SpringBoot 1일차

처음 스프링 깔고나서 폴더 구조

gradle - gradle을 쓰는 폴더

src : main, test가 나눠져있고 따라서 요즘은 test코드가 진짜 중요하다
- main
  - resources : 실제 자바 코드파일 제외한 xml, property같은 설정파일, html도 여기 들어감. 자바파일 제외한 모든 파일이 여기로
  - java : 실제 패키지와 소스파일
    - hellospringapplication : run 하면 막 뜸 -> localhost:8080에서 보면 화이트라벨에러 -> 성공함!
- test : test code가 들어가있는부분
- build.gradle : 가장 중요함. gradle이 버전설정하고 라이브러리 땡겨온다. 나중에 다시 공부하자.
  - sourceCompatiblility : 자바 버전 설명
  - dependencies에 있는 패키지들을 다운받는곳이 repositories - maven
- gradlew : 나중에 다시

라이브러리 살펴보기

처음에 선택한 라이브러리 얼마없긴함 - build.gradle의 dependencies에 나온것
근데 땡겨온 라이브러리 : External Libraries에 있음 - Gradle, maven같은 툴들은 의존관계를 다 관리해준다. 스프링부트스타터 웹 라이브러리가 의존관계를 다 분석해서 아예 의존관계있는애들을 다 땡겨옴
우측에 gradle을 눌러보면 디펜던시스
- compileclasspath에서 계속 하위폴더를 열수있는데 얘네가 다 의존관계에 있음 -> 계속 파고들고 그 라이브러리 다 땡겨옴
- (*)이라고 돼있는건 이미 땡기고 있다. 중복으로 다른데서 땡겨진거임.
- **스타터 웹**에 잇는 톰캣이 8080을 열어준거임. 자바 메인메소드만 실행해도 웹서버에 들어갈수있음
- 로그파일 : 웹에서 쓰는 로그 얘기하는듯? 로그백과 slf4j는 어떤 구현체로 로그를 출력할건가?
  - logback
  - slf4j
- testcompileclasspath
  - **junit**
  - assertj
  - mokito

웹페이지 만들기
강의록 참고하기!

템플릿 엔진을 이용하면 루프라던지..를 넣어서 index.html을 조종할 수 있음

controller:웹어플리케이션에서 첫번째 진입점
model : mvc 할때 그 모델

겟매핑 : get - post 할때 그 겟임.
리턴을했을때 hello면 resources 밑에 hello를 찾는다.
templates/hello.html의 ${data}는 모델에서 넘겼던 키값임.

```
@GetMapping("hello")
public String hello(Model model) {
    model.addAttribute( attributeName: "data",  attributeValue: "hello!!");
    return "hello";
}
```

```
<body>
<p th:text="'안녕하세요. ' + ${data}" >안녕하세요. 손님</p>
</body>
</html>
```

model.add~~
- 어트리뷰트 네임은 {data}와 매치되는 변수.
- 어트리뷰트 밸류는 {data}에 들어갈 변수값.
- 리턴값은 getmapping시 뷰 리졸버가 찾아갈 templates의 파일명..?인듯

스프링 웹 개발 기초

- 정적 컨텐츠 : 그냥 파일을 웹에 바로 나오게 하는것. 파일을 고객에게 바로 전달
  - 정적 컨텐츠 이미지 참고
    - 내장 톰캣서버가 hello-static 관련 컨트롤러를 찾았는데 없음1
      - 그러면 바로 hello-static.html이 있는지 찾아서 실행시킴
- MVC와 템플릿 엔진 : 가장 많이 함. jsp, php... 소위 말하는 템플릿 엔진. html을 그냥 주는게 아니라 서버에서 프로그래밍해서 html을 동적으로 내려주는것. controller, model, view(template엔진화면) -> MVC
- API : 안드로이드나 아이폰 클라이언트랑 개발한다면? 서버입장에서 json 데이터 포맷으로 내려줌. json만 내려주면 클라이언트가 알아서 화면 켜기

MVC와 템플릿 엔진
model, view, controller : 렌더링된 html을 고객에게 전달한다
- model : 비즈니스 로직과 관련있거나 내부적인걸 처리하는데 집중
- view : 화면을 그리는데 집중.

- controller : 비즈니스 로직과 관련있거나 내부적인걸 처리하는데 집중

MVC와 템플릿
http://localhost:8080/hello-mvc**?name=spring!!!!!!**
get post 메소드의 형식.
viewresolver는 html 찾아주는애?인듯!
model(key:value) -> model(name:spring)

API : 객체를 반환!!!!!!!
responsebody : http의 body부분에 내용을 직접 넣겠다.

```
@GetMapping("hello-api")
@ResponseBody
public Hello helloApi(@RequestParam("name") String name) {
    Hello hello = new Hello();
    hello.setName(name);
    return hello;
}
3 usages
static class Hello {
    2 usages
    private String name;

    no usages
    public String getName() {
        return name;
    }

    1 usage
    public void setName(String name) {
        this.name = name;
    }
}
```

위 방법이 api이고, 이것을 웹에서 열어보면 json으로 나온다.
@Responsebody가 들어있으면, viewresolver에게 주지 않고
http body에 바로 주면 되겠다고 한다.
근데 넘겨지는게 객체면 스프링 입장에서는 한번 생각하고
객체가 오면 기본 디폴트가 json방식으로 데이터를 만들어서 http 반응에 응답하기로 정해짐.
HttpMessageConverter
- JsonConverter : 객체라면 이거
- StringConverter : 그냥 문자라면 이거

회원관리예제 - 백엔드개발

```
    private static Map<Long, Member> store = new HashMap<>();
    1 usage
    private static long sequence = 0L;

    no usages
    @Override
    public Member save(Member member) {
        member.setId(++sequence);
        store.put(member.getId(), member);
        return member;
    }

    no usages
    @Override
    public Optional<Member> findById(Long id) {
        return Optional.ofNullable(store.get(id));
    }

    no usages
    @Override
    public Optional<Member> findByName(String name) {
        store.values().stream()
                .filter(member -> member.getName().equals(name))
                .findAny();
    }

    no usages
    @Override
    public List<Member> findAll() {
        return new ArrayList<>(store.values());
    }
}
```

save : 멤버를 저장함. sequence를 이용하여 id를 세팅해주고 store에 멤버의 id를 넣고 member리턴
findbyid : id로 멤버를 찾음. null일수도 있으니 optional.ofnullable로 감싸줌
findbyname : name으로 멤버를 찾음. 람다함수를 사용했고 store의 value들을 모두 돌면서 (stream) member의 이름이 name과 같다면 하나라도 찾기(findAny)
findall : store의 value들을 모두 어레이로 불러옴

테스트 케이스를 작성해서 잘 되는지 검증

모든 테스트는 순서가 보장이 안된다. 따라서 순서가 랜덤이어도 잘 돌아가도록 설계해야한다.
findall 하고 findbyname할 때 findall에서의 spring1, 2가 들어간듯!
그래서 데이터를 깔끔하게 clear해야함.
테스트를 하고 원본 코드를 짜는 것 -> 틀을 먼저 만드는것 -> 테스트 주도 개발(TDD)

회원 서비스 기능 구현

cmd opt v = 리턴 그냥 해줌
null일 가능성이 있으면 Optional로 감싸서 해주기

```
memberRepository.findByName(member.getName())
                .ifPresent(m -> {
                    throw new IllegalStateException("이미 존재하는 회원입니다.");
                });
```

와 같은 형식은 메소드로 빼주는게 좋음. control+T하면 메소드로 뽑기 가능. extract method

레포지토리는 그냥 되게 단순히 저장소에 넣다빼는 느낌이 나지만 -> 기계적으로 개발스럽게 언어선택
서비스는 비즈니스에 가까움. 조인 등등 -> 서비스는 비즈니스적으로 짜는게 좋음

=> 테스트 해보는 방법은 테스트케이스 활용

회원 서비스 기능 테스트

MemberService 클래스에서 단축키 cmd+shift+t => 똑같은 패키지에 이름까지 맞춰서 테스트파일 구현