Clean Architecture

演講者：Frank

2025/02/18

國立台北科技大學　資訊工程系

指導教授：鄭有進、謝金雲

SOFTWARE SYSTEMS LAB
軟體系統實驗室 NTUT

Outline

- What is Software Architecture?
- Layered Rule
- Dependency Rule
- Cross-Boundary Rule
- Pros and Cons

What is Software Architecture?(1)

The architecture of a software is the **shape** given to that system by those who build it.
The form of that shape is in the division of the system into **components**, the **arrangement** of those components, and the ways in which those components **communicate** with each other.

What is the purpose of the shape?

Development

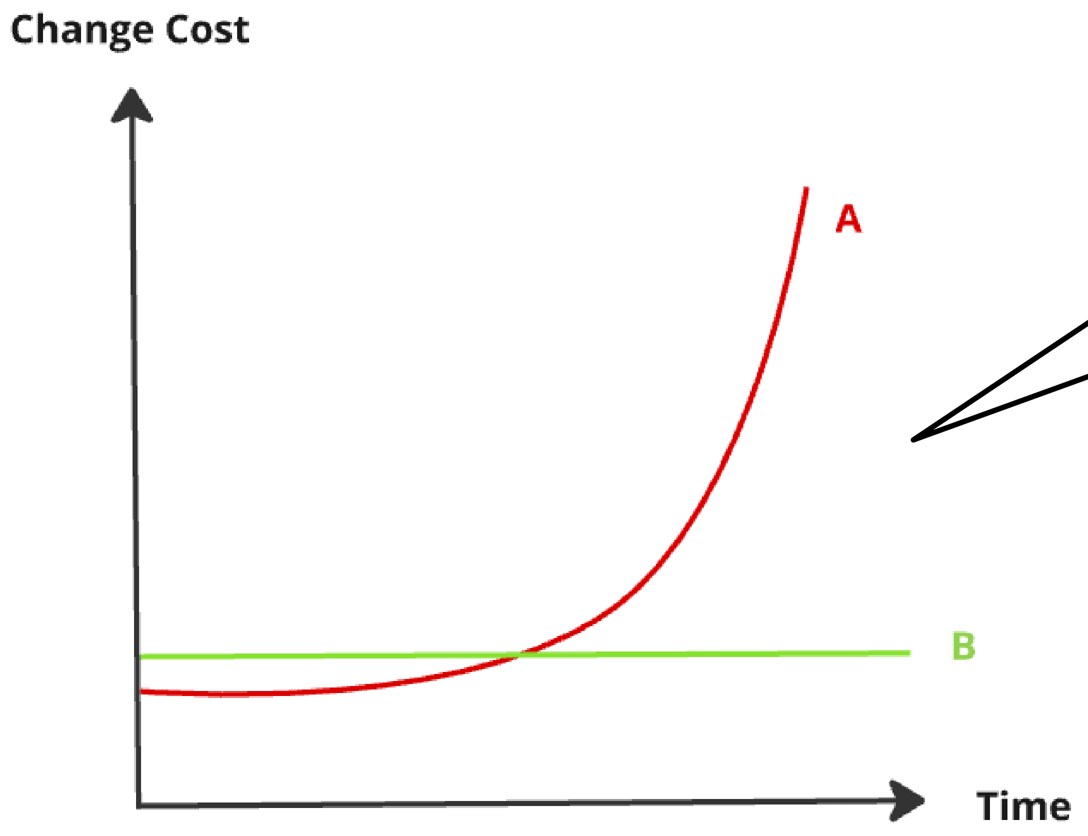
Deployment

Operation

Maintenance

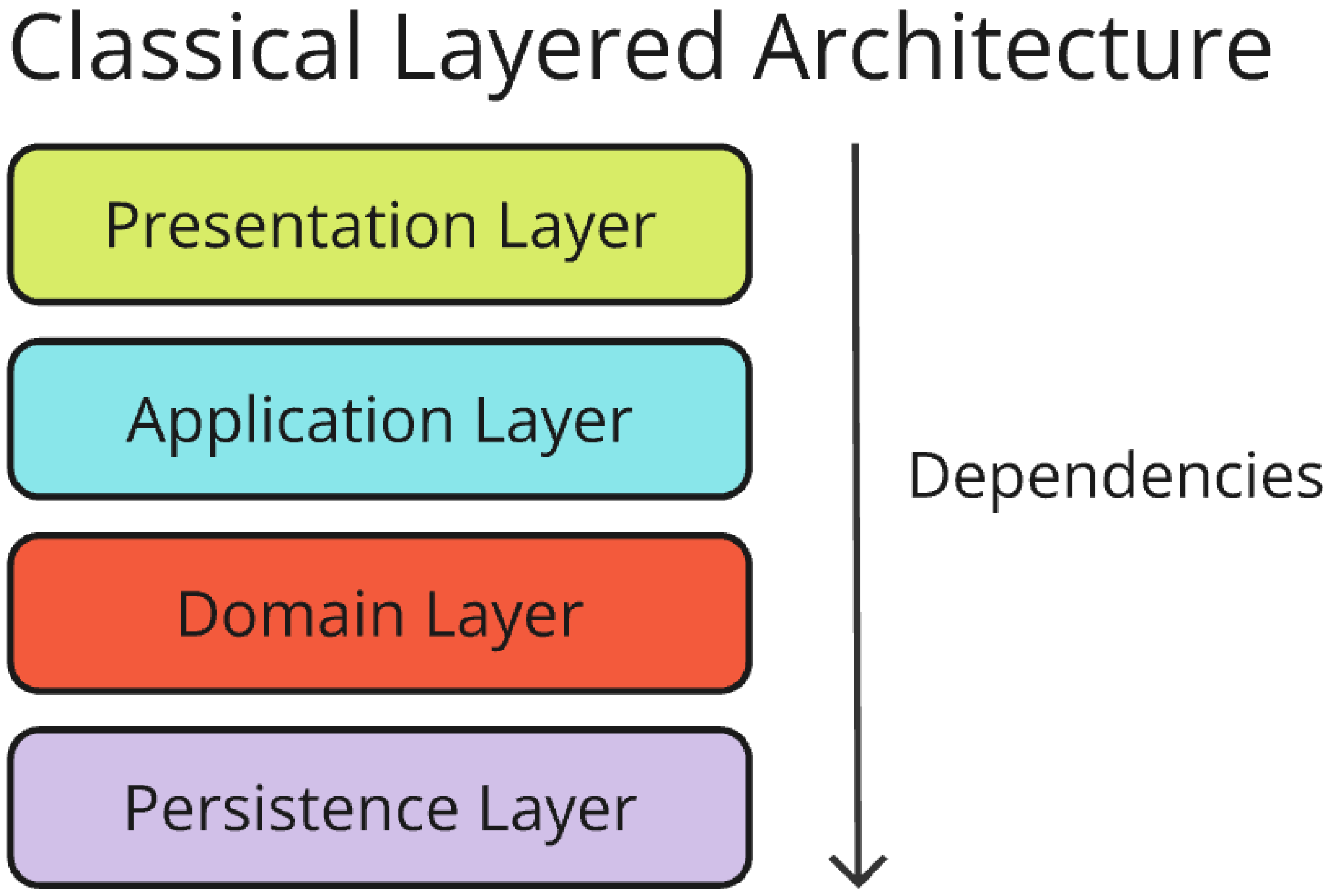
What is Software Architecture?(2)

The ultimate goal of software architecture is to minimize the lifetime cost of the system and to maximize programmer productivity.

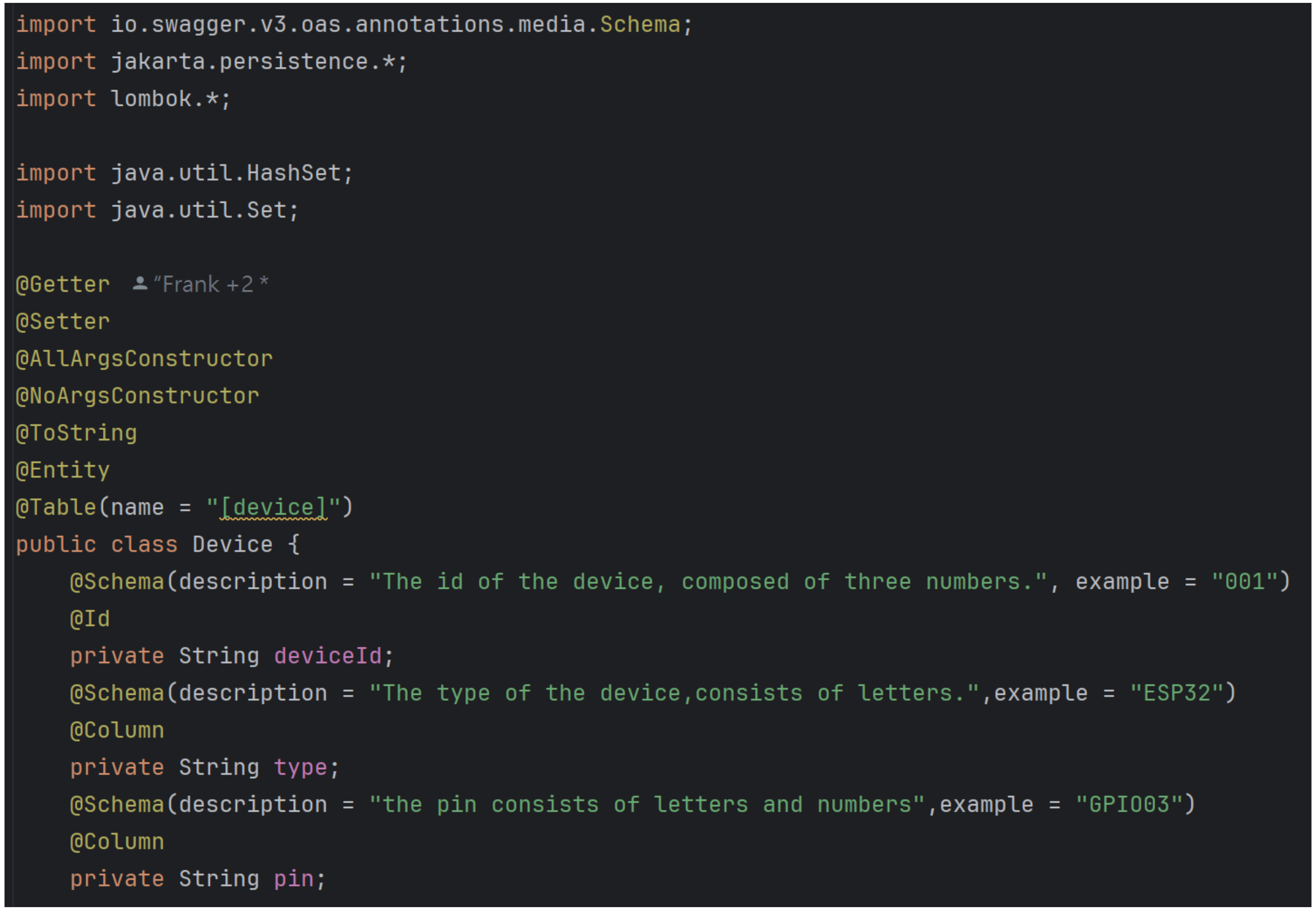


The cost of software changes is not related to time but to the scope of the change.

Layered Rule(1)



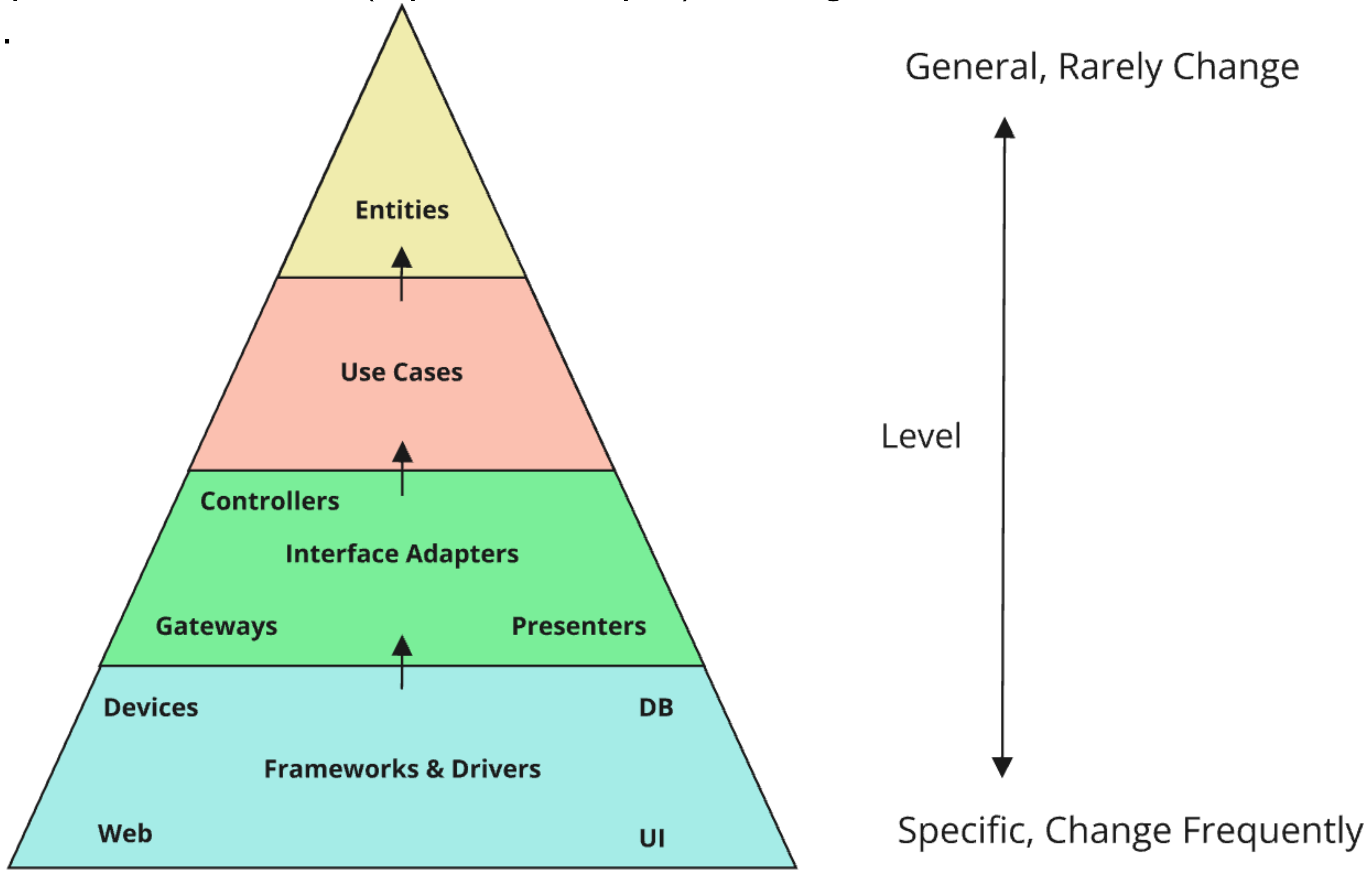
Layered Rule(1)



```
import io.swagger.v3.oas.annotations.media.Schema;
import jakarta.persistence.*;
import lombok.*;

import java.util.HashSet;
import java.util.Set;

@Getter  “Frank +2 *
@Setter
@AllArgsConstructor
@NoArgsConstructor
@ToString
@Entity
@Table(name = "[device]")
public class Device {
    @Schema(description = "The id of the device, composed of three numbers.", example = "001")
    @Id
    private String deviceId;
    @Schema(description = "The type of the device,consists of letters.",example = "ESP32")
    @Column
    private String type;
    @Schema(description = "the pin consists of letters and numbers",example = "GPIO03")
    @Column
    private String pin;
```

Layered Rule(2)

The farther a component is from I/O (input and output), the higher its level; the closer a component is to I/O, the lower its level.



DP Link

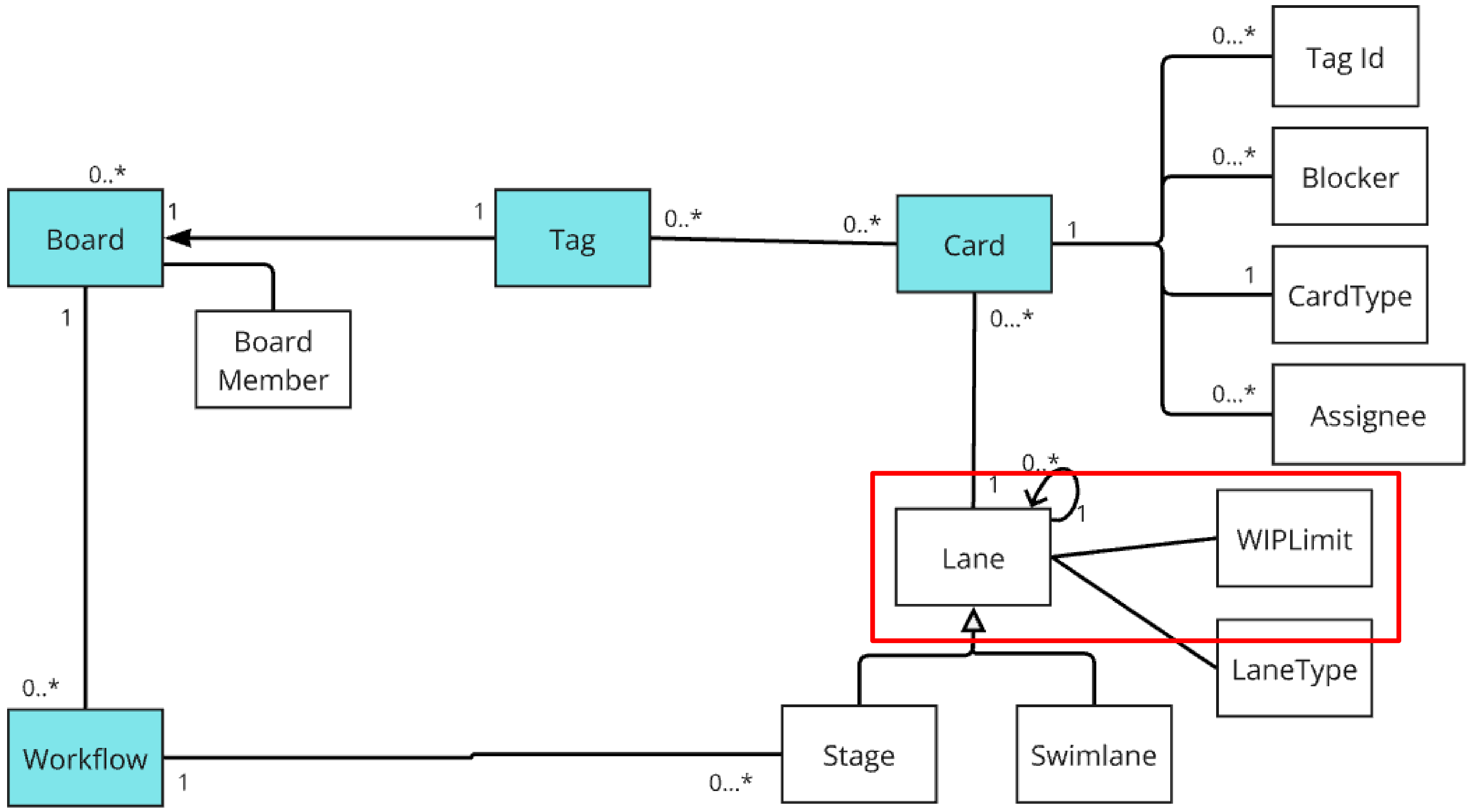
Layered Rule(3)

Entities：

Objects that encapsulate critical business rules and data, ensuring they remain pure business logic that can be used consistently across multiple applications within the enterprise. They are independent of databases, third-party dependencies, and user interfaces.

Layered Rule(4)

Example



Layered Rule(5)

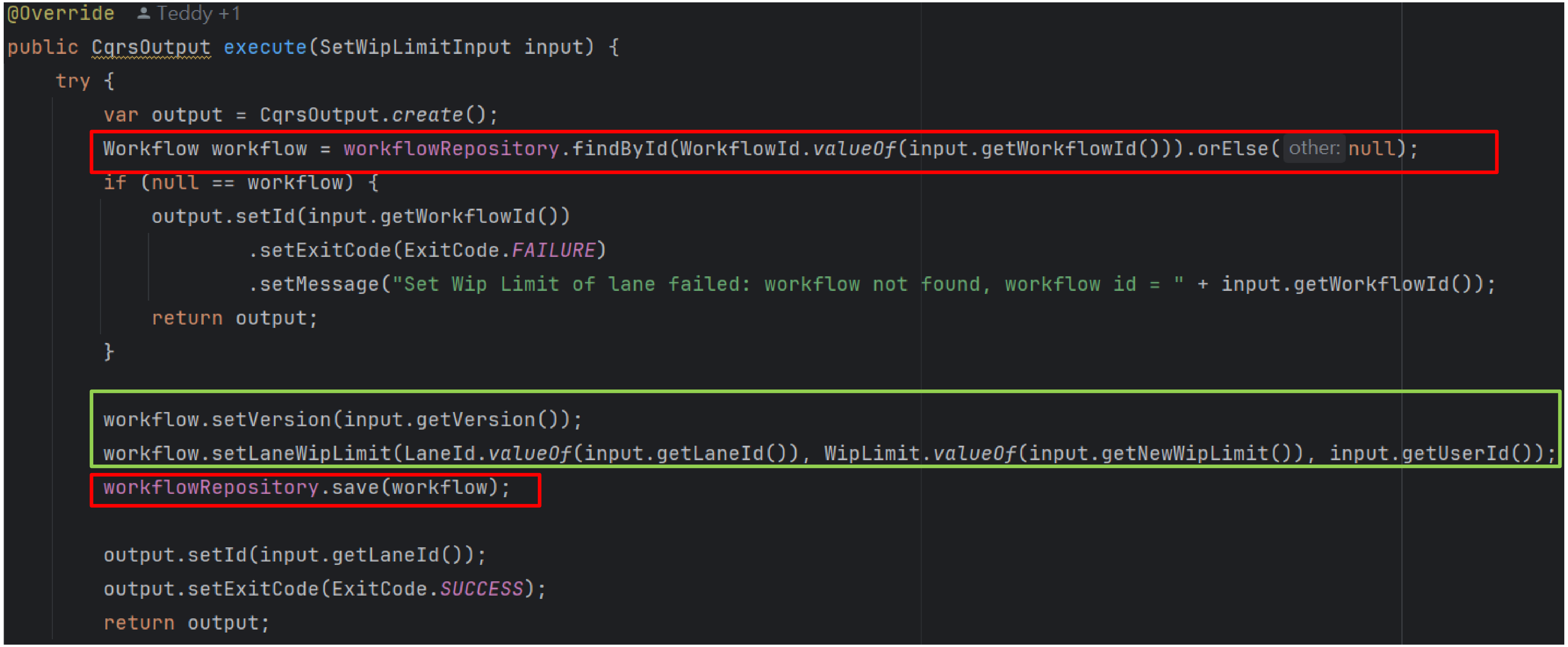
Example

Use Cases：

- Use cases are application specific business rules
  - Changes should not impact the Entities
  - Changes should not be impacted by infrastructure such as a database
- The use cases orchestrate the flow of data in/out of the Entities and direct the Entities to use their Critical Business Rules to achieve the use case.

Layered Rule(6)

SetWipLimitUseCase



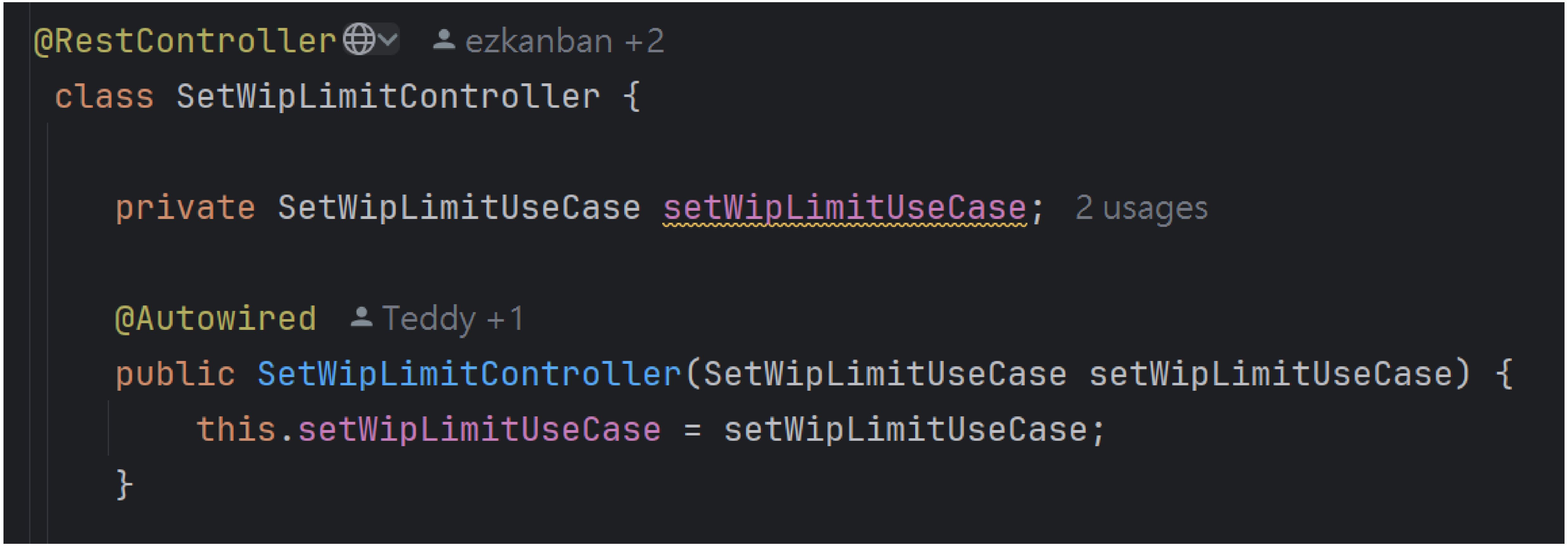
```
@Override  Teddy +1
public CqrsOutput execute(SetWipLimitInput input) {
    try {
        var output = CqrsOutput.create();
        Workflow workflow = workflowRepository.findById(WorkflowId.valueOf(input.getWorkflowId())).orElse( other: null);
        if (null == workflow) {
            output.setId(input.getWorkflowId())
                    .setExitCode(ExitCode.FAILURE)
                    .setMessage("Set Wip Limit of lane failed: workflow not found, workflow id = " + input.getWorkflowId());
            return output;
        }

        workflow.setVersion(input.getVersion());
        workflow.setLaneWipLimit(LaneId.valueOf(input.getLaneId()), WipLimit.valueOf(input.getNewWipLimit()), input.getUserId());
        workflowRepository.save(workflow);

        output.setId(input.getLaneId());
        output.setExitCode(ExitCode.SUCCESS);
        return output;
```

Layered Rule(7)

Interface Adapters：

- Converts data from data layers to use case or entity layers
    - Presenters, views and controllers all belong here
- No code further in (use cases, entities) should have any knowledge of the db.

Layered Rule(8)



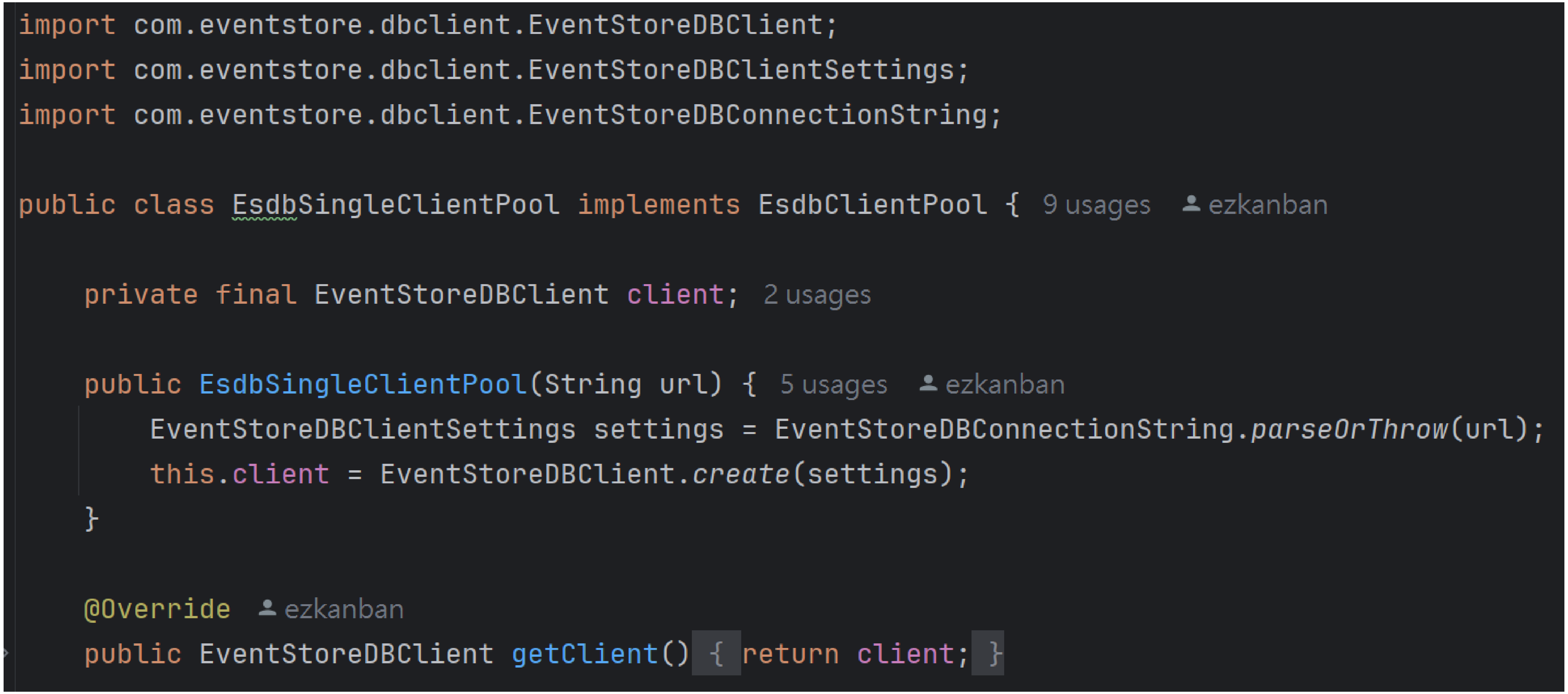
```
@RestController  ezkanban +2
class SetWipLimitController {

    private SetWipLimitUseCase setWipLimitUseCase;  2 usages

    @Autowired  Teddy +1
    public SetWipLimitController(SetWipLimitUseCase setWipLimitUseCase) {
        this.setWipLimitUseCase = setWipLimitUseCase;
    }
```

Layered Principle(9)

Frameworks and Drivers：

The outermost layer is generally composed of frameworks and tools such as the Database, the Web Framework, etc. Generally you don’t write much code in this layer other than **glue code** that communicates to the next circle inwards.

Layered Principle(10)



```
import com.eventstore.dbclient.EventStoreDBClient;
import com.eventstore.dbclient.EventStoreDBClientSettings;
import com.eventstore.dbclient.EventStoreDBConnectionString;

public class EsdbSingleClientPool implements EsdbClientPool {  9 usages   ezkanban

    private final EventStoreDBClient client;  2 usages

    public EsdbSingleClientPool(String url) {  5 usages   ezkanban
        EventStoreDBClientSettings settings = EventStoreDBConnectionString.parseOrThrow(url);
        this.client = EventStoreDBClient.create(settings);
    }

    @Override   ezkanban
    public EventStoreDBClient getClient() { return client; }
```

Dependency Rule(1)

Source code dependencies must point only inward, toward higher-level policies. **CA Link**

The Dependency Inversion Principle

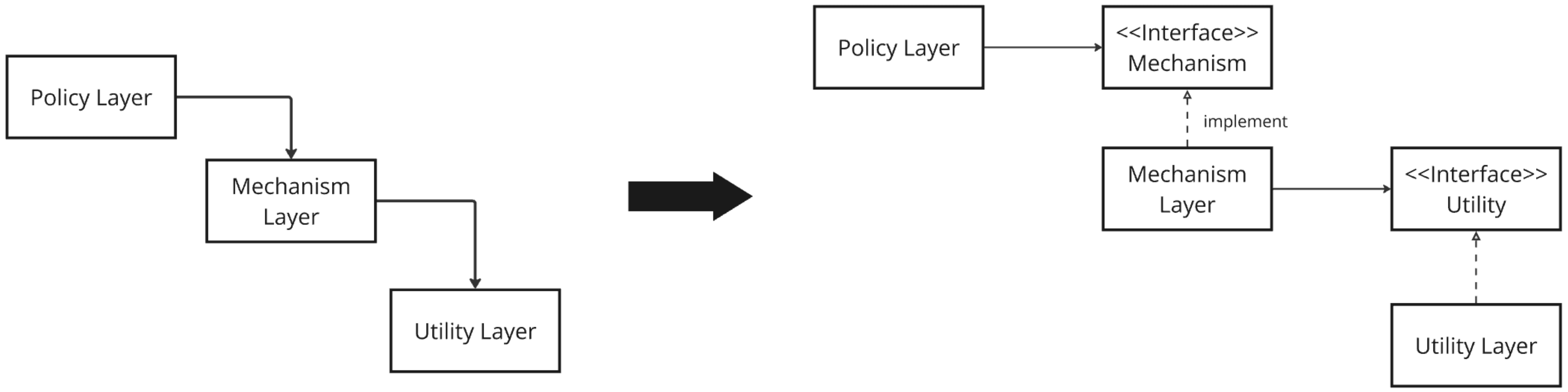
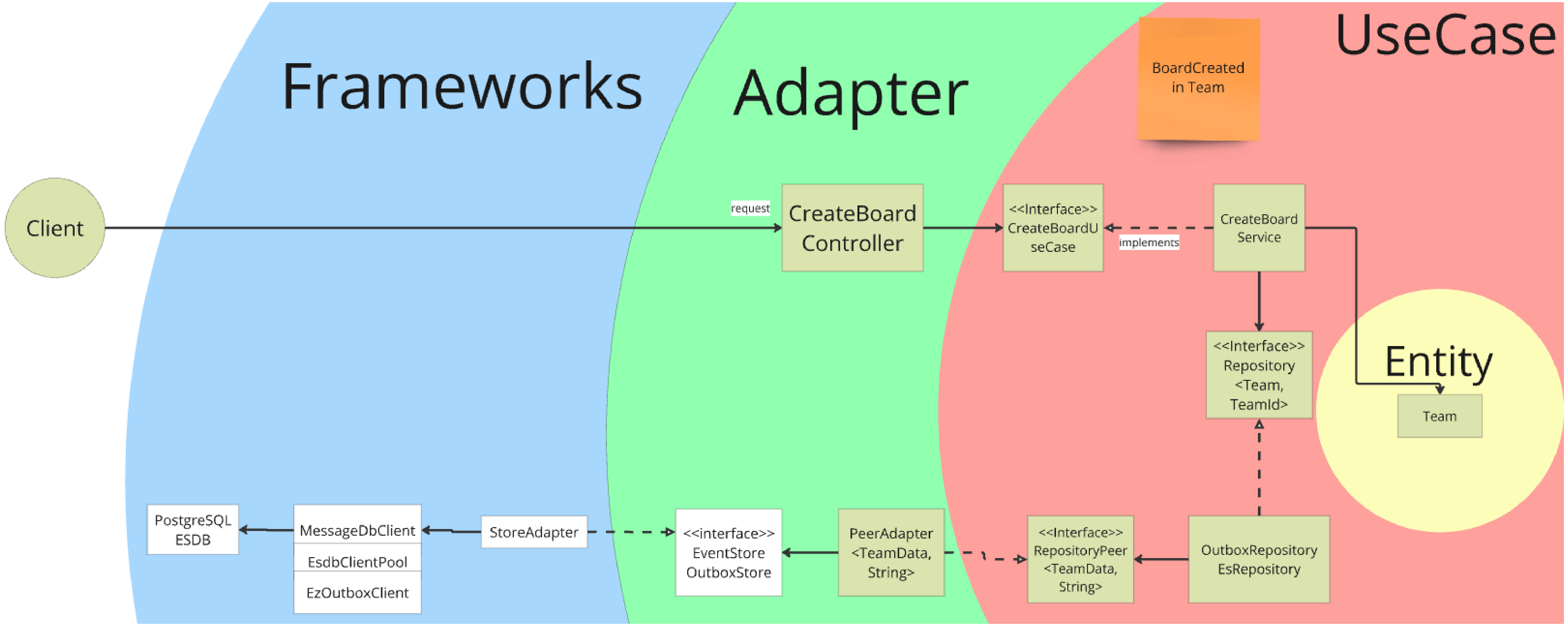


Figure 3: Simple Layers(Source: Robert C. Martin, "The Dependency Inversion Principle," C++ Report, May 1996, p. 7)
Figure 4: Abstract Layers(Source: Robert C. Martin, "The Dependency Inversion Principle," C++ Report, May 1996, p. 7)

Dependency Rule(2)

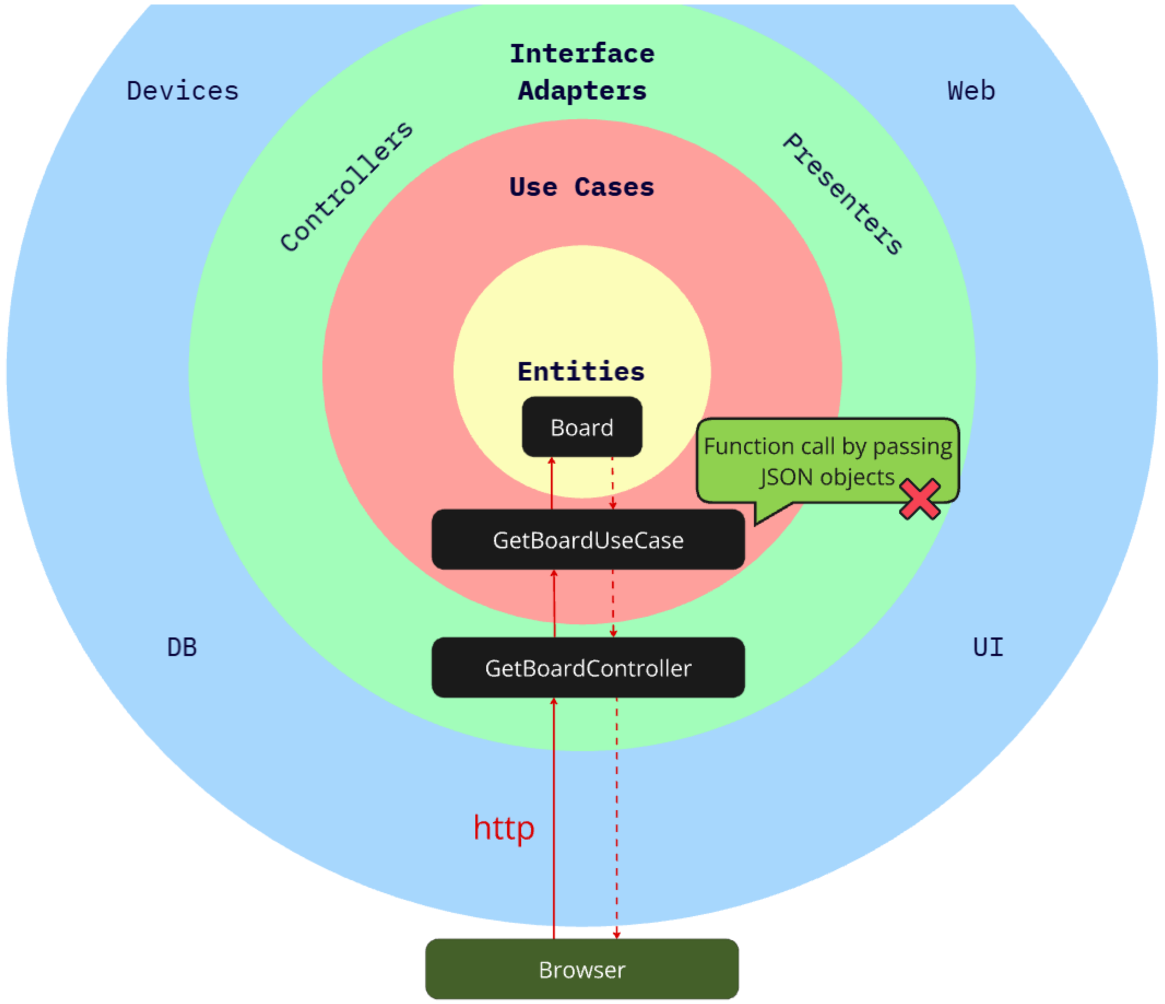
Example



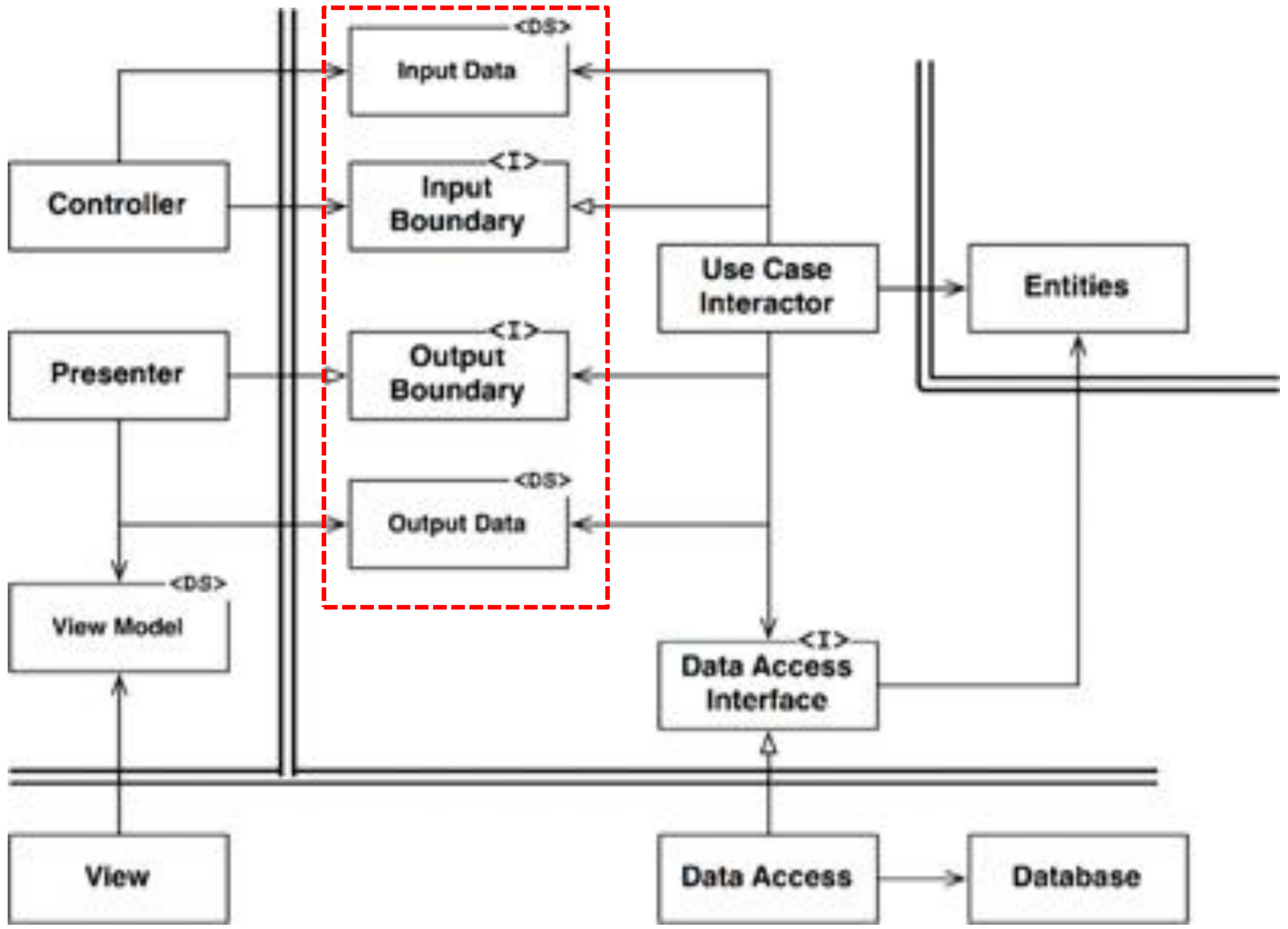
Frameworks
Adapter
UseCase
BoardCreated in Team
Client
request
CreateBoard Controller
<<Interface>> CreateBoardUseCase
implements
CreateBoard Service
<<Interface>> Repository <Team, TeamId>
Entity
Team
PostgreSQL ESDB
MessageDbClient
EsdbClientPool
EzOutboxClient
StoreAdapter
<<interface>> EventStore OutboxStore
PeerAdapter <TeamData, String>
<<Interface>> RepositoryPeer <TeamData, String>
OutboxRepository EsRepository

Cross-Boundary Rule(1)

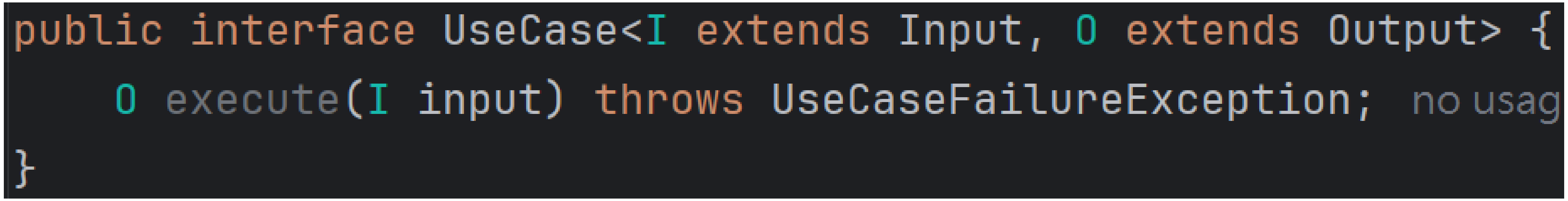
What should be done when objects need to cross boundaries?



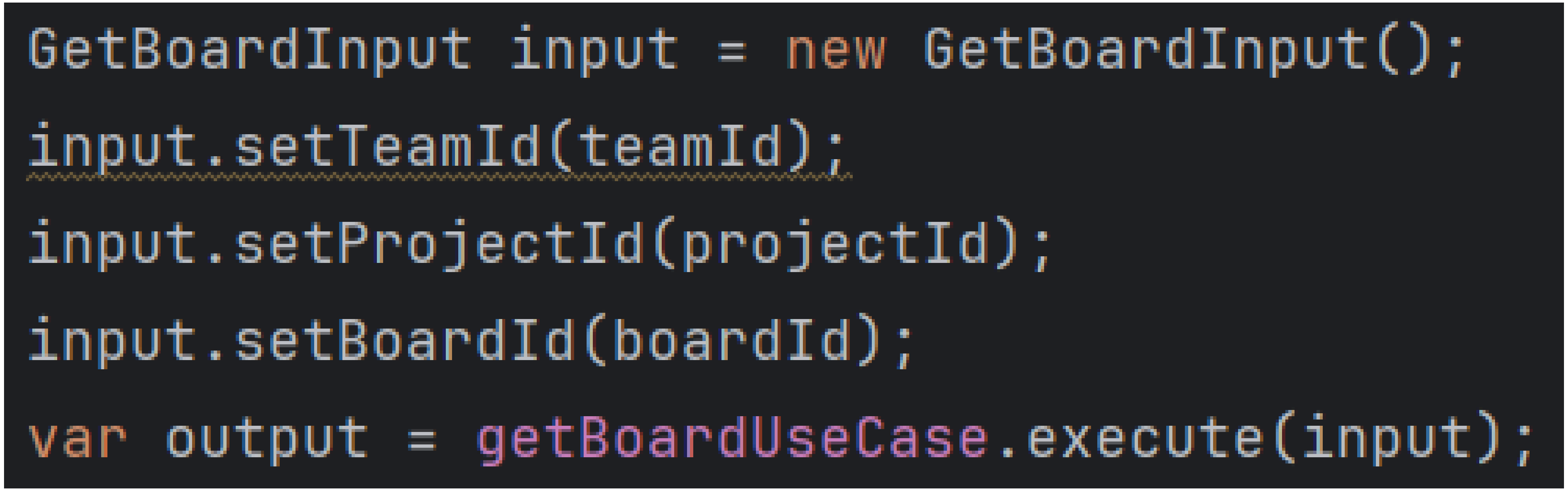
Cross-Boundary Principle(2)



Cross-Boundary Principle(3)

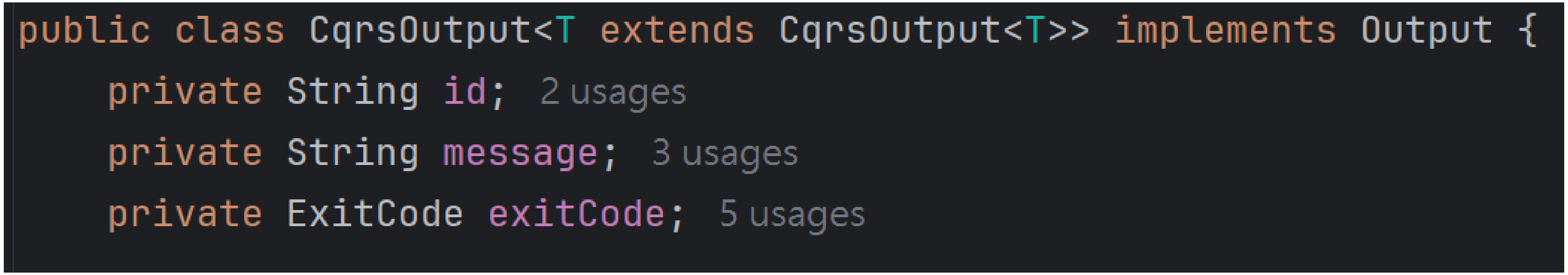


```
public interface UseCase<I extends Input, O extends Output> {
    O execute(I input) throws UseCaseFailureException;  no usag
}
```



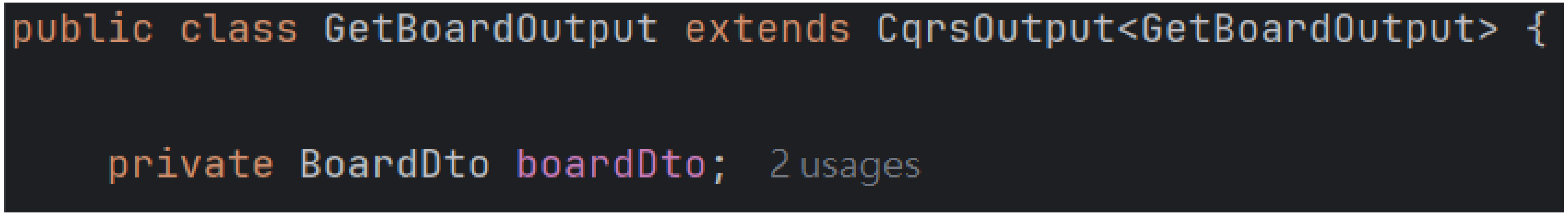
```
GetBoardInput input = new GetBoardInput();
input.setTeamId(teamId);
input.setProjectId(projectId);
input.setBoardId(boardId);
var output = getBoardUseCase.execute(input);
```

Cross-Boundary Rule(4)

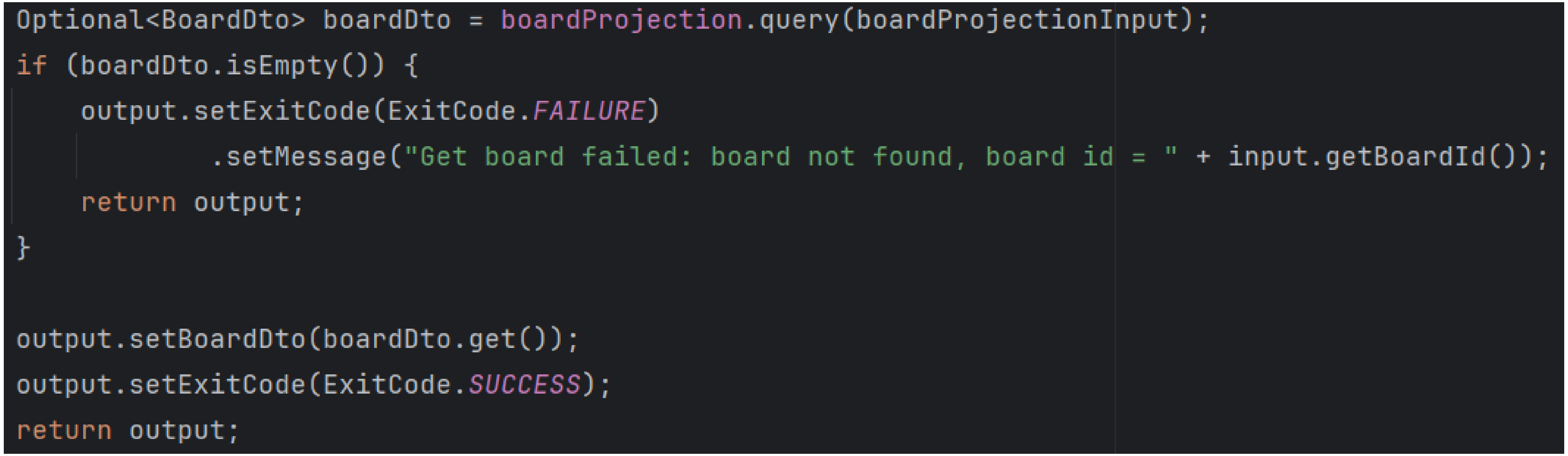


```
public class CqrsOutput<T extends CqrsOutput<T>> implements Output {
    private String id;  2 usages
    private String message;  3 usages
    private ExitCode exitCode;  5 usages
```

Cross-Boundary Rule(5)



```
public class GetBoardOutput extends CqrsOutput<GetBoardOutput> {

    private BoardDto boardDto;  2 usages
```



```
Optional<BoardDto> boardDto = boardProjection.query(boardProjectionInput);
if (boardDto.isEmpty()) {
    output.setExitCode(ExitCode.FAILURE)
            .setMessage("Get board failed: board not found, board id = " + input.getBoardId());
    return output;
}

output.setBoardDto(boardDto.get());
output.setExitCode(ExitCode.SUCCESS);
return output;
```

Pros and Cons(1)

Pros	Cons
Maintainable	Complicated(not easy for rookie)
Align with the **SRP**(four layer)	Not always necessary(CRUD App)
Testable	Performance Overhead(frequently transform data format)

Reference

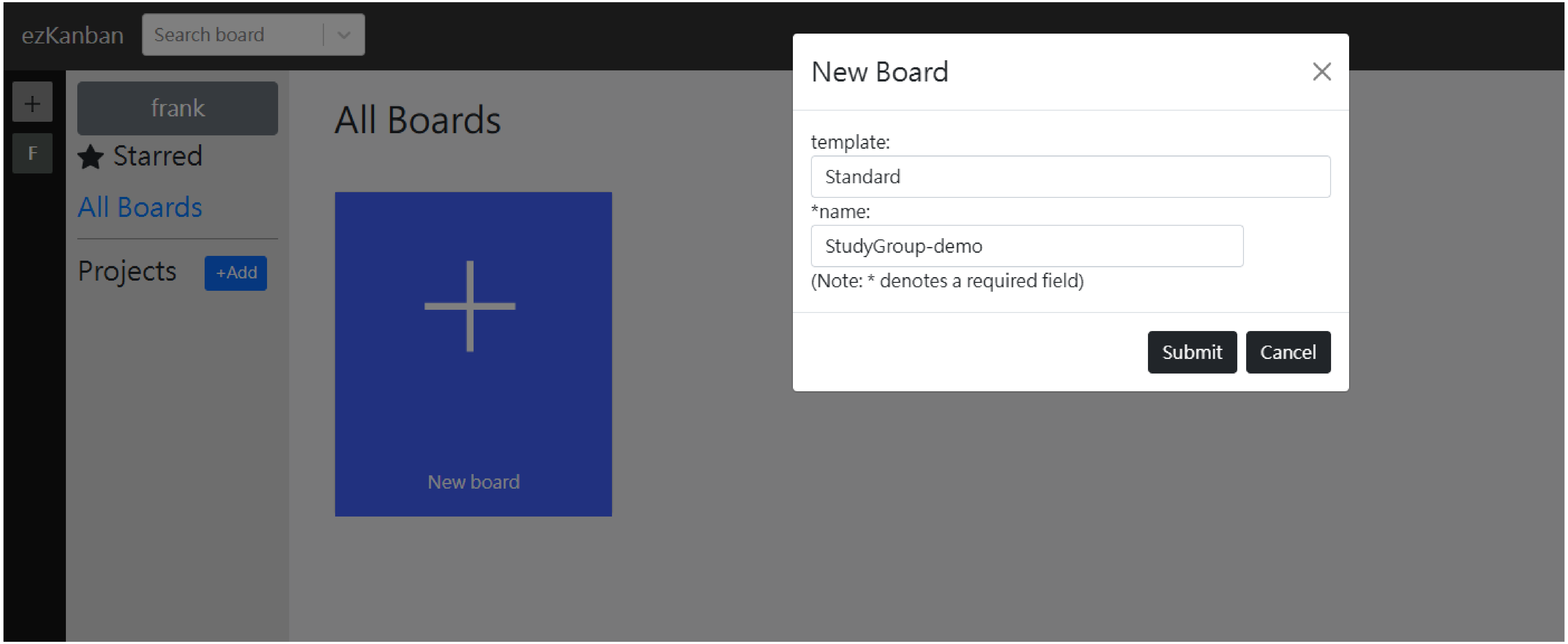
- Layered Architecture Is Good-Grzegorz Ziemoński
- Hexagonal Architecture Is Powerful-Grzegorz Ziemoński
- The Clean Code Blog-Robert C. Martin (Uncle Bob)
- Clean Architecture – Make Your Architecture Scream
- Clean Architecture三原則-搞笑談軟工Teddy
- Martin, Robert C. "The Dependency Inversion Principle," C++ Report, May 1996, p. 7.
- Martin, Robert C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017.

Thanks for listening

QA

CreateBoardInTeam Example

Back



All Boards

New Board

template:
Standard

*name:
StudyGroup-demo

(Note: * denotes a required field)

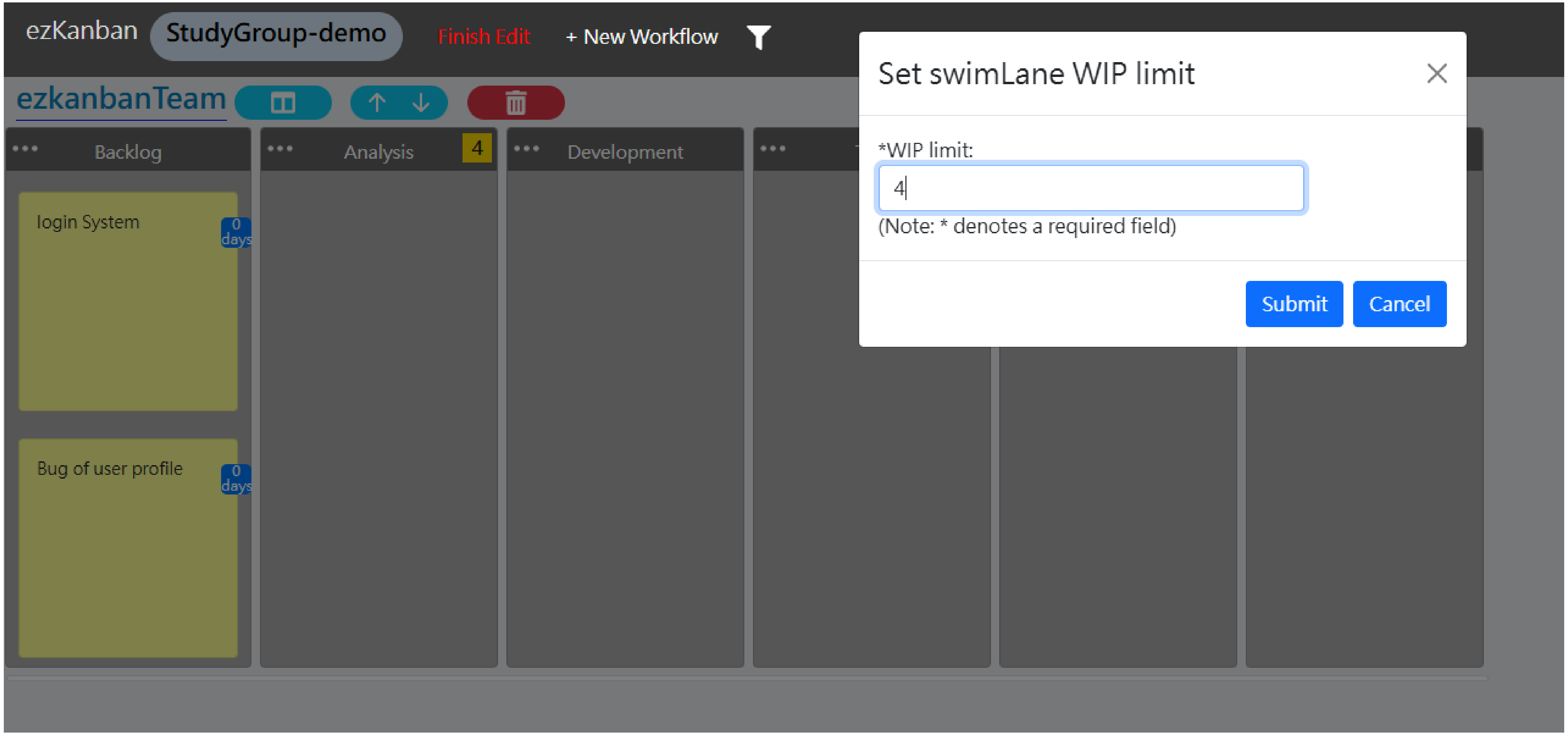
Submit Cancel

SOFTWARE SYSTEMS LAB 軟體系統實驗室 NTUT

SetWipLimit Example

BackToDomainModel

BackToUseCase



Set swimLane WIP limit

*WIP limit:

4

(Note: * denotes a required field)

Submit Cancel