|  |  |  |
| --- | --- | --- |
|  | 운영체제 | **운영체제란 컴퓨터 시스템의 자원들을 효율적으로 관리하며 사용자가 컴퓨터를 편리하고 효과적으로 사용할 수 있도록 환경을 제공하는 시스템입니다.**  사용자와 하드웨어 간의 인터페이스로서 동작하는 시스템 소프트웨어의 일종으로 다른 응용프로그램이 유용한 작업을 할 수 있도록 환경을 제공해줍니다. |
|  | 프로세스  메모리구조 | 1. 코드영역 : 실행할 프로그램의 코드가 저장되는 영역  CPU는 해당코드 영역에 저장된 명령어를 하나씩 가져가서 처리  2. 데이터 영역 : 전역변수와 정적변수가 저장되는 영역으로 프로그램이 시작하는 동시에  할당, 프로그램이 종료되면 소멸  3. 힙 영역 : 사용자가 직접 관리하는 영역이며 메모리 공간이 동적으로 할당 및 해제  (런타임) 동적으로 할당되므로 new로 생성한 object가 많아질수록 영역의 크기 증가  4. 스택 영역 : 함수의 호출에 따른 지역변수와 매개변수가 저장되는 영역 컴파일시  (컴파일) 크기 결정, 함수의 호출과 함께 할당, 종료되면 소멸 |
|  | 프로세스 | **프로세스란 프로그램이 돌아가고 있는 상태 즉 컴퓨터에서 작업중인 프로그램을 의미합니다.**  ● 프로세스별 각각의 영역을 가지기 때문에 별다른 동기화 작업이 필요하지 않음  ● 한 프로세스에서 오류가 생겨도 다른 프로세스에는 영향을 주지 않음 |
|  | 쓰레드 | **쓰레드는 한 프로세스 내의 실행 단위, 실행의 흐름을 의미합니다.**  ● Stack 영역만 별도로 가지고 나머지에 대해서는 해당 쓰레드를 포함한 프로세스의  자원을 공유  ● stack 영역 외부의 공유 자원에 대해 접근 시 동기화 작업이 필요 |
|  | 멀티프로세스,  멀티 쓰레드 | **멀티 프로세스란 두개 이상 다수의 프로세서가 협력적으로 하나 이상의 작업을 동시에**  **처리하는 것이고, 멀티 스레드는 하나의 프로세스에 여러 스레드로 자원을 공유하며 작업을 나누어 수행하는 것입니다.**  멀티 프로세스는 하나의 프로세스가 죽더라도 다른 프로세스에 영향을 주지 않아 안정성이  높지만 멀티 스레드보다 메모리 공간과 CPU 시간을 차지하는 단점이 있습니다. |
|  | 동기, 비동기 | 동기화란 요청과 결과가 동시에 일어난다는 말로 요청을 하면 시간이 얼마가 걸릴지라도 바로 결과가 주어져야 합니다.  비동기화는 요청과 결과가 동시에 일어나지 않는 것이며 작업을 동시에 처리할 수 있어 처리 속도를 향상 시킬 수 있습니다.  동기화는 설계가 간단하고 직관적이지만 결과가 주어지기 전까지 대기해야 하는 단점이 있고, 비동기화는 복잡하지만 자원을 효율적으로 사용할 수 있습니다. |
|  | 데드락 | **데드락이란 두 개 이상의 프로세스나 스레드가 서로 자원을 얻지 못해서 다음 처리를 하지 못하는 상태로 무한히 다음 자원을 기다리게 되는 상태를 말합니다.**  데드락의 특징  1. 상호 배제 : 자원은 한번에 한 프로세스만 사용  2. 점유 대기 : 하나의 자원을 점유하고 있으면서 다른 자원을 대기하는 프로세스가 존재  3. 비 선점 : 다른 프로세스에 할당된 자원은 사용이 끝날 때 까지 강제로 뺏을 수 없음  4. 순환 대기 : 프로세스의 집합에서 순환 형태로 자원을 대기하고 있어야함 |
|  | 프로세스 스케줄링 | **프로세스 스케줄링이란 시스템의 자원을 효율적으로 사용하기 위해 어떤 프로세스를 먼저 처리할지 순서를 정하는 것입니다.**  선점형 스케줄링 기법 : Round Robin, SRF  비 선점형 스케줄링 : Priority Scheduling, FCFS, SJF |
|  | 선점, 비선점 | **선점형 스케줄링이란 실행중인 프로세스나 스레드를 강제로 중단시키고 다른 프로세스를 실행시키는 것입니다.**  **비선점형 스케줄링은 실행중인 프로세스가 있다며 다른 프로세스들이 CPU를 강제로 뺏을 수 없는 스케줄링 방식입니다.** |
|  | G C  (가바지 컬렉션) | 가비지 컬렉션이란 동적으로 할당했던 메모리 영역 중 필요 없게 된 메모리 영역을 주기적으로 삭제하는 프로세스를 말합니다. |
|  |  |  |
|  | OSI 7계층 | **응용 계층** : 최종 목적지로 서비스를 수행하는 계층  **표현 계층** : 데이터 형식을 정의합니다.  **세션 계층** : 데이터가 통신하기 위한 논리적 연결을 담당하는 계층  **전송 계층 [세그먼트]** : 사용자들에게 신뢰성 있는 데이터를 전달하는 계층 ( TCP, UDP)  **네트워크 계층 [패킷]** : IP를 지정하고 라우터로 경로를 선택해 네트워크를 통해 데이터 전달하는 계층  **데이터 링크 계층 [프레임]** : 신뢰성 있는 전송을 위해 에러 검출, 흐름제어를 담당 물리적인 전송  **물리 계층 [비트]** : 데이터를 전기 신호로 변경하는 계층 |
|  | DNS,  DDNS | **DNS란** 도메인 네임 시스템의 약자로 IP주소를 사용자가에게 편리한 문자로 표현된 도메인주소로 바꿔주고 반대의 역할도 수행주는 시스템을 말합니다.  **DDNS란** Dynamic 도메인 네임 시스템으로 유동적으로 변하는 IP에 도메인주소를 매칭시켜주는 시스템을 의미합니다. |
|  | TCP / UDP | **TCP는** Transmission Control Protocol의 약자로 신뢰성 있는 통신을 위해 사용하는  프로토콜로 높은 신뢰성을 보장하지만 UDP보다는 속도가 느립니다. 3way, 4way handshaking으로 서버와 클라이언트가 1:1로 통신을 하고 흐름제어와 혼잡제어가 이루어 집니다.  **UDP는** User Datagram Protocol의 약자로 비연결형 프로토콜입니다.  손상된 데이터에 대해서 재전송하지 않고 신뢰성이 낮아서 스트리밍 같은 서비스에서 사용되며 1:1, 1:N, N:M연결이 가능합니다. |
|  | IP | **IP는** Internet Protocol의 약자로 데이터 패킷이 네트워크를 통해 이동하고 올바른 대상에 도착할 수  있도록 데이터 패킷을 라우팅하고 주소를 지정하기 위한 규칙의 집합입니다.  비연결성과 비신뢰성의 특징을 가지고 있습니다.  즉 인터넷에 연결된 모든 장치는 IP가 있고 이런 IP가 해당 장치의 주소라고 생각하면되고 이런 IP가 식별자 개념으로 생각하면된다. |
|  | 로드밸런서 | **로드밸런서**는 서버에 가해지는 부하를 분산해주는 장치 또는 기술을 통칭합니다.  클라이언트와 서버풀 사이에 위치하며 트래픽을 관리해줍니다. |
|  | HTTP  /  HTTPS | **HTTP란** Hyper Text Transfer Protocol의 약자로 웹상에서 정보를 주고 받을수 있는 통신 프로토콜 입니다.  **HTTPS란** HTTP의 보안상 문제를 해결하기 위해 나온 프로토콜입니다.  Body부분만 암호화합니다. |
|  | HTTP Method | **GET** : 조회  **POST** : 요청된 데이터를 처리하고 자원 생성  **PUT** : 요청된 자원이 있으면 갱신 없으면 생성  **PATCH** : 수정  **DELETE** : 삭제 |
|  | GET / POST | **GET** 방식이란 요청을 전송할 때 필요한 데이터를 Body에 담지 않고 쿼리스트링을 통해 전송하는 방식을 말합니다.  **POST** 방식은 전송해야 할 데이터를 HTTP메시지의 Body에 담아서 전송합니다.  GET 방식은 길이 제한이 있고 보안에 취약하고 멱등성을 보장합니다.  POST방식은 길이 제한이 없고 멱등성을 보장하지 않습니다. |
|  | 멱등성 | **멱등성이란** 여러 번 적용하더라도 결과가 달라지지 않는 성질을 의미합니다. |
|  | 정적 라우팅  /  동적 라우팅 | **정적 라우팅이란** 네트워크 경로를 수동으로 구성하고 선택하는 것을 말합니다.  네트워크 정체가 나타날 수 있으며 네트워크의 유연성을 저하시켜 성능을 제한합니다.  **동적 라우팅이란** 네트워크 조건에 따라서 런타임에 라우팅 테이블을 만들고 업데이트합니다.  목적지까지 가장 빠른 경로를 찾으려고 시도합니다.  트래픽 볼륨, 네트워크 장애 등 변화하는 조건에 대처가 가능합니다. |
|  | 쿠키  /  세션 | **쿠키란** 브라우저 로컬에 저장되는 키와 값이 들어있는 작은 데이터 파일입니다.  유효시간을 정할 수 있으며 브라우저가 종료와는 상관없이 시간에 영향을 받습니다.  **세션이란** 사용자의 데이터를 서버에 저장되는 데이터파일입니다.  브라우저를 종료할 때 까지 인증상태를 유지하비다. |
|  |  |  |
|  | DBMS | **DBMS란** DataBase Management System의 약자로 데이터베이스 관리 시스템으로 여러 사용자가 데이터 베이스에 접근해 사용할 수 있도록 해주는 소프트웨어를 의미합니다. |
|  | 스키마 | **스키마란** 데이터베이스의 구조와 제약조건에 관한 전반적인 명세를 기술한 메타데이터의 집합입니다. |
|  | 뷰 | **뷰는** 테이블에서 필요한 정보만을 추출하여 가상의 테이블로 만든 뒤, 해당 뷰로 조회하면 효율적으로 데이터를 조회할수있다. |
|  | 키 | 슈퍼키 ( Super Key )  후보키 ( Candidate Key )  기본키 ( Primary Key )  대체키 ( Alternate Key )  외래키 ( Foreign Key ) |
|  | 트랜잭션 | **트랜잭션이란** 데이터베이스의 상태를 변환시키는 하나의 논리적 기능을 수행하기위한 작업의 단위  또는 한꺼번에 모두 수행되어야 할 일련의 연산들을 의미합니다.  1, **Atomicity (원자성)** : 연산 전체가 모두 실행 or 모두 취소  2. **Consistency (일관성)** : 성공적으로 완료하면 일관성 있는 데이터베이스 상태 유지  3. **Isolation (고립성)** : 두 개 이상의 트랜잭션이 발생할 때 영향을 주면 안된다.  4. **Durability (지속성)** : 커밋 된 트랜잭션의 내용은 영구히 반영된다. |
|  | DELETE  TRUNCATE DROP | **DELETE** : 데이터는 지우지만 테이블 용량은 줄어들지 않고 원하는 데이터만 삭제가능  **TRUNCATE** : 전체데이터를 모두 삭제하는 방식, 테이블 용량이 줄어든다.  **DROP** : 테이블 자체를 완전히 삭제하는 방식 |
|  | DDL DML DCL | **DML (조작어)** : (Data Manipulation Language) DB내의 자료검색, 삽입, 갱신, 삭제를 위한 언어  ( SELECT, INSERT, UPDATE, DELETE)  **DDL (정의어)** : (Data Definition Language) 데이터베이스 구조를 정의, 수정, 삭제하는 언어  ( ALTER, CREATE, DROP)  **DCL (제어어)** : (Data Control Language) 데이터에 대해 무결성유지, 병행 수행 제어, 보호와 관리를 위한 언어  ( COMMIT, ROLLBACK ) |
|  | DB 인덱스 | **DB Index**란 테이블을 처음부터 끝까지 검색하지 않고 인덱스를 검색해서 테이블의 검색 성능을 향상시키기 위한 자료 구조입니다. 인덱스는 특정 열에 대한 정렬된 데이터 구조로 데이터베이스 관리 시스템이 데이터를 더 빠르게 찾을 수 있도록 도와줍니다. |
|  | SQL Injection | **SQL Injection**이란 사용자가 보안상의 취약점을 이용하여 임의의 SQL문을 주입하고 실행되게 하여 데이터베이스가 비정상적인 동작을 하도록 하는 행위입니다. |
|  | 정규화 | **1정규화** : 테이블의 칼럼이 하나의 값을 갖도록 테이블을 분해하는 것  **2정규화** : 기본키가 2개이상의 속성으로 이루어진 경우, 종속성을 제거한다.  **3정규화** : 기본키를 제외한 컬럼간의 종속성을 제거한다. |
|  | 수직 파티셔닝  /  수평 파티셔닝  (샤딩) | **파티셔닝은** 엔**티티의 컬럼 기준으로 분리하는 개념이다.**  **샤딩은** 엔티티의 로우(열) 기준으로 분리하는 개념이다.  파티셔닝은 주로 데이터 관리와 성능 최적화를 위해 사용되고 샤딩은 데이터베이스 시스템의 확장성을 향상시키기 위해 사용됩니다. |
|  | ORM | **ORM** 이란 Object -relational– Mapping의 약자로 객체클래스와 RDB에서 사용하는 데이터 테이블을 서로 매핑 시킨 것을 의미합니다. 예)JPA, MYBATIS, HIBERNATE |
|  |  |  |
|  | JVM | **JVM이란** Java Virtual Machine으로 운영체제와 Java Application의 사이를 중재해주는 가상머신입니다. |
|  | Java 실행과정 | 자바 소스코드(.java)를 컴파일러에 의해 바이트코드 파일(.class)로 변환  Class Loader를 통해 바이트코드 파일(.class)을 Runtime Data Area로 로딩  로딩된 파일을 Execution Engine(실행엔진)를 통해서 해석 및 실행 |
|  | JVM 메모리 구조 | 1. Method Area : 클래스에 대한 정보를 저장 클래스, 인터페이스, 메서드 저장  2. Heap Area : 런타임시 동적으로 할당하여 사용하는 영역  3. Stack Area : 스레드마다 존재하여 스레드가 시작할 때 할당 지역변수, 매개변수  4. PC Register : 스레드가 생성될 때마다 생성되는 영역 스레드가 현재 실행할 프레임의 주소 저장  5. Native Method Stack : C / C++등의 Low Level 코드를 실행하는 스택 |
|  | ==와 equals | **==은** 참조비교로 두 객체가 같은 메모리 공간을 가리키는 지를 확인하는 연산입니다.  **Equals는** 두 객체의 내부 값이 같은 지 내용을 비교합니다. |
|  | Java의 접근 제한자 | 1. **private** : 자기 자신 클래스 내에서만 접근 가능  2. **default** : 동일한 패키지 내에서만 접근 가능  3. **protected** : 동일한 패키지 내에서만 접근 가능 + 상속을 이용한 접근 가능  4. **public** : 접근에 제한이 없음 |
|  | 원시 자료형 (Primitive Type)  /  참조 타입  (Reference Type) | **Primitive Type은** 기본 타입의 크기가 작고 고정적이기 때문에 Stack영역에 저장됩니다.  주소는 없고 실제 값만 저장됩니다.  **Reference Type은** 데이터의 크기가 가변적이고 동적이므로 Heap영역에서 관리됩니다.  데이터는 Heap영역에서 관리되지만 주소 값은 Stack 영역에 저장됩니다. |
|  | 자바의 제네릭 | **제네릭은** 데이터의 타입을 일반화하는 것을 의미합니다. 클래스나 메서드에서 사용할 데이터의 타입을 컴파일시에 미리 지정하는 방법입니다. |
|  | 추상 클래스  /  인터페이스 | **추상 클래스는** 추상 메서드를 선언해 놓고 상속을 통해 자식 클래스에서 메서드를 완성하도록  유도하는 클래스로서 미완성 설계도라고 표현합니다.  **인터페이스도** 추상 클래스와 비슷하게 다른 클래스를 작성하는데 도움을 주는 목적으로 작성합니다. 다만 인터페이스는 추상클래스보다 추상화 정도가 높아 추상클래스와 다르게 구현부가 있는 일반 메서드, 일반 변수 멤버 등을 가질 수 없습니다. 기본설계도라고 부릅니다. |
|  | String  String Builder  String Buffer | **String**은 final char[] 형태이기 때문에 변경이 불가능합니다. 따라서 + 연산을 사용하면 두 String을 연결한 새로운 객체를 만들어서 저장하기 때문에 메모리 관리측면에서 치명적입니다.  **String Builder와 Buffer는** 가변성을 가지는 클래스로 문자열을 변경하는 것이 가능합니다.  다만 Builder는 동기화를 지원하지 않지만 속도가 빠르고 Buffer는 동기화를 지원하기 때문에 멀티 스레드 환경에서 사용이 가능합니다. |
|  | new String (“”)  “” 차이 | “”으로 문자열을 초기화 하게 되면 Java Heap 메모리의 String pool에 저장되게 됩니다.  반면에 new String(“”)은 String pool이 아닌 Heap영역에 각각 생성됩니다.  따라서 Heap영역에 새로운 주소지를 참조하게 됩니다. |
|  | Java 리플렉션 | **리플렉션이란** 메타정보를 프로그램에서 읽고 수정하는 행위를 말합니다.  메타정보란 패키지 정보, 타입 정보, 멤버 정보 등을 말합니다. |
|  | Java Optional | “NullPointerException”을 피하기 위해 null여부를 검사하게 되는데  Optional<T> 클래스를 사용해 NPE발생을 방지하도록 도와줍니다. |
|  | 자바 8  /  자바 11 | **자바 8** : 람다 표현식, Stream, interface의 default, Optional  **자바 11** : String, File 클래스의 메서드 추가, 람다 표현식 var키워드 사용가능 |
|  | DAO DTO VO | **DAO란** Data Access Object의 약자로 데이터베이스에 접근하기 위한 객체이며 데이터베이스 접근을 하기 위한 로직과 비즈니스 로직을 분리하기 위해 사용합니다.  **DTO란** Data Transfer Object의 약자로 계층간 데이터 교환을 위한 자바 빈즈입니다.  **VO는** Value Object로 Read-Only 속성을 지닌 값 오브젝트입니다. |
|  | 오버 라이딩  /  오버 로딩 | **오버 라이딩이란** 부모클래스로부터 상속받은 메소드를 자식 클래스에서 재정의 하는 것입니다.  부모클래스의 메소드를 재정의하는 것이므로 메소드의 이름, 매개변수, 리턴 값이 모두 같아야 합니다.  **오버 로딩이란** 자바의 한 클래스 내에 이미 사용하려는 이름과 같은 이름을 가진 메소드가 있더라도 매개변수의 개수 또는 타입이 다르면 같은 이름을 사용해서 메소드를 정의할 수 있습니다. 리턴 값만 다른 것은 오버로딩이 불가능합니다. |
|  | 단위 테스트 | **단위 테스트란** 테스트 가능한 가장 작은 소프트웨어를 실행하여 예상 동작을 검증하는 테스트  테스트 대상 단위는 클래스, 메소드가 있습니다.  테스트 단위의 크기가 작을수록 복잡성이 낮아지고 테스트 활용이 더 쉬워집니다.  대표적인 단위테스트 프레임워크는 Junit이 있습니다.  **TDD** ( Test Driven Development), 테스트 주도 개발은 선 개발 후 테스트를 진행하는 방식이 아닌  선 테스트 후 개발 방식의 프로그래밍 방법을 말합니다. |
|  | 통합 테스트 | **통합 테스트란** 단위 테스트보다 더 큰 동작을 위해 여러 모듈을 모아 의도대로 협력하는 지 검증하는 테스트입니다. 단위 테스트보다 더 많은 코드를 테스트하기 때문에 신뢰성이 떨어지고  에러 발생시 확인하기 쉽지 않아서 유지보수가 힘들다는 단점이 있습니다. |
|  | 인수 테스트 | **인수 테스트란** 사용자 시나리오에 맞춰 수행하는 테스트입니다.  즉 소프트웨어를 개발하고 인수자들에게 해당 소프트웨어를 인수하면 그 인수자들이 테스트하는 개념을 인수테스트라고 한다. (마지막 테스트) |
|  | Enum(열거형) | **Enum 클래스**란 Enumeration의 약자로 열거형 클래스입니다.  서로 관련이 있는 상수들끼리 모아 놓은 클래스로 외부에서 변경할 수 없어 안전하고 가독성이 좋은 코드를 구현할 수 있습니다. |
|  | 리팩토링 | **리팩토링이란** 코드의 의미와 결과의 변경 없이 코드의 구조를 바꾸는 것으로  주로 가독성을 높이고 유지보수를 편하게 하기 위해 사용하며 버그를 없애거나 기능을 추가하는 행위는 아닙니다. |
|  | Array  /  ArrayList  /  LinkedList | **Array는** 크기가 고정되고 데이터의 순서를 의미하는 Index가 존재하여 랜덤 액세스가 빠르지만 요소 추가 및 삭제가 느립니다.  **ArrayList는** 크기가 가변적이며 데이터의 순서를 의미하는 Index가 존재하여 랜덤 액세스가 빠르며 동적으로 크기를 조절할 수 있습니다.  **LinkedList는** 각 노드를 서로 연결시켜 만든 구조로 요소 추가 및 삭제가 빠르지만 랜덤 액세스가 느리며, 순차적으로 탐색해야 합니다. |
|  | Stack  /  Queue | **Stack은** FILO (First In Last Out)의 특성을 가진 자료 구조입니다. 가장 마지막에 추가된 요소가 가장 먼저 나가게 됩니다.  **Queue는** FIFO (First In First Out)의 특성을 가진 자료 구조입니다. 처음에 추가된 요소가 먼저 나가게 됩니다. |
|  | 해시 | **해시**는 자료를 효율적으로 저장하고 검색하기 위한 자료 구조입니다.  Key – Value로 쌍을 사용하여 데이터를 저장하고 특정 키를 사용하여 데이터를 빠르게 검색하는데 사용됩니다. Key 값을 이용해서 데이터를 검색하기 때문에 Key값은 중복이 불가능합니다.  해시 충돌 회피 기법으로는 체이닝 방식과 Open Addressing이 있습니다.  **체이닝 방식**은 하나의 Value 값을 가지는 것이 아닌 Value에 LinkedList를 이용해서 여러 개의 값을 저장하는 방법입니다.  **Open Addressing**은 중복되는 Key값을 바꿔서 저장하는 방식입니다. [ |
|  | Static  /  Dynamic | **Static (정적 변수)**는 클래스 레벨에서 공유되고, 클래스 이름을 통해 접근할 수 있으며 프로그램 실행 중에 계속 유지됩니다. 반면에 **Dynamic (인스턴스 변수)**는 객체마다 독립적으로 생성되며 객체를 생성해야만 접근할 수 있습니다. |
|  | 클래스 | **클래스는** 객체 지향 프로그래밍의 기본 구성 요소 중 하나입니다. 클래스는 객체를 생성하기 위한  설계 도면 또는 템플릿 역할을 합니다. 클래스는 객체의 속성과 메서드를 정의하여 모델링 하는데 사용됩니다. |
|  | 트리 | **Tree**는 비선형 자료 구조 중 하나로서 계층적 구조를 띄고 있습니다. |
|  | 힙 | **힙**은 최대값 혹은 최소값을 빠르게 찾기 위한 이진 트리로서 최소 힙의 경우 부모가 자식보다 작고  최대 힙의 경우에는 부모가 자식보다 커야 합니다. |
|  | 이진 탐색 트리 | **이진 탐색 트리란** 왼쪽 자식이 부모보다 작고 오른쪽 자식은 부모보다 큰 이진 트리입니다.  트리가 편향되지 않기 위해서 자가 균형 트리를 사용합니다. |
|  | 자가 균형 트리 | **이진 탐색 트리는** 편향될 경우 효율이 떨어지기 때문에 삽입과 삭제를 개선한 이진 탐색 트리를 자가 균형 트리라고 합니다. |
|  | 컴파일  /  런타임 | **컴파일**이란 자바 소스 코드를 바이트 코드 즉 .java 파일을 .class파일로 변환하는 과정을 말합니다.  보통 이클립스와 같은 IDE(Integrated Development Environment)환경에서는 실행을 시키면서 동시에 컴파일이 이루어지고 컴파일 과정을 거친 .class파일이 실행되는 과정의 시간을 **런타임**이라고 합니다. |
|  |  |  |
|  | Spring | **Spring**이란 Java의 웹프레임 워크로 Java로 다양한 어플리케이션을 만들기 위한 프로그래밍 틀입니다.  AOP(관점지향프로그래밍)의 특성을 가지며 중복코드의 사용을 줄여주고 비즈니스 로직을 더 간단하게 해줄 수 있습니다. |
|  | IOC  /  DI | **IOC**는 Inversion Of Control의 약자로 제어의 역전이라는 의미입니다.  객체의 생성부터 생명주기의 관리까지 모든 객체에 대한 제어권이 바뀐 것을 의미합니다.  클래스 간의 결합도를 줄이기 위해서 사용하는 디자인 원칙입니다.  **DI**는 Dependency Injection의 약자로 의존관계주입을 의미합니다.  어떠한 객체가 사용하는 의존 객체를 직접 생성하는 것이 아니라 주입 받아 사용하는 것입니다.  모듈 간의 결합도를 낮추기 위한 전략적인 방법입니다. |
|  | Bean | **Bean**이란 스프링 컨테이너가 생성해준 자바 객체로 DI 컨테이너가 관리하는 자바 객체를 의미합니다.  어노테이션을 통해 특정 객체를 빈으로 등록해 사용하고 빈으로 등록된 객체는 DI 컨테이너가 의존 관계를 주입시킵니다. |
|  | Annotation | **Annotation**은 코드 사이에 주석처럼 쓰이며 특별한 의미, 기능을 수행하도록 하는 기술입니다.  프로그램에게 추가적인 정보를 제공해주는 메타데이터라고 볼 수 있습니다. |
|  | 메타 데이터 | **메타 데이터는** 데이터를 위한 데이터를 말합니다.  대량의 정보 가운데에서 확인하고자 하는 정보를 효율적으로 검색하기 위해 사용합니다. |
|  | MVC 패턴 | MVC 패턴은 Model, View, Controller 세가지 구성요소를 사용해서 프로젝트를 구성한 패턴입니다.  **Model**은 서비스 로직 및 DAO객체가 있는 영역이다. (데이터베이스에 가까운 영역)  **View**는 사용자가 볼 결과물을 생성하기 위해 모델로부터 정보를 얻어옵니다.  **Controlle**r는 사용자의 요청을 파악한 후 데이터 Model을 이용해서 데이터를 View에 반영해서 사용자에게 알려주는 Model과 View 사이의 역할을 합니다. |
|  | 웹서버 /  WAS | **웹서버**는 정적인 데이터를 처리하는 서버로 이미지나 단순 html과 같은 정적인 리소스들을 전달하며 동적인 요청이 온다면 WAS에 동적인 요청을 보내고 처리한 결과를 다시 가져와서 클라이언트에게 전달해주는 역할을 합니다.  **WAS**는 위에 말한 것처럼 동적인 요청을 받았을 때 동적인 컨텐츠를 제공하기 위해 만들어진  Web Application Server입니다. |
|  | Servlet | **Servlet**이란 동적 웹페이지를 만들 때 사용되는 자바 기반의 웹 어플리케이션 프로그래밍 기술로서  클라이언트가 어떠한 요청을 하면 그에 대한 결과를 다시 전송해주는 역할을 합니다. |
|  | POJO | POJO란 Plain Old Java Object의 약자로 평범한 구식 자바 객체를 의미합니다.  인터페이스나 클래스를 구현하거나 확장하지 않은 단순한 클래스를 의미하는 것으로 종속관계가 있지 않아서 코드가 간결합니다. |
|  | 커넥션 풀 | **커넥션 풀이란** 어플리케이션이 DB를 사용하기 위해서 Connection이 필요한데 생성 및 소멸 비용이 크기 때문에 커넥션 풀을 미리 생성하고 어플리케이션이 시작하는 시점에 커넥션을 미리 생성하고 이것을 재활용하면서 사용하는 것을 말합니다. |
|  | AOP  /  OOP | **OOP (Object Oriented Programming)** 모든 데이터를 현실 사물에 빗대어 객체로 다루는 프로그래밍 기법으로 객체의 상속을 통해 재사용해서 반복되는 코드의 양을 줄여줍니다.  즉 객체들을 가지고 상호작용을 통해 프로그래밍 하는게 OOP이다.  **AOP (Aspect Oriented Programming**)은 관점지향 프로그래밍으로 관점을 기준으로 다양한 기능을 분리하여 보는 프로그래밍입니다. 중복되는 부가 기능과 비즈니스 로직을 모듈화 하는 것을 말합니다. |
|  | OOP의 특징 | 1. 캡슐화 : 클래스 안에 서로 연관이 있는 속성과 기능들을 하나의 캡슐로 만들어 데이터를 외부로부터 보호하는 것을 말합니다. (데이터 보호, 데이터 은닉)  2. 추상화 : 객체의 공통적인 속성과 기능을 추출하여 정의하는 것입니다.  어떤 객체가 수행하야 하는 핵심적인 역할만 규정하고 실제 구현은 각각의 객체에서 하도록 설계하는 것을 의미합니다.  3. 상속성 : 기존의 클래스를 재활용하여 새로운 클래스를 작성하는 자바의 문법 요소로서 상위클래스로부터 확장된 여러 개의 하위클래스들이 모두 상위클래스의 속성과 기능을 간편하게 사용할 수 있도록 합니다.  4. 다형성 : 어떤 객체의 속성이나 기능이 상황에 따라서 여러가지 형태를 가질 수 있는 성질을 의미합니다. (오버라이딩, 오버로딩) |
|  | OOP의 5원칙 | **S (Single Responsibility Principle)** : 단일책임원칙으로 모든 클래스는 각각 하나의 책임만 가져야 합니다.  **O (Open Closed Principle)** : 개방- 폐쇄 원칙으로 기존의 코드를 변경하지 않으면서 기능을 추가할 수 있도록 설계가 되어있어야 한다는 원칙입니다.  **L (Liskov Substitution Principle)** : 리스코프 치환 원칙으로 자식 클래스는 언제나 자신의 부모클래스를 대체할 수 있다는 원칙입니다.  **I (Interface Segregation Principle)** : 인터페이스 분리 원칙으로 목적과 관심이 각기 다른 클라이언트가 있다면 인터페이스를 적절하게 분리해줘야 한다는 원칙입니다.  **D (Dependency Inversion Principle) :** 의존 역전 원칙이란 클래스를 참조해서 사용해야 하는 상황이 생긴다면 Class를 직접 참조하는 것이 아닌 추상클래스 or 인터페이스를 참조하라는 원칙입니다. |
|  |  |  |
|  | 빅테크  /  핀테크 | **빅테크 :** 인터넷 플랫폼을 기반으로 한 거대 IT기업들을 뜻합니다.  **핀테크 :** 금융과 기술을 결합한 서비스 혹은 이를 제공하는 기업을 뜻합니다. |
|  | 데이터 모델링 | **데이터 모델링**이란 복잡하고 명확하지 않은 현실세계 데이터를 단순화시켜서 데이터로 저장하기 위한 작업입니다. 추상화, 단순화, 명확화의 특징을 가집니다.  (요구사항 분석 – 개념적 모델링 – 논리적 모델링 – 물리적 모델링) |
|  | 소프트웨어 생명주기 | 소프트웨어 개발 방법론의 바탕이 되는 것으로 소프트웨어 개발을 하기 위해서  요구사항정의 -> 분석 -> 설계 -> 구현 -> 테스트 -> 배포  생명주기 대표적인 모델로는 폭포수, 애자일 방법론이 있습니다. |
|  | 폭포수 방법론 | **폭포수 방법론**은 소프트웨어 개발 단계가 위에서부터 아래로 폭포에서 물이 떨어지듯이 순차적으로 진행되는 것을 말합니다. 다시 이전 단계로 가지 않고 계속 진행한다는 특징이 있습니다. |
|  | 애자일 방법론 | **애자일 방법론**이란 각 단계를 반복적으로 수행하면서 진행되는 것을 말합니다.  잦은 요구사항 변경이나 큰 프로젝트라 요구사항 분석 및 설계를 완벽하게 하기 어려운 경우에 적절한 개발 방법론입니다. |
|  | V Model | **V Model**은 폭포수 방법론의 확장된 형태로 테스트 단계에서 진행하던 테스트 케이스 설계를 분석 및 설계 단계에서 시작하여 프로젝트의 비용과 시간을 감소시키는 Model입니다. |
|  | 블록체인 | **블록체인**은 데이터 분산 처리 기술입니다. 네트워크에 참여하는 모든 사용자가 모든 거래 내역 등의 데이터를 분산, 저장하는 기술을 지칭하는 말입니다.  분산 저장을 하기 때문에 데이터의 위.변조가 어렵고 중앙관리자가 없다는 특징이 있습니다. |
|  | Front-end  /  Back-end | **Front – end** 개발은 사용자의 인터페이스 (UI)와 사용자의 경험 (UX)를 만드는데 초점을 맞춰서 개발을 하고, 주로 HTML, CSS, JavaScript를 사용합니다.  **Back – end** 개발은 서버 쪽 개발 분야입니다. 사용자에게 직접적으로 보이지는 않지만 기능을 구현하고 데이터를 저장 및 관리하는 작업을 합니다 주로 C언어와 Java를 사용합니다. |
|  | 형상관리자  Git / SVN | **형상관리**란 소프트웨어의 변경사항을 체계적으로 추적하고 통제하는 것으로 변경사항에 대한 원인과 변경사항이 기록되어 확인할 수 있는 것입니다.  **Git 과 SVN**의 차이는 중앙저장소(서버)는 동일하게 있고 Git은 개인저장소가 있습니다. |
|  | CI / CD | **지속적인 통합, 지속적인 배포**  어플리케이션 개발 단계부터 배포까지의 모든 단계를 자동화를 통해서 좀 더 효율적이고 빠르게 사용자에게 지속적으로 배포할 수 있는 것을 말합니다. |
|  | DevOps | **DevOps**란 Development 와 Operation의 합성어로  소프트웨어 개발자와 운영 전문가 간의 소통, 협업 및 통합을 강조하는 개발 환경이나 문화를 말합니다. |
|  | MSA | **MSA**는 Micro Service Architecture의 약어로 대형 소프트웨어 프로젝트의 기능들을 작고 독립적이며 느슨하게 결합된 모듈로 분해하여 서비스를 제공하는 아키텍처입니다. |
|  | Architecture | **Architecture**란 목표하는 대상에 대하여 그 구성과 동작 원리, 구성 요소 간의 관계 및 시스템 외부 환경과의 관계 등을 설명하는 설계도입니다. |
|  | 2 Tier | 클라이언트 – 서버 (C / S)  클라이언트 영역에서 요청을 하면 이게 서버로 가서 클라이언트가 요청한 결과물을 받는개념을 2단계로 표현한게 2Tier |
|  | 3 Tier | 클라이언트 – WAS – 서버 프리젠테이션 <-> 비지니스 <-> 펄시스턴스 보통 현업에서는 이런 3tier영역으로 나누어서 업무를 진행한다. 이렇게 영역을 나누면 협업에 유리, 유지보수 유리, 결합도 낮춰준다. |
|  | UML | **Unified Modeling Language**  시스템을 분석 정의할때 여러 다이어그램으로 시각화 하고 명세화하는 걸 통틀어 UML이라고 한다. |
|  | M / M | **Man / Month**  주로 현업에서 공수를 뜻하며 한달에 몇명이 해당 프로젝트에 투여했냐라는 개념이다. |
|  | White Box Test  /  Black Box Test | **White Box Test :** 구조기반 테스트  코드와 개발 설계 등의 소프트웨어 구현 정보를 기반으로 테스트케이스 도출하고 테스트 하는 방식입니다.  **Black Box Test :** 명세 기반 테스트  내부구조나 작동원리를 모르는 상태에서 입 출력만으로 판별하는 방식입니다. |
|  | Hot Deploy | 서버의 재시작 없이 응용프로그램의 동적 변경을 바로 적용시키는 기술입니다. |
|  | EAI / ESB | EAI란 기업 내 필요한 여러 어플리케이션들 사이에 상호 연동이 가능하도록 통합하는 솔루션입니다.  ESB는 기업 시스템에서 서비스 수준에서의 시스템 연계를 위한 체계입니다. |
|  | 계정계  정보계  대외계  채널계  운영계  기간계 | **계정계**는 본연의 금융 업무 처리 시스템으로 고객의 거래 데이터 자체를 다루는 영역으로  계좌 개설, 입금 출금, 계좌이체와 같은 업무를 합니다.  **정보계**는 계정계 데이터를 기반으로 영업점 및 각 부서의 업무 처리를 위해 필요한 고객의 거래 데이터에 대한 기록 및 통계를 관리하는 시스템입니다.  **대외계**는 은행 외부 기관과 연계되는 업무를 처리하기 위한 대외(금융 공동망) 연결 시스템입니다. 인터넷 뱅킹, 타행이체 등 과 같은 업무를 합니다.  채널계는 대외계 시스템 및 다양한 비대면채널들을 관리하는 시스템을 말합니다.  운영계는 IT시스템의 안정적인 운영을 위한 시스템입니다.  기간계는 새로운 시스템을 도입할 때 기존 시스템을 의미합니다 (Legacy) |
|  | 채널 | 고객이 은행과 거래할 수 있는 접점으로 대면채널과 비대면채널로 크게 분류할 수 있다. |
|  | 디버깅  /  Trouble Shooting | **디버깅**이란 컴퓨터 프로그램 개발 단계 중에 발생하는 시스템의 논리적인 오류나 버그를 찾아내고 원인을 밝히고 수정하는 작업을 말합니다.  **Trouble Shooting**이란 문제가 발생했을 때 원인을 규명하고 문제를 종합적으로 진단해 해결하는 것 즉 문제해결이라고 생각합니다.  즉 두가지의 차이는 디버그는 개발환경에서의 작업이고  트러블 슈팅은 실제 운용환경에서의 작업이라고 생각합니다. |
|  | PostMan | API 개발 및 테스트를 위한 프로그램으로 웹기반으로  간편하게 HTTP 요청을 생성하고 응답을 검사하고 테스트를 진행할 수 있습니다. |
|  | REST API | **Rest**는 Representational State Transfer의 약자로 자원을 이름으로 구분하여 해당 자원의 상태를 주고받는 모든 것을 의미합니다.  자원 (URI), 행위 (HTTP METHOD), 표현(Representation) 3가지로 이루어져 있습니다. |
|  | Auto COMMIT  /  MANUAL COMMIT | **자동 커밋은** 쿼리문을 커밋을 하면 즉시 데이터베이스 적용됩니다.  **수동 커밋은** 내가 직접 커밋 / 롤백을 해줘야 실행이 되고 쿼리문을 날려서 보여지는 것은 쿼리에 의해 DB가 임시 상태로 저장되는 형태이다.  수동 커밋 상태일 때 **COMMIT**을 누르면 임시 저장되었던 것들이 DB에 반영되는 것이고  **ROLLBACK**을 누르면 임시 저장되었던 것들이 다시 원래 상태로 돌아가는 것입니다. |
|  | GIGO | Garbage In Garbage Out으로 쓰레기가 들어오면 쓰레기가 나간다는 말입니다. |
|  | 디자인 패턴이란 | **디자인 패턴은** 선배들의 경험이 담긴 문제 해결방법을 말합니다.  특정 상황에서 발생하는 문제 패턴을 발견하고 해결방안을 기록으로 남긴 솔루션입니다. |
|  | 싱글톤 패턴 | **싱글톤 패턴이란** 객체의 인스턴스가 오직 1개만 생성되는 패턴을 의미합니다.  즉 어떤 클래스가 최초 한번만 메모리를 할당하고 그 메모리를 반복적으로 사용하는 패턴을 말합니다. |
|  | Inner Class | 클래스나 인터페이스 내부의 있는 클래스를 말하는 것으로 코드를 더 읽기 쉽고 오래 유지하기 위해 사용합니다. |
|  | 수신 / 여신 | **수신**은 신용을 받는다 라는 의미를 가진 단어로 은행이 대출을 받는 것을 말합니다.  한마디로 예금을 의미합니다.  **여신**은 신용을 준다 라는 의미를 가진 단어로 은행이 대출을 해주는 것을 의미합니다. |
|  | DB JOIN | **JOIN**이란 두 개이상의 테이블을 연결해서 데이터를 검색하는 방법입니다.  연결을 위해서 테이블이 적어도 하나의 컬럼을 공유하고 있어야 합니다.  대표적으로 테이블들의 교집합만 가지고 오는 INNER JOIN과  합집합을 가지고 오는 OUTER JOIN이 있습니다.  그 이외에도 LEFT OUTER JOIN, RIGHT OUTER JOIN, CROSS JOIN, SELF JOIN |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |