




면접 CS 공부

🕒 작성 일시	@2023년 5월 25일 오후 8:29
🕒 최종 편집 일시	@2023년 5월 27일 오전 1:18
📄 유형	Experience
👤 작성자	 종현 박
👥 참석자	
🗣️ 언어	

[JAVA](#)

[Spring](#)

[DB \(MySQL, PostgreSQL\)](#)

[Web](#)

[기타](#)

[면접 예상 내용](#)

[지원 회사](#)

▼ JAVA

- equals(), hashCode()
 - 공통
 - Object 메서드에서 포함된 메소드
 - 객체의 동등성 비교와 해시 기반 컬렉션 사용을 지원하기 위한 메서드
 - equals
 - 두 객체의 레퍼런스를 비교하여 동일한 객체인지 비교합니다.
(두 객체의 메모리 주소) - 동일성
 - 오버라이딩 규칙
반사성 (x.equals(x) 는 반드시 true)
대칭성 (x.equals(y) 가 true 면 y.equals(x)도 true)
추이성 (x.equals(y) 가 true, y.equals(z) 도 true 면 x.equals(z) 도 true)
 - 구현 방법

1. 자신이 참조인지 체크 한다. (==)
 2. instanceof 명령어로 자신의 타입이 맞는지 체크한다. (with null check)
 3. 핵심 필드 (동등성을 만족해주는 필드) 에 맞춰서 boolean 값을 반환한다.
- hashCode
 - native 메서드를 사용해서 메모리 주소를 참조한다.
(즉, a.equals(b) 는 false, a.hashCode() == b.hashCode() 도 false 이다)
 - 구현 방법
 1. int 변수 하나를 만든다.
 2. 핵심 필드 타입의 따라 다름
 - a. 기본 타입 이라면 기본타입.hashCode(field) (박싱된 기본타입)
 - b. 참조 타입 이라면 해당 타입이 equals 과 hashCode 를 재구성했다면 재귀적으로 hashCode 를 만들어 호출한다.
 - c. 배열 이라면 모든 원소가 핵심이라면 Arrays.hashCode 를 사용한다.
 3. 계산된 값을 지속적으로 갱신해준다.
 result = Integer.hashCode(a);
 result = 31 * result + Integer.hashCode(b);
 - 31 (쉬프트 연산, 소수(소수가 아닌 수를 곱한다면, 보다 같은 hash 값을 갖을 경우의 수가 많아짐으로 소수를 택한 것으로 생각됨.)
 - 만약 new Point(1, 2), new Point(1, 2) 가 있다고 치면
 - 둘은 사실상 같은 객체(동일성)가 아니지만 동등하다고 볼 수 있다. (동등성)
동등성을 이루기 위해서는 두 메서드를 반드시 오버라이드해야 한다.
 - 만약 equals 를 override 하고 hashCode 를 override 하지 않았다면, Collection Framework 를 사용할 때, 정상적인 로직을 기대할 수 없다.
 -
- String, StringBuilder, StringBuffer
 - call by reference vs call by value
 - serializable, cloneable interface 특징

- HashMap vs HashTable vs ConcurrentHashMap
- Java 8, Java 11 쓰는 이유
- Stream Paraller에 대해서 설명
- ForkedJoinPool에 대해서 설명
- GC에 대해서 설명
- **인터페이스와 추상클래스에 대해서 설명**
 - 공통
 - 둘다 IS-A 관계이다.
 - Default 메서드를 갖을 수 있다.
 - 인스턴스화 할 수 없다
 - 인터페이스 혹은 추상 클래스를 상속받아 구현한 구현체의 인스턴스를 사용해야 한다.
 - 인터페이스와 추상클래스를 구현, 상속한 클래스는 추상 메소드를 반드시 구현하여야 한다.
 - 인터페이스
 - static 변수를 갖을 수 있지만... 열거 타입으로 쓰지는 말자. (Enum 을 쓰자)
 - 추상 메소드 하나만 갖고 있는 인터페이스를 함수형 인터페이스라고 한다. (@FunctionalInterface) - 랴다로 사용가능
 - 다중 상속
 - 추상 클래스
 - 하위 클래스들의 공통점들을 모아 추상화하여 만든 클래스
 - 단일 상속
 - 클래스간의 연관 관계를 구축을 초점을 둔다.
- **volatile에 대해서 설명**
 - CPU 캐쉬에 등록되지 않고 메인 메모리에 등록되는 변수를 뜻한다.
 - 수정 가능 쓰레드 1개, 읽기 가능 쓰레드 여러개 로 사용된다.
 - 왜?
 - volatile int a = 0; 이라고 해보자

a++; 를 하면 어떻게 될까? a 를 Main Thread 에서 불러서 a 에 + 1 을 처리 해줄 것이다.

이때 만약 a를 부른 상태에서 +1 을 하기 직전에 다른 스레드에서 a++ 를 해서 a 를 가져왔다면 두 개 모두 1를 얻을 것이다.(안전 실패)

- 여러 스레드의 변경을 원한다면 synchronized 를 사용하거나 concurrent 라이브 러리를 사용하자

- 자바 접근 제어자
- 자바에서 불변 객체를 만드는 방식
- JVM 메모리 구조
- 직렬화 / 역직렬화
- SOLID 원칙
- 함수형 프로그래밍
 - stream
 - optional
- 컴파일 과정
- compiler vs interpreter
- **캡슐화 vs 정보은닉**
 - 정보 은닉의 목적은 기능의 교체나 변경에 대한 유연성을 제공하는 것이다. 코드가 타입이나 메소드, 구현등에 의존하는 것을 막아줌으로써 객체 간의 구체적인 결합도를 약화해 기능을 변경하거나 교체하기가 쉬워진다.
 - 정보 은닉이 캡슐화 보다 큰 범위이고 3 종류로 나눌 수 있다.
 - 객체의 구체적인 타입 은닉 (업캐스팅)
 - 객체의 필드 및 메소드 은닉 (캡슐화)
 - 접근 제어자 private 으로 필드나 메소드를 은닉 할 수 있다.
 - 구현 은닉 (인터페이스 & 추상 클래스)
- **CheckedException vs UncheckedException**
 - 컴파일 에러와 런타임 에러를 비교 하는 것이다.
 - 컴파일 에러는 에러를 수정 할 수 있어서 정상 로직에 다시 돌아갈 수 있도록 해야한다.

- 런타임 에러는 이대로 로직을 타면 실행에 오류가 발생된다.
(NullPointerException)
 - 런타임 에러는 try, catch 로 다시 정상 로직에 타게 하지 않는다.
 - java 에서는 service down 되고, Spring 에서는 에러 던지도록 한다. (런타임 예외가 발생하면 기본적으로 HTTP 응답 상태 코드를 반환하도록 설계되어 있음 - Spring의 예외 처리 메커니즘과 서블릿 컨테이너 으로 쓰레드 종료를 방지하고 예외를 처리하여 HTTP 응답을 반환)

▼ Spring

- Filter vs Interceptor
- AOP
- Bean Injection 방법
- POJO
- IOC/DI
- @Transactional
 - 성능을 개선하기 위해 메서드 내에서 스레드를 생성하여 비동기로 쿼리를 날리면 어떻게 될까요?(힌트: 병렬 처리보다 트랜잭션에 대해 묻는거다)
 - 스프링 내부에서 트랜잭션이 어디에 저장 될까요?
 - transactional 우선 순위
 - propagation, isolation level
- JPA vs MyBatis
- 스프링 통신 과정 (MVC)
- JDK Dynamic Proxy vs CGLib
- DL vs DI
- PreparedStatement vs Statement

▼ DB (MySQL, PostgreSQL)

- 인덱스
- 파티션
- explain analysis Query
- limit + offset

▼ Web

- 쿠키 vs 세션
- CORS

▼ 기타

- Non-Blocking vs Bloking
- Sync vs Async
- 싱글톤 패턴 / 팩토리 메소드 패턴 / 템플릿 메소드 패턴

▼ 면접 예상 내용

- ETL
 - 왜 만들었?
 - 기존 DB 가 MySQL 로 되어 있었고 몇몇 일부 테이블은 일정 주기마다 업데이트만 되는 방식이었음. 그런데 사용자 요구 사항에 시간대 변경하여 조회가 가능해야 되었고, 몇몇 테이블은 Join 을 너무 많이 해야 하는 상황이 나와 ETL 에서 미리 Join 된 마트 테이블을 생성하거나, 경량화된 테이블로 변경할 수 있었습니다. 또한, 이노 개발팀에서 앞으로의 RDB 방향성은 PostgreSQL 이였기 때문에, ETL 시스템을 만들었습니다.
- Node.js 로 만들었던데 왜? java 가 아니고?
 - Java는 컴파일 언어이고, Node.js는 인터프리터 언어임으로, 컴파일 시간을 줄이고 싶었습니다. 또한 Java 의 장점인 멀티 쓰레드 방식이 별로 필요 없다고 느껴졌습니다. 왜냐하면 테이블 update 주기가 5분, 1시간, 1일 이렇게 진행되고 1분, 30분 이런식으로도 추가될 가능성이 있다고 판단되어, 확장성에 용이하게 Node.js 를 사용하고 주기 단위로 node 를 여러 프로세스 로 관리하면 된다고 판단했기 때문입니다.
- 왜 NHN PAYCO 을 오려고 하는가?
 - 저는 애플리케이션을 잘 구축된 환경을 경험해 성장하고 싶기 때문입니다. 성공된 애플리케이션 특징으로는 신뢰성, 확장성, 유지 보수성으로 볼 수 있습니다. 이에 신뢰성에 중요하다고 생각된 PAYCO 결제 시스템을 경험해보고 싶었으며, 결제 시스템뿐만 아니라 다양한 도메인을 경험해 백엔드 개발자로 더욱 성장하고 싶었기 때문입니다.
- 이전 회사에서 프론트 작업을 많이 한 것 같은데..? 백엔드가 되고 싶은 이유는?
 - 다양한 트래픽을 경험하면서 시스템을 안정적이고 효율적으로 만들어 보고 싶다는 생각이 들었기 때문입니다. 예를 들어서 **데이터 중심 애플리케이션 설계**

책에서 확인 했던 내용으로 트위터 사용자에게 대한 응답 시간의 대한 목표를 갖고 해당 목표를 이루기 위해 팔로워 수에 따라 데이터를 불러오는 방식을 다르게 설계 한다는 것에 큰 영감을 받았습니다.

- JPA 를 사용한 적이 없는지?
 - 개인 사이드 프로젝트로 H2, JPA 를 사용한 적이 있긴 합니다만.. 크게 사용하는 법을 알지는 못합니다. 하지만, 이전 회사에서는 통신 트래픽을 발생함으로 Join 쿼리가 많아 JPA 보다는 MyBatis 가 더 효율적이 였었고, 직접 DB를 관리 할 수 있어서 저는 RDB 에 많은 역량을 키울 수 있었습니다. (limit/offset, index, partition, Explain analysis Query) JPA 도 DB 지식이 많아야 이해가 빨라져 습득력이 높을 것으로 예상되어 업무를 진행하는데 큰 차질은 없을 것입니다.

▼ 지원 회사

- 가고 싶은 곳 (결제 + B2C + 백엔드)
 - ~~NHN PAYCO Java 개발자~~ 서탈
 - NHN PAYCO 플랫폼 개발자
 - 카카오 페이 플랫폼 개발자
 - 카카오 페이 증권 서버 개발자
 - NHN Edu 에듀 서비스 백엔드 개발자
 - 야놀자 백엔드 플랫폼 개발자
- 2차 (B2C + 백엔드)
 - 패스오더 백엔드 개발자
 - 스테이지 파이프 백엔드 주니어
 - 롯데 정보 통신 Java 개발 / 운영
 - 페이 히어 서버 엔지니어