주소 해결에 사용됩니다.

데이터 보낼 때 겟방식과 호스트 방식이 있다.

안중요한건 겟방식,

아이디 비밀번호 같은 경우 겟방식으로 하면 안된다.

유니캐스팅/ 멀티캐스팅:

멀티캐스팅: 쓰레드 필요하다. 우리 코드 만든건 하나 밖에 접속이 안된다.

서버에 쓰레드

소켓 : 자바는 소켓을 통해 네트워크 통신을 한다.

자료구조

시험 코딩으로 본다. – 자바를 이어서 한다.

일단 무조건 공부에 머리를 가져다 박는 것은 좋은 생각은 아니다.

일단 배워야 하는 것이 무엇인지 리스트 업을 해야 한다.

그리고 프로그래머스 하루 2개,

자바의 정석 객체지향 부분 하루 1개 리뷰

문법이 중요한 것은 아니다.

생각을 해야하고, 해법을 찾아야 한다.

언어는 시간이 지나면 자연스럽게 올라온다.

문제는 생각이다.

자료구조와 함께 배우는 알고리즘 입문

어떤 정렬에 어떤 특징이 있는지 알고, API에서 받아서 내게 필요한 형태로 변환해서 쓰면 된다.

삽입정렬 – 일부 정렬되어 있으면 빠르지만, 전부 역방향으로 역정렬 되어 있으면 오히려 오래 걸린다.

선형 Queue – 선입선출

Enqueue- 큐에 넣는거, dequeue – 큐에 데이터 빼는거

실무에서 전혀 쓰지 않는다. 비효율적이다. 작업량이 너무 많아서. 데이터를 쓰면 쓸수록 기하급수적으로 느려진다.

원형 Queue

fron

Stack –

Push & Pop

기술은 시간이 지나면 해결이 된다.

문제는 그 기술로 무엇을 할 것인가?하는 것.

가장 나쁜 개발자를 코더라고 부른다. 코더는 욕이다.

아무 생각 없이 만드는 것.

창의력 가진 사람은 돈을 주고 개발자 사면 된다.

개발자 여성 경력 개발 생기지 않는다.

단 인맥과 자기 실력이 중요하다. 인맥이 잇어야 일을 받을 수 있다.

문제는 문제를 풀어야 감이 생긴다.

감이 있느냐 없느냐

재귀 호출 :Recursive Call

무한 루프다. – 개발자도 잘 못쓰는 경우 많다.

재귀호출은 반복문과 동일하다. 똑같은거 계속 반복한다.

큰 문제를 작은 문제로 분할하는 경우 100% 쓴다.

중복함수를 반복적으로 호출함으로써 메모리를 많이 사용하는 단점이 있다.

리스트 295 페이지

Linked list –

리스트는 자료를 순서대로 연결하여 저장하는 자료구조이며 주로 포인터를 이용하여 리스트를 연결한다.

포인터 = 레퍼런스

자기참조 object – 자기와 같은 타입을 가르칠 수 있는 것을 자기 참조 object라고 한다.

Class node

hasA 관계를 가지고 있다

우리가 쓰려고 하는건 single linkedList ->

실수로 링크가 끊어지면 다 잃어 버린다.

객체는 레퍼런스가 필요하기 때문이다.

싱긍, 더블,

싱글 환형, 싱글 더블 이렇게 4종류이다.

싱글 리스트

리스트는 뒤로 못간다. 때문에 중간에 멈추면 다시 처음부터 검색해야 한다.

하지만 역방향 으로 가르킨다면 – 앞으로도 뒤로도 갈 수 있다.

그러면 노드가 앞의 것도 뒤의 것도 알아야 한다.

뒤의 레퍼런스도 가지고 있는 것을 더블 링크드 리스트라고 한다.

기준

Head – 첫번째 노드를 가르키는 애 , 만약 첫번째가 삭제되면 두번재가 head가 되어야 한다.

새로 추가하는 것: new node –

Next 다음: 마지막 노드의 next는 null이어야 한다.

Prev 이전

Del 삭제

Cur: 잡일 – 검색을 한다든가 할 때 이 이름을 쓴다.

처음 삽입하는 경우, 중간에 삽입하는 경우, 마지막에 삽입하는 경우 다 경우의 수를 생각해야 한다.

삽입하는 경우

삽입전 노드 검색-> 커서를 활용(첫번재 삽입은 불필요)

앞의 것을 먼저 끊으면 뒤에것을 다 잃어 버린다. 그러므로 먼저 뒤랑 연결하고, 앞을 연결한다. 뉴노드의 넥스트에 커서의 레퍼런스를 넣어라.

커서의 넥스트에 뉴노드를 넣어라. 그러면 연결이 된다.

삭제하는 경우

커서를 사용해서 삭제 전의 노드를 검색 –

Delete로 삭제할 레퍼런스를 넣는다.

커서의 넥스트에 delete의 next를 넣고

삭제할 노드의 nest에 null 넣는다.

이름을 정하면 된다. 항상 그 용도로만 써야 한다. 안그러면 실수를 한다.

Binary Searh Hash : 이진검색의 개념

이진검색의 개념 : 정렬된 배열 또는 리스트에서 특정 값을 빠르게 찾는 알고리즘.

단점: 자료를 항상 정렬된 상태로 유지해야 한다.

재귀 함수 : 시험 – 다 짜세요. 혹은 채우기.

프로그래머의 수학공부

* 수학자가 들려주는 이야기 – 전지 / 애기들 그 책을 항상 옆에서 읽어 줘라.
* 사기 전에 한번 보고 / 읽을 만하면 사라 – 초등학생도 읽을 수 있는 수준.
* 자녀가 클때까지 절대 버리지 말아라.

인공지능 잘하려면 선형대수학을 해야 한다.

선형대수학을 잘하려면 미적분을 잘해야 한다.

이진트리 순회 :

이진탐색트리의 성능이 최하일 때 트리의 형태 –

선향 트리의 경우 – 최악의 경우.

최상은 왼쪽과 오른쪽이 대칭인 경우.

이진트리: 삭제 조심해야 한다.

자주 쓰는 API

|  |  |
| --- | --- |
| **for** (**int** i = 0; i < numberStr.length(); i++) {  *digits*[i] = Character.*getNumericValue*(numberStr.charAt(i));  } | //문자열로 표현된 숫자를 각 자리마다 분리하여 정수 배열에 저장 |
| // Arrays.sort(digits);를 활용하여 배열 정렬  **private** **static** **int**[] arrMaker(String number) {  // **TODO** Auto-generated method stub  // 숫자를 문자열로 변환하여 각 자리수에 접근  **int** space = String.*valueOf*(number).length();  **int**[] digits = **new** **int**[space];  String numberStr = String.*valueOf*(number);  // 각 자리수를 배열에 저장  **for** (**int** i = 0; i < numberStr.length(); i++) {  digits[i] = Character.*getNumericValue*(numberStr.charAt(i));  }  Arrays.*sort*(digits);    **return** digits;  } | // 숫자를 문자열로 변환하여 배열 생성 및  배열 정렬 |
| **import** java.util.\*;  Scanner sc = **new** Scanner(System.***in***);  exit = sc.nextInt(); |  |
| **int**[] answer = Deck.*randomMaker*(); | // 컴퓨터 난수 생성 |
| String numberStr = String.*valueOf*(number);  **int** len = numberStr.length(); | // String Class를 활용한 정수의 길이 생성 |