자기소개

통신 개념, 잘하는 언어

Java final 개념, http 통신 개념

자소서 기반 질문

수행했던 프로젝트 질문

학부생 시절 가장 재미있게 들었던 과목

그 과목 중 자신 있는 부분을 면접관에 설명해라. 네트워크, socket 프로그래밍, 앱 개발 쪽으로 하는게 좋을 듯

JAVA 질문. JAVA에 있는 자료구조 활용하여 문제를 풀음.

네트워크, OS 질문

소프트웨어 공학, 데이터베이스, 운영체제, 네트워크, 알고리즘 -> 로우 레벨에서 돌아가는 것 까지 알아야함(변수는 메모리 어디에 저장 되는가?)

왜 개발자가 되었는지

원하지 않는 일을 시키면 어떻게 해야 할지

야근을 해도 되는지

\*자바, DB, 네트워크 위주 공부

질문 할거?

자소서 질문

남들은 기피하는 어려운 과목들이란?

-> 전공 과목 중에 교수님께서 잘 가르치시지만 시험이 어려워 좋은 점수를 얻기 힘든 과목들이 있었습니다. 저는 과목을 듣고 난 다음에 저에게 도움이 될 수 있는 과목들을 좋아했기 때문에 남는 것은 없지만 점수를 얻기 쉬운 수업들은 기피했습니다. 수업을 열심히 듣고 공부해서 결과적으로는 좋은 점수도 얻고 많이 배울 수 있었습니다.

깊게 이해해서 보완된 사례는?

-> 지능형시스템 수업에서 크게 도움이 되었습니다. 이 수업에서는 데이터를 어떻게 분석할 수 있을지 이것을 가지고 어떤 방식으로 학습시킬지를 배웠습니다. 먼저 데이터를 어떻게 분석할지에 대해 이해하고 거기에서 그치는 것이 아닌 더 파고들고자 노력했습니다. 알고 있는 정의도 다시 생각해보고 완벽하게 이해하고자 했습니다. 시험 문제 대부분이 예시가 주어지고 어떤 방식으로 데이터를 분석해 나갈지 쓰는 것이었는데 저는 제가 알고 있는 지식에 대해 확신이 있었기 때문에 문제를 잘 해결할 수 있었고 좋은 점수를 받았습니다.

새로운 기술에 잘 적응한 사례는?

-> 매번 새로운 기술을 배울 때 처음은 어렵지만 나중에는 응용하는 저를 발견할 수 있었습니다. 최근에는 깃헙 블로그를 만들어보고자 여러 시도를 했습니다. 처음에는 깃헙도 어떻게 사용하는지 모르고 이해가 안 갔지만 적응하기 시작했고 나중에는 테마를 저의 스타일에 맞게 고칠 수 있게 되었습니다.

어느 환경에도 적응할 수 있는가 적응하기 힘들었던 환경은?

-> 자기소개서에서도 썼던 교환학생 생활이 제일 적응하기 힘들었습니다. 왜냐하면 자기소개서에도 썼 듯이 해외라는 곳이 저에게 처음이었기 때문입니다. 영어나 프랑스어를 원어민 수준으로 할 수 있지 못했던 터라 걱정이 많았지만 좋은 사람들을 만나고 영어를 알아듣고자 노력해서 잘 적응할 수 있었습니다.

다른 스타일로 고쳤다고 하는데 본인의 스타일과 어떤 점이 달랐는가?

-> 포트폴리오에 나와있는 코드가 서로의 스타일들이 합쳐진 결과물입니다. 제가 스테이지를 구현하면 다른 개발자 분이 코드의 통일성을 위해 수정을 했습니다. 저는 게임 오브젝트 이름과의 매치를 위해 배열을 1부터 시작했는데 다른 개발자 분은 배열의 시작은 무조건 0이라며 수정을 했습니다. 그리고 저는 한 줄을 길게 쓰기 보다는 여러 줄에 나눠서 썼습니다. 다른 개발자 분은 한 줄에 최대한 많은 것을 표현하는 스타일이었습니다. 스타일마다의 장단점이 있지만 둘의 스타일이 서로 극단적이지 않고 적절히 조화를 이루는 것도 좋은 방법이라는 것을 알 수 있었습니다.

커뮤니케이션 능력을 발휘해서 해결한 사례는?

-> 저번 학기에 졸업 작품 같이 학기 내내 하는 프로젝트를 했습니다. 개발자는 저 혼자 였고 나머지 세명은 기획자, 디자인을 맡았습니다. 3D Co-op 컴퓨터 게임을 개발했습니다. 네 명이 처음 도전하는 분야 였고 익숙하지 않았기 때문에 시행착오가 많았습니다. 3D 모델링을 맡은 학우가 소통이 적은 편이어서 만드는 과정이 쉽지 않았지만 제가 나서서 제작의 시행착오를 줄이고자 스케쥴 조정과 난이도 조절을 했습니다. 그 결과 게임은 성공적으로 마무리 되었고 교내 대회에서 우수상을 수상했습니다.

한국인에게 익숙한 기능과 UI를 제공해준다 하였는데 구체적으로 어떤 점이 그러 한가?

-> 네이버 개인화 서비스를 웹 상에서는 위에 아이콘으로 배치함으로써 사용자들에게 보기 쉽게 해주었습니다. 그리고 통일적인 UI를 통해 사용자들이 메일을 주로 사용했다 하여도 다른 개인화 서비스를 이용하고자 할 때 더 쉽게 접근할 수 있습니다.

수신확인이 가능해서 상대방의 메일 읽음 여부에 대한 답답함을 줄인 것과 내게 쓰기 기능이 저에게 그리고 제 주변 사람들 사이에서는 좋은 기능으로 통합니다. 주로 학교 수업을 학교 컴퓨터로 듣는 저와 학우들은 수업 내용을 타이핑하고 저장하기 위해 내게 쓰기나 네이버 메모를 많이 이용했습니다. 간편하고 시간이 지나도 지워지지 않고 확인 가능하기 때문입니다.

외국인들에게 어떻게 네이버 메일을 써보게 해줄 것인가?

-> LINE을 활용하면 좋을 것 같습니다. 자기소개서에서 썼 듯 제가 처음 구글에 접한 계기도 구글스토어 때문이었습니다. 이런 방식처럼 LINE을 이용하는 사용자들을 상대로 네이버 메일 이벤트를 통해 접하게 하면 메일을 써보면서 네이버 메일의 좋은 점들을 알게 될 것입니다.

네이버 개인화 서비스 중 발전시켜야 할 부분은?

-> 현재로서도 좋은 기능을 제공해주고 있지만 트랜드인 인공지능을 결합한 기능이 보편화 되면 좋을 것 같습니다.

중급 -> 중간 규모 프로젝트 및 시스템을 개발할 수 있으며 주요 이슈 트러블슈팅(시스템에서 발생하는 복잡한 문제들을 종합적으로 진단해 처리.)을 할 수 있을 정도로 내부구조에 대해 이해하고 있음

C-> 메모리 구조 : Code Area -> 코드 자체를 구성하는 메모리 영역. Hex or bin file memory, 프로그램 명령이 위치. 기계어로 제어되는 메모리 영역

Data area->전역변수, 정적변수(static)-지역(변수)이 사라져도 변수는 유지되어야 하는 경우,배열, 구조체 등이 저장.

프로그램 실행 시 생성. 종료되면 시스템에 반환. 함수 내부에 선언된 static 변수는 프로그램 실행 시 공간만 할당. 함수 실행 시 초기화.

초기화된 데이터는 data 영역에 저장. 초기화 안되 데이터는 BBS Block Stated Symbol에 저장

Heap area->프로세스가 사용할 수 있도록 미리 예약 되어 있는 공간. 동적으로 메모리 할당 시 위치하는 메모리 영역. 메모리 주소 값에 의해서만 참조되고 사용.

Stack area(high address)->프로그램이 자동으로 사용하는 임시 메모리 영역, 지역변수, 매개변수, 리턴 값 등 잠시 사용 후 사라지는 데이터 저장.

함수 호출 시 생성. 함수 끝나면 시스템에 반환됨.

사이즈는 각 프로세스마다 할당. 프로세스가 메모리에 로드 될 때 스택 사이즈가 고정되어 있어 런타임 시 스택 사이즈 변경 불가능.

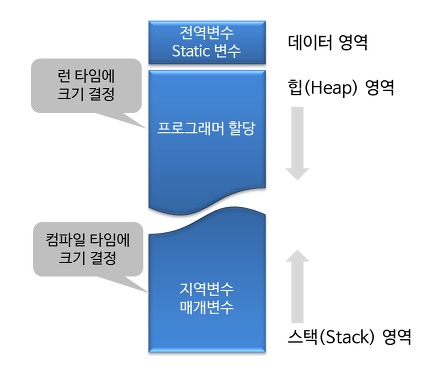
명령 실행 시 자동 증가, 감소하기 때문에 보통 메모리 마지막 번지를 지정

메모리 공간

할당 시기-> 프로그램 실행될 때 마다

할당 장소->메인 메모리 RAM

할당 용도 -> 프로그램 실행 시 필요한 메모리 공간 (지역변수-함수 내 , 전역변수) 할당

매개변수->파라미터. 함수의 인풋으로 제공됨.

정적 변수-> 선언 위치는 지역변수와 같음. 특정 선언 지역에서만 접근 가능. 초기값을 주지 않을 경우 항상 0으로 초기화. 프로그램 실행시킬 때 단 한번만.

Malloc -> void 포인터를 반환하여 개발자가 알맞은 용도로 변환하여 사용할 수 있게 함.

Calloc(몇 개, size (int)) 값 모두 0으로

Realloc(void\*, 바꾸고 싶은 공간 크기 sizeof(int)\*5)

에러

런타임 에러-> 설계 실수. 무한루프. 나누기 0을 한 경우

컴파일 타임 에러 -> 문법적 오류. Warning or error. Syntax error->조건문이나 반복문 등 형식이 맞지 않음. 데이터 타입 등

논리 에러-> 논리 오류. 의도하지 않은 결과가 나옴.

Java

클래스-> new 를 통해 객체를 만들 수 있음

Animal cat = new Animal() 클래스에 의해서 만들어진 객체를 인스턴스. cat이라는 객체는 Animal의 인스턴스. 인스턴스는 특정 객체가 어떤 클래스의 객체인지 관계 위주 설명 시. Cat은 Animal의 인스턴스.

과자틀은 클래스고 과자틀에 의해서 만들어진 과자들이 객체

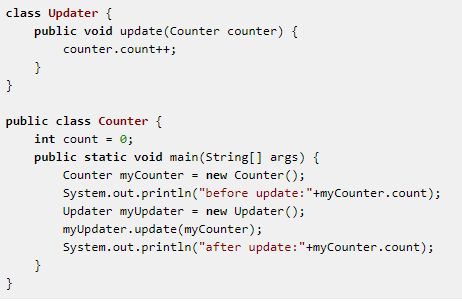
객체 변수-> 인스턴스 변수, 멤버 변수, 속성. 클래스에 의해 생성되는 것은 객체 . 객체 변수는 서로 공유되지 않는다.

메소드->함수. 반복적인 작업을 할 때 어떤 입력값을 주었을 때 어떤 리턴값을 돌려준다 식.

Local variable 변수 -> 메소드 내에서만 쓰임.

메소드에 값(primitive type) 전달하는 것과 객체(reference type) 전달하는 것엔 차이가 큼.

메소드로 객체를 전할 경우 메소드에서 객체의 객체변수 값을 변경할 수 있음.

이래야 count값 증가함.

즉 다른 클래스의 지역 변수(속성값)를 건드리고 싶을 때 사용하는 방법.

상속 inheritance-> 자식이 부모로부터 물려 받는 것. 자식클래스 extends 부모클래스

IS-A 관계. Dog is a animal 같은 것. 하위 개념. Animal dog = new Dog() (개로 만든 객체는 동물 자료형이다.)는 되지만 반대는 안됨. 부모 클래스로 만든 객체는 자식 클래스의 자료형으로 사용할 수 없다.

Method Overriding-> 부모 클래스와 동일한 형태(입출력 동일)의 같은 메소드를 구현하면 자식 클래스의 메소드가 더 높은 우선순위를 갖게 된다.

Method overloading-> 자식클래스에 부모 클래스의 메소드를 생성할 때 입력항목만 다르게 하는 것.

\*자바는 다중 상속을 지원하지 않는다. (애매모호한 경우가 발생하기 때문. 두 클래스에 같은 메소드가 있을 때는 어떤 것을 우선순위로? 그래서 다른 언어에선 우선순위 등을 적용.)

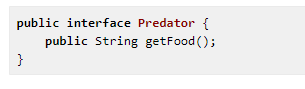
생성자 Constructor-> 메소드명 == 클래스명. 리턴 자료형이 없는 메소드. New 클래스명(input,,) 생성자가 있으면 이후 객체 생성시 지켜주지 않으면 오류남.

Default 생성자-> 생성자인데 입력도 내부에도 아무 내용 없음.

생성자도 오버로딩 가능. 즉 입력 다른 여러 생성자 만들어도 됨.

Interface-> public interface Name{} public class A extends B implements Name{}->인터페이스 구현

A가 Interface의 자료형을 타입으로도 쓸 수 있음.

implements한 클래스는 무조건 getFood 구현.

Zookeeper가 동물들의 종류에 의존적인 클래스에서 동물들의 종류와 상관 없는 독립적인 클래스가 되었다.

컴퓨터 -> Zookeeper

USB Port->Predator

Hard Disk, Camera->Tiger, Lion..

Implements A,B 가능

인터페이스는 일반 클래스와 달리 extends A,B 가능 다중 상속 됨.

자식 인터페이스로 생성하 객체의 자료형은 부모 인터페이스로 사용하는 것이 가능.

다형성 Polymorphism-> 객체지향 프로그래밍 특징. 이를 이용하면 복잡한 if else 조건문을 간단하게 처리할 수 있다.

* 하나의 객체가 여러 개의 자료형 타입 가질 수 있는 것. Tiger는 Tiger 클래스의 객체, Animal의 객체, Barkable, Predator 인터페이스의 객체 Barkable bark = new Tiger()가능.

A instanceof B->A객체가 new B로 만들어진 객체인가?

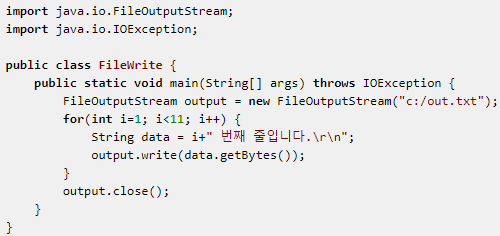
**추상 클래스-> 하나 이상의 추상 메소드를 가지며 객체 생성 불가능. 새로운 클래스를 작성하는데 있어서 바탕이 된다. 기존 클래스의 공통 부분을 뽑아 내서 조상 클래스로 만드는 것.**

**인터페이스는 추상메소드로만 구성,**

추상 클래스 Abstract Class-> 인터페이스의 역할, 구현체도 가지고 있는 자바의 돌연변이 클래스.

Public abstract class Predator extends Animal{ public abstract String getFood()} 몸통 없음. Abstract 클래스 상속하는 클래스에서 이 메소드 구현해야함. Extends Predator

Abstract 안 붙은 메소드도 넣을 수 있음. 그러면 Tiger, Lion 같은 객체들이 그 메소드 모두 사용할 수 있게 됨.

새로운 파일 생성.

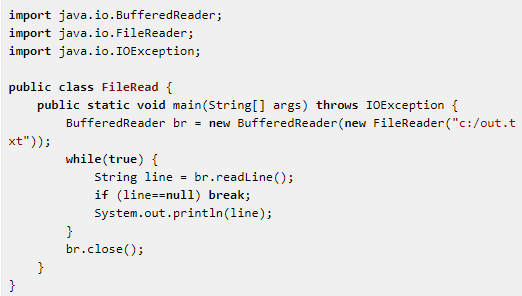
close생략 가능. 자바가 프로그램 종료 시 닫아 줌. 하지만 닫지 않고 다시 사용하려 하면 오류 남.

String을 byte로 변환해야해서 불편함.

FileWriter를 쓰면 그냥 바로 보낼 수 있음. 하지만 \r\n써야 하니 PrintWriter를 쓰고 pw.println(data)하면 알아서 붙어 줌.

FileWriter fw2 = new FileWriter("c:/out.txt", true); 이렇게 뒤에 true 붙이면 추가모드로 열음. printWriter도 마찬가지.

FileInputStream은 byte배열로 파일 읽어서 불편.

읽을 라인 없으면 null return.

Package-> 자바 클래스들을 모아 넣는 자바 디렉토리. 서브패키지는 . 이용

Import package; 클래스 분류에 용이하고 패키지가 다르면 동일 클래스명 사용 가능.

Private-> 변수, 메소드는 해당 클래스에서만 접근 가능.

Default-> 접근제어자 설정 하지 않으면 해당 패키지 내에서만 접근 가능.

Protected-> 변수, 메소드는 동일 패키지 내 클래스 또는 클래스 상속받은 외부 패키지의 클래스에서 접근 가능.

Public-> 변수, 메소드는 어떤 클래스에서도 접근 가능.

\*\* public이 붙은 클래스는 반드시 파일명과 이름이 같아야 한다. 가독성

클래스 내의 클래스인 inner class에도 접근제어자 붙일 수 있다.

Static-> 변수, 메소드 앞. 항상 값이 변하지 않는 경우. 메모리 이점 얻을 수 있음. 같은 곳의 메모리 주소만 바라보게 되므로 다른 객체들이 공유할 수 있음.

클래스를 설계할 때 멤버변수 중 모든 인스턴스에 공통적으로 사용해야하는 것에 static.(원래는 서로 다른 값)

Static이 붙은 멤버 변수는 인스턴스를 생성하지 않아도 사용할 수 있다. (자동적으로 클래스가 메모리에 올라갈 때 생성 됨)

Static method-> 안에서는 인스턴스 변수 접근이 불가능함. 유틸리티 성 메소드 작성 시 많이 사용.

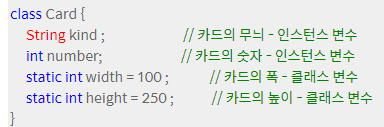
인스턴스 변수는 인스턴스를 생성해야만 존재할 수 있는데 static이 붙은 메서드를 호출할 때 인스턴스가 생성되어 있을지도 않을 수도 있어서 허용 안함.

반대는 가능함. 인스턴스 변수가 존재한다는 것은 static이 붙은 변수가 이미 메모리에 존재한다는 것을 의미.

메서드 내에서 인스턴스 변수를 사용하지 않으면 static 붙이는 거 고려. 호출 시간이 짧아져 효율을 높임. (안 붙이면 실행 시 호출되어야 할 메서드를 찾는 과정 추가적으로 필요)

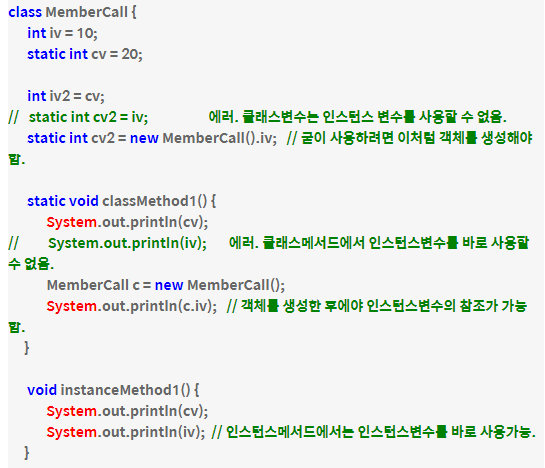
\*\*클래스 설계 시 static 사용지침

* **클래스의 멤버변수 중 모든 인스턴스에 공통된 값 유지시켜야 하면 static 붙여주기**
* 메서드 중 인스턴스 변수를 사용하지 않는 메서드에도 고려

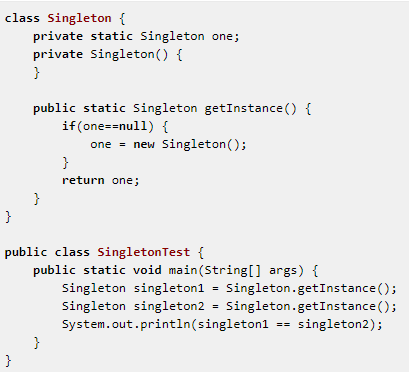
width, height는 모든 인스턴스가 같은 값 유지. 모두 같은 저장공간 참조.

클래스메소드-> 클래스 변수만을 사용하는 메소드.

인스턴스 메소드-> 인스턴스 변수와 관련된 작업을 함. 작업 수행 시 인스턴스 변수 필요로 함.

\*\* 인스턴스 멤버(변수와 메서드)는 반드시 객체를 생성한 후 참조 호출이 가능함. 인스턴스 간에는 문제 없음. 이미 생성되어 있다는 것을 전재.-> 이럴 경우는 드믐.

Final-> 값을 변경하지 못하게 함. 변경하려 하면 예외 처리.



Singleton pattern-> 디자인 패턴 중 하나. 단 하나의 객체만을 생성하게 강제하는 패턴. 클래스를 통해 생성할 수 있는 객체는 하나.

Try, catch, finally

Class extends RuntimeException 해서 만들 수 있음. throw new class();

RuntimeException-> 실행 시 발생하는 예외. 발생 할 수도 안 할 수도 있음.

Exception-> 컴파일 시 발생하는 예외. 프로그램 작성 시 이미 예측가능한 예외

Transaction-> 하나의 직업 단위

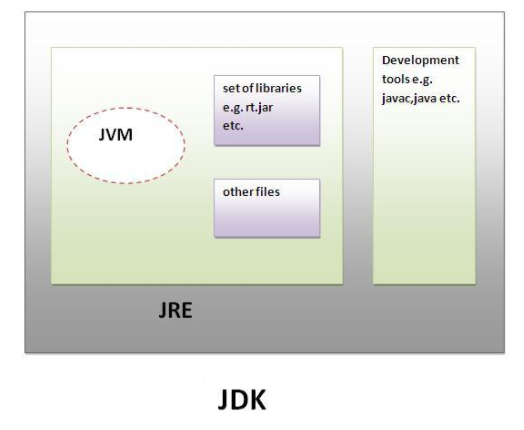
Rollback-> 작업들 모두 취소하는 행위 / 트랜잭션 관리 위해 사용. 모두 취소를 안 하면 논리적으로 안 맞는 경우가 생길지도.

Process-> 동작하고 있는 프로그램.

Thread-> 두 가지 이상 일을 동시에 할 수 있음. Extends thread. Thread.start()로 시작. Run()함수에 작업 쓰기 try{thread.sleep();}catch{}

* join메소드 사용해서 메인에서 먼저 스레드 종료 안 시키게 함.종료될 때 까지 대기.
* 보통 implements Runnable 많이 씀. Thread t = new Thread(new Test(i)); 인터페이스로 인해 상속 및 기타 등등에서 좀 더 유연한 프로그램으로 발전한 것.

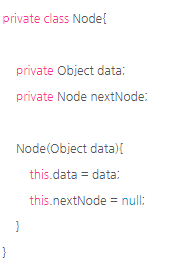
JVM-> Java Virtual Machine. 자바 소스코드로부터 만들어지는 자바 바이너리 파일 .class를 실행 할 수 있음. 플랫폼에 의존적. 리눅스의 JVM과 윈도우의 JVM은 다르다. 컴파일된 바이너리 코드는 어떤 JVM에서도 동작 가능.

* 역할 : 바이너리 코드를 읽는다, 검증한다, 실행한다.
* 역할 : 실행환경(Runtime Environment) 규격 제공. 필요한 라이브러리, 기타 파일

JRE-> Java Runtime Environment 약자. JVM이 자바 프로그램 동작 시킬 때 필요하 라이브러리 파일들, 기타 파일들을 가지고 있음. JRE는 JVM의 실행 환경 구현

JDK->자바 개발 도구 Java Development Kit. JDK 는 JRE+개발을 위해 필요한 도구(javac, java) 포함.

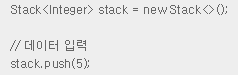
**소스 party.java->컴파일러 오류 안생기면 party.class bytecode-> JVM을 통해 party.class 실행. JVM은 바이트 코드를 해당 플랫폼에서 이해할 수 있는 형태로 해석해서 프로그램을 실행시켜줌.**



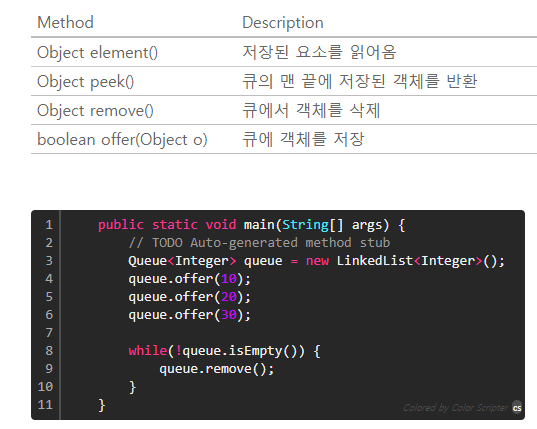
노드 만드는 법.

스택 만들 때 헤드 값을 이동시키면서 top값이 맨 마지막이 되도록 유지.

큐는 헤드 값과 꼬리 값 유지.



push는 데이터를 앞에, offer은 데이터를 뒤에 입력. poll()하면 됨.



C#(Unity)-> using System;

namespace abc{

class Hello{

public static void Main(){

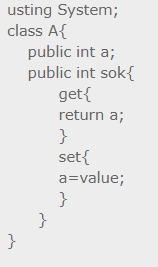
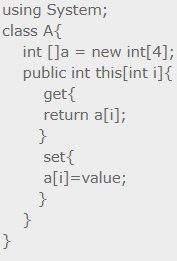
Console.WriteLine("HelloWorld"); } }}

Int []a = new int[4]; int[] a = new int[]{1,2,3,4}; int[,] a = new int[2,3]; 이차원 배열 각 줄 요소 수 안 같아도 됨.

length로 요소 구하기. Enum-열거형

접근 제한자 생략하면 private.

필드-> 클래스 안 변수. / 메소드-> 클래스 안 함수



a.sok = 3; Console.Write(a.sok); 그냥 입력, 출력하면 된다.

인덱서-> 객체를 배열처럼 취급. Get, set 으로 구성.=>

상속-> class B:부모클래스명 부모클래스부터 차례로 생성자 작동.

Sealed-> 상속되지 않도록 할 클래스.

추상클래스(abstract) 상속하여 사용하려면 override 붙여줘야 함.

Interface-> 추상메서드, 속성, 인덱서, 이벤트만 멤버로 가질 수 있음. Public abstract void add();

Public abstract int b{ get; set;} 모든 메서드 public 취급하므로 생략 가능. 여러 개 인터페이스 가능. Class X : A,B,C interface A{}

Namespace-> 같은 이름의 클래스 등이 경합을 일으키지 않도록 함.

네임스페이스 안에 또 다른 네임스페이스 만들 수 있다.

다른 네임스페이스 사용하려면 using namespace or Namespace.Class

FileStream fs = newStream("A.txt", "FileMode.Open"); //A.txt는 파일이름이고 FileMode는 열기모드이다

StreamReader r = new StreamReader(fs); string s = r.ReadLine(); r.Cloe();

자동으로 garbage collection 함.

제공되는 데이터형-> 값형 value type과 참조형 reference type으로 구분. 둘 다 선언과 동시에 스택에 변수를 위한 메모리 공간 생성. 실제 데이터가 저장되는 공간은 다름.

-> 값형: 할당된 스택 메모리에 값 저장

->참조형: 힙에 객체를 위한 메모리 공간 할당. 주소에 대한 hash value 즉 참조 값을 스택 메모리에 저장. Index table에 해시 값, 실제 메모리 주소 저장.

실수 쓸 때 숫자 뒤에 유형 안정해주면 디폴트인 더블로 됨. F, D, M(decimal)을 뒤에 붙임. 정수형 L(long)도 마찬가지.-> 내장형 데이터 built-in type

사용자 정의형 데이터 user-defined type –> enum 열거형, struct 구조체

enum UserType{

Admin,//0 Master//1, Guest=5, Invalid //6 } //UserType.Admin이라 하면 상수 0이 나옴.

public struct Dog{

public string name; //멤버 변수

public Dog(string name){}//생성자. 메서드} ///Dog.name

문자열-> 값형. 배열->참조형

const 상수-> 변하지 않는 문자나 숫자 값. 선언과 동시에 초기화.

Readonly-> 선언과 동시에 초기화하지 않아도 됨. Static도 써줌.

Convert->Convert.ToInt32(string); 언어마다 컴파일러 다르니깐 어디서나 호환이 되도록.

c->obj->.exe 우리가 만드는건 obj 플랫폼에서(컴파일러) 실행되는건 exe

java->.class / JVM 여기선 JVM이 c#은 .cs파일

\*\*객체지향-> 객체지향의 핵심인 클래스는 참조형에 해당하는 사용자 정의 데이터형.

New 키워드를 통해 힙에 데이터 저장 위한 메모리 할당. 할당된 힙 메모리의 실제 주소를 해시 함수 통해 구하고 해시 값을 스택 메모리에 지정. 객체/인스턴스를 선언하면 참조 값을 저장하기 위한 스택 메모리 할당.

객체지향 프로그래밍에서는 실세계를 바탕으로 모델링화하여 사용자 정의 데이터형인 클래스를 만든다.

특성정보-> field 멤버변수. 행동양식->method.

자식 클래스 is a 부모 클래스

C#-> 다중 상속 금지. 여러 개 인터페이스 구현 가능

추상 메서드-> 구현이 없는 빈 메서드. 인터페이스의 메서드와 같이 상속 받은 클래스에서 구현해야 함.

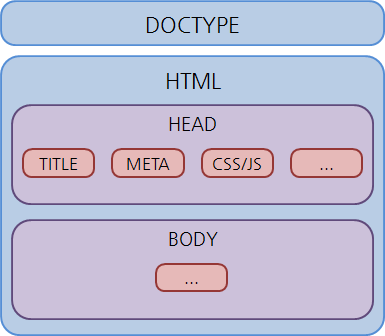
생성자, 소멸자 있음(~ClassName(){})

Internal-> 참조 목록에 있는 클래스를 포함하여 현재 웹사이트 또는 프로젝트 클래스에서 접근 가능.

Private-> 같은 클래스에서만

Protected-> 같은 클래스, 파생클래스 /protected internal

Public string SocialNumber{ get{return social;} set{}}

 HTML-> <meta charset="utf-8"> 다국어로 작성된 문서.

ASP.NET

Server-> 자료나 자원을 제공 Client->필요한 자원을 요구.

웹 애플리케이션-> 웹을 기반으로 실행되는 프로그램.

URL-> 자원의 위치를 표현하는 주소

HTTP-> HyperText Transfer Protocol 자원을 서로 주고받기 위한 약속

HTML->HyperText Markup Language 자원을 표현하고 자원 사이 이동 쉽게 할 수 있도록 개발된 언어

**사용자가 웹 브라우저 주소 입력 URL 입력해서 요청하면 서버에서 HTTP 따라 요청을 해석하고 URL가 가리키는 문서를 찾아 클라이언트에 보냄. 컴퓨터는 HTTP에 따라 응답 해석 후 문서를 웹 브라우저에 전달함. 웹 브라우저는 HTML을 해석해서 화면에 뿌림.**

**DNS root name server-> .comTLD->Route ## name server-> ip**

Static Web page-> 정적. 서버에 있는 문서가 수정되지 않으면 항상 같은 웹 문서.

Dynamic Web Page-> 동적. 실시간으로 변화하는 내용에 따라 저장된 데이터를 바탕으로 그때 그때 만들어짐. 서버에서 동적 웹 페이지 코드를 이용해 여러 상황 고려한 문서를 만듦. 필요한 자료는 DB로부터 얻음.

ASP.NET-> 웹 애플리케이션을 구축하기 위한 프로그래밍 프레임워크. -> 제품을 구성하는 부분.

Component-> 프로그램을 구성하는 재사용할 수 있는 작은 블록. 버튼, 입력 상자 등.

ASP는 DB연동을 위한 ADO 및 다른 프로그래밍 언어로 개발된 다양한 컴포넌트 등을 이용해 스크립트 한계를 극복. 서버 객체, 서버 컴포넌트

**이벤트 기반 프로그래밍 모델** -> 재사용성 높고 공유, 작성, 읽기 쉬움

컨트롤 기반 페이지-> 웹 폼과 서버 컨트롤들로 만들어짐.

CommonLanguage Runtime 통한 언어 독립적 플랫폼 제공. VB, C#, J# 등으로도 만들 수 d있음.

Code-behind 모델-> 코드 부분 분리.

성능 향상-> 컴파일 언어 사용하여 페이지 작성. 컴파일된 페이지를 Caching 기법에 의해 메모리에 올려놓음. Garbage Collector로 객체 관리. 메모리 누수로 인한 서버 다운 줄임.

다양한 웹 브라우저 지원.

IIS기반.

가상 디렉터리-> 물리적으로 하위 디렉터리가 아니어도 웹에서 웹사이트의 하위 디렉터리로 인식됨.

(없으면 보안 문제-루트 디렉터리만 알아내면 다 접근 가능하게 됨, 공간부족 현상이 발생)

ASP.NET 페이지 구성-> **업무 로직 business logic인 소스 코드 부분, 시각적인 부분인 HTML**

단일 파일 페이지 모델 Single-File -> 코드, HTML이 하나의 파일에 위치.

코드 숨김 파일 페이지 모델 Code-behind Page Model-> 코드는 cs파일.

.aspx가 상속 받는 객체고 .cs가 상속해주는 객체.

<%@ Page Language=”C#” AutoEventWireup=”true” CodeFile=”Circle.aspx.cs” Inherit=”Circle” %>

AutoEvent-> 통상적인 event handling 하는데 있어서 자동으로 해주겠다.

<head runat=”server”></head>

Protected void btnArea\_Click(object sender, EventArgs e){}

//,,이벤트 처리기 이름, 이벤트 발생 객체 정보, 이벤트 정보

Window Form (Win Form)-> Windows 응용 프로그램 개발에서 사용자 인터페이스를 구성하는 모든 종류의 창 또는 대화 상자.

Web Form-> 웹 응용 프로그램 작성.

서버 컨트롤->서버에서 운영되는 객체. 두가지 유형 runat=”server”

* **HTML 서버 컨트롤** : 표준 HTML 태그와 상응하는 서버 기반 컨트롤. C#에서 이거 컨트롤 할 수 있도록 맵핑
* **웹 컨트롤** : 윈도우 응용 프로그램 작성시 사용하는 컨트롤이 웹 폼에서 사용되는 경우. 태그 앞에 asp: 붙음. 서버에서 동작하는 하나의 .NET 클래스.

->표준 웹 컨트롤, 유효성 검사 컨트롤, 데이터 컨트롤, 로그인 관련 컨트롤

HTTP에서 제공하는 대표적 전송 방식

* GET : user가 input 엘리먼트에 입력한 정보를 URL 통해 서버로 전송. ? 뒤에 오는 문자열-> Query String. 매개변수=변수값. &로 연결 . 16진수많은 양의 데이터를 보내기엔 적합하지 않음. 2000자 이상은 처리 못함. 보안상 문제. URL복사하면 동일 정보 볼 수 있다.
* POST : 웹 브라우저가 서버로 보내지는 페이지 요청에 사용자가 input 엘리먼트에 입력한 내용 포함시켜 전달. 내용 노출 안됨. 정보의 양 한계 없음. 새로 고침 하면 만료된 페이지라 뜸.

View state-> 뷰상태. ASP.NET에서 보내는 hidden 필드는 압축된 형태로 페이지에 있는 모든 서버 컨트롤 상태 정보를 갖음.

Public partial class-> 보이지는 않지만 다른 코드 파일과 합쳐서 하나의 클래스가 된다.

ASP.NET에서 웹 폼의 처리 과정

* 웹 브라우저가 페이지에 대한 요청을 IIS 웹 서버에 보냄
* 웹서버는 요청한 aspx 파일 확장자가 ASP.NET 등록 되어 있는지 판단. Asp, html일 경우 참여 안함
* 처음 요청 된 것이면 ASP.NET은 자동으로 응용 프로그램 영역을 만듦. 모든 코드 컴파일 후 컴파일된 파일 캐싱. ASP.NET은 페이지의 컴파일된 버전 재사용
* BMI.aspx를 마치 일반 프로그램 처럼 행동. Page Load와 같은 이벤트 발생. 메모리에 있는 .NET 객체와 함께 동작
* 코드 실행 끝나면 ASP.NET은 적절한 html태그로 표현 되었는지 확인
* 최종 페이지는 사용자에게 보내지고 응용 프로그램 종료.
* 요청-> 등록된 파일? -> 응용 프로그램 생성? -> 요청된 페이지 컴파일? -> 페이지 초기화, 페이지 이벤트 발생, 이벤트 처리기 실행-> 한 컨트롤씩 HTML표현->응답

Id.Items.Add() Clear()

IsPostBack-> 처음 열리는 페이지-false 컨트롤 이벤트 처리한 뷰 상태 정보 담은 페이지-true

서버 컨트롤 또는 Response 객체의 메서드를 통해 만들어진 HTML 페이지는 buffer에 저장된다. 이 페이지는 웹 폼 실행이 끝나면 사용자에게 전송 됨. Response.Write() / Request.PhysicalApplicationPath

->Flush() 현재 버퍼에 있는 내용 사용자에게 전달. 이로 인해 사용자 안 지루하게.

다른 페이지 이동-> Response.Redirct() Server.Transfer()

Page Class 주요 이벤트

* Init : 요청 처리시 필요한 객체와 변수 초기화
* Load : 모든 컨트롤이 로드, 초기화, 속성 설정
* Control Events / Control Event Handler() 버튼 클릭, 텍스트 변경 등 컨트롤 이벤트 처리
* PreRender Page\_ HTML 문서 생성 전 최종적으로 컨트롤 갱신
* Nload 요청 처리하는데 사용한 객체와 변수 자원 반환.

같은 ASP.NET 응용 프로그램에 있는 페이지들은 자원, 환경 구성 등 공통적 내용 공유.

ASP.NET 파일 종류

* aspx : 웹 페이지 파일. 서버 컨트롤 제공
* ascx : 사용자 정의 컨트롤 파일. 사용자 직접 접근 불가
* web.config : XML. 개별화된 보안 설정, 상태 관리, 메모리 관리. 환경설정configuration
* global.ascx : 전역 응용 프로그램 파일. 전역 변수, 전역 이벤트에 응답

Application\_EndRequest : 사용자에게 보여주기 직전. 요청 시 페이지 코드 처리된 후 실행. 여기에 쓰면 모든 페이지에 바닥글 같은 거 추가 가능(공통 사항 추가)

* cs : 코드 숨김 파일. C#

ASP.NET 응용 프로그램 폴더

* App\_Code : 동적으로 컴파일 되는 소스 파일
* App\_GlobalResources : 모든 페이지가 접근할 수 있는 전역 자원 //local
* App\_Data : 텍스트 파일, SQL DB File
* App\_Theme : 표준화되고 재사용 가능 테마
* Bin : 컴파일 된 .NET 컴포넌트 DLL이 저장. 이곳에 있는 어셈블리 탐지해서 모든 페이지에 사용할 수 있도록 함.

LinkButton-> 하이퍼링크 형식

[컨트롤 클래스].[이벤트] += new System.EventHandler([이벤트 처리기]);

정규식을 통해 검사도 가능

유효하지 않으면 Page.isValid -> false . 성공-> Response.Redirect /ErrorMessage 속성

다른 페이지로 이동하면 데이터는 소멸 된다.

**Session**-> Server 쪽에 저장. 네트워크 환경에서 정보 교환을 위한 논리적인 연결. View State랑 방식 같음. 복잡한 저장이나 철저한 보안이 요구되는 경우. 서버의 메모리에 데이터 저장. 데이터는 사용자에게 전달 되지 않음. 성능 저하 될지도 Session[“data”]. 전역적 사용 가능

* 세션 상태 잃는 경우 : 웹 브라우저 닫거나 다시 시작. 같은 페이지 웹 브라우저의 다른 창 통해 접근. 일정 시간동안 요구 없음. Session.Abandon() 호출해서 세션 종류

Session ID-> 식별자. 클라이언트와 서버 사이 연결에 대한 정보. 이거 전달하는데 쿠키 사용 하거나 URL에 포함.

**Cookie**-> client에 저장. 4096 바이트. 사용자 접근 가능. Response.Cookies[“key”].Value = 정보;

Request.Cookies[“key”].Value로 읽어 옴. 삭제하려면 만료 기간 지난 시점으로 쿠키 설정.

간략한건 **View State**에 저장.

상태 관리-> 응용 프로그램에서 정보 저장, 관리하는 방법 ViewState[“Note”] 페이지의 객체

DB 사용-> connection -> cmd(명령문,conn) -> conn.Open(); SqlDataReader = cmd.ExecuteRader()->rd.Read()->rd.Close()->conn.Close()

마스터 페이지-> 웹 사이트에서 공통적으로 사용되는 구조 정의하는 새로운 개념의 페이지.

Asp:ContentPlaceHolder 이 안에 들어갈 콘텐츠는 asp:Content

인증 Authentication : 사용자가 등록된 사용자인지 판단. 인증 안되면 익명 사용자.

권한 부여 Authorization : 특정 페이지 보거나 글쓰는 것과 같은 인증된 사용자가 할 수 있는 행동 결정하는 과정. /asp.NET-> 폼 인증(텍스트, 외부 service), 윈도우 인증(윈도우 사용자 정보)configuration deny user=”?” 익명 / 비회원이 볼 수 있도록 Public 폴더 만들기

포트폴리오

c언어 바이너리 파일 입출력-> fread, fwrite

자바-주사위 게임 : 사용자 화면 전환 클래스. 시작화면에서 몇 명의 플레이어가 있을지 선택하는 클래스. 플레이할 캐릭터 선택하는 클래스. 게임 기능 총집합한 클래스. 캐릭터 정보 가지고 있는 클래스. 캐릭터의 움직임은 쓰레드로. 캐릭터 그려주는 클래스. 주사위 랜덤 값을 구해 애니메이션을 보여주는 클래스. 함정, 게임 끝 클래스.

Unity 2D :

규칙을 정해두고 그에 맞게 코딩을 했다 했는데 어떤 규칙?

* 배열의 시작은 항상 0으로. 가독성을 위한 변수 이름 선정.

벡터값을 이용해 아래로 내려가게 하고 자이로센서 x축 값을 받아와 회전 시킴.

퍼즐 풀 때 게임오버가 있기 때문에 항상 리셋이 있었음.

Web Server

동시 접속은 어떻게 했나?

* 쓰레드로 구현. 각 쓰레드 안에서는 데이터 원할히 주고 받을 수 있게.

소켓으로 리퀘스트 메시지를 받고 GET 메소드 뒤 url 추출. 버전, 서버 주소 추출. 요청한 url이 없으면 index.html 전송. File\_read로 파일 읽어오고 헤더 사이즈 만큼 메모리 할당 후 응답 보낼 packet을 만듦. 헤더 사이즈 메모리, 200, requestVersion, buf\_file, “index.html” 로 패킷 만들기. Header 전송 후 file 전송한다. 보안이 필요한 경우는 Request 메시지부터 Authentication 추출 후 base64형태의 비밀번호와 같은 지 확인. 없으면 400 Bad request이지 확인. 301 moved permantly-> 다른 URL로 데이터가 이사. 404 not found 뒤를 다 NULL로 보냄. 그 후 close socket.

ASP.NET

지도 API는 자바스크립트를 사용함. 테마도 자바스크립트로 적용 시킴. 병원 정보를 가져오는 것은 C#내에서 수행. url에 query string을 붙이고 webclient를 통해 string을 다운 받는다. 그것을 XDocument로 파싱하고 item 키워드의 자손들 XElement로 추출해서 value를 얻어낸다.

Multicasting Chatting

멀티캐스팅 IP 주소를 입력하고 포트 번호를 입력하면 일치한 사람들끼리 같은 네트워크 상에 있다면 대화할 수 있다. 자신과 같은 대화명을 가진 상대방이 말을 걸으면 강제적으로 1대1이 되어 이름을 바꾸게 한다.

멀티캐스트-> 한 번의 송신으로 메시지나 정보를 목표로 한 여러 컴퓨터에 동시 전송.

일반적인 전송 프로토콜은 사용자 데이터그램 프로토콜 UDP다. UDP는 전송 과정에서 데이터그램이 누락되거나 순서가 바뀔 수 있어 신뢰성이 낮다. 그래서 손실을 감지하고 재전송하기 위해 PGM과 같은 신뢰 가능한 멀티캐스트 프로토콜이 개발 되었다.

Mode를 설정해서 ip, port,대화명 설정, 대화모드, 1대1

이벤트 생성-> 대화상자 생성-> 이벤트 제거-> 종료

대화상자 프로시저에서 사용자의 입력에 따라 기능 수행. Ip, port, 대화명 설정하면 대화 시작. 소켓 스레드 두개 생성.

WaitForSingleObject로 write, read 이벤트 기다리게 해서 두 스레드가 혼동이 되지 않게?

데이터 보내는 스레드-> 송신용 소켓 생성. AF\_INET , to.sin\_addr.s\_addr=inet\_addr

소켓을 UDP로 설정. Ip, port 번호 설정. Write 이벤트가 발생하면 사용자가 데이터 입력했다는 뜻으로 데이터 보내는 작업 시작. 보낼 배열에 대화명, 대화 내용, 식별 아이디(threadID)을 \r\n을로 구분시켜 배열에 넣고 보냄. 만약 같은 대화명 가진 사람이 보내면 특정 명령어+랜덤 포트 번호+본인의 대화명 합쳐 데이터로 보냄. 그 후 1대 1 모드로 바꾸고 1대 1 대화를 위한 스레드 두개를 생성하고 현재 대화 스레드 두개는 종료 시킴

데이터 받는 스레드->

소켓 UDP로 설정. Ip, port 번호 설정. setsockopt으로 멀티캐스팅을 위한 가입. 데이터를 받으면 \r\n단위로 쪼개고 대화명, 대화 내용, 상대방 식별 아이디로 구분. 본인이 보낸 데이터가 아니고 대화명이 안같으면 그대로 출력. 같으면 대화명이 겹친다는 메시지를 띄우고 모드를 바꾼다. 데이터를 받았을 때 특정 명령어를 받았으면 1대1 대화 모드인 것이고 내 대화명과 같으면 1대1 대화를 위한 스레드 두개를 생성한 후 현재 스레드 두개는 종료.

1대1 대화 스레드 두개는 사용자가 대화명을 바꾸면 멀티 대화 모드로 변경.

받는 스레드.

영화예매 프로그램

사용자 용으로 영화, 장르, 영화관, 음식, 상영날짜를 검색할 수 있고 영화 예매 내역을 수정, 삭제 가능. 검색 기능 제공.

관리자 용으로 영화 정보 입력, 수정 가능하고 사용자의 구매내역을 삭제할 수 있음. 검색기능으로 특정 상황 별 인기 많은 영화 검색 가능.

콘솔 기반으로 CMD 형식인 프로그램.

구조체들을 만들어서 SQL 명령문에 대한 결과들을 받아오게 함. EXEC SQL 사용. CURSOR 만들고 명령어 입력하고 SQL OPEN -> SQL FETCH cursor INTO :구조체;

자기소개- 어떤 강점을 가졌는지 지원 이유 입사 후 어떻게 회사에 기여할 것인지

지원 회사와 연관된 내용

안녕하십니까. 저는 이번 웍스모바일 인턴에 지원한 장연수입니다.

* 담담하게 본인 역량 어필-> 꼬리 질문-> 호감으로 매력.
* 저는 ~역량을 가지고 있습니다. 이러한 경험을 통해 ~역량을 갖게 됐습니다. 이 역량을 토대로 이 회사에 기여하겠습니다.
* 역량 = 지식+기술+태도
* 어떤 기술, 지식을 가지고 있고 지식을 획득할 수 있는 태도를 가지고 있다. 이를 토대로 어떤 사업에 기여하고 싶다.
* 역량 + 가치관 이야기도 괜찮음. 가치관을 가지게 된 계기는 ~ 경험때문입니다.
* 너무 욕심 내지 말고 있는 그대로를 보여주는게 좋기 때문에 내가 평소에 쓰는 언어를 사용하는 것이 좋다.

습득력(이해력), 적용력, 팀 커뮤니케이션 능력

인턴으로 뽑아주신다면 항상 배움의 자세로 임하겠습니다.

------------

안녕하십니까. 저는 이번 웍스모바일 기술 부분 인턴에 지원한 장연수입니다.

저는 빠른 습득력과 적응력을 가지고 있습니다. 이를 토대로 전산 기초에 대한 지식을 쌓아 올렸습니다. 그리고 학부 시절 많은 프로젝트를 통해 여러 컴퓨터 언어를 익혔고 팀 커뮤니케이션 능력을 향상시킬 수 있었습니다. 저는 복수전공으로 엔터테인먼트소프트웨어 융합전공을 하였습니다. 이 과의 수업은 전반적으로 각기 다른 과의 사람들이 모여 프로젝트를 한학기동안 진행하는 것이었습니다. 저는 프로젝트에서 주로 개발자의 와 팀장 역할을 맡았습니다. 수업마다 다른 컴퓨터 언어로 코딩을 하거나 프로그램을 쓸 경우가 많았는데 수업의 특정상 제대로 배우지 못했습니다. 그래서 프로젝트를 진행하면서 스스로 찾아가며 공부했고 팀이 원하는 결과물을 팀원들과 함께 만들 수 있었습니다.

저의 이런 습득력과 적응력, 소통 능력을 토대로 웍스모바일에 기여하고 싶습니다.

저는 학교에서 많은 프로젝트들을 진행했습니다. 거기서 기억에 남는 것은 유니티로 게임을 만드는 수업이었습니다. 저는 개발자였고 디자이너, 기획자 이렇게 모여 팀을 이뤘습니다. 그 수업에서 처음으로 Unity, C#을 접했는데 시간 관계상 기초만 배우고 바로 프로젝트에 돌입했습니다. 처음에는 낯설고 두려웠지만 팀에서 필요로 하는 기술을 기준으로 공부를 하다 보니 자신감이 생겼고 구현을 잘 할 수 있게 되었습니다. 제가 구현한 게임을 팀원들이 플레이하면서 게임의 난이도, 디자인을 조정했습니다. 그 결과 좋은 퀄리티의 게임을 만들 수 있었습니다.

저는 이처럼 저의 적응력과 소통 능력을 가지고 웍스모바일에 기여하고 싶습니다.

지원 동기

* 왜 기업이 나를 뽑아야 하나? 회사도 우리에게 vision을 원한다
* Vision-> 내가 기업에 쓰일 수 있는 사람인가. 열심히 할 수 있는 동기.
* 그 기업에서 무언가를 하고 싶다. 입사해서 어떠 사업에 기여하고 싶다.
  + 사회, 경제, 문화의 변화에 대한 이야기
  + 해당 기업의 사업 비전
  + 비전을 실현시킬 나의 역량.
* 현대 사회는 ~문제점을 갖고 있습니다 -> 하지만 ~기업에서는 ~문제를 해결하기 위한 ~사업을 하고 있습니다.-> 저는 저의 관심과 능력을 바탕으로 ~기업 ~사업을 통해 ~문제점을 해결하고 싶습니다.

학창시절 네트워크에 대해 배우면서 처음으로 서버에 대해 접하게 되었습니다. 네트워크의 구조에 대해 알아가고 어떻게 통신하는지 알아갈수록 더욱 흥미가 가게 되었습니다. 멀리 떨어져 있는 사람들과 서로 이어준다는 것이 매력적이었습니다. 이 점이 제가 웍스모바일 인턴에 지원한 이유입니다. 웍스모바일의 인턴에 참여하면 제가 하고 싶었던 개발을 할 수 있을 것이라 생각하기 때문입니다.

회사 내 비효율적인 업무가 문제. 회사 내 업무를 효율적이게 해줄 라인 웍스가 이 문제를 해결해 줄 것이라 생각.

웍스 모바일에서 제공해주는 서비스 중 라인웍스는 기업을 대상으로 한 서비스입니다. 그만큼 보안이 중요하고 오류가 없어야 한다고 생각합니다. 이를 위해서는 꼼꼼함과 이해력이 필요하다고 생각합니다. 오류나 보안 문제를 체크할 때 빠짐없이 할 수 있는 꼼꼼함과 더불어 프로그램을 이해한 것을 바탕으로 생길 수 있는 문제를 스스로 찾아 보완해야하기에 이해력도 중요하다 생각합니다. 저는 이 두가지에 자신이 있기 때문에 지원하게 되었습니다.

마무리 멘트

* 내가 못다한 이야기를 풀어내는 시간
* 마지막에 멘트를 꼭 해야 하는가? 모든 걸 쏟아 붓지 마라
* 내 모든 걸 보여주려 하며 과한 행동 부담. 면접에서 굳이 힘주지 마라.
* 대화의 point : 얼마나 성실한가? 상대방에 성실해라. 면접은 시험이 아니다!! 대화다!

Java final 개념

* 해당 객체가 오로지 한 번 할당될 수 있음을 의미. 항상 상수이지는 않음.
* 메소드 -> 오버라이드하거나 숨길 수 없음.
* 클래스-> 상속할 수 없음. 상속 계층 구조에서 마지막 클래스.

Static

* 멤버 변수 : 클래스 변수, 같은 메모리 공유, 특정 인스턴스에 종속 안됨. 인스턴스 만들지 않고 사용 가능
* 메소드 : 클래스 메소드, 오버라이드 불가. 상속 클래스에서 보이지 않음.

클래스 설계 시 static 사용지침

* **클래스의 멤버변수 중 모든 인스턴스에 공통된 값 유지시켜야 하면 static 붙여주기**
* 메서드 중 인스턴스 변수를 사용하지 않는 메서드에도 고려

Static final 쓰는 이유-> 프로그램 처음부터 끝까지 바뀌지 않는 논리. 인스턴스가 만들어질 때마다 새로 메모리를 잡고 초기화시키지 말고 쭉 쓰기.

항상 쓰지는 않음. 각 인스턴스마다 서로 다른 final 멤버 변수를 생성자에서 초기화시키는 경우 static 사용 안함

http 통신 개념 : Hyper Text Transfer Protocol

* WWW상에서 정보를 주고받을 수 있는 프로토콜. TCP, UDP, 포트번호 80
* 클라이언트와 서버 사이에 이루어지는 요청, 응답 프로토콜
* 클라이언트인 웹 브라우저가 HTTP를 통항 서버로부터 웹페이지나 그림 정보 요청-> 서버는 이 요청에 응답. 필요 정보를 해당 사용자에게 전달.
* Server, Client Model
* 이거 자체는 connectionless. 연결 하고 request-> response 후 접속 끊음. 그래서 연결 정보 저장 위해 쿠키나 세션이 나옴.
* HTTP 요청메시지-> 요청 라인, 헤더, 바디. 헤더와 바디 사이 구분 위한 공백
  + 요청라인-> 요청 방식Method-GET,POST, URL, HTTP Version
* HTTP 응답메시지-> 상태 라인 Status-Line, 헤더, 바디. 헤더와 바디 사이 공백
  + 상태 라인-> HTTP Version, 상태 코드-200, 응답 이유-OK
* HTTP1.1 -> Connection 당 하나의 요청을 처리. 동시 전송 불가능. 응답 순차적.

Head Of Line Blocking-> 특정 응답 지연. Pipelining을 사용하는데 1번째->2,3 번째 응답이 가는데 1번째 응답 처리 완료때까지 2,3은 대기해야함.

RTT 증가-3 way handshake, 무거운 header 구조

* HTTP2 : Multiplexed Streams 한 커넥션으로 동시에 여러 개 메시지 주고 받음. 응답은 순서에 상관없이 stream으로 주고 받음.

리소스간 우선순위를 설정해서 css파일이 image파일보다 늦게 안오도록.

Server Push-> 서버는 클라이언트의 요청에 대해 요청하지 않은 리소스도 마음대로 보내줄 수 있다. 안그러면 HTML 요청하고 나중에 CSS요청하고 이래야함.

Header 정보 압축.

학부생 시절 가장 재미있게 들었던 과목

그 과목 중 자신 있는 부분을 면접관에 설명해라. 네트워크, socket 프로그래밍, 앱 개발 쪽으로 하는게 좋을 듯

네트워크, OS 질문

소프트웨어 공학, 데이터베이스, 운영체제, 네트워크, 알고리즘 -> 로우 레벨에서 돌아가는 것 까지 알아야함(변수는 메모리 어디에 저장 되는가?)

왜 개발자가 되었는지

* 처음 컴퓨터공학과를 선택한 것은 아버지가 관련 일을 해서 친숙해서 였는데 정확히 어떤 것을 배우는지 몰랐다. 처음 c언어 수업을 배울 때 재미있음을 느꼈다. 무에서 유를 창조하고 내가 원하는 프로그램을 만들 수 있다는 매력이 있었다. 그 후 개발자가 되겠다고 꿈을 꾸고 여기까지 달려왔다.

원하지 않는 일을 시키면 어떻게 해야 할지

* 학부 시절이라면 하기 싫은 일을 피하려고 했겠지만 직장에서는 일에 대한 책임이 있기 때문에 원하지 않더라도 피하지 말아야 한다고 생각합니다. 왜 원하지 않는지에 대해 생각하고 해결하는 방안을 생각할 것입니다.

야근을 해도 되는지

* 업무를 일과시간 내에 끝내는 것이 좋겠지만 입사 초기에 업무에 숙달되지 못했다면 야근을 해서라도 저의 업무를 마무리 짓겠습니다.

쿼리의 반응속도를 빠르게 하려면?

멀티스레드 환경에서 신경써야 하는 부분?

* 데이터 동기화. 임계 영역 만들기-> 오직 한 스레드만 접근 허용하게 함. 세마포어-상호배제에 기반. 세마포어의 값이 0이면 자원에 접근할 수 없도록 하고 0보다 크면 접그하고 동시에 1을 감소. 종료하고 나갈 때는 1증가시켜 다른 프로세스 접근 가능.

다른 직무?

-> 저의 역량을 봐주셔서 감사합니다. 하지만 저는 ~ 능력에 자신있어 기술 직무에 일을 하고 싶습니다. 혹시나 저의 역량이 그렇게 평가 되서 배치를 받게 된다면 근무는 해보겠지만 추후에 기술영역도 일해보고 싶습니다.

다른 곳 근무?

-> 집 떠나서 근무하는 것은 굉장히 어려울 것이라 생각을 해봅니다. 하지만 제가 교환학생으로 집을 떠나 생활해 보면서 힘들었지만 요리에 자신이 생겼습니다. 다른 곳에 발령이 난다면 그 부분만큼은 자신 있을 것 같습니다.

-프로그램 실행 과정

Fetch : 메모리 상에 존재하는 명령어를 CPU로 가져옴 -> Decode : 가져 놓은 명령어를 CPU가 해석 -> Execution : 해석된 명령어의 명령대로 CPU가 실행

전처리기->#으로 시작하는 지시자의 지시에 따라 소스코드 적절히 변경하는 작업

컴파일러 -> 소스코드가 어셈블리 코드로 번역

어셈블러->CPU가 이해할 수 있는 2진 코드로 바꿈

링커->프로그램 내 참조하는 함수, 라이브러리들 하나로 묶는 작업

실행파일->명령어들이 들어가 있고 메모리 공간에 올라가고 난 다음 CPU에 의해 실행

DataBase

자료 Data-> 현실세계에서 측정이나 관찰을 통해 얻은 처리되지 않은 데이터

정보 information-> 자료를 처리하거나 분석해서 값이나 지식을 얻은 것.

데이터베이스 -> 공용, 운영, 통합(중복 없다), 저장 데이터

* 특징 : 데이터 중복 최소화, 계속적인 변화, 실시간 접근, 동시 공유, 내용에 의한 접근, 뛰어난 자료간 연계성
* 속성 Attribute : DB 구축하는데 가장 작은 논리적 단위
* 개체 Entity : 정보를 나타내는 논리적 단위. 하나 이상의 속성 조합. 개체 타입, 어커런스

스키마 -> DB의 전체적 논리 설계. 관계들이 갖는 제약조건 정의. Meta data, data dictionary.

* 외부스키마, 개념스키마, 내부스키마

DBMS -> 데이터 집단, 데이터 관리하는 프로그램 집단. 데이터 정의, 조작, 제어 기능.

데이터베이스 언어-> .데이터 정의어, 조작어, 제어어

개체관계모델 E-R Model : 개체 내 관계, 개체와 개체 사상관계.1대1, 1대N 등

논리적 데이터 모델-> 관계형, 계층형, 네트워크 데이터 모델

이상 현상-> 관계 스킴-데이터 논리적 구조이 잘못. 데이터 중복. 삭제, 삽입, 갱신 이상.

관계형 데이터베이스

* 도메인, 튜플-하나의 행, 릴레이션-한 개 이상 속성들.

무결성 제약 조건 : 정확성 부여

* 도메인 무결성-값들이 속성 정의 각 속성 값이 반드시 원자 값이어야 한다.
* 개체 무결성-기본키 따라 연산 중복 없음. 기본 키를 구성하는 어떤 속성도 NULL을 가질 수 없음. 튜플 고유하게 식별 불가하기 때문.
* 참조 무결성-외래 키의 값은 참조 릴레이션의 어떤 튜플의 기본 값과 같다. 외래키가 자신을 포함하고 있는 릴레이션의 기본 키를 구성하고 있지 않으며 널 값을 가진다.

SQL-> 관계 DB를 위한 표준질의어, 비절차적 Data 언어.

SELECT [ALL | DISTINCT] 속성리스트(,)

FROM 테이블

[WHERE 조건식]

[GROUP BY 속성..[HAVING 조건식]]

[ORDER BY 속성.. [ASC|DESC]];

SELECT, HAVING 에서만 집계 함수 쓸 수 있음. COUNT, MAX, MIN, SUM, AVG / 속성 IN, EXISTS

트랜젝션-> DB 연산들 모아 놓은 것. 논리적 작업 단위.

* 특성 : 원자성- 트랜잭션 구성하는 연산들 모두 실행 시 all or nothing,
* 특성 : 일관성-트랜잭션 수행 후 일관된 상태 유지
* 특성 : 격리성, 고립성 – 수행 중 트랜잭션 연산 결과에 다른 트랜잭션 접근 못함.
* 특성 : 지속성, 영속성-성공적 완료 후 DB에 반영한 결과는 영구적
* 연산 : commit-완료, rollback-실패, 작업 취소

회복 연산 : Dump-> 다른 곳에 백업, log-변경 전 후 값 기록, redo-전체 복구, undo-로그 사용 연산 취소 / 즉시갱신, 지연갱신, 검사시점, 미디어 회복 기법

2단계 로킹 규약-> 직렬 가능성 보장, 데드락 발생. 로킹 기법-> 공용 : read, 전용 : write

OS

OS의 목표 -> 프로그램 편리하게 사용, 하드웨어를 효율적이게 사용.

* 사용자로부터 하드웨어 디바이스 특징 숨기기

OS 정의-> resource allocator-하드웨어를 효율적이고 공정하게 나눈다, control program.

컴퓨터 시스템 구조-> 하드웨어, OS(하드웨어 사용Control & Coordinate-조직화)

CPU – main memory <-> local buffer로 데이터 이동

Device controller->CPU에게 operation 끝을 알림. Interrupt로.

Context Switching-> interrupt 다른 프로그램 수용 중 interrupt 발생하면 바꾸는거.

Main Memory-> CPU가 접근 가능한 큰 저장장소. Random Access, 휘발성. 명령어, 데이터 저장.

Caching-> 속도 차이 줄이기 위함. Information 임시 복사해두고 요청하면 여기서 정보 줌.

Kernel-> 운영 체제의 핵심 부분. 응용 소프트웨어를 컴퓨터 하드웨어에 연결 시켜줌. 컴퓨터에서 항상 돌아가는 프로그램.

OS Structure

* Multiprogramming – 효율성. Job scheduling-데이터 처리 순서 정함
* TimeSharing-Multitasking – 많은 작업들이 한번에 실행되는 것 처럼 보이게 하기 위해 짧은 시간에 switch. CPU Scheduling-CPU가 어떤 작업 할지. Virtual Memory-프로그램 일부만 넣어서

User Mode = 1-> CPU<-> Memory , I/O

Kernel mode = 0-> 여기서만 실행 되는 애들 있음.WRITE같은거. OS에서만

Process Management-> Process : 활동중인 객체. <-> Program : passive entity

* Single thread-> Program Counter 1개 . 다음 실행될 명령 위치. 다중 쓰레드는 쓰레드 당 한 개의 program counter..
* Process-> text-code, program counter, processor register, stack(임시 데이터, 함수), heap-malloc, data section
* Process 여러 개면 linked list로 연결. Scheduling이 골라서 수행.

Memory Management-> 언제 무엇이 메모리에 있을지 결정. Bit map으로 빈공간 알아내기

* Data- processing 전 후에 memory에, Instruction- 실행 위해 메모리에
* 누구에게 사용되는지 track 유지. Process & Data를 memory에 in out 할지.
* \*Address Binding-> program내 주소는 0부터 시작. physical에서는 OS가 0부터 시작.
  + Absolute address(compile 후 주소 0. CPU에 의해 생성) -> physical address(실행 될 때). 맨날 주소 바뀜.
  + Binding Time-> compile, load때는 볼 필요 없음. 다 볼거 아님. Execution 일 때만 봄.
  + CPU가 메모리 통해 접근할 수 있도록 해줌.
  + Disk->Main Memory->Cache->HW register
* 메모리 나누다가 아주 작은 빈 공간 생기거나-외부 쓰고있는 block 내 빈 공간-내부.
* Segmentation-> 크기 가변적. Table에 base, limit. 외부 단편화
* Paging-> 메모리를 고정된 크기로 나눔. 내부 단편화. Page와 Frame.

Storage Management-> 누가 무엇을 접근할 수 있는지 제어. File의 local view 제공.

* OS에서 information storage의 logical view를 제공한다
* File은 물리적 요소들을 논리적 storage unit으로 추상화 시킨다.
* Cloud로 발전해서 이제 파일이 어디 있는지 신경 안씀
* 파일들 만들고 삭제, 파일을 secondary storage로 mapping.
* 파일은 연속적이고 논리적인 주소 공간
* 디렉토리는 파일의 정보를 저장.

Mass-Storage Management->크거나 영구보관 해야하는 거 관리

* Free space Management, Storage 할당, Disk Scheduling-track에 헤드 접근 시간? 디스크 헤드 움직이는 경로
* Disk Scheduling : 찾는 시간 최소화 시키기. SCAN-시작부터 0->맨끝.

Magnetic disk->Main Memory->cache-> hadware register

Buffering-> 임시 저장, caching-> performance 향상 위함. Spooling->프린터기에 보내기 전 저장.

Protection-> 사용자나 프로세스의 접근을 control

Security-> 외부나 내부 공격 막기

OS 서비스 제공 기능

* User Interface, Program Execution, I/O Operations, File system Manipulation, Communication, Error detection

System Call-> OS와 유저, 프로그래머 사이. High-level language로 쓰임. API-더 쉽게 만들어줌. API를 통해 direct한 call 대신으로 씀.

* 운영 체제의 커널이 제공하는 서비스에 대해 응용 프로그램 요청에 따라 커널에 접근하기 위한 인터페이스.
* 커널 <-> 사용자모드
* 유형 : 프로세스 제어, 파일 조작, 장치 관리, 정보 유지, 통신.

Parameter Passing-> register를 통해 파라미터 전달. 프로그램에 의해 스택에 들어가고 OS에 의해 pop 됨.

Process State

* New -> ready <- waiting(interrupt)-> running -> terminated
* 하나의 프로세스만이 CPU를 사용할 수 있다.
* Scheduler dispatch : ready 여러 개 중 하나 pick->running
* Running->ready갈 때 register에 PCB로 저장
  + Process Control Block : process state, number, program counter 등

Context Swtich-> 정보까지 저장해서 자리 옮기기.

Interrupt by other <-> System Call by myself

Overhead-> OS가 CPU쓰면 idle해짐. 실행 안 한 시간.

Process Scheduling-> CPU 사용 최대로 빠르게 프로세스 switch

* Scheduling queue : job queue(system에 있는 process), ready queue, device queue
* Queue header가 있고 linked list처럼 이어져 있음. Head, tail
* 프로그램 클릭-> process ready queue. 완전 종료->PCB 존재하지 않음.
* Long-term scheduler = job -> ready queue에 보냄. Short-term=cpu->CPU에 할당할 process 선택. 호출 빈번.
* Round Robin-> 일정 시간 내에서만 사용 가능. 못하면 다음 기회. Time interrupt 발생. 하나당1/n~n-1 기다림. 시간 길이 올릴수록 반응 시간 올라가고. 낮을수록 오버헤드 증가(context switch 많이 해서)

Critical section problem-> entry, exit section. 자원 공유 하는 부분

* Mutual Exclusion-상호 배제. 하나만 허용
* Progress-> empty면 사용 가능. empty이지 않게
* Bounded waiting-> n-1까지 최대 기다리게 하기.
* 세마포어,뮤텍스로 동기화.

데드락 ->두 개 이상 작업이 서로 끝나기를 기다리는 것.

* 4가지 조건 : Mutual Exclusion-상호 배제 프로세스들이 필요로 하는 자원에 대해 배타적인 통제권 요구, No preemption 비선점-자원 다 쓸 때까지 내놓을 수 없음, Hold and wait 점유대기-할당된 자원 가진 상태에서 다른 자원 기다림, circular wait-순환대기
* Prevention- 4가지 조건 중 하나 안일어나게, avoidance- 요청할 때 마다 체크, detection

Virtual Memory Management-> 프로그램이 필요한 부분만 적재.

* Demand paging, Segmentation / virtual memory->memory map->physical memory

Page Replacement-> 공간 부족 시 페이지 교체.

Thrashing-> 컴퓨터 작업 느려짐. 원인은 메모리나 기타 자원 고갈. 필요 연산에 비해 너무 적음. Page fault 때문일지도.

Controller-> 속도 조절 위해 버스로 제어.

Device Driver-> device detail 압축. 디바이스 관리.

Direct Memory Access-> CPU 개입 없이 memory<-> Disk local 버스 제어권으로 개입 없이 사용 가능.

SW -> Kernel , Kernel I/O subsystem->general, Device driver->specific

HW-> Device controller-어떤 전기 신호? , Device

객체지향

**\*\*객체지향-> 각 객체는 저마다 특징을 가지고 있는데 이 각 특징에 맞게 객체를 다루는 것을 말함..**

객체지향의 핵심인 클래스는 참조형에 해당하는 사용자 정의 데이터형.

New 키워드를 통해 힙에 데이터 저장 위한 메모리 할당. 할당된 힙 메모리의 실제 주소를 해시 함수 통해 구하고 해시 값을 스택 메모리에 지정. 객체/인스턴스를 선언하면 참조 값을 저장하기 위한 스택 메모리 할당.

객체지향 프로그래밍에서는 **실세계를 바탕으로 모델링하여 사용자 정의 데이터형인 클래스를 만든다. 실세계의 현상을 컴퓨터상에 객체로 모델화 함으로써 컴퓨터를 자연스러운 형태로 사용하여 다양한 문제를 해결하기 위한 프로그램기법. 객체라는 작은 단위로서 모든 처리를 기술하는 프로그래밍 방법.**

**작은 문제들을 해결할 수 있는 객체들을 만든 뒤 객체들을 조합해서 큰 문제를 해결하는 상향식 해결법.** 객체를 독립성, 신뢰성을 높게 만들어 놓으면 그 이후엔 객체를 수정 없이 재사용 가능.

특성정보-> field 멤버변수. 행동양식->method.

자식 클래스 is a 부모 클래스

* 캡슐화 encapsulation : 프로그램 세부 구현을 외부로 드러나지 않도록 특정 모듈을 내부로 감추는 것. 인터페이스 구현. 정보 은닉. 데이터와 데이터를 다루는 도구를 집합하는 그 자체.

만일의 상황(타인의 조작)에 대비해서 외부에서 특정 속성인나 메서드를 사용자가 사용할 수 없도록 숨겨놓는 것.

* 상속 inheritance : 부모 클래스의 특성과 기능을 그대로 물려받는 것을 말한다. 캡슐화를 유지하면서도 클래스 재사용이 용이하도록 해 줌. 오버라이딩 . 추상화를 구조화함.
  + Method Overriding-> 부모 클래스와 동일한 형태(입출력 동일)의 같은 메소드를 구현하면 자식 클래스의 메소드가 더 높은 우선순위를 갖게 된다.
* 다형성 polymorphism : 하나의 변수명, 함수명 등이 상황에 따라 다른 의미로 해석될 수 있음. 오버로딩. 같은 메소드에 대해 클래스에 따라 다른 행위를 할 수 있음.
  + Method overloading-> 자식클래스에 부모 클래스의 메소드를 생성할 때 입력항목만 다르게 하는 것.

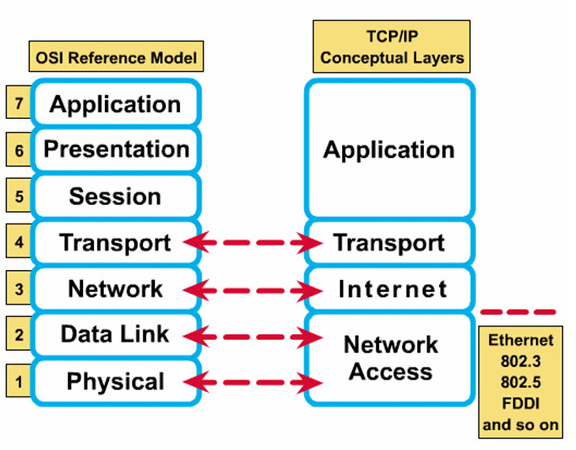
클래스-> 객체의 상태를 저장하는데 사용될 속성을 정의, 객체가 이해할 수 있는 메시지와 메시지에 응답하는 과정 정의. 메소드 정의. 즉 속성과 행위를 정의. 사용자 정의 데이터형.

네트워크

Packet switching-> queue, store and forward 전달될 때 노드들 사이에 개별적으로 경로가 제어. 짤막한 데이터 트래픽에 적합. 패킷을 일렬로 보내고 받고.

Circuit switching-> no sharing. 발신자와 수신자 간 독립적이며 폐쇄적인 통신 연결. 1대1 연결. 채널이라고도 함. 대역폭 낭비가 심함. Good for real time traffic

**OSI 7계층**

* 7 Application : DNS, FTP, HTTP. 응용계층, 사용자가 네트워크에 접근할 수 있도록 함. 우편 같은 서비스 제공. /message
* 6 Presentation : JPEG, MPEG. 표현, 데이터의 의미 해석. 하나의 표현 형태로 변환
* 5 Session : SSH, TLS. 세션. 통신 세션 구성. 포트 연결(양 끝단). 통신장치 간 동기화
* 4 Transport : TCP, UDP 전송. 양 끝단 사용자들이 신뢰성 있는 데이터 주고 받을 수 있게 해줌. 게이트웨이 /segment
* 3 Network : IP, ICMP 네트워크. Logical address를 담당. 여러 개의 노드를 거칠 때마다 경로를 찾아줌. 라우팅. 라우터 /packet, datagram
* 2 Data Link : MAC, Ethernet 데이터링크. Point to Point 간 신뢰성 있는 전송 보장. 물리적 계층을 통한 데이터 전송에 신뢰성 제공. 컴퓨터와 컴퓨터 간에 프레임 데이터를 전송. 이웃 network element들 간의 데이터 전송. 에러검출/재전송/흐름 제어. 브릿지, 스위치 /frame
* 1 Physical : Ethernet.RS-232C 물리. 물리적 매체를 통해 bit 흐름을 전송하기 위해 요구되는 기능 조정.
* 

바이러스 -> 자기 복제. 오브젝트를 받거나 실행하면 발생.

Worm-> 컴퓨터 시스템 파괴, 지연. 바이러스와 달리 다른 프로그램 감염시키지 않고 자기 자신을 복제하며 통신망을 통해 퍼짐.

Dos = Denial of Service-> DDos(distributed, 분산) 다른 사람들의 컴퓨터에 악성코드를 설치해 원격에서 제어가능하게 한다. 좀비피시. 특정 웹사이트를 공격. 서버 다운 시킴. 마비.

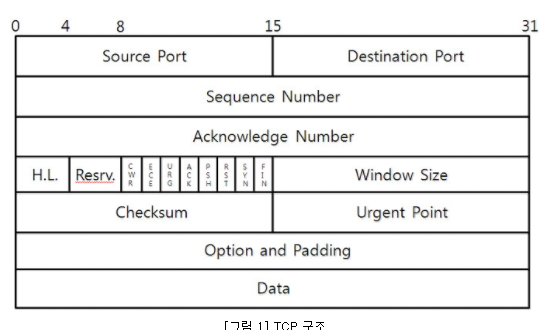
Packet sniffing-> 모든 패킷을 읽거나 기록함.

IP spoofing-> false source address로 패킷을 보냄.

Server-Client model, P2P architecture

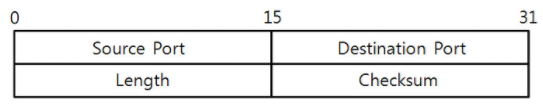
Hybrid of client-server and p2p. -> Voice Over IP에서 사용. 클라이언트끼리 p2p. 서로 알아내기 위한 건 서버. IP 찾아 줌.

Socket-> process(host와 같이 돌아가는 프로그램)는 메시지를 소켓을 통해 주고 받는다. 주고받기 위해서는 식별자가 필요. IP&Port Number

TCP-> 신뢰성 있음. 연결 지향성. Timing, 최소 throughput, security 제공 안해 줌. Data loss는 보장. 메시지 수신 확인. 보내진 순서 보장

SYN bit=1, seq=x->SYN=1, seq=y, ACK = 1, ACK num=x+1->ACK = 1, ACK num = y+1 /클라이언트->서버->클

UDP-> 신뢰성 없음. 모든 거 지원안해줌. 메시지 확인 불가. 도착 순서 예측 불가. 속도가 더 빠름. /DNS, IPTV, VoIP,게임



http 통신 개념 : Hyper Text Transfer Protocol / Application Layer

* WWW상에서 정보를 주고받을 수 있는 프로토콜. TCP, UDP, 포트번호 80
* 클라이언트와 서버 사이에 이루어지는 요청, 응답 프로토콜
* 클라이언트인 웹 브라우저가 HTTP를 통해 서버로부터 웹페이지나 그림 정보 요청-> 서버는 이 요청에 응답. 필요 정보를 해당 사용자에게 전달.
  + 클라이언트가 서버와 TCP 연결 개시-> 서버는 클라이언트로부터 TCP 연결 accept-> 브라우저와 웹 서버 간에 HTTP msg 주고 받기-> TCP연결 종료
* Server, Client Model
* 이거 자체는 connectionless. 연결 하고 request-> response 후 접속 끊음. 그래서 연결 정보 저장 위해 쿠키나 세션이 나옴. Stateless 정보 저장 안함.
* HTTP 요청메시지-> 요청 라인, 헤더, 바디. 헤더와 바디 사이 구분 위한 공백
  + 요청라인-> 요청 방식Method(GET,POST), URL, HTTP Version
* HTTP 응답메시지-> 상태 라인 Status-Line, 헤더, 바디. 헤더와 바디 사이 공백
  + 상태 라인-> HTTP Version, 상태 코드-200, 응답 이유-OK
* HTTP1.1 -> Connection 당 하나의 요청을 처리. 동시 전송 불가능. 응답 순차적.

Head Of Line Bloccking-> 특정 응답 지연. Pipelining을 사용하는데 1번째->2,3 번째 응답이 가는데 1번째 응답 처리 완료때까지 2,3은 대기해야함.

RTT 증가-3 way handshake, 무거운 header 구조

* HTTP2 : Multiplexed Streams 한 커넥션으로 동시에 여러 개 메시지 주고 받음. 응답은 순서에 상관없이 stream으로 주고 받음.

리소스간 우선순위를 설정해서 css파일이 image파일보다 늦게 안오도록.

Server Push-> 서버는 클라이언트의 요청에 대해 요청하지 않은 리소스도 마음대로 보내줄 수 있다. 안그러면 HTML 요청하고 나중에 CSS요청하고 이래야함.

Header 정보 압축.

* 연결에 2가지 있음
  + Nonpersistent HTTP : 하나 연결 당 하나의 오브젝트만. 하고 바로 끊음.
  + Persistent : 여러 오브젝트를 하나의 TCP연결로 보냄.
    - Pipeline 없으면 클라이언트는 새 요청을 전 요청이 서버에 받아 졌을 때만 보낼 수 있음.
    - Pipeline 있으면 HTTP 1.1 요청을 상대 응답에 상관없이 보냄.

FTP-> File Transfer Protocol. 원격 호스트 사이 파일 전송. Client(연결 재시), server(다른 호스트) 모델

* Client가 포트21로 FTP서버 연결 TCP -> control connection 권한 받음-> control connection 통해 원격 대상 직접 찾기-> 서버는 전송 파일 전송 명령 받고 두번째 TCP연결 오픈.-> 1개 파일 보내고 서버는 data connection(포트 20) 닫음.

전자메일

* 세가지 요소 : user agent-UA(mail reader), mail servers(mailbox 있음), SMTP
* SMTP Simple Mail Transfer Protocol
  + TCP 사용. 메시지는 7bit ASCII.
  + A는 UA를 사용해서 메시지 구성한다.-> A의 UA는 A의 메일 서버로 메시지를 보낸다. 메시지는 message queue에 저장-> SMTP client는 A와 B사이를 연결해주는 TCP connection을 오픈-> SMTP client는 메시지를 TCP통해 보냄-> B의 메일 서버는 B의 mail box에 메시지를 넣음-> B는 그의 UA에서 메시지를 읽음.
  + 여러 개의 오브젝트는 여러 개의 메시지로 보내짐
* Message format : header lines(TO: From: Subject: ), body
* Mail Access Protocol : 서버로부터 회수
  + PoP: Post Office Protocol, IMAP : Internet Mail Access Protocol, HTTP:gmail 등

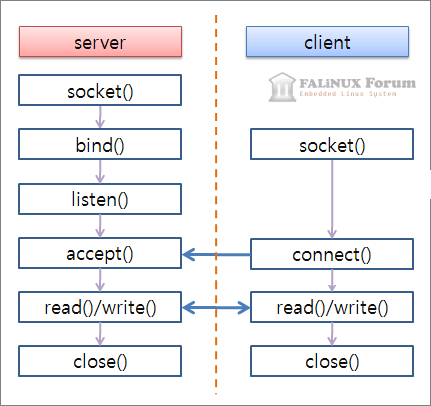
DNS-> Domain Name System. Host name을 ip address로 바꿔줌. 인터넷 도메인 주소 체계

* 도메인 이름을 IP주소로 변환하고 라우팅 정보를 제공하는 분산형 데이터베이스 시스템. 트리 구조.
* \*브라우저가 URL 입력하면 -> DNS Client가 호스트 네임 포함한 쿼리를 DNS Server에 보냄-> DNS Client는 호스트네임의 IP를 받음-> 브라우저는 IP 주소로 TCP 연결 재시.
* TLD Server : com, org, net 이런거. 탑 레벨 도메인. Local name server cached.

Socket programming -TCP

* Serversocket, client쪽에선 socket 만듦-> server socket.accept() 해서 새로운 소켓 만듦.

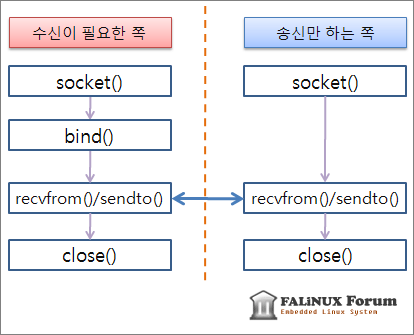
소켓에 ip, port번호

* 

Socket Programming – UDP

* Client, server사이 연결 없음.

Datagramsocket 만듦. 클라이언트 쪽에서 server ip랑 포트 번호 들어간 데이터그램 만들어서 보내기.

* 

Rdt-> Reliable data transfer. UDT에 신뢰성 더하기. 데이터에서 검사하고 패킷 만들어 보내고 다시 받을 때 또 검사.

* 3.0 : ACK를 일정 시간 만큼 기다리고 안오면 다시 재전송. ACK에 0, 1 붙여서 구문.

Pipelined Protocols

* Go-Back-N : receiver는 무조건 받은 애만 보냄. 그리고 주는 쪽은 time out되면 ACK 받은 숫자 후부터 보냄.
* Selective Repeat : sender는 무조건 ACK를 못 받은 것만 재전송.

IP->Internet Protocol

* 송신 호스트와 수신 호스트가 패킷 교환 네트워크에서 정보를 주고받는 데 사용하는 정보 위주의 규약(protocol). OSI 네트워크 계층에서 호스트의 주소 지정과 패킷 분할, 조립 기능 담당.
* V4 : 32 bit 식별자 .
  + 라우터는 여러 개의 인터페이스(호스트, 라우터와 물리적인 링크 연결)
  + 호스트는 한 개의 인터페이스
  + Subnet. 커다란 네트워크 분할. Class A,B,C. broadcasting, multicast
  + DHCP Dynamic Host Configuration Protocol- 네트워크 조인 할 때 호스트가 dynamically하게 ip address를 network server로부터 얻게 함. 브로드캐스팅 사용. Transaction ID로 누군지 알아내기.
  + NAT : Network Address Translation 한 개의 IP로 여럿이 쓰는 거
* V6 : 128bit 식별자.
* Tunneling : Ipv4->Ipv6로 바꾸는 법 ipv6안에 ipv4있는 모양.

Checksum -> 보내는쪽에서 데이터를 다 더하고 0->1, 1->0으로 바꿈. -> 받는 쪽에서 다 더하고 0->1 바꾼 후 검사합과 더해서 0이면 오류 없음.

MAC Media Access Control : 2개 이상 보내면 collision.

* MAC protocols : Channel Partitioning – TDMA(시간 나누기), FDMA(주파수)
  + Random Access Protocol-> 충돌 발견과 그에 따른 회복 필요. CSMA-idle 할 때 보내고 busy할떄 idle까지나 랜덤 시간 기다리기
* 네트워크 세그먼트의 데이터링크 계층에서 통신을 위한 네트워크 인터페이스에 할당된 고유 식별자. 48 bit.

Ethernet-> LAN에 제일 많이 사용. 데이터 링크 계층에서 MAC 패킷과 프로토콜 형식 정의.

* Connectionless(no handshaking), 신뢰성 없음. Unsorted CSMA/CD

스위치(link)-> 여러 개 동시 전송 가능. Full deplex. Store and forward , forwarding table

Base station-> 유선망과 무선망 연결.

Ad hoc mode-> 필요 따라서 소대 구성. Node들 스스로 네트워크 꾸밈.

알고리즘

알고리즘-> 어떤 작업을 수행하기 위해 입력을 받아 출력을 만드는 과정

* 효율성과 정확성을 중심으로 설계.

알고리즘 수행시간-> 입력 크기에 대해 수행시간이 어떤 비율로 소요되는지 표현.

* O 표기법.

재귀-> 같지만 크기가 다른 문제를 관계로 해결.

점화식-> 함수를 자신보다 더 낮은 변수에 대해 함수와의 관계로 표현.

선택정렬-> 가장 큰 원소를 A[n]과 교환. 가장 작은 원소를 A[0]과 교환

for(i=0; i<MAX-1; i++) {

min = i;

for(j=i+1; j<MAX; j++) {

if(arr[j] < arr[min]) min = j;

}

temp = arr[i];

arr[i] = arr[min];

arr[min] = temp;

}

버블정렬-> 인접한 원소끼리 크기 보면서 오른쪽보다 왼쪽이 더 크면 자리 바꿈. 선택정렬 같이 한번 할 때 마다 제일 큰 수가 뒤로 감.

* For(i=0;i<arr.length-1;i++){ for(j=0;j<arr.length-i-1;j++) j>j+1?}

삽입정렬->자료 배열의 모든 원소를 앞에서부터 차례대로 이미 정렬된 배열과 비교하며 자신의 위치를 찾는 것.

* For(i=1;i<length;i++){
* Temp=arr[i];
* J=i-1;
* While(j >=0 && arr[j] > temp){
* Arr[j+1] = arr[j];
* J=j-1;
* }
* Arr[j+1] = temp;
* }

병합정렬-> 반으로 나누고 전, 후반 독립적으로 정렬한 후 합치기.

* Mergesort(); 두번 하고 merge()하기. 임시 배열에 두 시점 비교하면서 작은 거 임시 배열에 넣기. 그후 남아있는 애들 넣고 복사. A, p,r일 때 p<r일때만

public static void mergeSort(int arr[], int l, int r) {

if(l<r) {

int mid = (l+r)/2;

mergeSort(arr, l, mid);

mergeSort(arr, mid+1, r);

merge(arr, l, mid, r);

}

}

public static void merge(int arr[], int l, int mid, int r) {

int i = l;

int j = mid+1;

int k = l;

int temp[] = new int[arr.length];

while(i<=mid && j<=r) {

if(arr[i] < arr[j]) {

temp[k++] = arr[i++];

}else {

temp[k++] = arr[j++];

}

}

while(i<=mid)

temp[k++] = arr[i++];

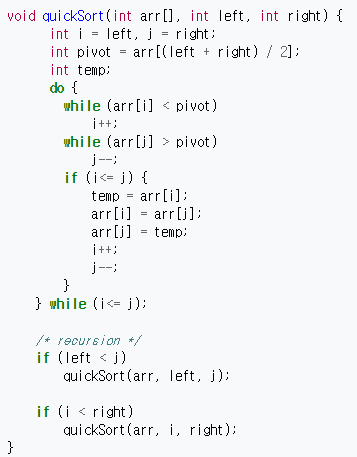
while(j<=r)

temp[k++] = arr[j++];

for(int index=l; index<k; index++)

arr[index] = temp[index];

}

퀵정렬-> 한 숫자 기준으로 그 숫자보다 작은거, 큰거 나눠서 정렬. 임시배열 필요.

* P<r일 때 partition해서 q값 얻어내고 quicksort q-1, quicksort q+1
* Partition(A[],p,r) {x=A[r]; i=p-1;

For(j=p to r-1){ if(A[j] <= x) then A[++i]<->A[j];}

A[i+1]<->A[r]; return i+1;

}

힙정렬-> 힙- 완전이진트리, 자식노드 작거나 같음. 부모노드는 자식 노드보다 큰 값.O(nlogn) – 직계 관계에서만 만족. Partial order부분 트리에서

* 알고리즘 : n개의 노드에 대한 완전 이진트리 구성. 루트 노드, 왼쪽 오른쪽 자식 노드
* 최대 힙을 구성(내림차순 정렬). – 부모 노드가 자식 노드보다 큰 트리. 단말 노드를 자식 노드로 가진 부모 노드부터 구성하며 아래부터 루트까지 올라오며 순차적으로 만듦.
* 가장 큰 수(루트에 위치)를 가장 작은 수와 교환.

private static void heapSort(int[] arr) {

// TODO Auto-generated method stub

int length = arr.length - 1;

maxHeapBuild(arr);

for (int i = length; i >= 1; i--) {

swap(arr, 0, i);

length-=1;

maxHeapify(arr, 0);

}

}

private static void maxHeapBuild(int[] arr) {

int length = arr.length;

// 마지막 노드의 부모 노드에서부터 시작

for (int i = length / 2; i >= 0; i--) {

maxHeapify(arr, i);

}

}

private static void maxHeapify(int[] arr, int i) {

int length = arr.length;

int leftChild = i\*2+1;

int rightChild = (i\*2)+2

int k = 0;

//자식 노드가 없을 경우에는 종료한다.

if(leftChild > length || rightChild > length) {

return;

}

//더 큰 Child를 k에 할당한다.

if (arr[leftChild] > arr[rightChild]) {

k = leftChild;

} else {

k = rightChild;

}

//부모 노드가 자식보다 큰 경우 종

if (arr[i] >= arr[k]) {

return;

}

//swap

swap(arr, i, k);

maxHeapify(arr, k);

}

public static void swap(int arr[], int i, int j) {

int temp = arr[i];

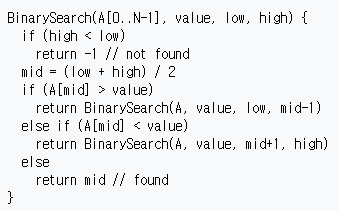
arr[i] = arr[j];

arr[j] = temp;

}

라딕스 정렬 -> 일의 자리 수 -> 십의 자리수 순으로 정렬. 0~9까지 큐에 순서대로 집어넣고 빼기

O(자리수\*n)

이진 탐색 -> 이진트리는 왼쪽이 더 작고 오른쪽이 더 큼

이진 검색트리에서 삭제 시 자식이 두개 있으면 삭제 후 오른쪽 노드에서 제일 왼쪽에 있는 애가 삭제된 곳으로 올라옴.-> 오른쪽 한번 가고 왼쪽으로 쭉 내려가기. 반대의 경우도 가능.

AVL 트리-> 모든 노드의 균형인수는 0,1,-1 중 1개.

해시테이블-> 원소가 저장될 자리를 원소 값에 의해 결정. 키값. 테이블에 키 값 저장.

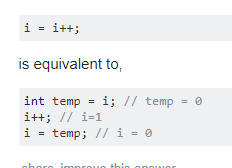
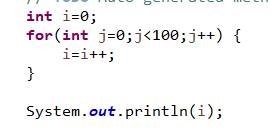
* 문제 : 충돌 발생. -> 체이닝. 같은 주소로 해싱하는 원소는 연결리스트로
  + 개방주소 방법: 다른 위치에 저장. 선형조사-> 충돌 시 다음 자리 후보로
* 조건 : 입력원소가 해시테이블 전체에 고루 저장. 계산은 간단히

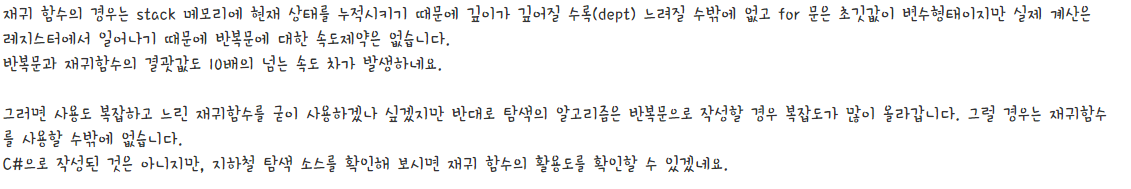
그래프-> 사물이나 현상 관계를 정점과 간선을 이용해서 모델링.

* 표현 : 인접 행렬, 인접 리스트-정점에 인접한 정점들 리스트로 표현.
* 탐색 : BFS-너비 우선 탐색. Queue-> 삽입되는 순서는 가까운 거리 방문 순서. FIFO
  + 방문 표시하고 큐에 넣고 큐가 비어있지 않을 때 까지 큐에서 빼고 그 것의 인접행렬 구하고 방문표시하고 enqueue
* 탐색 : DFS-깊이 우선 탐색. STACK, LIFO
  + 방문 표시 하고 그 것의 인접행렬만큼 for문. 방문 안했으면 다시 DFS
* 최소 신장트리 : 신장트리는 모든 정점이 연결된 트리. 간선 가중치 합 가장 작은거.
* 프림알고리즘 : 임의의 정점에서 시작-> 최소비용으로 연결 할 수 있는 다른 정점 연결-> 정점 연결하며 트리 확장-> 모든 정점 연결할 때 까지
* 크루스칼 알고리즘 : 가중치가 작은 간선부터 차례로 선택해서 트리 만들기.
* 최단 경로 : 다익스트라 알고리즘 -> 가중치가 모두 0 이상. 시작 정점에서 가장 가까운 정점(가중치가 제일 작은 연결-> 2,…,N번째 가까운 인접 정점 찾아 연결.
* 최단 경로 : 벨만-포드 알고리즘 -> 음의 가중치 허용. 한 정점 기준으로 간선 최대 1,2,,,n-1개 사용하는 최단 경로 찾기. 한 스탭에서 꼭 간선 다 봄. 최단 거리 업데이트 되는 노드 없어질 때 까지 반복. 음의 가중치로 인해 업데이트 무한히 되면 탈출.

자료구조

* 스택 -> LIFO – Last In First Out
* 큐 -> FIFO – First In First Out
* 링크드 리스트-> 각 노드가 데이터와 포인터로 구서. 한 줄로 연결되어 있는 방식으로 데이터 저장.





해시 함수 -> 데이터의 효율적 관리를 목적으로 임의의 길이의 데이터를 고정된 길이의 데이터로 매핑. 키값, 해시값으로 매핑하는 과정을 해싱.

* 적은 리소스로 많은 데이터 효율적 관리
* 보안에 사용. 키와 해시값 사이 직접적 연관이 없어서 해시 값만 가지고 키 복원하기 어려움.

임계 구역 -> 병렬 컴퓨팅에서 둘 이상의 스레드가 동시에 접근해서는 안되는 공유 자원을 접근하는 코드 일부. 지정된 시간 만큼 대기.

* 세마포어 : 공유 자원에 대한 접근 제한.

**WAS -> Web Application Server**

* 인터넷에서 HTTP를 통해 사용자 컴퓨터나 장치에 어플리케이션 기능을 수행해주는 미들웨어.
* 웹서버와 웹 컨테이너의 결합. 클라이언트의 요청이 있을 때 내부의 프로그램을 통해 결과를 만들어내고 이를 클라이언트에 전달
* Tomat.
* 동적 데이터 (웹 컨테이너에서), 정적 데이터 처리
* DB와 연결되어 데이터를 주고 받거나 데이터 조작이 필요한 경우

**Apache -> http 웹 서버. http 요청을 처리**

* 정적 데이터 처리
* 이미지나 단순 HTML 파일 처리에는 빠르고 안정적.