**[자바]**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1. 자바 | | | | |
| OOP | Object Oriented Programming(객체 지향적 프로그래밍)  객체 지향 언어의 특성  : 캡슐화(Encapsulation) > 정보 은닉화  상속(Inheritance), > 코드의 중복작성을 방지,  프로그램의 크기, 코드작성 시간, 비용을 절감  다형성(Polymorphism) > 같은 이름의 함수 호출에 대하여 각 객체에 따라  다른 동작을 할 수 있도록 구성하는 것  (오버로딩, 오버라이딩과 밀접한 관계) | | | |
| 객체(오브젝트) | 클래스라는 추상적인 개념을 실체화(Instance)한 것 | | | |
| 클래스(추상) | OOP에세 사용하는 추상적 개념의 단위 | | | |
| 절차지향 vs 객체지향 | 절차지향 | | 객체지향 | |
| 장점 | 단점 | 장점 | 단점 |
| 실행속도가 빠르다 | 유지보수 어려움  재사용 어려움  스파게티 코드가 되기 쉬움 | 재사용성 용이  유지보수 용이  자연적인 모델링 | 실행처리 속도가 느리다 |
| 데이터 보다 알고리즘에 치중 | | 알고리즘 보다 데이터에 치중 | |

|  |  |
| --- | --- |
| 자바와 C언어의 차이점 | |
| **자바** | **C언어** |
| 보안이 강함  객체 지향적인 언어  운영체제, CPU 호환성이 강함 | 구조화된 프로그램 개발 가능  호환성이 좋음  작고 효율적임 |
| 자바는 처음 설계부터 인터넷 기반으로 시작합니다.  네트워크용 프로그래밍 언어  그만큼 메모리를 많이 사용하고 속도가 느림 | 하드웨어를 직접 제어할 수 있는 것이 C언어,  속도를 요하는 프로그램에 아주 유용한 프로그래밍 언어. |

|  |  |
| --- | --- |
| 가비지 컬렉션(= 자동 메모리 관리), 컬렉터 | |
| 가비지 컬렉션 | 가비지 컬렉션은 가비지 컬럭터가 메모리의 힙 영역에 할당된 더 이상 사용되지 않는 객체를 다른 객체가 사용 할 수 있도록 제거하는 것  \*특징  - 프로그래머가 구현 할 수 없음  - 대상은 객체이지 참조가 아님  - 대상은 살아있는 스레드가 더 이상 참조하지 않는 객체  - 객체에 할당된 메모리를 명시적으로 해체할 수 없음  - 객체를 어떠한 순서로 정리하는지 알 수 없음  - 언제 수행되는지 정확히 알 수 없음  - JVM이 결정하며 우리는 단지 요청만 할 수 있음 |

|  |  |
| --- | --- |
| String | 불변객체  객체는 한 번 생성되면 할당된 메모리 공간이 변하지 않음  문자열 연산이 많은 경우, 그 성능이 좋지 않음  간단하게 사용가능  내부 데이터를 자유롭게 공유 |
| StringBuffer | 변하기 쉬움  문자열을 추가하거나 변경 할 때 주로 사용하는 자료형 |
| StringBuilder | 문자열의 내용을 수정하는 메소드들을 제공  단일 스레드 환경에서만 사용  스레드에 안전한지 여부가 전혀 관계 없는 프로그램을 개발할 때 사용하면 좋다. |

|  |  |  |
| --- | --- | --- |
| 2. 컴파일? (\*\*\*) vs 인터프리터와 비교 | | |
| 컴파일 vs 인터프리터 | 컴파일러 | 인터프리터 |
| 번역단위 | 전체 | 한 줄씩 |
| 실행속도 | 빠름 | 느림 |
| 목적 프로그램 | 느림 | 빠름 |
| 메모리 할당 | 목적프로그램 생성시 사용 | 사용 안 함 |
| 비교 | 컴파일언어는 한번에 번역을 해서 속도가 느리지만 나중에는 빠르다.  수정하는 순간 컴파일을 다시 해야 함 | 인터프리터는 유지보수가 편함,  늘 일정한 비용을 내야 한다. |
| 자바는 인터프리터와 컴파일의 장점을 조화시킨 언어 | | |
| 3. 자바 컴파일 | | |
| a. 1차 컴파일 -> 2차 인터프리터(메서드)  b. 1차 컴파일 -> 2차 JIT 컴파일 | | |
| 4. 자바 컴파일 | | |
| Test.java -> (javac.exe) -> Test.class -> (java.exe) -> 기계어 | | |

|  |  |  |
| --- | --- | --- |
| 5. 자료형 | | |
| a. 자바의 자료형? | 1. 기본형(원시형)  - 정수(byte, short, int, long)  - 실수(float, double)  - 문자(char)  - 논리(boolean) | 2. 참조형(사용자 정의형)  - 클래스 : 문자열(String)  - 인터페이스 : 컬렉션  - 배열 |
| b. 값형 vs 참조형 비교 | 기본 자료형으로 선언된 변수에는 실제 값이 저장된다 스택  참조 자료형으로 선언된 변수에는 인스턴스에 대한 주소 값이 저장된다 | |
| 값형 8개 특징 | - 정수  byte  short  int  long  - 실수  Float  double  - 문자  char  - 논리  Boolean : true or false | |
| 참조형 | 클래스 : 클래스로부터 생성된 객체(인스턴스)의 시작 주소값을 가진다  인터페이스 : 인터페이스를 구현한 클래스의 인스턴스 시작 주소값을 가진다  배열 : 생성된 배열의 시작 주소값을 가진다 | |

|  |  |
| --- | --- |
| 6. 변수 + 생명주기 | |
| a. 지역 변수? | - 메서드나 제어문 안에서 선언된 것  - 메모리가 할당되었다가 제어문을 빠져나가면 소멸 |
| b. 멤버 변수? | - 객체 생성될 때 생성  - 소속된 인스턴스가 소멸될 때 같이 소멸. |
| c. 객체 변수 | - 각 인스턴스의 개별적인 저장공간. 인스턴스마다 다른 값 저장가능  - 인스턴스 생성 후, ‘참조변수. 인스턴스 변수명’으로 접근  - 인스턴스를 생성할 때 생성되고, 참조변수가 없을 때 가비지 컬렉터에 의해 자동 제거됨 |
| d. 정적 변수 | - 제일 먼저 만들어짐  - 프로그램 종료 시 소멸, 가장 마지막에 소멸 |

|  |  |
| --- | --- |
| 7. 형변환 | |
| a. 값형식 형변환 |  |
| b. 참조형식 형변환 | byte b, int n  b = n //명시적 형변환  작은 자료형에 큰 자료형을 대입하면 오버플로우가 발생해서  n = b //암시적 형변환 |

명시적 형변환 vs 암시적 형변환

명시적 : 큰 자료형에서 작은 자료형으로 넣을 때

암시적 : 자동으로 > 작은 형이 큰 타입의 형태가 될 때

|  |  |
| --- | --- |
| 데이터(Data) 영역 | 전역 변수와 static 변수가 할당되는 영역  프로그램의 시작과 동시에 할당되고, 프로그램이 종료되어야 메모리에서 소멸됨 |
| 스택(Stack) 영역 | 함수 호출 시 생성되는 지역 변수와 매개 변수가 저장되는 영역  함수 호출이 완료되면 사라짐 |
| 힙(Heap) 영역 | 필요에 의해 동적으로 메모리를 할당 할 때 사용  할당해야 할 메모리의 크기를 프로그램이 실행되는 동안 결정해야 하는 경우  (런타임 때) 유용하게 사용되는 공간 |

|  |  |
| --- | --- |
| 8. 콘솔 입력 방식 | |
| - 손 코딩 | 사용자의 키보드 입력 : System.in  InputStream - byte   * InputStreamReader - character   BufferedReader - 문자열  사용자가 요구할 때 버퍼에서 읽어오게 한다.  BufferedReader를 이용하면 속도를 향상, 시간의 부하를 줄일 수 있음.  BufferedReader의 readLine이라는 메소드를 이용하면 줄 단위로 사용자의 입력을 읽을 수 있다. |
| - Scanner 사용 | next - 단어  nextLine - 라인  nextInt - 정수 |
| - 문자열 입력,  숫자 입력 |  |

|  |  |
| --- | --- |
| 9. Calendar | |
| - 손 코딩 |  |
| - 현재시간, 특정시간 |  |
| - 원하는 항목 출력 |  |
| - 연산(시각 + 시각, 시각 - 시각) |  |

|  |  |
| --- | --- |
| 10. 연산자 | |
| - 손 코딩 |  |
| - 알고리즘,  자료구조에 사용 |  |
| - 숫자형 연산  (연산 결과는 더 큰 피연산자 타입으로 반환) |  |
| - 문자열(equals) |  |
| 11. 메서드 | |
| - 뭐? 왜? | - 반복적으로 코드가 발생했을 때 그것을 재사용하기 위한 기본적인 단위 |
| - 사용했을 때와  안 했을 때 차이 | - 코드의 가독성이 좋다. |
| a. 매개변수, 파라미터,  아규먼트  (가인자, 실인자) | 가인자 - 메서드 선언부에서 인자값을 받아주는부분  실인자 - 인자값을 메서드 실행부에서 넘겨주는 값 |
| b. 반환값 |  |
| c. 오버로딩?  왜? 장점? | 한 클래스 내에서 메서도 이름은 같은데 인자 값을 바꿔 사용  같은 이름의 메소드를 여러 개 가지면서 매개변수의 유형과 개수가 다르도록 함 |
| d. 오버라이딩?  왜?? | 부모 > 자식 클래스 메서드 재정의  상위 클래스가 가지고 있는 메소드를 하위 클래스가 재정의 해서 사용 |
| 오버로딩vs오버라이딩 |  |
| e. Call by Value,  Call by Reference | 매개변수를 전달할 때의 차이  Call by Value : 값에 의한 호출 값을 복사해 넘기기 때문에 원본 손상X  Call by Reference : 참조에 의한 호출 > 명시적으로 클래스 객체를 주고 받을 때 |

|  |  |
| --- | --- |
| 12. 재귀 메서드 : 자기 자신을 반복적으로 호출 하는 형태 | |
| - 손 코딩 |  |

|  |  |
| --- | --- |
| 13. 값 바꾸기 | |
| - 손 코딩  (정렬 알고리즘..)  - 컵 3개 |  |

|  |  |
| --- | --- |
| 14. 제어문 | |
| - 손 코딩 |  |

|  |  |
| --- | --- |
| 손 코딩 | |
| a. 프로그래밍 기본 개념-> 프로그램의 흐름? | |
| b. 알고리즘  -> 제어문 + 연산자 | |
| c. 자료구조  -> 배열  -> 컬렉션 | : 순수배열, ArrayList, HashMap //필수  : Stack, Queue, Tree //여유 |

|  |  |
| --- | --- |
| 15. 문자열 | |
| - 손 코딩  - ex)주민번호 검사 등.. |  |

|  |  |
| --- | --- |
| 16. 배열(\*\*\*\*\*\*) : 자료형의 종류가 아닌 자료형의 집합을 의미 | |
| 크기가 정해져 있음 | |
| - 멤버 체크 |  |
| - 문제 체크 |  |

|  |  |
| --- | --- |
| 리스트 : 리스트는 배열의 약점을 보완하기 위해서 만들어진 자바의 내장 자료형 | |
| - List | 크기가 정해져 있지 않음 |
| - ArrayList | 객체의 수가 많아지면 자동으로 크기 조절  index를 빨리 가져올 수 있음 -> 내부적으로 배열을 이용하기 때문  Size() : 데이터 크기도 쉽게 알 수 있음  데이터 추가 삭제마다 데이터를 옮김 -> 시간이 많이 필요 |
| - LinkedList | 노드가 하나의 링크 필드에 의해서 다음 노드와 연결되는 구조  삽입, 삭제가 빠르다  탐색이 느리다 |
| - **HashMap** | key, value 값으로 데이터를 저장  key값은 중복이 되지 않고, value값은 중복이 허용 |
| 17. 클래스(메인) \*\*\*\*\*\*\*\*(ex44 ~ ex87) | |
| - 클래스 뭐냐?  객체 생성, 구현 특징 | 객체들을 여러 개 만들기 위한 하나의 틀(설계도)  객체 : 사물 또는 개념  클래스 내에는 여러 가지의 필드와 메서드를 가짐 |
| - 캡슐화, 정보 은닉화, 유일성, 구현 등.. | 캡슐화 > 객체의 속성과 동적을 하나로 묶어줌., 데이터의 일부를 감추어 외부와의 상호작용은 메서드를 통하여 할 수 있게 한다(정보은닉) |
| a. 클래스 선언 | -> 객체 생성 연습 |
| b. 접근 지정자  (4가지 - 범위 구분) | public : 모든 접근을 허용..  private : 외부에서 접근이 불가능. 즉, 같은 클래스 내에서만 접근이 가능. protected : 상속받은 클래스 또는 같은 패키지에서만 접근이 가능.  default : 기본 제한자로써 아무것도 붙지 않고,  자신 클래스 내부와 같은 패키지 내에서만 접근이 가능.  사용이유 : 외부로부터 보호해야 할 데이터를 숨기기 위해서 > 보안성(정보은닉화)   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **한정자** | **클래스 내부** | **동일 패키지** | **하위 클래스** | **그 외의 영역** | | public | ● | ● | ● | ● | | protected | ● | ● | ● | X | | default | ● | ● | X | X | | private | ● | X | X | X |   접근 영역: public > protected > default > private |
| c. 객체 멤버, |  |
| - 정적 멤버(static) | 클래스에 고정된 필드와 메서드  객체를 생성하지 않고 클래스로 바로 접근해서 사용가능  메모리 할당을 딱 한번만 하게 되어 메모리 사용에 이점  Static 변수의 값을 공유 |
| d. 생성자.  왜 | 클래스명과 메서드명이 동일, 리턴 타입을 정의 하지 않음  필수 적인 행동을 인스턴스 생성시에 제어할 수 있음  객체 생성의 편리, 가독성을 높이기 위해, (가독성, 객체 생성 편리 등) 오버로딩 |
| e. setter, getter. 왜? | 캡슐화-> 개체 무결성 유지 |
| f. 상속. 설명, 왜? 장점 | - 파생, 구현, 확장 등..  - 코드 재사용(\*\*\*) -> 물리적 -> 패키지 import  자식 클래스는 부모클래스의 모든 속성과 동작을 물려받는다.  자식 클래스에만 필요한 기능이 있다면 추가 또는 변경 가능 |

|  |  |
| --- | --- |
| 18. Object 클래스 | |
| - 자바에서의 최상위 클래스 |  |
| - 상속,  참조형 형변환 등.. |  |

|  |  |
| --- | --- |
| 19. 추상 클래스 | |
| - 추상 클래스 | - 구현부 가짐  - 클래스를 통해 만들어낸 객체가 객체의 모습이 아님  (클래스가 객체를 생성 할 수 없도록) |
| - 추상 메서드 | - 내용이 비어있는 메서드 |

|  |  |
| --- | --- |
| 20. 인터페이스 | |
| 인터페이스 | - 구현부 못가짐  - 추상메서드로만 이루어져 있으며 메서드의 선언만 가능  - 서브 클래스에서 메서드를 반드시 구현하도록 만들기 위해서 사용  (서로 다른 클래스를 연결해 주는 연결 장치)  - (자바는 public static final 변수 o)  - 다중 상속은 인터페이스에서만 가능함 |

|  |  |  |
| --- | --- | --- |
|  | 추상클래스 | 인터페이스 |
| 공통점 | 클래스  하위 클래스에서 모든 추상메서드를 구현해야 함 | |
| 차이점 | 추상메서드 외 일반 멤버 변수와 메서드를 가질 수 있다.  Extends를 사용  단일 상속만 가능  작업의 레벨 분할을 위해서 사용 | 추상메서드와 static final변수만 사용  Implement 를 사용  중복구현 가능  공동 작업을 위한 상호간의 인터페이스를 위해 사용 |

|  |  |
| --- | --- |
| 22. final //여유 | |
| final | - 자바는 const 존재 x -> final 키워드 대신 |

|  |  |
| --- | --- |
| 21. 추상클래스 -> 대리점(프린터 예제) | |
|  |  |

|  |  |
| --- | --- |
| 23. 패키지 //여유 | |
| 패키지 | - 수많은 클래스를 관리하기 위해  - 특정 클래스 ? -> java.io.Input 클래스, java.sql.Item 클래스  - 패키지 없이 클래스 선언 -> default package에 들어감(패키지x)  -> import 할 수 없음 import 패키지명.클래스명; |

|  |  |
| --- | --- |
| 24. Ex63\_cast(\*\*\*)\_ 참조형에서 사용 | |
| 캐스팅 | 캐스팅(casting)이란 개발자가 프로그램 코드로 행하는 강제 타입변환 |
| 업캐스팅 | 서브클래스가 슈퍼클래스 타입으로 변환되는 것을 뜻함.  업 캐스팅을 하면 객체 내에 있는 모든 멤버를 접근할 수 없고  슈퍼 클래스의 멤버만 접근할 수 있다. |
| 다운캐스팅 | 업 캐스팅 된 것을 다시 원래대로 되돌리는 것.  업 캐스팅과 달리 명시적으로 타입을 지정해야 한다. |
| 상속 | 코드 재 사용성 |

|  |  |
| --- | --- |
| 25. 제네릭 | |
| 제네릭 | - 코드상에서 타입을 미리 결정하지 않고 컴파일 시 문맥에 따라 자료형을 결정  - 제네릭 컬렉션  - 생산성  - int, String, boolean -> Object -> Generic |

|  |  |
| --- | --- |
| 26. 인터페이스(\*\*\*) | |
| - 왜? 역할?  사용 o, 사용 x |  |

|  |  |
| --- | --- |
| 27. enum | |
| enum | - 수업 x -> 업무 o  - 개념  - <select>, <radio>  - 열거형(숫자 상수를 이용하는 것 보다 직관적임 > 상수를 묶어서 관리)  -> 코드 안정성, 생산성, 의사 소통 등.  단순 상수로 사용  값 설정해서 사용  메서드 오버라이드  Switch / case에 사용  인터페이스의 구현체가 될 수 있음  메서드 안에서 사용하면 안됨,  잘못된 데이터가 사용될 가능성이 적다. |

DOM과 SAX의 차이

DOM 은 몇 번이고 원하는 요소 정보를 바로 찾아가 추가 및 수정할 수 있는 수단인데

비해, SAX는 문서를 처음에서 끝까지 순차적으로 처리하며 읽는 기능으로만 처리할 수 있다.

DOM 은 전체 문서를 메모리상에 올려놓고 처리하므로 원하는 요소를 바로 찾아가 추가 및 수정할 수 있는데

비해, SAX는 문서를 처음에서 끝까지 순차적으로 처리하며 문서의 일부분만을 메모리에 올려 처리하므로

메모리를 적게 사용하는 반면 특정 요소를 찾아 수정하는 기능은 불가능하다.

DOM 은 XML 문서에 대한 구조적 접근이 필요한 경우, 문서 정보를 쉽게 파악하고자 할 때 사용되며 SAX 는 문서의 일부분만 읽을 때 데이터 변환 시, 유효성 처리 시 사용한다

라이브러리는 라이브러리를 가져다가 사용하고 호출하는 측에 전적으로 주도성이 있다.

반면, 프레임워크는 그 틀 안에 이미 제어 흐름에 대한 주도성이 내재한다.

프레임워크는 가져다가 사용한다기보다는 거기에 들어가서 사용한다는 느낌/관점이다.

프레임워크는 전체적인 흐름을 스스로가 쥐고 있으며 사용자는 그 안에서 필요한 코드를 짜 넣는다.

반면 라이브러리는 사용자가 전체적인 흐름을 만들며 라이브러리를 가져다 쓰는 것이다.

|  |  |
| --- | --- |
| 28. 컬렉션 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |
| - 모든 종류 정리(특징(구조) -> 용도) | |
| - 구조 파악 -> 사용(코드, 손코딩) | : ArrayList, HashMap, Stack, Queue(LinkedList) |

|  |  |
| --- | --- |
| List 계열 | 특징: 순서가 있고, 중복을 허용 (배열과 유사)  장점: 가변적인 배열(배열이 자동으로 늘어남)  단점: 원하는 데이터가 뒤쪽에 위치하는 경우 속도의 문제  방식: equals( )를 이용한 데이터 검색  ArrayList LinkedList |
| Set 계열 | 특징: 순서가 없고, 중복을 허용하지 않음  장점: 빠른 속도  단점: 단순 집합의 개념으로 정렬하려면 별도의 처리가 필요  HashSet TreeSet |
| Map계열 | 특징: Key(키)와 Value(값)으로 나뉘에 데이터 관리,  순서는 없으며, 키에 대한 중복은 없음  장점: 빠른 속도  단점: Key의 검색 속도가 검색 속도를 좌우  HashMap TreeMap |
| Tree 계열 | 각 Element(Node)들의 균형된 분포가 관건  Balanced Tree -  Node들의 분포가 고른 Tree구조  어떤 Node를 찾아도 검색 depth가 일정하게 구성  TreeSet TreeMap |
| Stack 클래스 | 특징: 아래가 막혀있는 순서 있는 구조  장점: 위쪽의 데이터를 빈번하게 접근하는 경우에 유리  단점: 선형 리스트의 단점  메소드: push( ), pop( ), peek( ) (예: FILO(first in last out))  가장 최신것이 가장 빨리 나온다. |
| Iterator  /  Enumeration | Enumeration(JDK1.0) / Iterator(JDK1.2)  거의 모든 자료구조 클래스들이 구현하는 인터페이스  어떠한 자료구조 클래스에 관계없이 자료들을 조회하고 싶은 경우에 사용  Interface에서 정의된 기능들  아직 조회할 데이터가 남아 있는가  Enumeration - hasMoreElements( )   /  Iterator - hasNext( )  다음 데이터를 가져온다  Enumeration - nextElement( ) / Iterator – next( )  데이터의 삭제 : Iterator – remove( ) |

|  |  |
| --- | --- |
| 29. 파일입출력 | |
| - 코딩 문제 |  |
| - 읽기/쓰기(BufferedReader, BufferedWriter) |  |

|  |  |
| --- | --- |
| **동기방식 vs 비 동기 방식 차이** | |
| 동기방식 | 정의 : 작업이 들어온 순서에 맞게 차근차근 진행  작업 요청을 했을 때 그 요청의 결과값을 간접적으로 받는 것  특징 : 순서에 맞게 빨리빨리 진행되는 장점이 있지만, 여러 가지의 일을 동시에 처리할 수 없는 단점이 있다. |
| 비동기 방식 | 특징 : 실행 함수가 끝난 뒤 되돌아오는 반환 함수,  콜백 함수를 활용하여 일 처리 순서를 제어  작업 요청을 했을 때 그 요청의 결과값을 간접적으로 받는 것  자바스크립트에서 말하는 비동기로직이란 여러 개의 프레임으로 명령을 분산해 적재 직접 적재하는 방법(setTimeout, requestAnimationFrame, setInterval), 시스템이 적절하게 적재하는 방법(이벤트 리스너 지정 등)  특징 : 여러 가지 일을 한꺼번에 처리할 수 있지만, 동기 방식보다 처리 속도가 떨어짐  해당 요청작업은 별도의 쓰레드에서 실행하게 되며, 비동기 요청처리 결과 전달 방법에는 콜백(Call Back) - 요청 처리가 완료되면 콜백 호출과 결과 조회 - Future. 비동기 작업 결과를 조회해볼 수 있는 핸들러. 요청이 완료되었는지 확인 기능   1. 사용처 : 1. 1프레임에 너무 많은 명령을 적재하면 최초 1프레임의 렌더링이 늦어져 유저가 2. 화면을 볼 수 있는 대기 시간이 길어진다. 3. 2. 애니메이션처럼 시간자체를 지연하고 싶어 지연한다. 4. 3. 시스템이 수신하는 데이터(이벤트)는 시스템만 루프를 돌며 대기하고 스크립트는 5. 발생한 후 통보를 받는 식으로 하여 부하를 줄일 수 있다.   4. 한 프레임에서 실행되는 명령에는 시간 제약이 있으므로 무거운 로직은 여러 프레임에 걸쳐 처리하도록 해야 한다 |

**[오라클]**

1. 오라클 11g 사용 - ANSI SQL(표준 SQL), PL/SQL

2. 클라이언트툴 - SQL Developer, SQLGate, Toad

3. 모델링, ERD, 정규화 - eXERD, er-win 등

|  |  |
| --- | --- |
| 3. 정규화 | |
| 정규화 | 자료의 손실이나 불필요한 정보를 없애고,  데이터 일관성이나 데이터 중복을 최소화 하기 위해 자료 구조를 변경하는 작업 |
| 목적 | 자료 저장에 필요한 공간을 최소화  자료 삽입, 갱신, 삭제에 따른 이상 현상 제거  자료 구조의 안정화 등 |
| 이상 현상 | 1. 입력 이상 - 데이터 입력 시 필요 없는 속성까지 입력해야 하는 현상  2. 수정 이상 - 데이터 수정 시 원하지 않는 데이터까지 수정되는 현상  3. 삭제 이상 - 데이터 삭제 시 필요한 데이터까지 삭제되는 현상 |
| 종류 | 1. 제 1 정규화, 1NF(Normal Form)  - 모든 컬럼(속성)은 원자 값을 가진다.  - 여러 개의 값을 동시에 가지는 컬럼은 존재할 수 없다.  - 모든 컬럼에는 값이 존재해야 하고 중복 값이 발생하면 안 된다.  - 어느 한 쪽이 다른 한쪽을 꼭 참조해야만 한다. (PK - F),  관개를 맺어야 한다.  - 정규화를 하면 조인을 해야 한다.  (정규화가 많아질 수록 조인을 많이 할 수 밖에 없음)  -- 1 : N  2. 제 2 졍규화, 2NF  - 키가 아닌 모든 컬럼(속성)은 키 전체에 종속이어야 한다.(완전 함수 종속)  - 키의 일부 속성에만 의존적인 속성이 있어서는 안 된다.  - 키에 종속적이지 않는 속성이 있어서는 안 된다.  3. 제 3 정규화, 3NF  - 키가 아닌 컬럼에 의존하면 안 된다.  - 다른 컬럼에 의해 계산되는 컬럼이 존재하면 안 된다.  4. 제 4정규화, 4NF |

|  |  |
| --- | --- |
| 4. 자료형 | |
| 숫자형 | a. NUMBER  - 정수 & 실수  - 38자리 숫자를 표현(20BYTE\_ 최대, 자유롭게 사용, 크기에 따라 용량이 변동)  - NUMBER(PRECISION, SCALE)  - PRECISION : 소수 이하를 포함한 전체 자리 수(1~38). 정밀도  - SCALE : 소수점 이하 자리 수. 스케일  b. PLS\_INTEGER  c. BINARY\_INTEGER  d. 서브타입  - DECIMAL, FLOAT, INTEGER, REAL -> 거의 NUMBER를 사용, 자리 수 주의! |
| 문자형 | - CHAR + STRING  - 오라클에서의 문자 집합(character set)  -. 단일 바이트 vs 바이트  -. 고정폭 vs 가변폭  - 9i 부터 기본적으로 UTF-8을 사용한다.  a. CHAR / NCHAR  - 고정 자리수  - CHAR(N) : N자리 문자열(N : 바이트) \_ N값을 생략할 수 없음  - N 최대 자리수 : 2000바이트  - EX) CHAR(3) : 3바이트 문자열 자료형  - 장점(속도 빠름), 단점(공간 낭비)\_ 남은 자리수는 공백으로 처리  - EX) 이름, 주민번호, 전화번호\_ 최소한의 움직임이 있는 문자열,  0~N까지의 최대 자리 숫자를 표현  b. VARCHAR2 / NVARCHAR2  - 가변 자리수  - VARCHAR2(N) : N자리 문자열(N : 바이트) \_ N값을 생략할 수 없음,  주로 많이 사용함, CHAR과 성능 차이가 거의 나지 않음  - N 최대 자리수 : 4000바이트\_ 숫자는 바뀔 수 있음(오라클의 버전에 따라?!)  - EX) VARCHAR2(10) : 10바이트 문자열 자료형  - 장점(공간 절약), 단점(속도 느림, CHAR에 비하여)\_ 남은 공백 반환  - EX) 주소, 게시판 글 제목\_ 가변 폭이 있는 데이터  c. N의 유무  - 문자셋의 차이  - N이 없으면 -> UTF-8 사용  - CHAR(3) : 3바이트(영어 3글자, 한글 1글자) 즐겨 씀, 물리적으로 효과적임  - N이 있으면 -> UTF-16 사용  - NCHAR(3) : 3글자(유니코드 지원, 영어, 한글 모두 3글자, 잘 사용하지 않음) |
| 날짜 / 시간형 | a. DATE(\*\*\*)  - B.C 4712년 1월 1일 ~ A.D 9999년 12월 31일까지 표현이 가능  - 7BATE  - 날짜와 시간을 모두 포함한다.(Calendar)  - 최소 단위가 초 단위이다.(그 이하는 판별 불가능)  b. TIMESTAMP  - DATE보다 더 하위 단위까지 표현이 가능  - 10억분의 1초까지 표현  c. INTERVAL  - 특정 시각이 아닌 시각과 시각이 차이를 저장(시간 저장)  NUMBER로 사용가능하기 때문에 잘 사용하지 않음 |
| 대용량 자료형 | - 오라클 문자열 상수가 최대 4000자까지 표현 가능(영어에 한해서)  a. LOB, Large Obect  - 대용량의 텍스트나 멀티미디어 데이터 등  활용도가 높지 않음, 용량이 큼(느려짐)  - 4GB  - 참조형\_외부 참조 주소값을 사용함, 최대한 안쓰는 것이 좋음  가. CLOB  - 싱글 바이트 문자열\_ 일반적으로 사용  나. NCLOB  - 멀티 바이트 문자열  다. BLOB  - 이진 데이터. 동영상, 이미지 등..(문자열이 아닌 나머지 형)\_  b. LONG  - 호환성 때문에 남아있음.  - LOB를 사용 |

|  |  |  |
| --- | --- | --- |
| DDL(Data Definition Language) : 데이터 정의 언어  DB Object를 생성하고 관리한다.(테이블, 뷰, 함수 등) 데이터베이스 관리자(담당자), 프로그래머(일부) | | |
| CREATE | 생성 | CREATE TABLE 생성할 테이블명(         필드명1 타입명,         필드명2 타입명,  primary key(필드명) }; |
| ALTER | ADD 새로운 필드를 추가  MODIFY 생성되어 있는 테이블의 필드를 수정  DROP 생성되어 있는 테이블의 필드를 삭제 | ALTER TABLE 테이블명 적용옵션(  필드명 타입명 ); |
| DROP | 삭제 | DROP TABLE  테이블명; |
| TRUNCATE | DROP 후 CREATE |  |

|  |  |
| --- | --- |
| DML(Data Manipulation Language) : 데이터 조작 언어  SQL 80%이상의 비중을 차지함 데이터베이스 관리자(담당자), 프로그래머(\*가장 많이 사용\*) | |
| SELECT | 조회  SELECT 필드명1, 필드명2 ... FROM 테이블명1, 테이블명2 WHERE 필드명3='필드값3'  AND 필드명4='필드값4' ORDER BY 필드명5 [ASC|DESC]; |
| INSERT | 추가  INSERT INTO 테이블명 (필드명1, 필드명2) VALUES (필드값1, 필드값2); |
| DELETE | 삭제  DELETE FROM 테이블명 WHERE 필드명1='필드값1' AND 필드명2='필드값2'; |
| UPDATE | 변경  UPDATE 테이블명 SET 필드명1='필드값1', 필드명2='필드값2'  WHERE 필드명3='필드값3' AND 필드명4='필드값'4'; |

|  |  |
| --- | --- |
| DCL(Data Control Language) : 데이터 제어 언어  사용자와 데이터에 대한 접근 권한 부여, 박탈 등 데이터베이스 관리자(담당자) | |
| COMMIT | 트랜잭션의 작업 결과를 반영 |
| ROLLBACK | 트랜잭션의 작업을 취소 및 원래대로 복구 |
| GRANT | 사용자에게 권한 부여 |
| REVOKE | 사용자 권한 취소 |

|  |  |
| --- | --- |
| 6. 연산자 | |
| 연산자 | 1. 산술 연산자  - +, -, \*, /  - % : SQL은 나머지 연산자 없음(함수(메서드)로 제공)  2. 비교 연산자  - >, >=, <, <=  - =(==), <>(!=)  3. 논리 연산자  - AND(&&), OR(||), NOT(!) \_ 모두 영문으로 작성  4. 문자열 연산자  - ||  5. SQL 연산자  - JAVA(obj intstanceof Class)  - IN, BETWEEN, LIKE, IS NULL, NOT NULL, ANY, ALL.... |

|  |  |  |
| --- | --- | --- |
| 7. select - 문제풀이 연습하기 | | |
| a. 기본 select | | - SELECT 컬럼리스트 FROM 테이블  1. 컬럼 리스트  - 원하는 컬럼만 가져오기  - 수직 필터링 |
| b. 컬럼 제어(별칭) | |  |
| c. 조인(inner, outer) | | 2개 이상의 테이블 내용을 한번에 가져오는 기술  관계를 맺고 있는 테이블끼리만 가능함   1. 내부 조인, Inner join(\*사용 빈도수 높음\*)   - 각 테이블에 조건으로 들어간 컬럼 값을 조합해서 결과 셋으로 반환  - 부모테이블(PK) - 자식테이블(FK)이 일치하는 행만 유효한 행으로 판단  - 단순 조인 -> (유효한 행 추출) -> 내부조인  2. 외부 조인, Outer join(\*실무에서 많이 사용\*)  - 내부 조인 + a  - 양쪽 테이블에 모두 동시에 존재하는 내용만을 가져온다. (교집합)  3. 셀프 조인, Self join(빈도수 낮음)  - 1개의 테이블을 사용해서 조인  4. 크로스 조인, Cross join, 단순 조인(X, 사용하지 않음, 의미 없음)  - 왼쪽의 한 행과 오른쪽의 한 행을 붙여서 새로운 행을 만들어냄(A \* A ...)  - JOIN을 기준으로 양쪽의 테이블의 값을 한 행에 보여줌  (필요한 데이터와 필요 없는 데이터가 섞여있음)  -> 데이터를 알 수 있지만 보기에 매우 불편함  -> 그래서 잘 사용하지 않음 |
| d. 함수들 | 수학 | 1. ROUND 함수  - NUMBER ROUND(컬럼명)\_ 정수형태로 출력(소수 이하 반올림)  - NUMBER ROUND(컬럼명, 자릿수)\_  - 반올림  2. FLOOR, TRUNC 함수  - 절삭 함수, 무조건 내림 함수  - 반올림 O -> X, 반올림 X -> X  - NUMBER FOOLR(컬럼명)  - NUMBER TRUNC(컬럼명 [,자릿수])  - 현재 숫자와 가장 가까운 작은 정수 중 최대 정수를 반환한다.  3. CEIL 함수  - 무조건 올림  - NUMBER CEIL(컬럼명)  - 현재 숫자와 가장 가까운 큰 정수 중 최소 정수를 반환한다.  4, MOD 함수  - 나머지 함수  - NUMBER MOD(피제수, 제수) |
| d. 함수들 | 문자열 | 1. UPPER, LOWER, INITCAP 함수  - VARCHAR2 UPPER(컬럼명)  - VARCHAR2 LOWER(컬럼명)  - VARCHAR2 INITCAP(컬럼명)  2. SUBSTR 함수  - 문자열 추출 (서수는 1부터 시작함(\*주의!\*))  - VARCHAR2 SUBSTR(컬럼명, 시작위치, 개수)  - VARCHAR2 SUBSTR(컬럼명, 시작위치)  - 문자열.subString(시작위치, 끝위치) : JAVA  - 문자열.subString(시작위치) : JAVA  3. LENGTH 함수  - NUMBER LENGTH(컬럼명)  - 문자열의 길이  4. INSTR 함수  - indexOf()  - 특정 문자가 출현하는 위치를 반환  - NUMBER INSTR(대상문자, 검색문자)  - NUMBER INSTR(대상문자, 검색문자, 시작위치)  5. LPAD, RPAD 함수  - VARCHAR2 LPAD(컬럼명, 갯수, 문자)  - VARCHAR2 RPAD(컬럼명, 갯수, 문자)  - 남는 공간에 특정 문자 채우기  6. LTRIM, RTRIM, TRIM  - VARCHAR2 LTRIM(컬럼명 [, 문자])  - VARCHAR2 RTRIM(컬럼명 [, 문자])  - VARCHAR2 TRIM([문자 FROM] 컬럼명)  7. REPLACE, TRANSLATE 함수  - 문자열 치환  - VARGHAR2 TRNSLATE(컬럼명, 문자, 문자) : 1문자 치환  - VARGHAR2 REPLACE(컬럼명, 문자열, 문자열) : 문자열 치환  8. DECODE 함수(\*사용 빈도수 높음\*)  - 문자열 치환  - VARCHAR2 DECODE(컬럼명, 문자열, 문자열)  9. CASE 함수  - java : switch case문  - DECODE와 유사, CASE는 범위 검사가 가능 |
| d. 함수들 | 날짜 | 1. SYSDATE  - 현재 시각 반환  - DATE SYSDATE \_ 인자값이 없어서 소괄호 없음  2. LAST\_DAY  - 해당 날짜가 속한 월의 마지막 날짜  3. MONTHS\_BETWEEN  4. ADD\_MONTHS  - DATE ADD+MONTHS(DATE, NUMBER) |
| e. where 절 | | - 원하는 행만 가져오기  - 수평 필터링  - 조건절(조건을 만족하는 행만 가져오기) -> 비교, 논리 연산자, 함수 등 사용  - 모든 행을 대상으로 질문  - BETWEEN : 범위 조건을 지정  - (WHERE) 컬럼명 BTWEEN 최소값 AND 최대값  - (WHERE) NOT 컬럼명 BTWEEN 최소값 AND 최대값  - IN : 열거형 비교  - (WHERE) 컬럼명 IN (열거값)  - (WHERE) 컬럼명 NOT IN (열거값), 열거 값을 뺀 나머지의 값 출력  - (WHERE) NOT 컬럼명 IN (열거값), 뒤에 있는 조건을 전부 부정, 값은 같음  - LIKE : 문자열 대상  - (WHERE) 컬럼명 LIKE 패턴문자열 (\_, %)  - NULL : 특정 컬럼이 NULL상태인지 검사  - (WHERE) 컬럼명 IS NULL  - (WHERE) 컬럼명 IS NOT NULL  - (WHERE) NOT 컬럼명 IS NULL |
| f. order by절 | | - 결과 셋의 레코드(행) 정렬  - 특정 컬럼 값을 가지고 정렬 -> 오름차순, 내림차순  - 정렬 대상 -> 숫자, 날짜, 문자  - 일반적으로 NULL값은 빼고 정렬을 많이 한다.  - ORDER BY 컬럼명 [ASC|DESC] \_ 기본 오름차순 |
| g. group by절, | | - 행(데이터 1개)들을 특정 기준에 맞춰 그룹을 짓는 역할  - 왜 그룹을 지을까? -> 그룹내의 행(데이터)을 대상으로 집계함수를  - GRUOP BY + 집계 함수  - WHERE절 GROUP BY절 ORDER BY절 |
| Having절 | | - 조건절(WHERE와 같은 역할)  - WHERE : FROM과 연동 -> 단일 레코드에 대한 조건  - HAVING : GROUP BY와 연동 -> 집합 함수의 결과에 대한 조건  - GROUP BY 컬럼명 HAVING 조건 |
| h. 집계함수 | | 1. COUNT : 개수  2. SUM : 합계  3. AVG : 평균  4. MAX : 최대값  5. MIN : 최소값  \* 주의 집계함수로 절대 할 수 없는 행동 \*  1. 컬럼 리스트에는 집계함수와 일반 컬럼을 동시 사용 불가  2. 집계(그룹) 함수는 WHERE절에 사용이 불가  : WHERE절은 단일 데이터를 처리 하는 구문(그룹 함수는 올 수 없다) |
| i. 서브쿼리 | | - 쿼리 안에 또 다른 쿼리가 있는 형태  - SELECT안의 SELECT  - 소괄호로 분류 |
| j. rownum | | - 행의 일련 번호를 반환  - 결과 셋으로 레코드 순서를 가져오는 컬럼  - FROM절 바로 직후에 할당된다.(\*매우 중요\*) |
| k. union | | - 두 테이블을 결합할 때 중복되는 행은 자동으로 제거(DISTINCT 적용) |

|  |  |
| --- | --- |
| 8. 테이블 생성 | |
| - 컬럼 자료형?? |  |
| - 제약 사항(5가지) | 1. NOT NULL  - 반드시 컬럼 값을 기재 (필수 값들은 NOT NULL로 정의하는 것이 좋음)  - 컬럼 레벨로만 정의 가능  2. PRIMARY KEY : 행을 구분하기 위한 유일성을 보장 받음  - 기본 키, 모든 테이블에 1개 이상이 있어야 함  - 키(KEY) : 속성, 컬럼  - 엔티티 객체들을 구분하는 식별자 -> 행과 행을 구분하는 유일한 식별자(\*\*)  - 자동으로 NOT NULL(NULL값을 가질 수 없음)  - 유일한 값을 가진다.(UNIQUE)  - 일반적으로 테이블에는 PK 1개가 있다.(1개 이상의 PK도 존재 가능 - 복합키)  3. UNIQUE  - 해당 컬럼 값은 테이블 내에서 유일해야 한다.(PK와 거의 유사함)  - 데이터의 유일성을 보장 받기 위해 사용  - 행을 구분하는 용도가 아니다!!!!!  - NULL을 가질 수 있다. (ENIQUE + NOT NULL -> PRIMARY KEY) ,  NULL은 데이터로 중복이 가능하다  4. CHECK  - 열거형 제약 (미리 정해진 값 외에는 사용 불가)  - CHECK (컬럼명 IN (제약사항 열거)  - CHECK (컬럼명 BETWEEN 최소값 AND 최대값), |

|  |  |
| --- | --- |
| 9. 시퀀스 | |
| - 어디다? | - 인덱스를 만들기 위해서 시퀀스 객체를 사용했습니다. |
| 시퀀스 | 연속된 숫자 데이터가 필요한 경우 사용  대부분 식별자 생성 용도로 사용(유일성을 보장) – 번호 생성기 같은 역할을 함  독립적임(테이블에 종속적이지 않음), 한 곳에서만 쓰는 것을 권장함  테이블 하나당 하나의 시퀀스를 만들어서 사용하는 것이 좋음  시퀀스를 부른 횟수를 기억하기 때문에 실패나 성공의 결과에 상관없이 계속 카운트 됨  식별자이기 때문에 연속적인 숫자일 필요는 없음(\*유일성\*) |

|  |  |
| --- | --- |
| 10. 뷰 | |
| - 왜?  - 장점? | - 코드 재사용, 간결성 -> 의사 소통 원활  1. 복잡하고 길고 반복되는 질의를 단순화 시킨다. (특히 조인문)  2. 보안에 유리하다.  - 특정 유저에게 일부 데이터만 접근 시킨다.  - 하나의 테이블을 여러 용도로 사용이 가능하다. |
| 종류 | 1. 단순 뷰  - 하나의 기본 테이블에 의해 정의된 뷰  - DML을 사용할 수 있다.(INSERT, UPDATE, DELETE)  2. 복합 뷰  - 두 개 이상의 기본 테이블에 의해 정의된 뷰(서브쿼리,조인 등...)  - DML을 사용할 수 없다. |

|  |  |
| --- | --- |
| 11. 트랜잭션 | |
| - 어느 업무에 적용??  - 2~3가지 업무 | - 글을 삭제하는데 오류가 나서 댓글만 삭제했을 경우  기존에 삭제했던 댓글을 복구 해야 하기 때문에  문제가 생김 -> 이런 문제를 해결하기 위해 트랜잭션으로 묶어줘야 한다.  - 쿼리 조작의 일련의 과정, |
| 트랜잭션 | - 오라클에서 발생하는 여러 개의 SQL 명령문들을 하나의 논리적인 작업 단위로 묶어 처리하는 기술  - 모두 성공하면 정상 종료를 하고 하나라도 실패하면 전체를 취소시킨다.  - 데이터의 일관성을 유지하면서 안정적으로 데이터를 복구한다.  - 명령을 실행했던 구간, 단위 |
| 명령어 | 1. COMMIT  2. ROLLBACK  3. SAVEPOINT |
| 12. 인덱스 | |
| 인덱스 | - 검색을 빠른 속도로 하기 위해서 제공  - SQL 명령문의 처리 속도를 빠르게 하기 위해서 컬럼에 대해서 생성되는 객체  - B\* 트리 구조(자바 TreeSet과 유사)  - 오라클은 인덱스를 따로 생성하지 않아도 자동으로 생성이 된다.  (해당 테이블의 PK, UNIQUE 컬럼들이 자동으로 인덱스가 생성), 유일한 값  - 명시적으로 인덱스를 생성하는 경우  (주로 WHERE절의 대상이 되는 컬럼을 사용 권장) 총 관리자 |
| 사용해야 하는 경우 | 1. 테이블의 행 개수가 많은 경우  2. WFERE 절에 해당 컬럼이 많이 사용이 되는 경우  3. 검색 결과가 전체 데이터의 2%~4% 정도 일때  - 100건의 테이블 중 2~4개의결과를 가져올 때  4. JOIN에 사용되는 컬럼(PK, FK)  5. NULL을 포함하는 컬럼이 많은 경우(NULL값은 색인에 제외된다.),  색인은 NULL값은 찾을 수 없음 |
| 사용하지 않는 경우 | 1. 테이블의 행 개수가 적은 경우  2, 검색 결과가 전체 데이터의 많은 비중을 차지하는 경우 |
| 종류 | 1. 고유  2. 비고유  3. 단일  4. 결합  5. 함수 기반 |

|  |  |
| --- | --- |
| 13. 프로시저 | |
| 프로시저 | 특정 작업을 수행 하는, 이름이 있는 PL/SQL BLOCK 이다.  매개 변수를 받을 수 있고, 반복적으로 사용 할 수 있는 BLOCK (재사용 가능)  보통 연속 실행 또는 구현이 복잡한 트랜잭션을 수행하는 PL/SQL BLOCK을 데이터베이스에 저장하기 위해 생성  공유된 메모리 자원 이용 |

|  |  |
| --- | --- |
| 14. 트리거 | |
| 트리거 | - 특정 사건이 발생하면 미리 준비해 놓은 일을 자동으로 실행하는 역할  - 데이터베이스가 미리 정해놓은 조건을 만족하면 자동으로 실행되는 업무  - 직접 호출은 불가능하고 미리 정해놓은 조건이 발생할 때만 실행된다.  - 프로시저의 일종  - 테이블 감시를 통한 업무를 진행(과하면 시스템이 느려질 수 있음) |
| - 1~2가지 업무 | - 테이블의 변경 사항이 생겼을 때, 레코드의 추가, 수정, 삭제 시 사용 |

|  |  |  |
| --- | --- | --- |
| 파일 시스템 vs DBMS차이점 | | |
|  | 파일 시스템 | DBMS |
| 데이터 관리 | 필요한 파일들을 별개로 정의하고 구현. | 사용자 공용(shared) |
| 데이터 독립성 | 파일들이 프로그램에 종속됨 | 프로그램에 독립적으로 데이터가 관리됨 |
| 논리적 독립성 | 데이터 구조가 바뀌면 프로그램을  그에 맞게 변경 | 데이터베이스의 논리적 구조 기술만 변경 |
| 물리적 독립성 | 디스크장치가 바뀌면 데이터의 표현방식이 바뀌어 프로그램을 수정함. | 물리적 기술만 변경 |
| 데이터 접근 | 개별 접근 or 공통 접근 루틴 | DBMS를 통한 접근  (논리적, 물리적 독립성 유지) |
| 데이터 중복성 | 파일 간의 데이터 중복, 비슷한 내용을 포함하고 있어도 상호 교환되지 않는다  일관성, 보안성, 경제성 문제 발생 | 데이터 일관성, 보안성을 유지하고,  경제성을 높임 |
| 무결성 | 데이터 불일치로 인한 무결성 결핍이 높음 | 일관성, 제약조건을 통한 무결성 상승 |
| 동시접속 | 협의되지 않은 다수의 프로그램 접근시 제어가 어렵다 | 동시 액세스(concurrent access) 제공 |
| 조인 불가, 중복저장 할 수 있음 > 일관성 훼손 | | 중복저장 불가능 > 일관성 유지 가능 |

[JDBC] : 참고 http://hyeonstorage.tistory.com/111

- 손코딩

a. 반환값 없는 질의

b. 반환값 있는 질의

|  |  |
| --- | --- |
| 1. Connection | |
| Connection | 네트워크 연결  SQL문장 실행 시키기 위해 사용 |

|  |  |
| --- | --- |
| 2. Statement | |
| Statement | executeQuery()나 executeUpdate()를 실행하는 시점에 파라미터로 SQL문을 전달전달되는 SQL 문은 완성된 형태로 한눈에 무슨 SQL 문인지 파악하기 쉽다.  하지만, SQL문을 수행하는 과정에서 매번 컴파일을 하기 때문에 성능상 이슈가 있다. (이 컴파일을 Parsing 한다고도 표현)  사용 : Dynamic SQL을 사용할 경우, 디버깅이 좋음 |
| PreparedStatement | 준비된 Statement. 이 준비는 컴파일(Parsing)을 이야기하며, 컴파일이 미리 되어있기에 Statement에 비해 성능상 이점이 있다. 보통 조건절과 함께 사용되며 재사용이 되는데, ? 부분에만 변화를 주어 지속적으로 SQL을 수행하기 때문에 한눈에 무슨 SQL 문인지 파악하기는 어렵다.  사용 : 사용자 입력 값으로 쿼리를 생성하는 경우  쿼리 반복수행 작업일 경우 |
| CallableStatement | DB에 생성된 저장 프로시저를 사용  수행 속도가 빠름 |

|  |  |
| --- | --- |
| 3. ResultSet - 레코드 접근 방법(if, while) | |
| ResultSet | SQL 문 중에서 Select 문을 사용한 질의의 경우 성공 시 ResultSet을 반환  ResultSet은 SQL 질의에 의해 생성된 테이블을 담고 있음   '커서(cursor)' > ResultSet에서 특정 행에 대한 참조를 조작 |
| cursor | first() : 커서를 첫 번째 행으로 옮긴다.  last() : 커서를 마지막 행으로 옮긴다.  beforeFirst() : 커서를 첫번째 행 이전으로 옮긴다.  afterLast() : 커서를 마지막 행 다음으로 옮긴다.  next() : 커서를 다음 행으로 옮긴다.  previous() : 커서를 이전 행으로 옮긴다. |

**[HTML]**

|  |  |  |  |
| --- | --- | --- | --- |
| HTML | | | |
| - HTML5 |  | | |
| - 웹 표준화? | -> 구현해봤냐 -> 웹 표준화에 맞춰서 구현했다.  수업을 표준화에 맞춰서 수업했기 때문에 이전상태가 어땠는지 알지 못한다. | | |
| HTML의 역할 | 1. 웹 페이지 제작(웹 문서)  2. 뼈대(골격)와 내용물을 작성하는 용도  3. 서식 작업(디자인 작업) -> 절대 X | | |
| HTML 구성 요소 | 1. 태그(tag), 엘리먼트(Element)  - 문서의 골격을 형성  - 각종 구성 요소 출력(텍스트, 이미지, 링크, 테이블 등)  - <태그명>  a. 태그 형식  ㄱ. <태그명></태그명>  ㄴ. <태그명>, <태그명/>  b. 태그 내용의 형식, Content Type  - <시작태그> 내용 </끝태그명>, 정해진 패턴 내에서 작성해야 함  ㄱ. 자식 엘리먼트  - 시작 태그와 끝 태그 사이에 또 다른 태그만이 올 수 있다.  ㄴ. PCDATA  - 시작 태그와 끝 태그 사이에 문자열만 가지는 형태  ㄷ. Empty  - 빈 태그(단독태그)  ㄹ. 혼합형, Mixed Type  - 시작 태그와 끝 태그 사이에 자식태그, 문자열도 올 수 있는 형태  2. 속성(Attribute)  - 태그의 성질을 정의하는 데이터(문법은 정해져 있음)  - 보조적으로 도와주는 역할(여러 가지 형태로 출력 할 수 있도록 제어)  3. 텍스트(PCDATA, Parsed character Data)  - 출력하고자 작성한 문자  - 화면에 출력되는 데이터 역할(출력용 데이터, 화면에 100% 출력됨)  - 파싱 되는(해석기가 해석하는) 문자 데이터  - 문법 검사를 하는 영역  (태그의 <>와 문장의 <>를 구분 하지 못하기 때문에 에러가)  -> PCDATA 영역 -> 문법검사를 안 하는 영역으로 바꾸기  4. 엔티티(Entity)  - 정해져 있는 표기법을 사용하면 브라우저가 약속된 표현으로 바꿔서 출력  5. 주석(Comment)  - 다중 라인 주석 안에 다중 라인 주석이 들어 갈 수 없음(단일라인은 가능함) | | |
| HTML의 속성의 유형 | 1. 속성명="속성값" : 값을 지정하는 것에 따라 결과가 달라짐  2. 속성명 : 조작 불가능  - 쓰면 적용 / 안 쓰면 적용 안 함  - 플래그 타입 속성 | | |
| HTML의 수치 | 1. 숫자  - 픽셀(pixel)단위 : 모니터에 출력되는 점 1개  - 절대값  2. 숫자%  - 백분율  - 상대값(\*\*항상 부모의 영역을 기준으로 함  (자식요소가 부모요소를 벗어나지 않음), 화면 크기에 따라서 달라짐) | | |
| 웹에서의 경로  (HTML,  CSS,  Javascript,  서버 측) | : A(현재 페이지, 항상 중심은 나) -> B(찾을 대상, 웹에서 접근 가능한 자원)  '\' : 파일 경로  '/' : 웹 경로  1. 상대경로  - .: 현재 페이지가 들어있는 폴더  - .. : 부모 폴더  - / 구분자  - 항상 현재 페이지가 어디인지를 꼭 숙지!  - 나를 기준으로 남을 찾는 표현(내 위치가 바뀌면 남의 위치도 바뀌게 된다.)  2. 절대경로  - 반드시 경로의 시작을 '/'로 한다.  - 어떤 미리 약속된 특정 위치를 기준으로 하는 경로 표현(기준이 최상위 폴더)  - 처음에 나오는 / : 최상위 루트 폴더  3. 로컬경로(파일경로)  - 웹에서 사용 안함  4. 외부경로  - http:// ~  - 남의 사이트 꺼를 가져올 때 사용(변동 사항이 생기면 대처 할 수 없음) | | |
| 태그 | 인라인 태그 | | 블록 태그 |
| - 자신의 내용을 앞, 뒤의 요소의 내용과 같은 라인에 출력  - 내용물의 크기가 태그의 영역 | | - 자신의 내용만을 한 라인에 출력  - 앞뒤의 내용이 있어도 절대로 같은 라인에 출력하지 않음  - 내용물의 크기와 상관없이 항상  너비 100%를 자기 영역이라고 판단 |
| 규칙 | 1. 블럭태그는 자식으로 블럭 태그 or 인라인 태그를 가질 수 있다.  2. 인라인 태그는 자식으로 인라인 태그만을 가질 수 있다.  3. 블럭 태그 중 <p>만은 자식으로 인라인 태그만을 가질 수 있다. | |

**[CSS]**

|  |  |
| --- | --- |
| HTML | |
| - CSS3 |  |
| HTML의 역할 | 1. HTML 페이지에 서식 작업   - 서식 작업(HTML -> CSS), 디자인 |

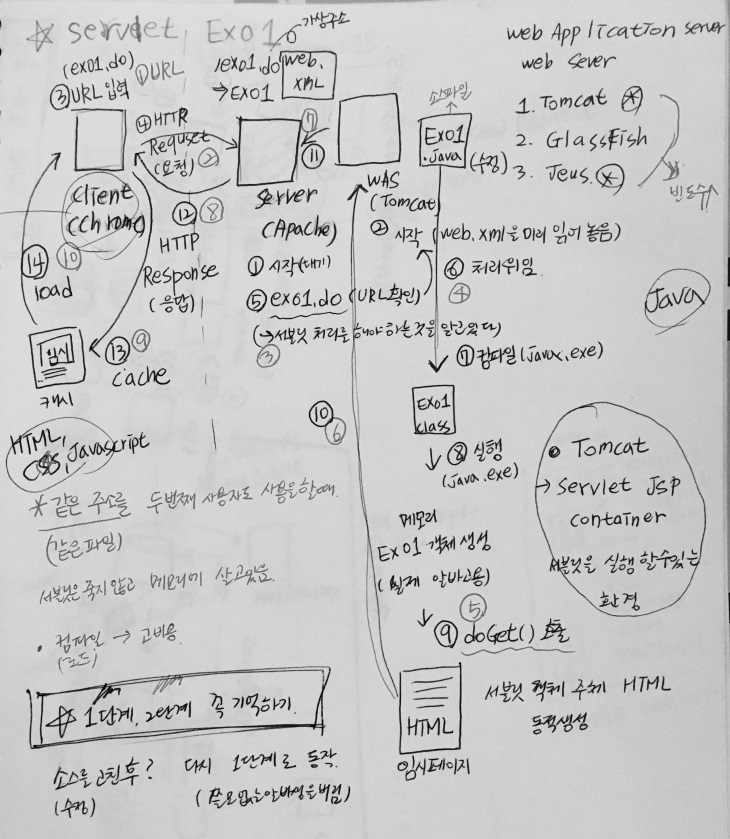
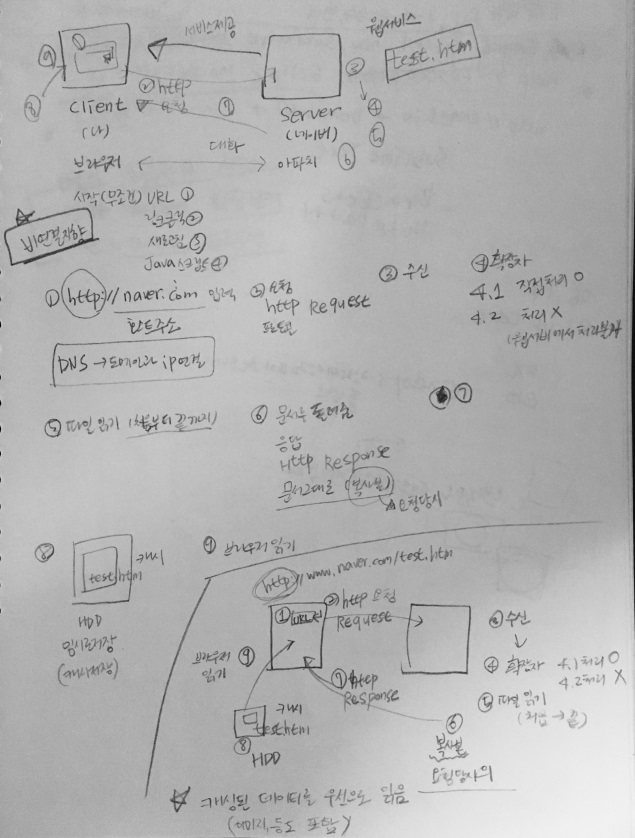
**[Javascript]**

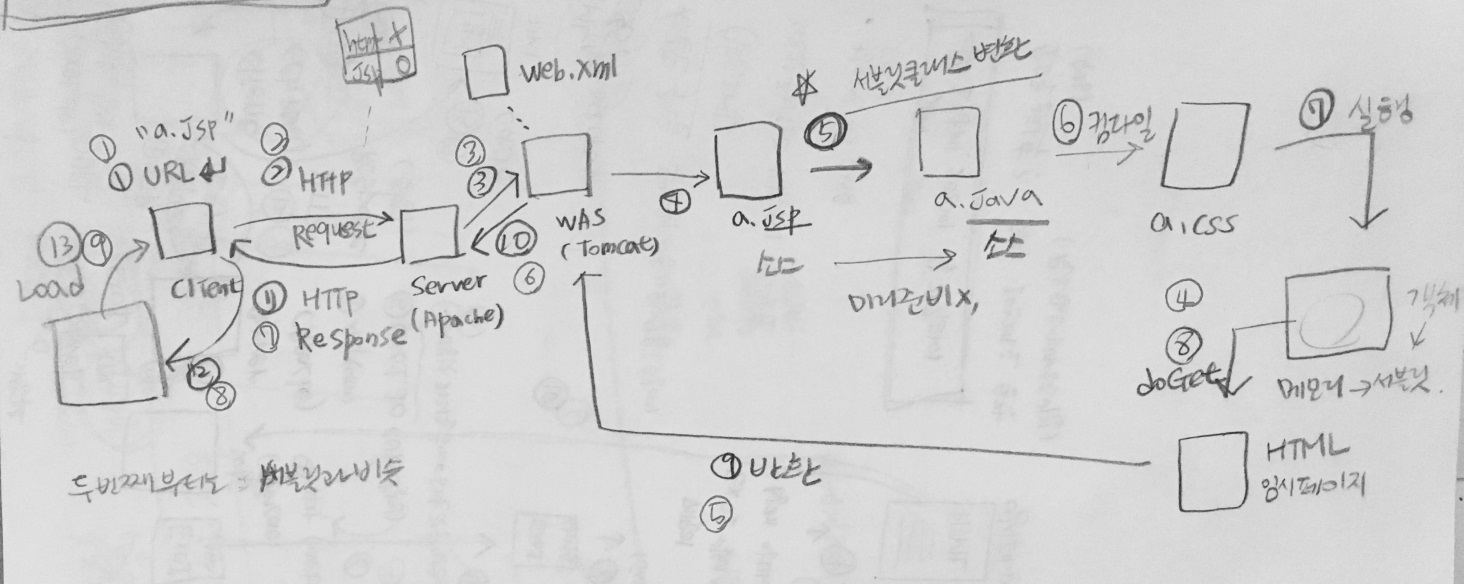
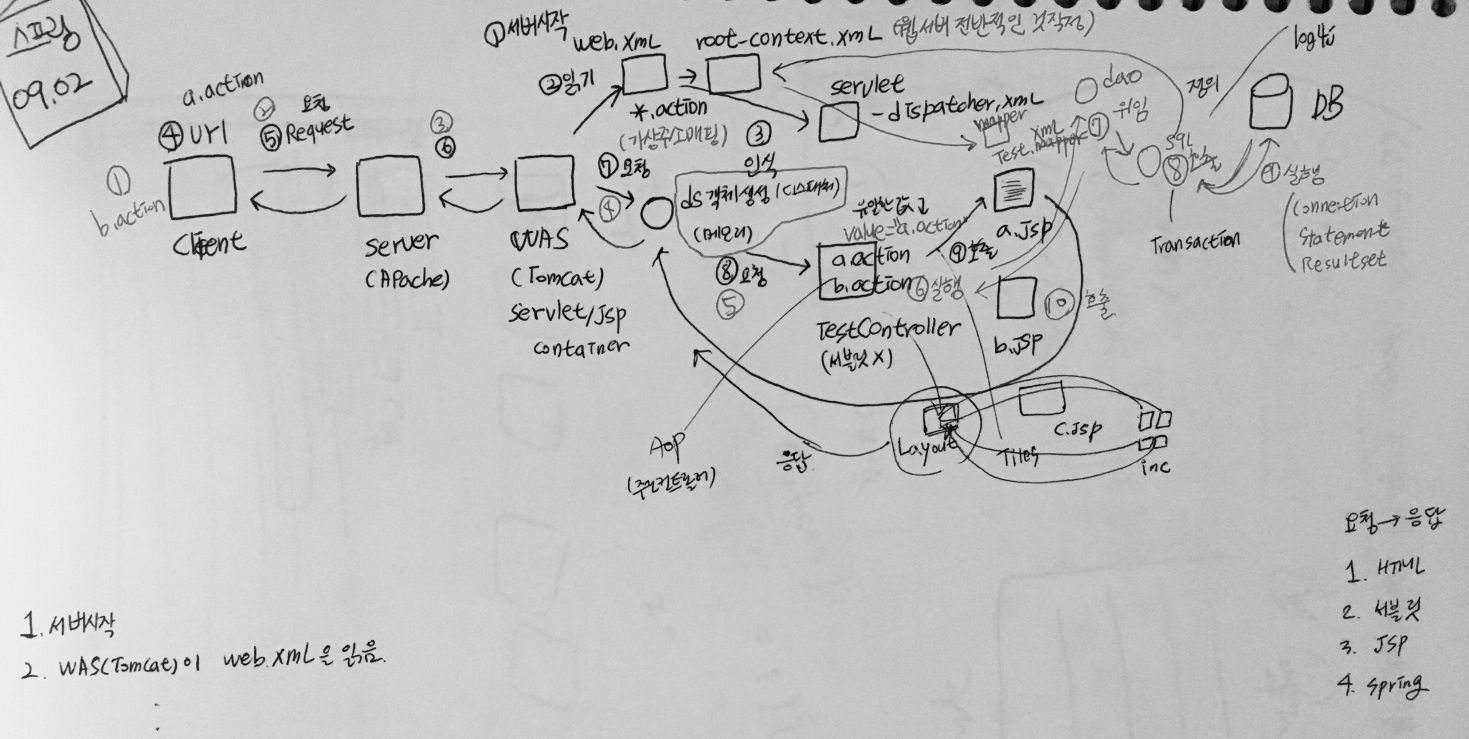
|  |  |  |
| --- | --- | --- |
| Javascript | | |
| 1. function(ex37~39) |  | |
| 2. DOM | - Javascript DOM -> HTML  - Java DOM -> XML  - 태그나 속성 피씨데이터 주석 이런 등등에 문서의 구성 요소들을 트리 구조로  모델화 시켜놓은 방식  -> 접근, 수정, 삭제, 추가 등 HTML 트리 구조로 만들어 놓은 것 | |
| BOM | - 브라우저 객체 모델(BOM : Browser Object Model)  - window, screen, location, history, navigator 객체 등 | |
| **BOM** | | **DOM** |
| - 초기 모델, 폐쇄적  - 이미 족보가 완성되어있음 (정해진 트리 구조)  - 직책을 가지고 태그를 구분 (하는 역할로 구분)  - 브라우저의 웹 페이지 로딩  -> 웹 페이지의 각종 태그 -> 특정태그  -> 정해진 계층구조로 탐색  - 특정태그: Window, Document, a, img, form, form  - 모든 태그에 접근 가능한 게 아니라 일부 계층 구조에 이미 속해있는 태그에 한해서 접근가능      ex) <p>, <table>같은 태그에는 접근 할 수 없다  - 태그의 name속성값 식별  - 자바스크립트가 제공하는 배열을 통해서 식별  - window, document, body, images, forms, links, elements.. | | - XML   - 족보가 소스에 따라 다름   - 문서의 트리 구조에 따라서 접근   - 부모, 자식관계를 찾아서 식별   - 특정 메서드로 접근   - getElementById("아이디") : 태그의 id속성값 식별   - getElementsByTagName("태그명") : 태그명 식별   - 문서내의 태그들의 계층구조를 통해서 접근 가능   - Document구조를 트리 구조로 구조화 시켜서 접근   - Document내의 모든 태그에 접근이 가능하다   - 기능이 BOM에 비해 확장되어 있다 |

**[Servlet] / [JSP]**

|  |  |  |
| --- | --- | --- |
| 1. 그림  - JSP 파일의 수명 |  | |
| 1. Servlet??   존재 의미? 역할? | 웹페이지를 동적으로 생성하는 서버 측 프로그램 혹은 그 사양  [JSP](https://ko.wikipedia.org/wiki/JSP)가 HTML 문서 안에 Java 코드를 포함하고 있는 반면,  서블릿은 자바 코드 안에 HTML을 포함하고 있다는 차이점 | |
| 3. Servlet -> JSP 전환 |  | |
| 4. MVC 패턴 | - Model(DTO) 계층간의 교류 데이터를 나르는 단위  - view 출력 담당  - Controller 제어 담당 | |
| MVC Model 1 | MVC Model 2(\*\*\*) |
|  |  |
| 사용자로부터 요청을 JSP가 받아  (JSP 에서 사용자가 요청)  Java Bean(DTO, DAO)을 호출해 처리  개발 속도가 빠르고 배우기 쉽지만  프레젠테이션로직과 비즈니스로직이 혼재  JSP 코드가 복잡해져 유지 보수가 어려움 | View 는 JSP로, Controller 는 Servlet 을 사용  보여지는 부분은 HTML이 중심이 되는 JSP,  다른 자바 클래스에게 데이터를 넘겨주는 부분은 Java  코드가 중심이 되는 Servlet 이 담당  Model 영역 에서는 DTO, DAO를 통해 Mysql 과 같은 Data Storage 에 접근 |
| 5. EL | <%= %> , out.println()과 같은 자바코드를  더 이상 사용하지 않고 좀더 간편하게 출력을 지원하기 위한 도구.  배열이나 컬렉션에서도 사용되고, JavaBean의 프로퍼티에서도 사용  (page → request → session → application) | |
| 6. JSTL | JSP는 자신만의 태그를 추가할 수 있는 기능을 제공  커스텀 태그처럼 연산이나 조건문이나 반복문인 if문, for문, DB를 편하게 처리할 수 있는 것 | |

**[Spring]**





|  |  |
| --- | --- |
| 디버깅 방법  디버깅 : 컴퓨터 프로그램 상의 오류를 찾아내어 바로잡는 과정 | |
| 브레이크 포인트 | 조사하고 싶은 시점에 진행을 멈추고 각 객체에 들어가 있는 값들을 확인  디버그모드 실행 |
| 메모리 디버깅 | 특정 변수에 대해 조사하고 싶을 때, 변수의 주소 값을 취해 해당 주소의 메모리가 변경될 때 중단점이 히트  변수가 바뀌는 부분을 알고 싶을 때 유용 |
| 원격 디버깅 | 원격지에서 실행되는 프로그램을 빌드한 소스를 이용해 원격지에서 디버깅 하는 방법  원격지에서 실행되는 프로그램에 대해, 기본 디버깅처럼 수행가능, 심볼파일 필요 |
| 메모리 덤프 | 윈도우가 뻗거나 치명적인 오류가 발생했을 때 메모리의 상태를 파일로 기록  이 파일을 WinDbg라는 프로그램을 이용해 분석하여 오류가 난 원인을 분석, 고침  WinDbg는 다른 식으로도 사용 가능 |
| 로그 | 프로그램 개발 시 많이 쓰이며 특히 게임의 서버에서 많이 쓰이는 방법  오류가 났을 때의 상황을 다시 재현하기 힘들기에 오류가 발상했을 때  유저의 PC에 오류 정보에 대한 기록을 남기거나 서버 쪽으로 전송해 개발자가 분석하고 디버깅 할 수 있도록 함 |