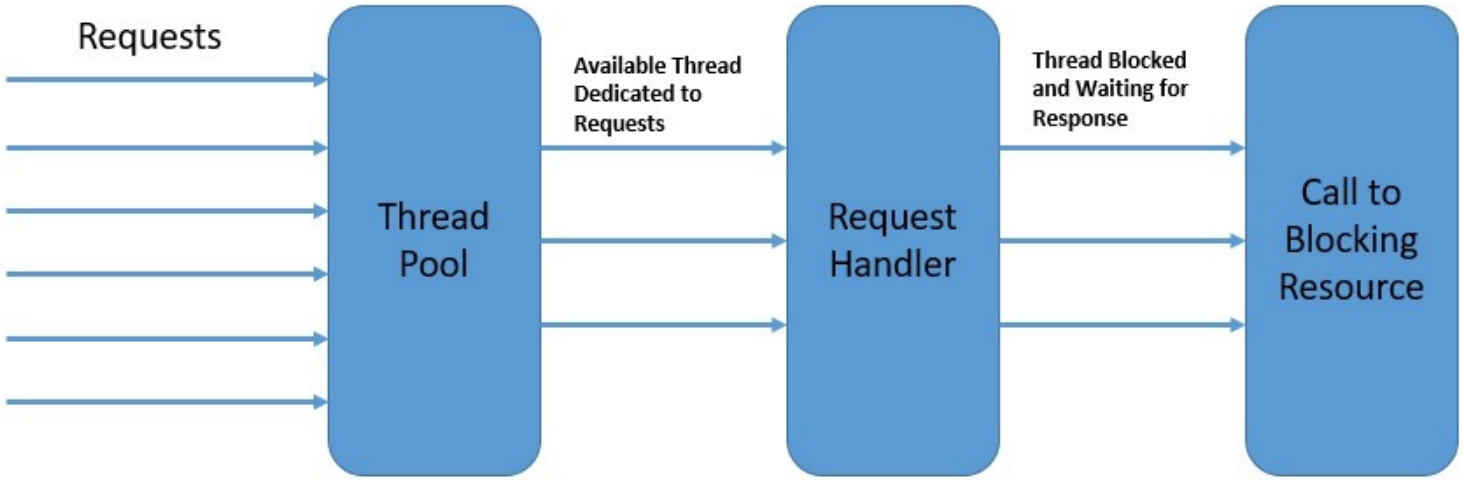
Why Spring

- 조민규 -

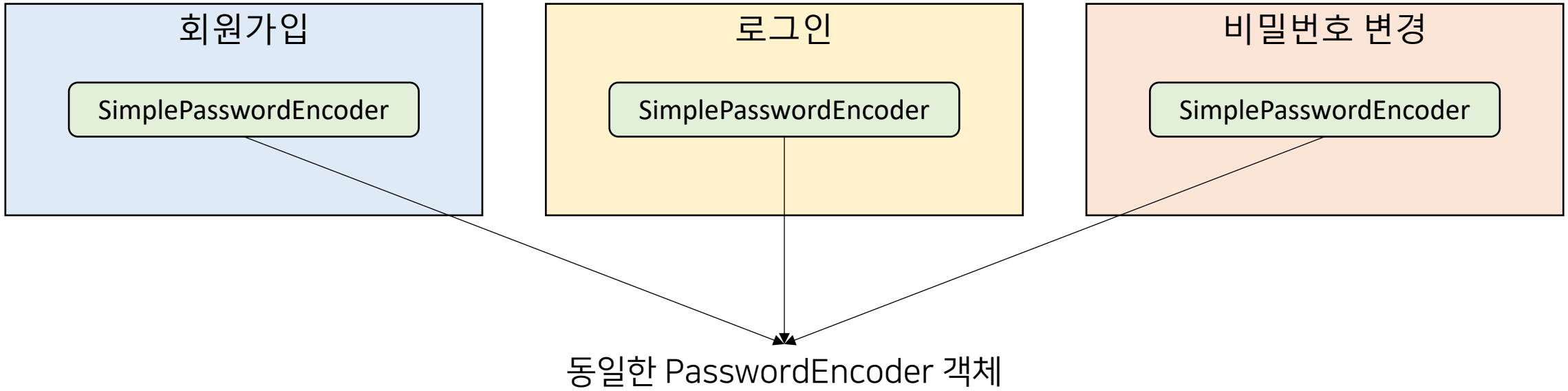
Spring이란

- 자바 기반의 웹 애플리케이션 프레임워크 → 코틀린도 지원
- 멀티쓰레드 기반의 동장 방식 → WebFlux
- 싱글톤 기반의 빈 객체 관리



Spring이 싱글톤을 사용하는 이유

- 싱글톤: 1개의 객체 만을 생성하여 관리
- 스프링에 여러번 빈(객체)을 요청하더라도 매번 동일한 객체를 반환



Spring이 싱글톤을 사용하는 이유

- 대규모 트래픽을 처리할 수 있도록 (대규모 엔터프라이즈 환경)
- If not 싱글톤...
  1. 매번 클라이언트에서 요청이 올 때마다 로직을 처리하는 빈을 생성
  2. 1초에 500번 요청이 오고, 요청을 처리하기 위해 5개의 객체가 필요함
  3. 1초에 2500개의 새로운 객체가 생성
  4. GC의 빈도가 잦아져 애플리케이션의 중단 시간 증가(stop-the-world)

→ 빈을 싱글톤으로 관리하여 1개의 요청이 왔을 때 여러 쓰레드가 공유

Spring의 핵심 특징

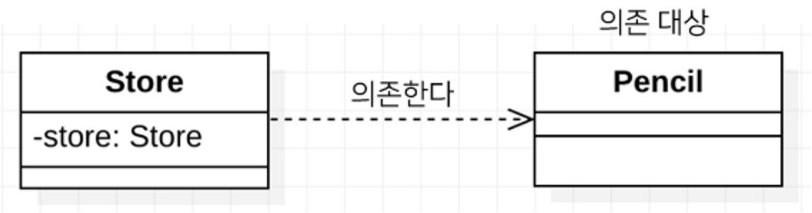
- IoC/DI(제어의 역전/의존성 주입)
- 서비스 추상화
- AOP(Aspect Oriented Programming)

IoC/DI(제어의 역전/의존성 주입)

- 두 객체 간의 관계를 결정해주는 디자인 패턴
- 인터페이스를 사이에 둬서 클래스 레벨에서는 의존관계가 고정되지 않도록 하고 런타임 시에 관계를 다이나믹하게 주입
- 유연성을 확보하고 결합도를 낮출 수 있음(다형성)

IoC/DI(제어의 역전/의존성 주입)

- 상점에서는 연필을 팔음(상점은 연필에 의존함)
- 상점에서는 연필 말고 책을 팔고 싶다면 → Store 클래스 변경 필요




```
public class Pencil {


}
```

```
public class Store {

  private Pencil pencil;

  public Store() {
    this.pencil = new Pencil();
  }


}
```

IoC/DI(제어의 역전/의존성 주입)

- 연필, 책, 지우개 등을 묶은 Product를 만들고 주입해주자!
- 결국 핵심은 객체지향의 다형성!

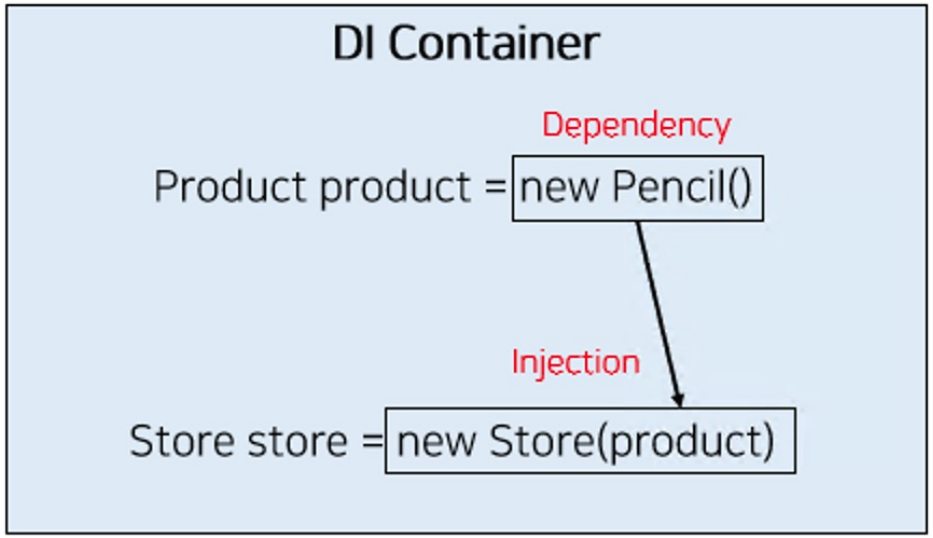
```
public interface Product {

}
```

```
public class Pencil implements Product {

}
```

```
public class Store {

    private Product product;

    public Store(Product product) {
        this.product = product;
    }

}
```

IoC/DI(제어의 역전/의존성 주입)

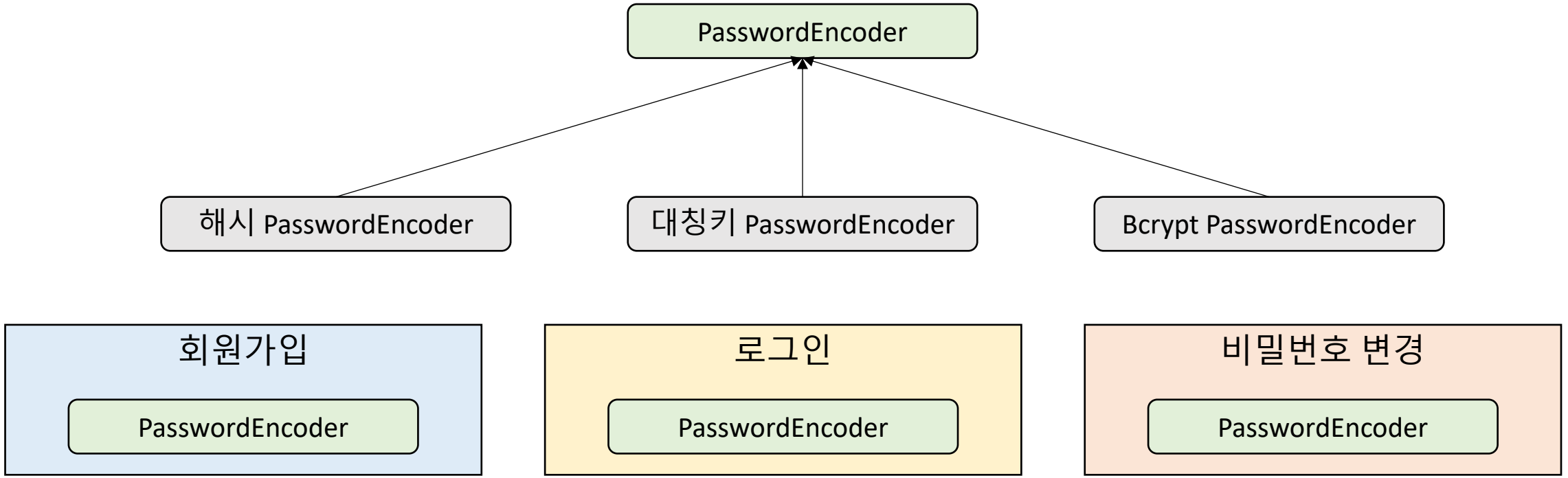
- 개발자가 new로 객체를 만들지 않음
- Spring(DI 컨테이너)가 알아서 만들고 주입을 해줌




```java
@RestController
public class LoginController {

    private final LoginService loginService;

    @Autowired
    public LoginController(final LoginService helloService) {
        this.loginService = helloService;
    }

    @GetMapping("/login")
    public ResponseEntity<String> login() {
        return ResponseEntity.ok(loginService.login());
    }

}
```

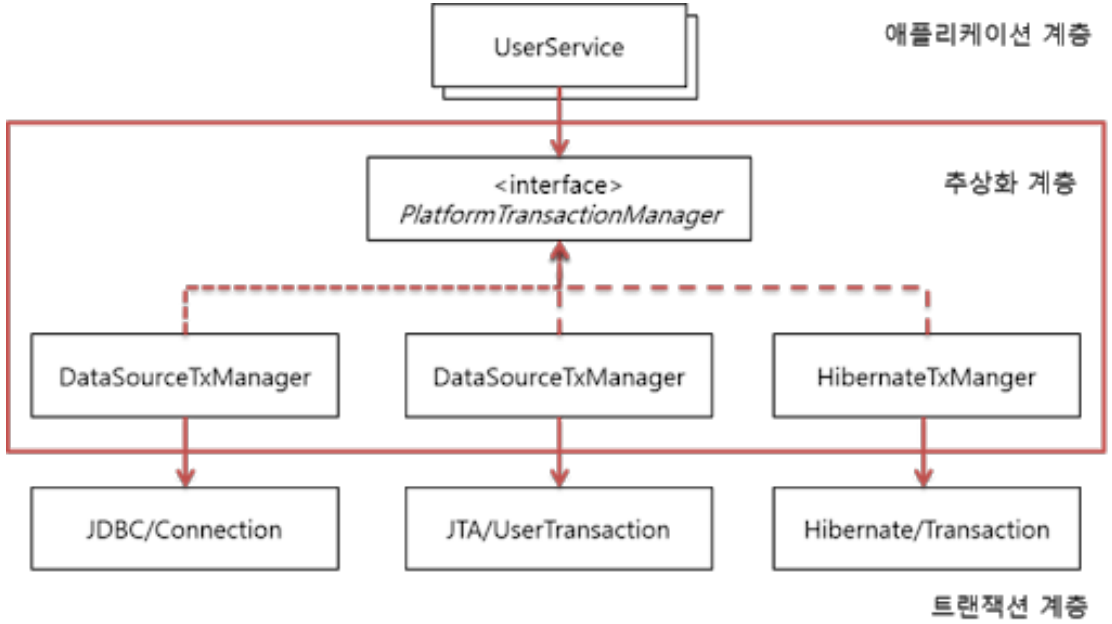
IoC/DI(제어의 역전/의존성 주입)

- 유연성을 확보하고 결합도를 낮출 수 있음
  - 유연성 확보: PW 암호화 알고리즘이 변경되어도 코드를 수정하지 않아도 됨
  - 결합도 낮춤: 각각의 서비스들은 어느 PW 암호화를 사용하는지 몰라도 됨



서비스 추상화

- 특정 환경이나 서버, 기술에 종속되지 않으며 유연하게 개발 가능
- DB 접근을 위해 어떠한 기술을 써도 동일하게 트랜잭션 처리 가능
- 추상화를 위해 프록시(Proxy) 패턴 등이 매우 자주 사용됨
- 트랜잭션, 캐시 기타 등등



서비스 추상화

- 캐시 도구로는 Redis, Ehcache, Memcache 등이 있음
- Spring의 캐시 추상화 → 캐시 도구가 변경되어도 코드 영향 X

```
public interface FileRepository extends JpaRepository<MyFile, Long> {

    @Cacheable(key = "resourceId", value = "value")
    Optional<MyFile> findByResourceId(final String resourceId);

}
```

AOP(Aspect Oriented Programming)

- 핵심 로직이 아닌 공통의 부가적인 기능들을 독립적으로 모듈화
- 깔끔한 코드를 유지할 수 있고, 객체지향스럽게 개발

AOP(Aspect Oriented Programming)

- 사용자 로그인의 경우, 입력값이 Null인지는 핵심 로직이 아님
- 핵심 로직은 입력받은 이메일과 패스워드가 올바른지 검사하는 것!

```java
@Service
public class LoginService {

    public String login(String email, String pw) {
        if (email == null || pw == null) {
            throw new IllegalArgumentException();
        }

        // ... 비밀번호 검사 이후 로그인 처리

    }
}
```

AOP(Aspect Oriented Programming)

- 유효성 검사는 숨겨서 처리하고 핵심 로그인 로직만 집중

```java
@Getter
@RequiredArgsConstructor
public class LoginRequest {

    @Email
    private final String email;

    @NotBlank
    private final String pw;

}
```

```java
@Service
public class LoginService {

    public String login(LoginRequest loginRequest) {
        // ... 비밀번호 검사 이후 로그인 처리

    }

}
```

AOP(Aspect Oriented Programming)

- Spring은 캐시, 트랜잭션 등에서 AOP를 사용함
- 우리가 원하는 공통 로직들도 AOP로 커스텀하게 만들 수 있음
  - Ex) 메소드의 실행 시간 측정 등

```java
@Service
@RequiredArgsConstructor
public class UserService {

    private final PlatformTransactionManager transactionManager;
    private final UserRepository userRepository;

    public void addUsers(List<User> userList) {
        TransactionStatus status = this.transactionManager.getTransaction(new DefaultTransactionDefinition());
        try {
            for (User user : userList) {
                if (isEmailNotDuplicated(user.getEmail())) {
                    userRepository.save(user);
                }
            }
            this.transactionManager.commit(status);
        } catch (Exception e) {
            this.transactionManager.rollback(status);
            throw e;
        }
    }

    private boolean isEmailNotDuplicated(String email) {
        return true;
    }
}
```

```java
@Service
@RequiredArgsConstructor
public class UserService {

    private final PlatformTransactionManager transactionManager;
    private final UserRepository userRepository;

    public void addUsers(List<User> userList) {
        TransactionStatus status = this.transactionManager.getTransaction(new DefaultTransactionDefinition());
        try {
            for (User user : userList) {
                if (isEmailNotDuplicated(user.getEmail())) {
                    userRepository.save(user);
                }
            }
            this.transactionManager.commit(status);
        } catch (Exception e) {
            this.transactionManager.rollback(status);
            throw e;
        }
    }

    private boolean isEmailNotDuplicated(String email) {
        return true;
    }
}
```

핵심 비지니스 로직만 남길 수 있음

Spring의 핵심 특징

- IoC/DI(제어의 역전/의존성 주입)
- 서비스 추상화
- AOP(Aspect Oriented Programming)