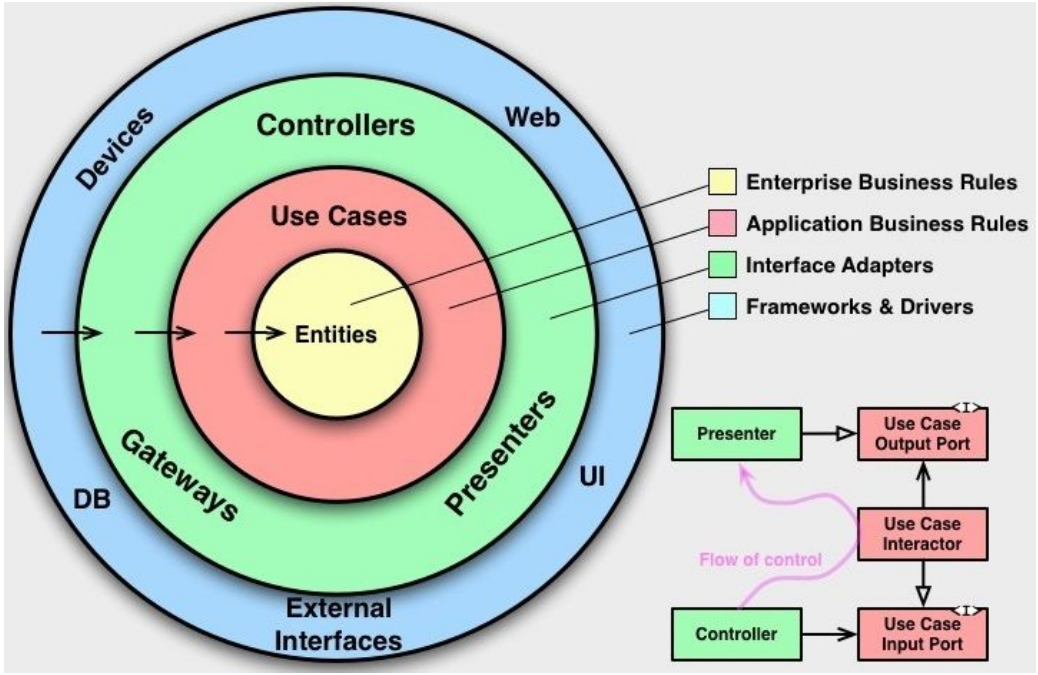
32장
프레임워크는 세부사항이다

프레임워크와 아키텍처

- 당신의 코드는?



저의 경험상...

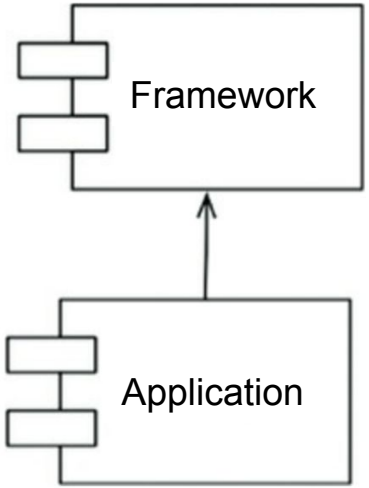
- 프레임워크를 사용하지 않는 경우는 본 적이 없고,
- 프레임워크가 바깥쪽 원에 위치하는 것을 본 적도 없음.

그래서...

- 프레임워크의 버전이 달라졌거나
- 새로운 프레임워크가 등장해서 갈아타고 싶거나
- 제품의 요구사항이 프레임워크와 궁합이 맞지 않거나

프레임워크는 세부사항이다.

- 이 사태의 원인: 프레임워크를 바깥쪽 원 (=세부사항)으로 다루지 않았기 때문
- 해결책: 프레임워크를 세부사항으로 다루자



왜 우리는 프레임워크를 세부사항으로 다루지 않는가?

- 프레임워크 제작자들은 프레임워크에 대한 비대칭적 헌신을 요구한다.
  - 프레임워크를 우리가 만들 소프트웨어가 감싸도록 요구한다.
  - 프레임워크 기능들을 업무 객체에 임포트해서 사용하라고 요구한다.
  - 프레임워크가 제공하는 기반 클래스로부터 파생클래스를 만들라고 요구한다.
- 프레임워크 제작자들은 아무런 영향을 받지 않을 뿐더러, 자신들의 프레임워크의 유용성을 검증 받는 것

왜 우리는 프레임워크를 세부사항으로 다루지 않는가?

- 프레임워크에 익숙하지 않으면, 제작자들의 문서에 의존하다가 일을 그르침
- 프레임워크에 익숙하면, 빠르게 개발하거나 원래 편한 방식으로 하다가 일을 그르침

예시: Django

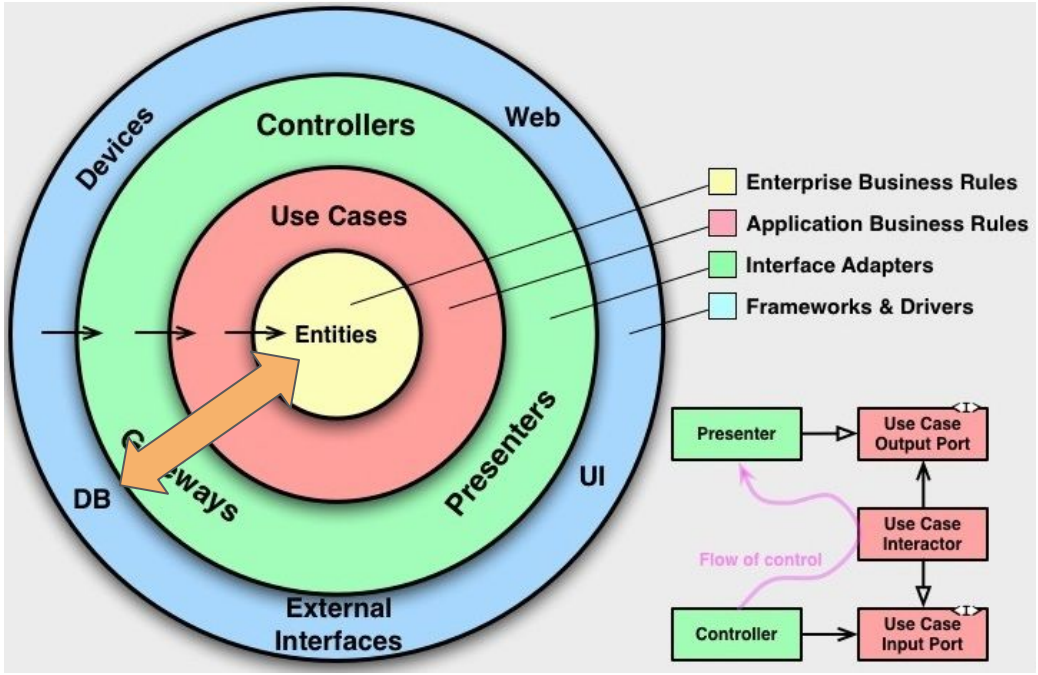
polls/models.py

```python
from django.db import models


class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')


class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
```

예시: Django


Devices
Controllers
Web
Use Cases
Entities
DB
Presenters
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers
Presenter
Use Case
Output Port
Use Case
Interactor
Flow of control
Controller
Use Case
Input Port

예시: Django

- 이렇게 DB와 Entity와 Framework가 결합되면서, 고려사항이 폭발적으로 늘어남
- Entity 객체들 간의 상속관계를 만들거나, 구성(Composition)을 만들고 싶을 때, 데이터베이스 테이블에 어떤 영향이 갈 것인지 고려해야함.

프레임워크와 애플리케이션

- 프레임워크를 성급하게 애플리케이션과 결합시키는 것은 연애도 안해보고 결혼하는 것과 같다.
- 그리고 알고보니 결혼상대가 엄청 이기주의적인 사람인 것과 같다.

해결책

- 결혼하지 말라!
- 프레임워크가 핵심 코드 안으로 들어오지 못하게 하라.
  - 만일 자신들의 기반클래스를 사용하라고 하면, 프록시를 만들고, 이 프록시로 만든 모듈을 업무 객체에 플러그인으로 써라
  - 의존성 규칙을 준수하라

예시: Spring @Autowired

```
@Service
public class BookService {
    private BookRepository bookRepository;

    @Autowired
    public BookService(BookRepository bookRepository){
        this.bookRepository = bookRepository;
    }
}
```

예시: Spring @Autowired

```
@Service
public class BookService {

    private BookRepository bookRepository;

    public BookService(BookRepository bookRepository){
        this.bookRepository = bookRepository;
    }

}
```

예시: Spring @Autowired

```
@Configuration
public class ApplicationConfig {
    @Bean
    public BookRepository bookRepository(){
        return new BookRepository();
    }

    @Bean
    public BookService bookService(){
        return new BookService(bookRepository());
    }
}
```

예외: 언어와 표준 라이브러리

- C++과 STL, JAVA와 표준 라이브러리의 결합은 불가피
- 하지만, 비용에 대해 알고 사용하고, 가능하면 제한하라

결론

- 앞의 예시도 사실 @Autowired를 객체 내부에 쓰면 더 좋다는 내용을 순서를 바꾼 것
- 확실히 우리의 프레임워크에 대한 사랑이 맹목적이라는 것을 깨달음
- 하지만, 상황에 따라 적절한 균형을 찾는 것도 중요한 것 같음