기술면접 오답노트

면접 때 틀렸던 내용 모음집

김온지

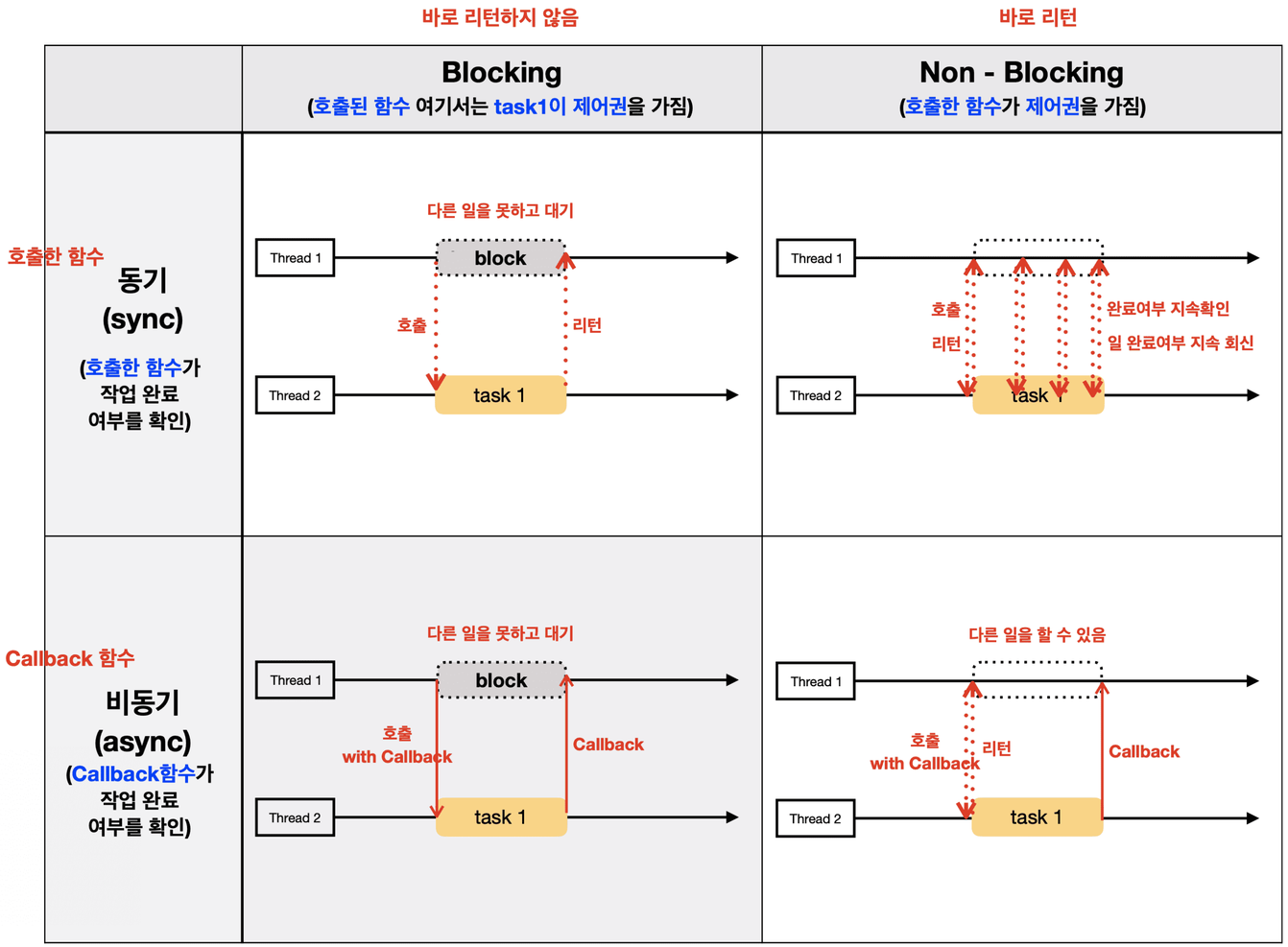
오답 목록

- 스푼라디오 데엔 - 0개
- SI 회사 백엔드 - 1개
- 중견 데엔 - 4개
- 서비스 백엔드 - 20% 답 못함, 20% 틀림

컴공 CS

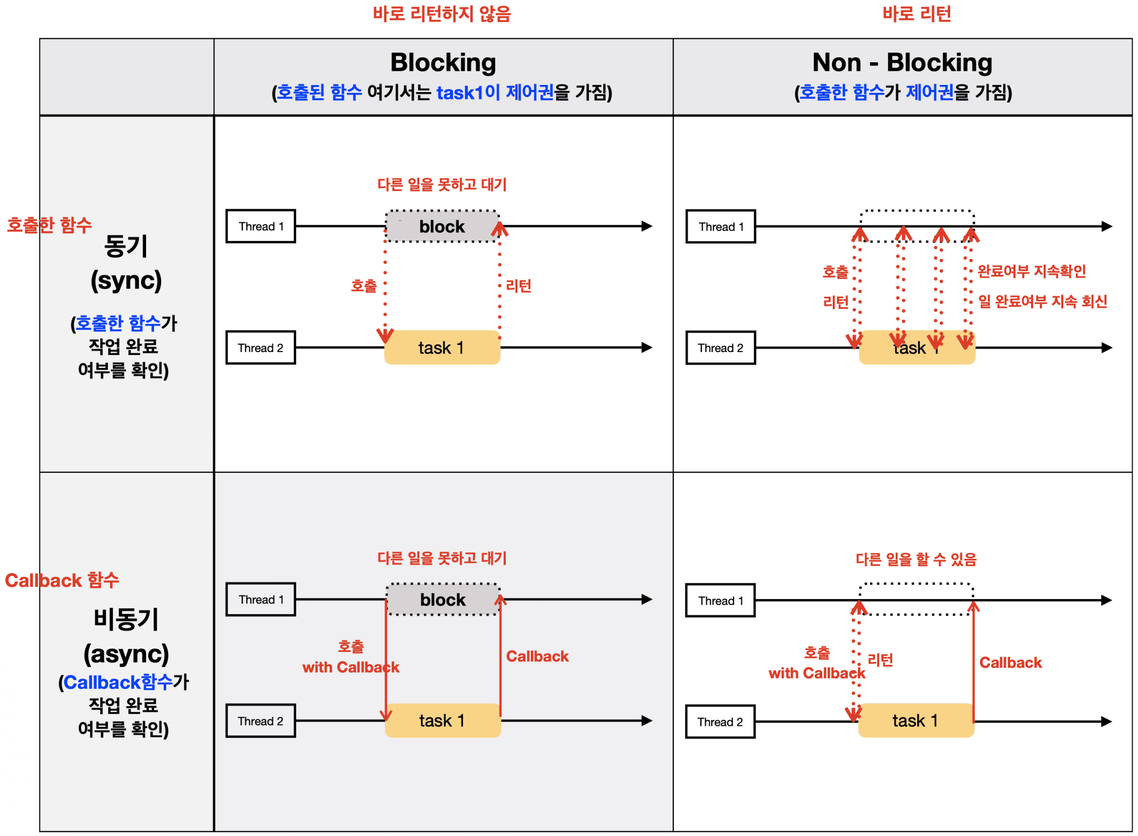
Sync, Async, Blocking, Non-Blocking

SI 백엔드, 서비스 백엔드



Java에서 Async Blocking은 어떻게 구현되어 있나요?

서비스 백엔드

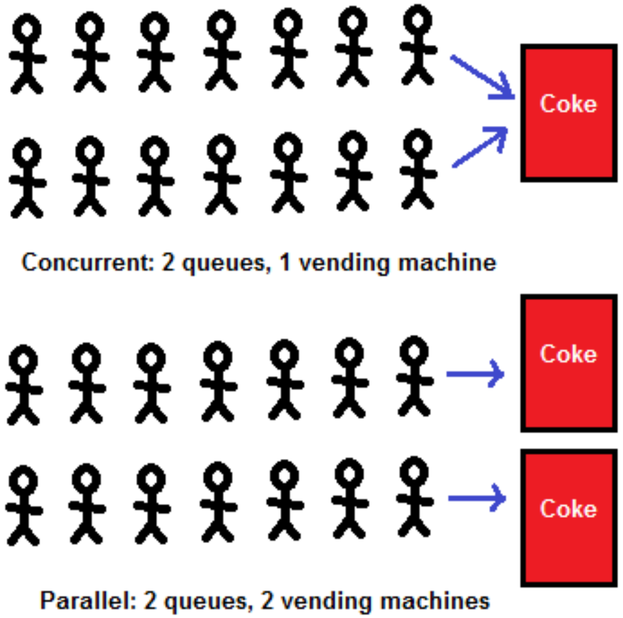



```java
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class FutureExample {
    public static void main(String[] args) {
        // ExecutorService를 사용하여 스레드 풀을 생성합니다.
        ExecutorService executor = Executors.newSingleThreadExecutor();

        // Callable 객체를 정의합니다. 이 객체는 비동기 작업을 수행합니다.
        Callable<Integer> task = () -> {
            // 비동기 작업을 시뮬레이션하기 위해 잠시 대기합니다.
            Thread.sleep(2000);
            return 123;
        };

        // 작업을 제출하고 Future 객체를 반환받습니다.
        Future<Integer> future = executor.submit(task);

        try {
            // 비동기 작업이 완료될 때까지 대기하고, 완료되면 결과를 반환받습니다.
            Integer result = future.get(); // 결과를 기다리는 동안 블록됩니다.
            System.out.println("Result: " + result);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            // ExecutorService를 종료합니다.
            executor.shutdown();
        }
    }
}
```

동시성(Concurrency) 병렬성(Parallelism) 차이

중견 데이터엔지니어

- 모두 동시에 어떤 일을 처리하지만,
- Concurrency는 스위칭으로 한번에 여러 작업을 동시에 다룸
- Parallelism은 여러 작업을 각각의 실행단위로 물리적으로 동시에 실행

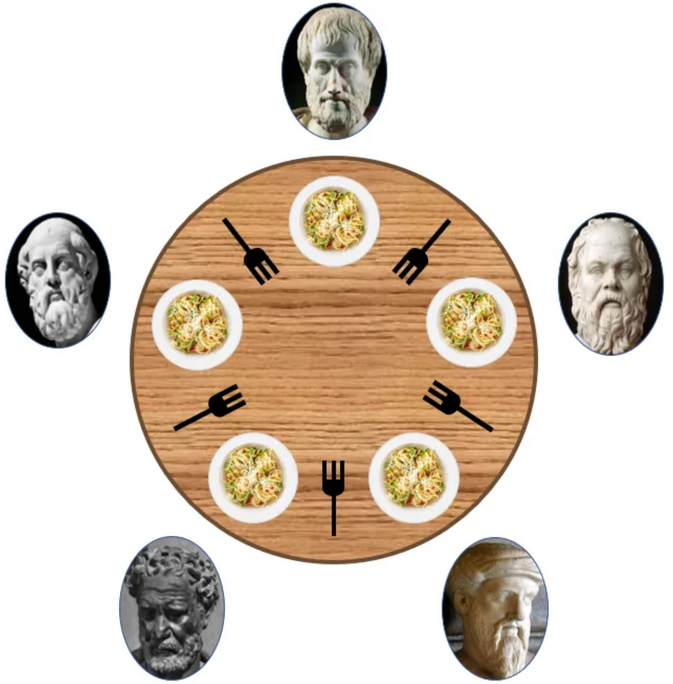


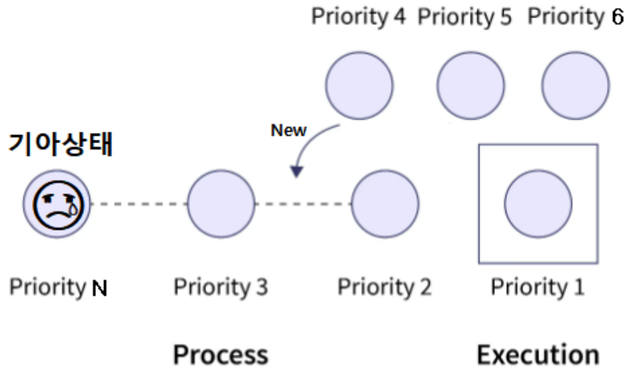
- 파이썬은 GIL으로 인해 멀티 쓰레드를 사용해서 동시성으로 수행
- 싱크로, 공유변수, 선후관계

기아상태와 데드락 차이

중견 데이터엔지니어

- 둘다 자원의 할당을 계속해서 대기
- 데드락
  - blocked 상태에서 발생
  - 발생하지 않는 이벤트를 대기
- 기아상태
  - ready 상태에서 발생
  - CPU 할당을 대기





쓰레드의 컨텍스트 스위칭이 가벼운 이유

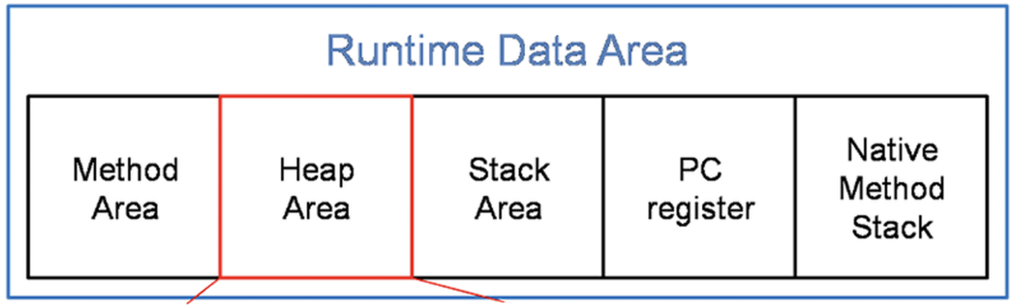
중견 데엔

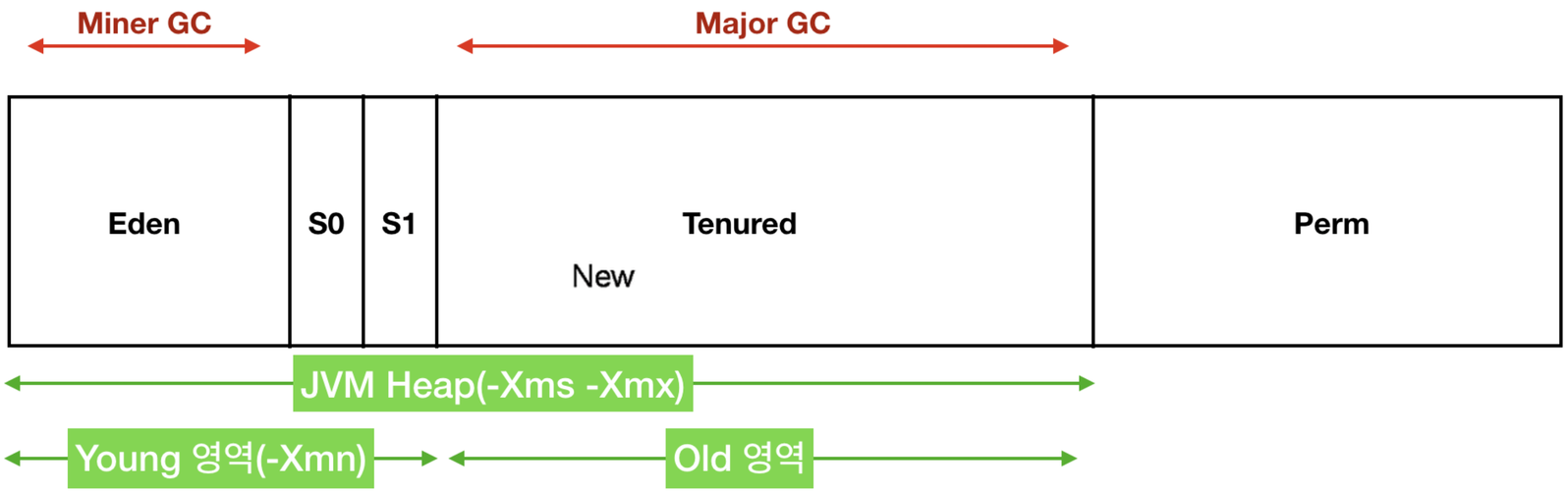
- 프로세스 컨텍스트 스위칭은 1~4번 수행
- 스레드 컨텍스트 스위칭은 1번만 수행
- 메모리 주소 공간이 바뀌지 않기 때문

1. 실행 컨텍스트 백업
2. CPU 캐시 비움
3. TLB(Table Lookaside Buffer)를 비움
4. MMU(memory management unit) 변경

Java

Java Heap

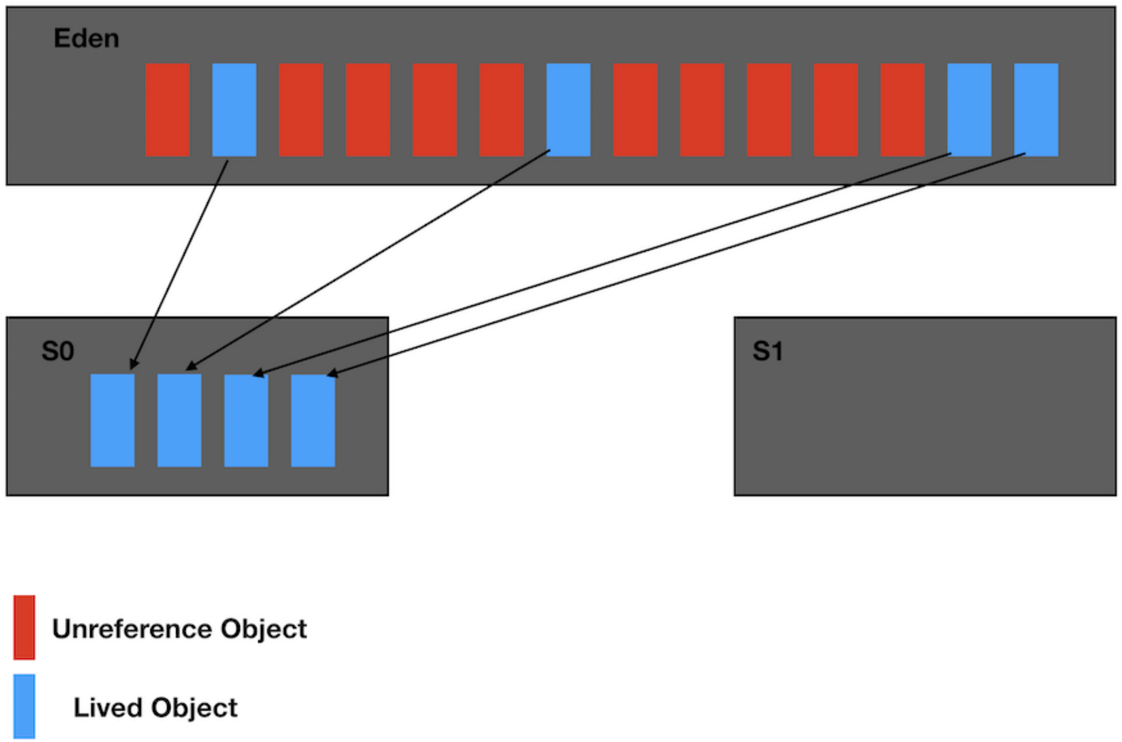

Runtime Data Area
Method Area
Heap Area
Stack Area
PC register
Native Method Stack

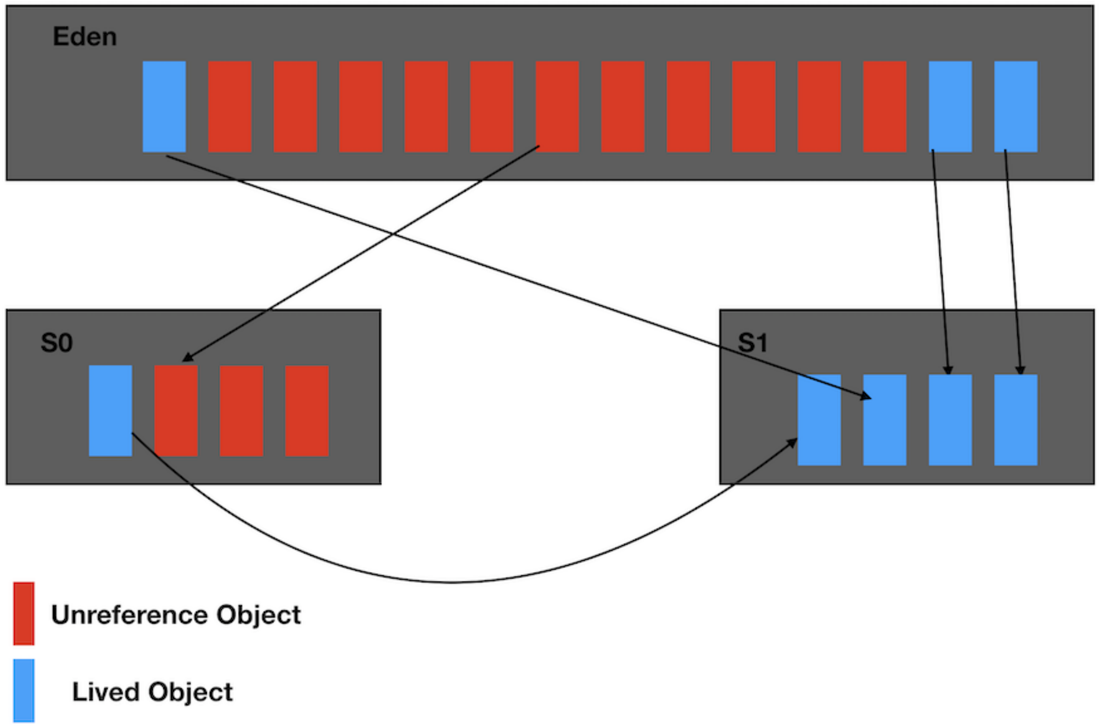

Miner GC
Major GC
Eden
S0
S1
Tenured
New
Perm
JVM Heap(-Xms -Xmx)
Young 영역(-Xmn)
Old 영역

Minor GC

비교적 적은 수 객체 검사

Eden 꽉차면, Minor GC





survivor같이 검사, 이동

Major GC

설정 수 만큼 살아 남은 객체가 Old 영역으로 이동

- Old 영역 꽉차면 실행
- Stop the World
- 모든 객체를 검사하므로 시간이 오래걸림
- GC 모니터링 필요

GC의 종류

- Serial GC -> CPU 성능 낮을 때 사용, 하나의 쓰레드만 사용
- Parallel GC -> CPU 코어 수만큼 쓰레드 만들어서 Minor GC에 사용, Major는 1개
- Parallel Old GC -> 모두 코어수만큼 쓰레드 만들어서 사용
- Concurrent Mark Sweep Collector
  - Major GC와 App. 쓰레드를 동시에 수행
- G1 GC
  - Young, Old를 나누지 않고 동일한 사이즈의 힙공간으로 나눔
  - 가장 liveness가 적은 곳을 GC

JIT 컴파일러란 무엇이며, Java에서는 언제 동작하나요?

- 메서드 호출 횟수 + 메서드 루프 횟수 -> 자격 판단
- 큐에서 대기 후, 컴파일 됨
- 이후에는 컴파일 된 기계어가 실행됨
- 스택 상의 교체(on-stack replacement, ORS)라고 부른다.

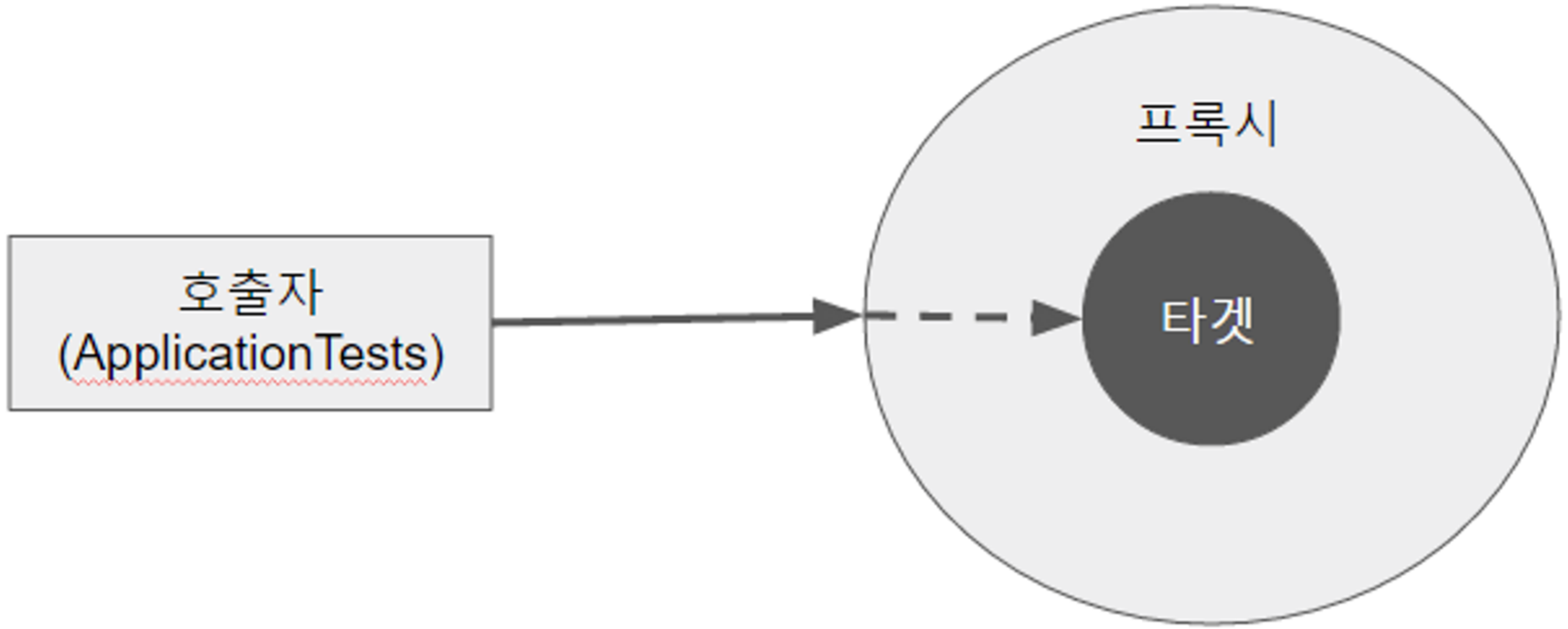
hashcode() 또는 equals()가 잘못 구현되었을 때 어떻게 동작?

```
public boolean add(E e) {
    int hash = e.hashCode();  // hashCode() 호출
    int bucketIndex = getBucketIndex(hash);  // 버킷 인덱스 계산

    if (bucketContains(bucketIndex, e)) {  // 해당 버킷에서 equals() 호출하여 중복 확인
        return false;  // 이미 존재하면 추가하지 않음
    }

    addToBucket(bucketIndex, e);  // 버킷에 객체 추가
    return true;
}
```

스프링 프레임워크

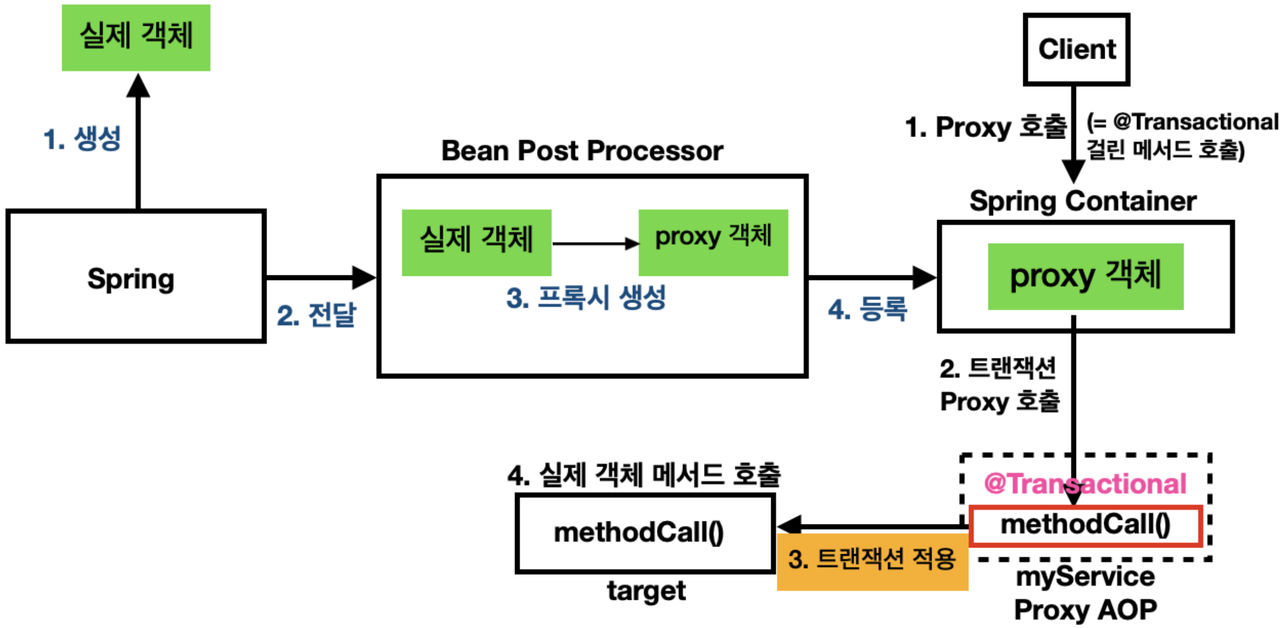
AOP 동작 원리

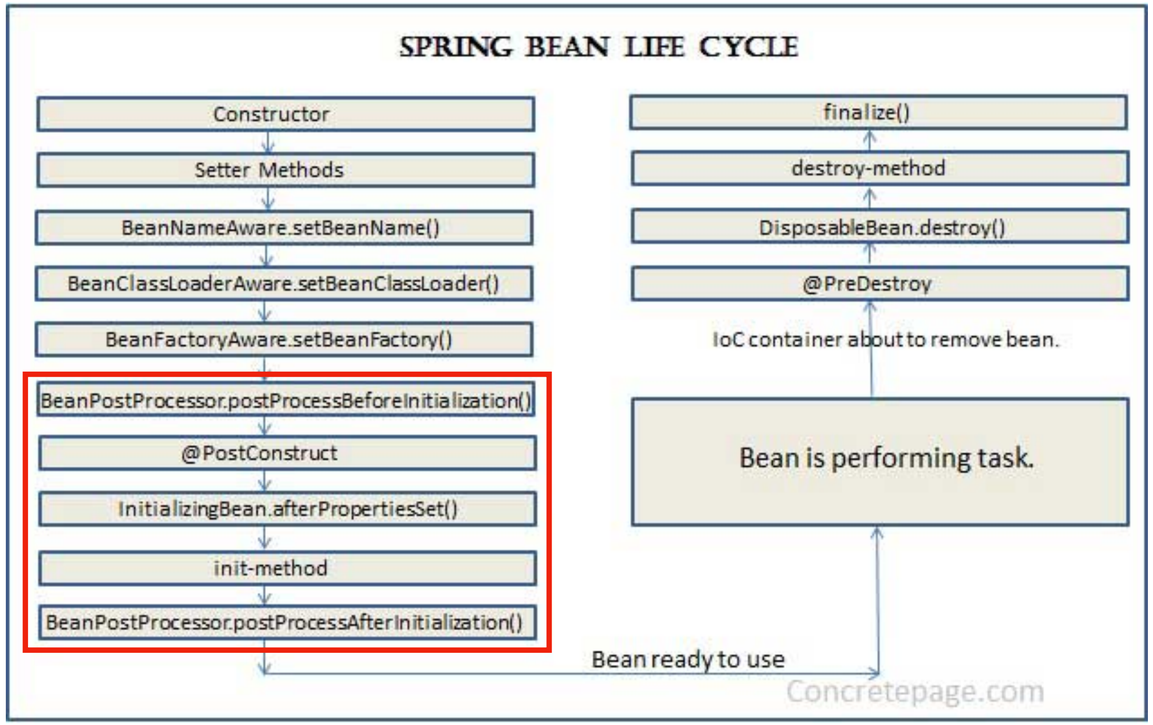
Dynamic Proxy, CGLIB

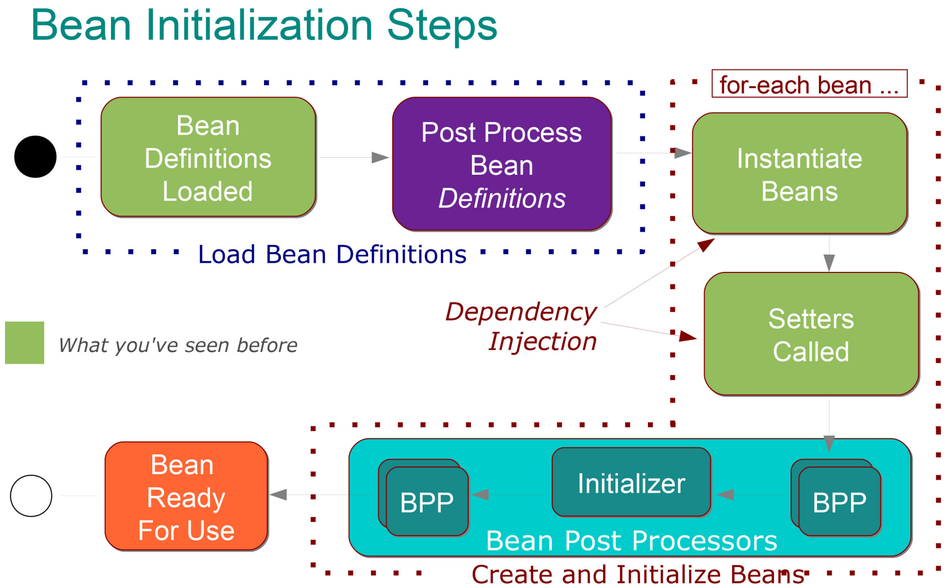


AOP 동작 원리

Dynamic Proxy, CGLIB

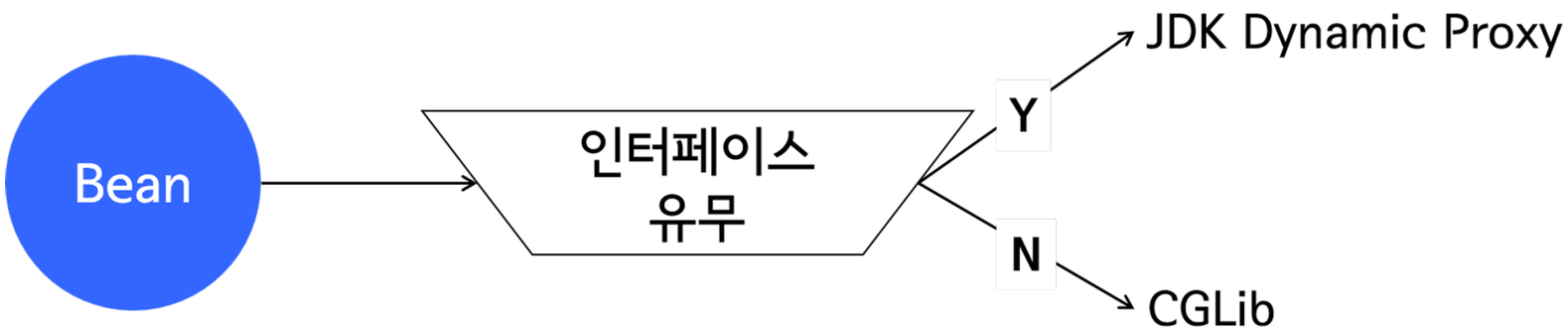






AOP 동작 원리

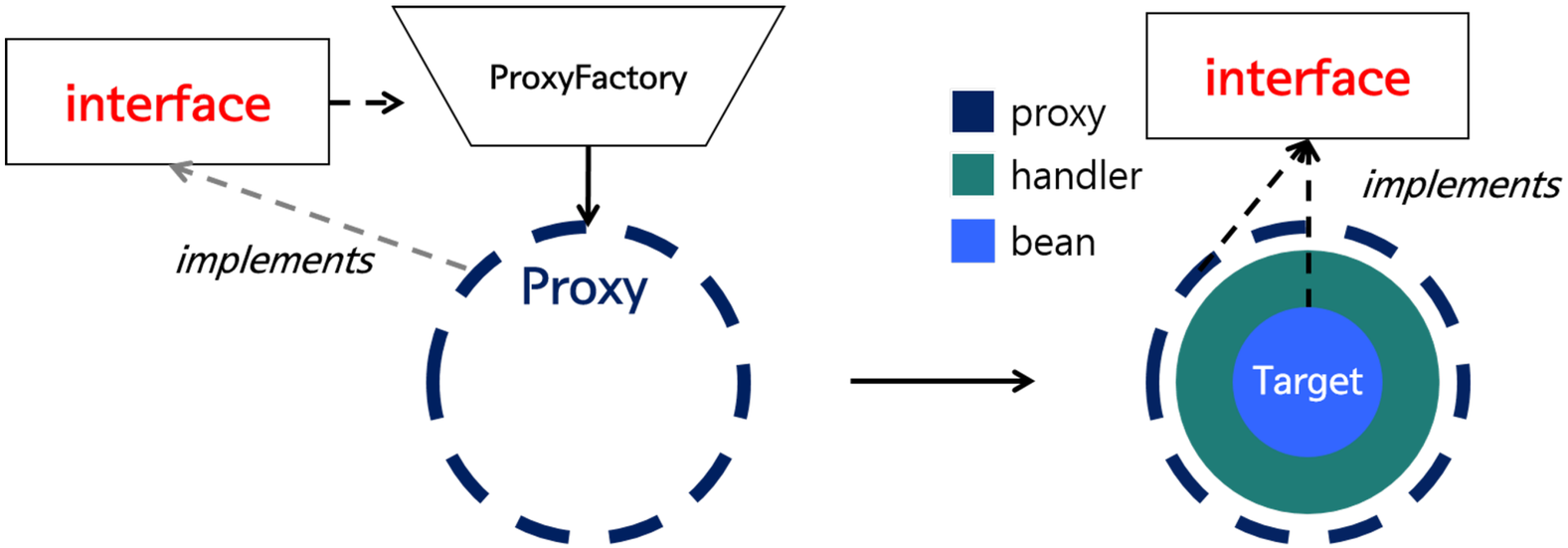
Dynamic Proxy, CGLIB



AOP 동작 원리

Dynamic Proxy

```
Object proxy = Proxy.newProxyInstance(ClassLoader        // 클래스로더
                                    , Class<?>[]         // 타겟의 인터페이스
                                    , InvocationHandler // 타겟의 정보가 포함된 Handle
                                    );
```

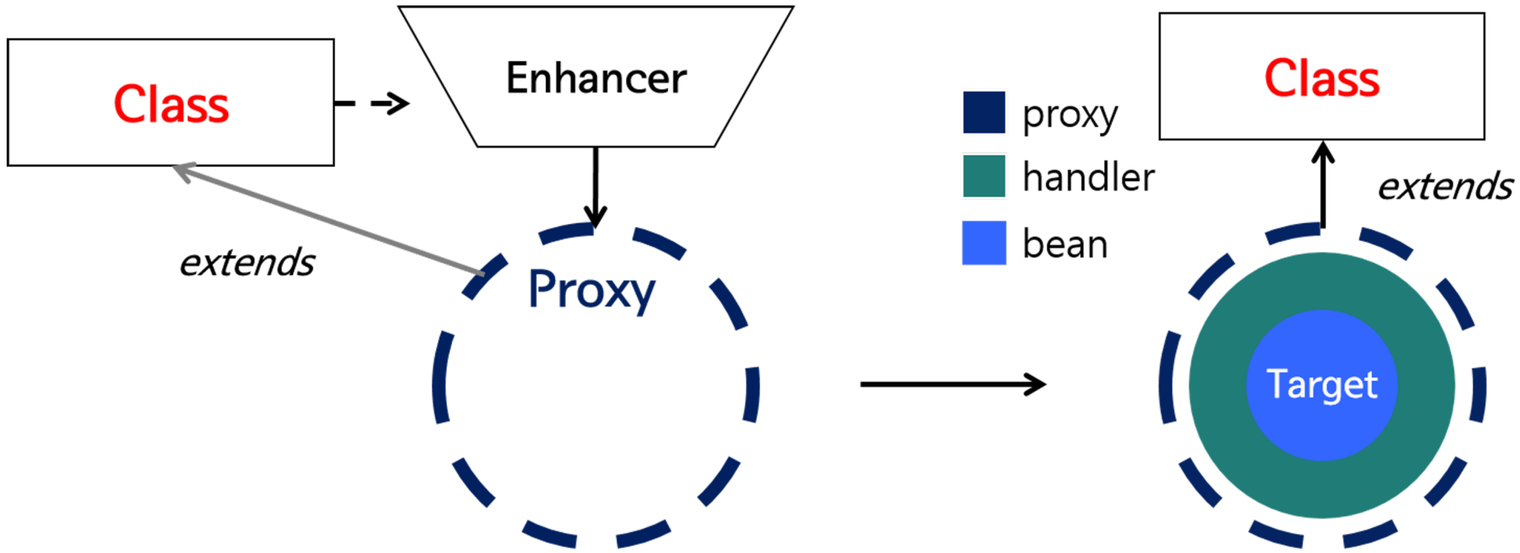


- Java Reflection 사용
- 성능 구림
- 인터페이스 있을 때
- 클래스로 DI시 에러

AOP 동작 원리

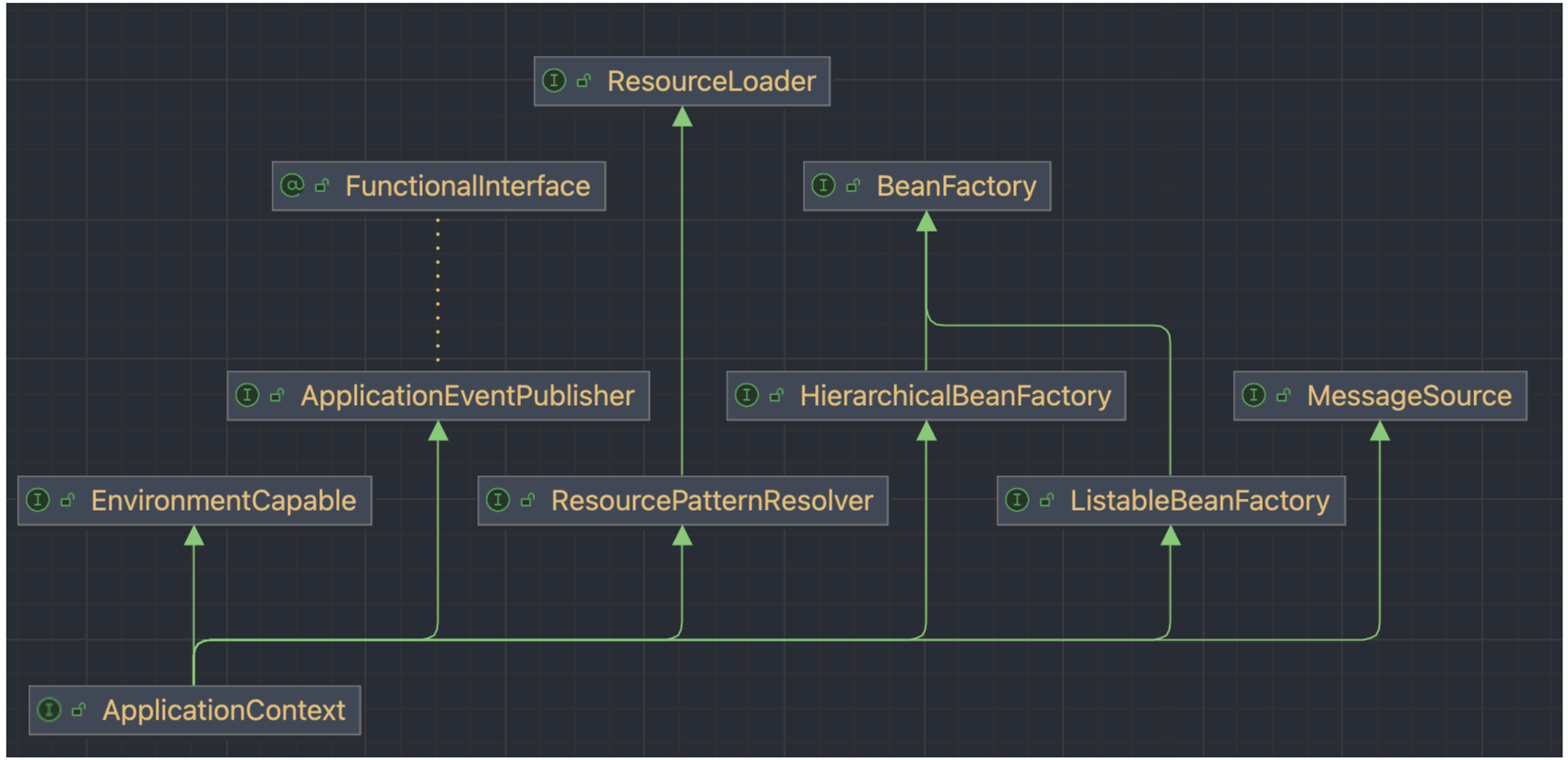
CGLIB

```
Enhancer enhancer = new Enhancer();
enhancer.setSuperclass(MemberService.class); // 타겟 클래스
enhancer.setCallback(MethodInterceptor); // Handler
Object proxy = enhancer.create(); // Proxy 생성
```



- 바이트 코드 조작
- 클래스에도 동작
- default 생성자가 필요

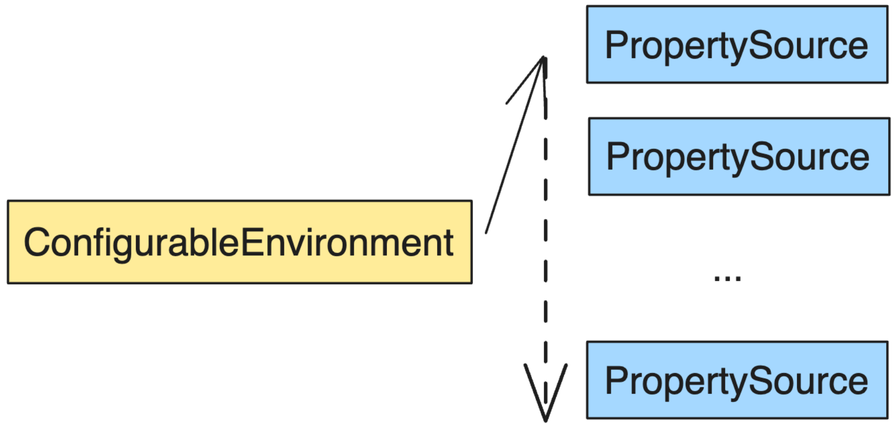
ApplicationContext는 무엇을 상속했고 역할은 무엇인가요?



EnvironmentCapable

원하는 PropertySource 추가 가능

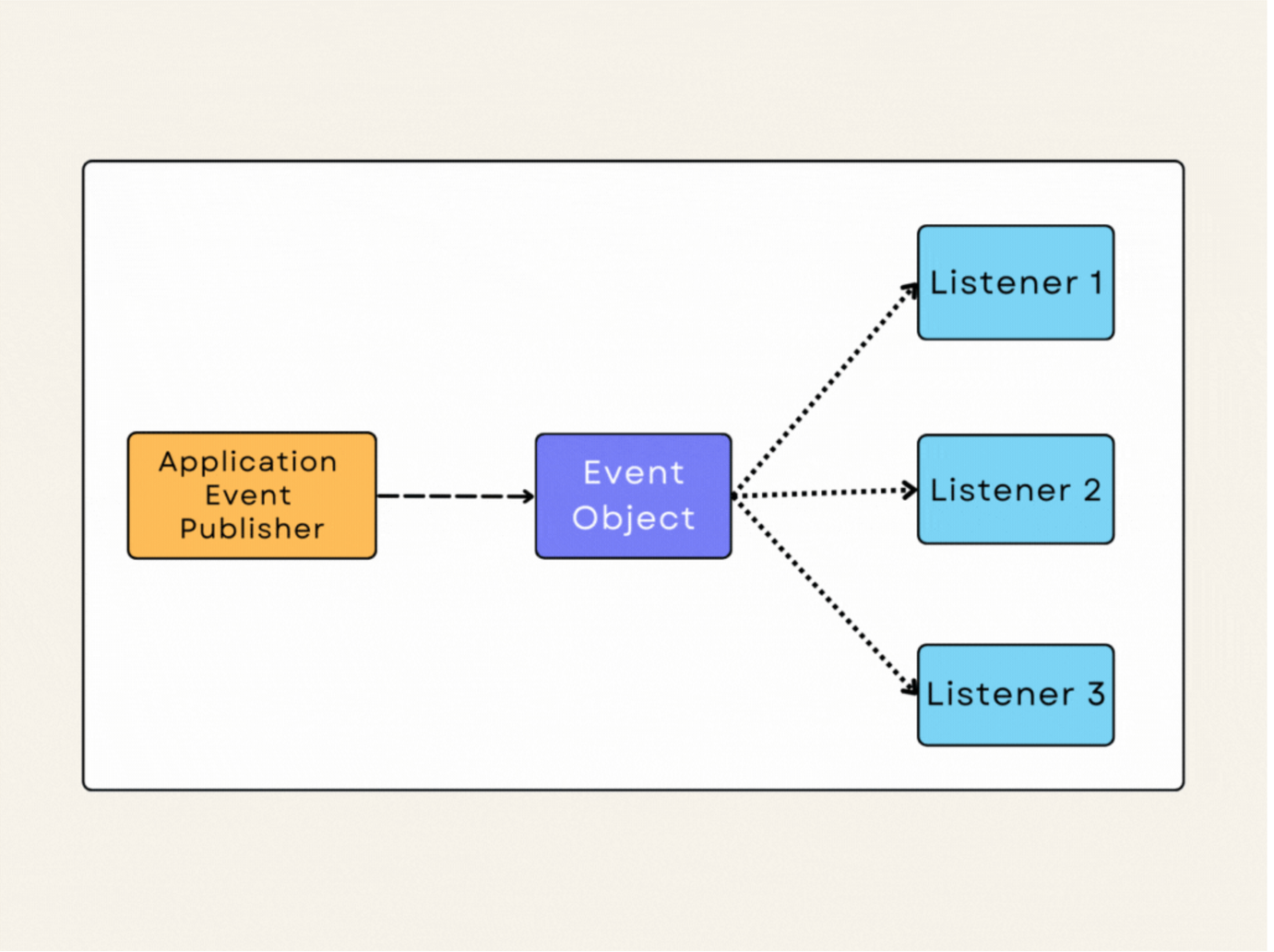
```java
public interface EnvironmentCapable {

    Return the Environment associated with this component.

    Environment getEnvironment();

}
```



환경변수, application.properties, 시스템

ApplicationEventPublisher

@EventListener



그 외

- BeanFactory -> Ioc 컨테이너, DI, 빈 생성 담당
- MessageSource -> 국제화 담당
- ResourcePatternResolver -> java.net.URI 확장해서 리소스 로딩 담당

프로젝트 관련

- 비정규화된 테이블 쓰면 mongoDB랑 똑같지 않냐
- Postgresql mvcc 메커니즘 말해봐라
- 인덱스 어떻게 구현되어 있냐
- 카프카 메시지 전송 실패하면 가격이 안매겨질 텐데 이건 어떻게 예외 처리했냐