クリーンアーキテクチャー　コードを事例に

概要

こちらを参考にクリーンアーキテクチャを紐解く

domains

参考のコードは以下の通り。(言語は TS)

```
import moment from "moment-timezone";
import { ID } from "../type";

export class Todo {
  private _id: ID;
  private _title: string;
  private _description: string;
  private _createdAt: moment.Moment;
  private _updatedAt: moment.Moment;

  get id(): ID {
    return this._id;
  }

  set id(id: ID) {
    this._id = id;
  }

  get title(): string {
    return this._title;
  }

  set title(title: string) {
    this._title = title;
  }

  get description(): string {
    return this._description;
  }

  set description(description: string) {
    this._description = description;
  }

  // ビジネスルールを格納
  isTitleFilled(): boolean {
    return this._title.length > 0;
  }

  isDescriptionFilled(): boolean {
```

```
    return this._description.length > 0;
  }

  // 省略 日付周りの定義も実施しています

  // idや時刻を生成しない
  // ドメインとしては「タスク」を生成するだけなので、システムで必要なidなどはentityで実施しない
  constructor(title: string, description: string) {
    this._title = title;
    this._description = description;
  }
}
```

- domeinsはビズネスルールのカプセル化した層
- ここではtodoのidやtitleなどを定義して、値を得たり代入したりルールの格納といった処理をクラスの中に定義している

applications

ここでは CRUD(Create,Read,Update,Delete の頭文字)を定義している。以下例は Create のみ。

```
import uuid4 from "uuid4";
import moment from "moment-timezone";
import { Todo } from "../../../domain/Todo";
import { ITodoRepository } from "./ITodoRespository";

export class CreateTodo {
  private taskRepository: ITodoRepository;

  constructor(taskRepository: ITodoRepository) {
    this.taskRepository = taskRepository;
  }

  execute(title: string, description: string) {
    const task = new Todo(title, description);

    if (!task.isTitleFilled() || !task.isDescriptionFilled()) {
      throw new Error("ビジネスルールを破っているためエラー");
    }

    // アプリケーション要件的な要素はインスタンス化で設定せず、setterで設定
    task.id = uuid4();
    task.createdAt = moment();
    task.updatedAt = moment();

    return this.taskRepository.create(task);
  }
}
```

- ここでは`domains`を利用してタスクを生成している
- ここで全体`id`や`createdAt`などをここで生成
- 生成するのは`domains`で可能
- 保存処理はどのように実現するのか？・・・`getways`が担当
- `import { ITodoRepository } from "./ITodoRespository";`を呼び出しているがこちらのコードは以下の通り

```
import { Todo } from "../../../domain/Todo";
import { ID } from "../../../type";

// usecase層と interface層(gateways(repository))を繋げる抽象インターフェース
export interface ITodoRepository {
  findAll(): Promise<Array<Todo>>;
  find(id: ID): Promise<Todo | null>;
  create(todo: Todo): Promise<Todo>;
  update(todo: Todo): Promise<Todo>;
  delete(id: ID): Promise<null>;
}
```

- `domains`から`Todo`の型を import している
- ここでは`findAll`など`todo`を取得したり操作したりする関数をこちらで定義している
- このコードの置き場はプロジェクトによって異なる
- `Repository`に関しては`ITodoRepository`に依存していれば何にでも取り替えることができ、DB の取り替えが簡単になる

getways

- ここではインメモリで todo を管理している(リロードしたら消えるやつ)

```
import { Todo } from "../../../domain/Todo";

export let inMemoryTodo: Todo[] = [];
```

- 上記`inMemoryTodo`の配列に todo を保存していく

```
import { ITodoRepository } from
"../../../application/usecases/Todo/ITodoRespository";
import { Todo } from "../../../domain/Todo";
import { ID } from "../../../type";
import { inMemoryTodo } from "./InMemoryTodo";

export class TodoRepository implements ITodoRepository {
  private inmemoryTodo: Todo[];
  constructor() {
    // @ts-ignore
    inMemoryTodo = [
      new Todo("todo01", "インメモリtodo01"),
```

```
      new Todo("todo02", "インメモリtodo02"),
    ];
    this.inmemoryTodo = inMemoryTodo;
  }

  async create(todo: Todo): Promise<Todo> {
    new Promise(() => setTimeout(() => {}, 1000));
    this.inmemoryTodo.push(todo);
    return todo;
  }

  // そのほかのメソッドは省略
}
```

- ここでは以下の処理を実装している
  - domainから Todo のインスタンスを利用してインメモリにデータを初期化している
  - 作成、削除などの関数を作成している
- sqlでITodoRepository.tsに依存した Repository を作ればすぐに取り替えが可能

presenters

- web や DB などの外部から input を内部で扱いやすくする
- usecaseが生成したデータが外部が扱いやすいように変換する

```
import { Todo } from "../../domain/Todo";
import { ID } from "../../type";

interface TodoResponse {
  id: ID;
  title: string;
  description: string;
}

export interface ITodoOutputSerializer {
  todo(todo: Todo): TodoResponse;
  todos(todo: Todo[]): TodoResponse[];
}
```

```
import { Todo } from "../../domain/Todo";
import { ITodoOutputSerializer } from "./ITodoSerializer";

const serialize = (todo: Todo) => {
  return {
    id: todo.id,
    title: todo.title,
    description: todo.description,
  };
};
```

```
export class TodoSerializer implements ITodoOutputSerializer {
  todo(todo: Todo) {
    return serialize(todo);
  }
  todos(todo: Todo[]) {
    return todo.map((mTodo) => serialize(mTodo));
  }
}
```

- 上記コードは Todo リストの output を変換するコード
- 最初のコードで定義したinterfaceを 2 つ目のコードのクラスで継承している

controllers

- usecaseやrepositoryなどを利用し input/output の処理を実装

```
import { TodoRepository } from "../gateways/memory/TodoRepository";
import { TodoSerializer } from "../presenters/TodoSerializer";
import { GetTodo } from "../../application/usecases/Todo/GetTodo";
import { GetTodoList } from "../../application/usecases/Todo/GetTodoList";
import { CreateTodo } from "../../application/usecases/Todo/CreateTodo";
import { UpdateTodo } from "../../application/usecases/Todo/UpdateTodo";
import { DeleteTodo } from "../../application/usecases/Todo/DeleteTodo";
import express from "express";

type Request = {
  req: express.Request;
};

export class TodoController {
  private todoSerializer: TodoSerializer;
  private todoRepository: TodoRepository;

  constructor() {
    this.todoRepository = new TodoRepository();
    this.todoSerializer = new TodoSerializer();
  }

  async create({ req }: Request) {
    const { title, description } = req.body;
    const usecase = new CreateTodo(this.todoRepository);
    const result = await usecase.execute(title, description);

    return this.todoSerializer.todo(result);
  }
  // その他のメソッドは省略
}
```

infrastucture

- フレームワークや DB の詳細を格納

```
import express = require("express");
import { TodoController } from "../interfaces/controller/TodoController";

const todoController = new TodoController();
const router = express.Router();
router.post("/todo", async (req: express.Request, res: express.Response)
=> {
  const result = await todoController.create({ req });
  res.send(result);
});

// その他のエンドポイントは省略

export default router;
```

- ここでは`express`を使っているようだ