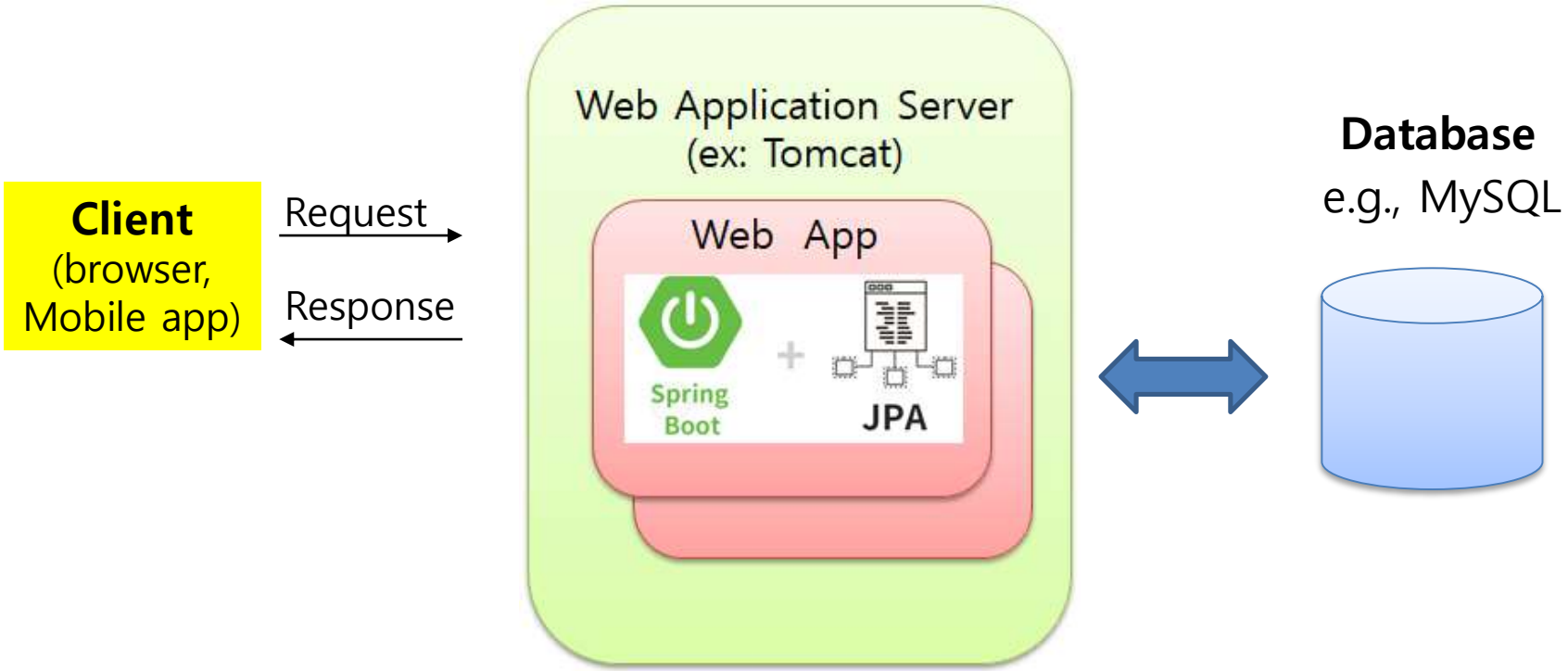
웹 프레임워크

교과목 목표

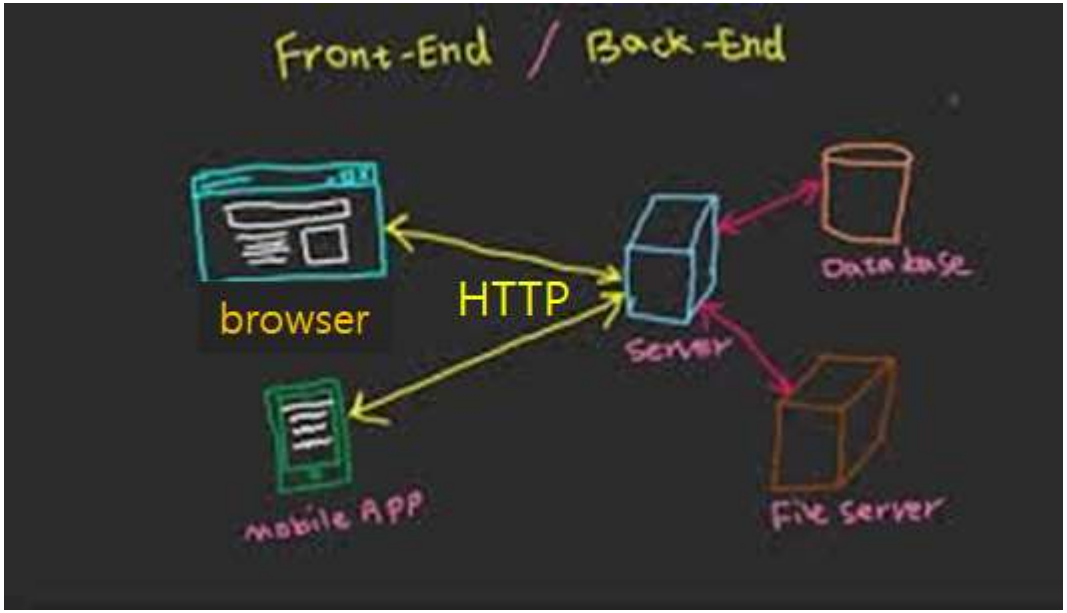
- Learn how to develop web applications with Spring/Spring Boot and JPA



1. 웹 시스템

웹 환경하에서 서로 다른 종류의 컴퓨터간에
상호 작용을 하기 위한 소프트웨어 시스템

- 클라이언트-서버 기반
- HTTP, 메시지 지향적(XML/JSON)
- 플랫폼 중립적, 독립적



웹 개발 기술

언어: HTML, CSS and JavaScript
라이브러리/프레임워크:
jQuery, Bootstrap, React, Angular

언어: PHP, **Java**, Ruby, Python
프레임워크: **Spring**, Rails, Django
데이터베이스: **MySQL**, PostgreSQL, Oracle
운영체제: Windows, **Linux**
버전관리: CVS, **Git**

2. SW 품질

Great software is... more than just one thing

기능 품질

구조 품질

It's going to take more than just a simple definition to figure out exactly what "great software" means. In fact, *all* of the different programmers out there I talked about a *part* of what makes software great.

First, great software must satisfy the customer. The software must do what the customer wants it to do.

Building software that works right is great, but what about when it's time to add to your code, or reuse it in another application? It's not enough to just have software that works like the customer wants it to; your software better be able to stand the test of time.

Second, great software is well-designed, well-coded, and easy to maintain, reuse, and extend.

Win your customers over

Customers will think your software is great when it does what it's supposed to do.

Make your code as smart as you are.

You (and your co-workers) will think your software is great when it's easy to maintain, reuse, and extend.

프레임워크란?

- 프레임워크는 '구조 품질'을 보장
  - SW 구조 그리고 기반되는 클래스를 제공

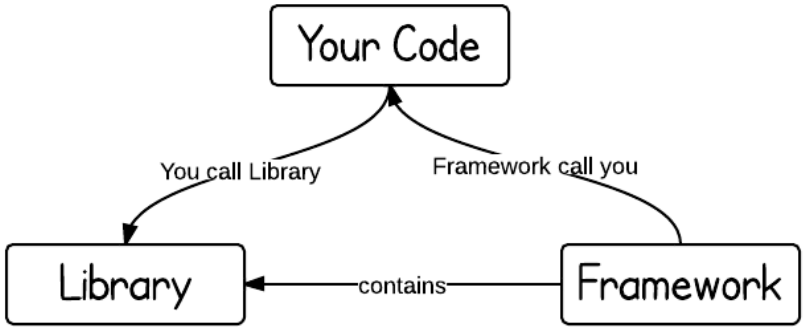
프레임워크의 중요성

- 프레임워크는 반제품 처음부터 만들지 않아도 됨
  - 애플리케이션 구조 및 코드의 상당 부분을 제공
  - 개발자는 애플리케이션의 핵심 로직에 집중 가능

- 프레임워크 장점
  - 높은 생산성
  - 코드 품질 보장

라이브러리 vs 프레임워크

- 중요한 차이점은 "Inversion of Control"
  제어의 역전
- 라이브러리
  - 클래스의 집합으로서 코드의 재사용성 지원(예: math)
  - 제어의 주체는 개발자: 코드에서 라이브러리 함수를 호출
- 프레임워크
  - 제어의 주체는 프레임워크: 프레임워크에서 여러분의 코드를 호출(제어의 역전)
  - 프레임워크에서 기본적인 골격을 잡아놓았기 때문에 우리는 제어의 흐름에 맞게 코드를 작성해 두면 프레임워크에서 호출



We can think of a *library*
as a certain function of an application,
a *framework*
as the skeleton of the application

3. 스프링이란?

Lightweight
JAVA Application
Framework
(Open source)

POJO(vs EJB) 기반의
엔터프라이즈 애플리케이션
개발을 쉽고 편하게

스프링이란?

- 자바 애플리케이션을 개발하는데 필요한 하부구조 (infrastructure)를 포괄적으로 제공

- 스프링이 하부 구조를 처리하므로 개발자는 애플리케이션 개발에 집중 -> Simplify Java Enterprise Development

3.1 스프링 주요 특징


애플리케이션 구성
의존 관계 주입
Dependency Injection
횡단 관심사 분리
관점 중심 프로그래밍
Aspect Orientated Programming
순수 객체
(POJO)
이식 가능한 서비스 추상화
Portable Service Abstraction

1) POJO
(Plain Old Java Object)

- Simple Java class that is not required to implement any special interface or inherit from any specific class
- Configured and managed by the Spring container

상속, 인터페이스가 없는 아주 간단한 객체

POJO

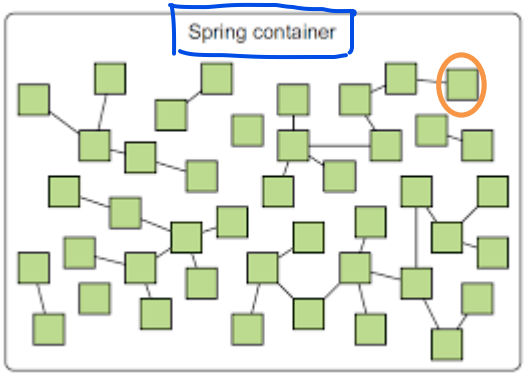
POJO의 일종이다

```
public class Person {
    private String name;
    private int age;

    public Person() {
    }

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // getters and setters
    ...
}
```

스프링에서는 (객체 = bean)

```
<bean id="person"
            class= " kr.ac.hansung.Person">
    <constructor-arg value="John Doe" />
    <constructor-arg value="30" />
</bean>
```

POJO를 관리해줌



person

∴ 우리는 객체를 만들지 않고 설정만 하면 됨

2) DI
(Dependency Injection)

```
@Component
public class PersonService {
    private Person person;

    @Autowired
    public PersonService(Person person) {
        this.person = person;
    }

    public void printPerson() {
        System.out.println("Name: " + person.getName() +
                           ", Age: " + person.getAge());
    }
}
```

person 이 무엇인가에 따라 다르게 출력될 것

The PersonService class is a service that depends on a Person object.
It uses constructor injection to receive a Person object,
which is provided by the Spring container.

DI

- The goal is to promote loose coupling between objects
- Instead of creating objects and their dependencies directly, objects are created and configured by an external entity, known as the Spring IoC container

객체간 의존성이 낮아짐, 의존성을 신경쓸 필요 없어짐

3) AOP
(Aspect-Oriented Programming)

```
public class UserService {
    public void addUser(User user) {
        // Add the user to the database
    }
}
```

We want to log information about when the addUser method is called

Non-AOP Approach

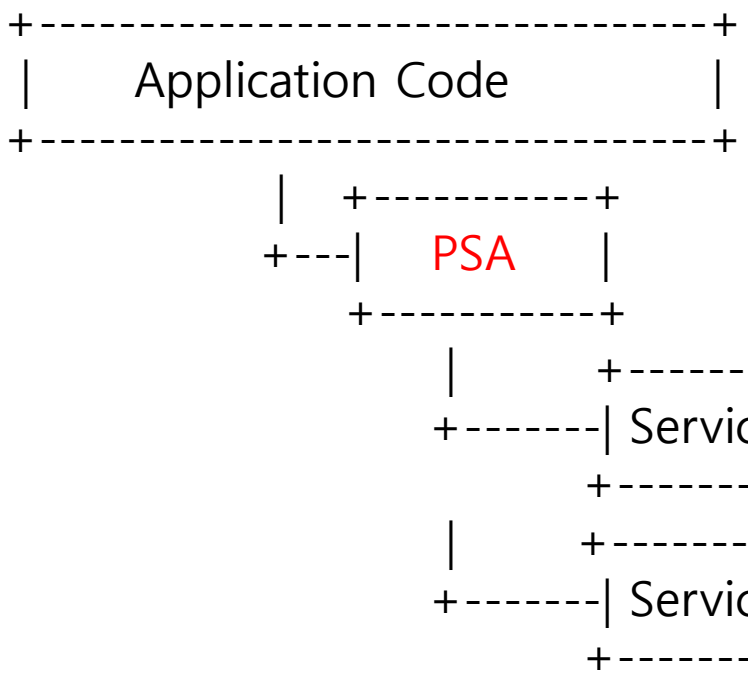
```
public class UserService {
    public void addUser(User user) {
        // Log information about adding the user
        System.out.println("Adding user: " + user.getUsername());

        // Add the user to the database
    }
}
```

AOP

AOP Approach

```
@Aspect
public class LoggingAspect {

    @Before("execution(* kr.ac.hansung.UserService.addUser(..)) && args(user)")
    public void logBefore(JoinPoint joinPoint, User user) {
        System.out.println("Adding user: " + user.getUsername());
    }

}
```

4) PSA
(Portable Service Abstraction)



PSA란 추상화된 것을 가지고 있어, Service Provider를 1에서 2로 바꿔도 지장이 없도록 함

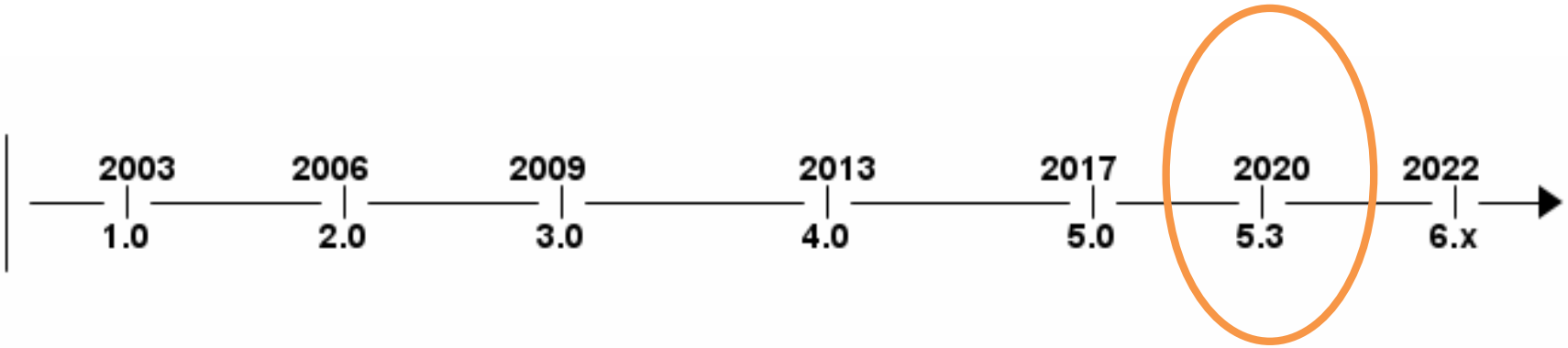
∴ 개발자는 신경 쓸 필요없다

The application code interacts with a PSA layer,
which provides a common interface for working with different service providers.
The service providers are the specific implementations of the service

PSA

- By using PSA in Spring, the application code can be written to work with the common interface defined by the PSA layer, without having to worry about the specific implementation of the service
- This makes it easier to switch between different service providers, without having to modify the application code

3.2 스프링 버전



3.3 스프링 웹 사이트(공식)

- www.spring.io

- Spring Framework Documentation Version 5.3.30
  - https://docs.spring.io/spring-framework/docs/5.3.x/reference/html/

마무리

∴ 프레임워크를 쓰자

- 스프링을 통해 Java Enterprise 시스템 개발이 용이
  - 비즈니스 로직에 집중 가능하여 생산성 증대
  - 재사용 및 유지 보수 용이, 확장성을 가진 코드 설계

- 전자 정부 표준 프레임워크는 스프링 기반

- 소프트웨어 공학의 주요 설계 패턴을 경험할 수 있는 기회가 있음
  - 좋은 프로그래밍 습관 형성이 가능

SW를 개발하는 중요한 전략의 하나는
"항상 프레임워크 기반으로 접근하라"
- Rod Johnson