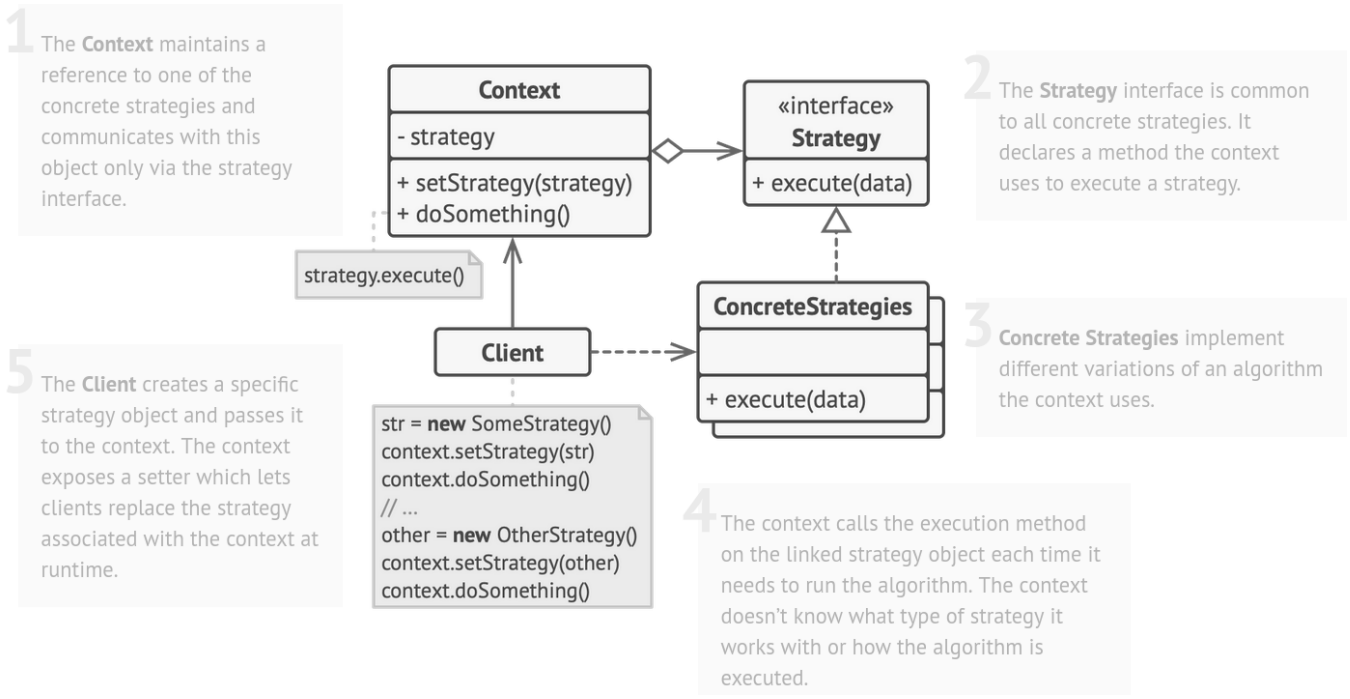
Design patterns

Pour plus de détails sur les patrons de conceptions, voir Introduction à la conception.

Repository Pattern

Le repository est une d'application du patron Strategie pour le stockage de données.



Dans le cadre du stockage des données, l'interface Strategy va contenir les méthodes pour créer, lire, modifier et supprimer les éléments du stockage.

Les ConcreteStrategies implémente l'interface Strategy et vont effectuer ces opérations CRUD sur différents supports (bases de données, fichiers json, fichiers XML, etc).

Enfin le Context va représenter l'objet qui doit être écrit, ce peut être un DTO (comme on va voir plus tard), ou encore une entité du domaine.

Exemple

- Interface des repositories

```
public interface UserRepository {
    User get(Long id);
    void add(User user);
    void update(User user);
    void remove(User user);
}
```

- Implémentation du Repository

```
public class PostgresUserRepository implements UserRepository {
    @Override
    public User get(Long id) {
        // récupération dans la base de donnée
    }

    @Override
    public void add(User user) {
        // ajout dans la base de donnée
    }

    // ...
}
```

- Enfin l'objet lui même

```
public class User {
    private Long id;
    private String userName;
    private String firstName;
    private String email;

    // getters and setters
}
```

DAO vs Repositoy

DAO et Repository sont deux patrons qui sont souvent confondus, cependant il y a une différence.

Les DAO sont plus proche du stockage physique des données, souvent étant un mapping direct de tables par exemple. Les repository sont une abstraction plus haute, et peuvent elle même utiliser

des dao pour fonctionner.

Par exemple si chaque User connait la liste de ses messages, on peut avoir un MessageDao et un UserDao (pour accéder aux deux différentes tables) et on peut utiliser un repository pour lier les deux :

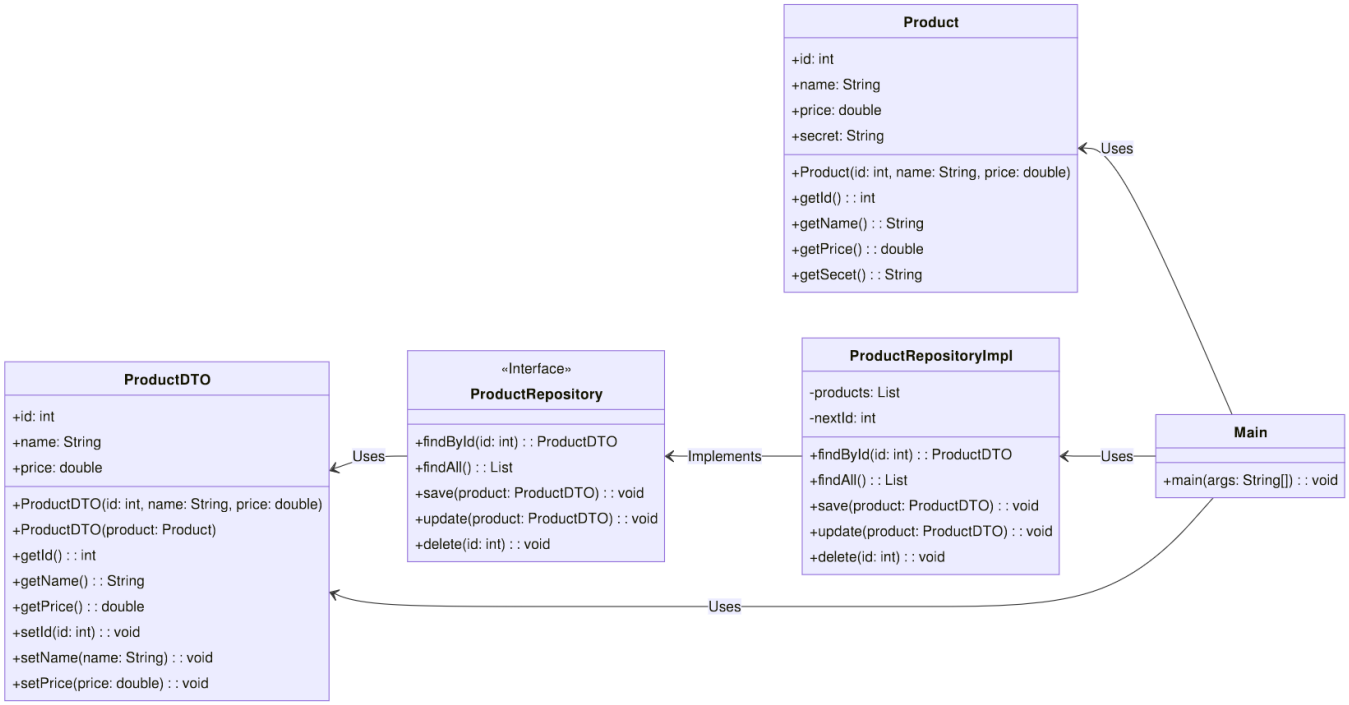
```
public class PostgresUserRepository implements UserRepository {
    private UserDaoImpl userDaoImpl;
    private TweetDaoImpl tweetDaoImpl;

    @Override
    public User get(Long id) {
        UserSocialMedia user = (UserSocialMedia) userDaoImpl.read(id);

        List<Tweet> tweets = tweetDaoImpl.fetchTweets(user.getEmail());
        user.setTweets(tweets);

        return user;
    }
    // ...
}
```

DTO Pattern

Un DTO (Data Transfer Object) est un objet permettant de transférer des informations entre différents éléments du programme.

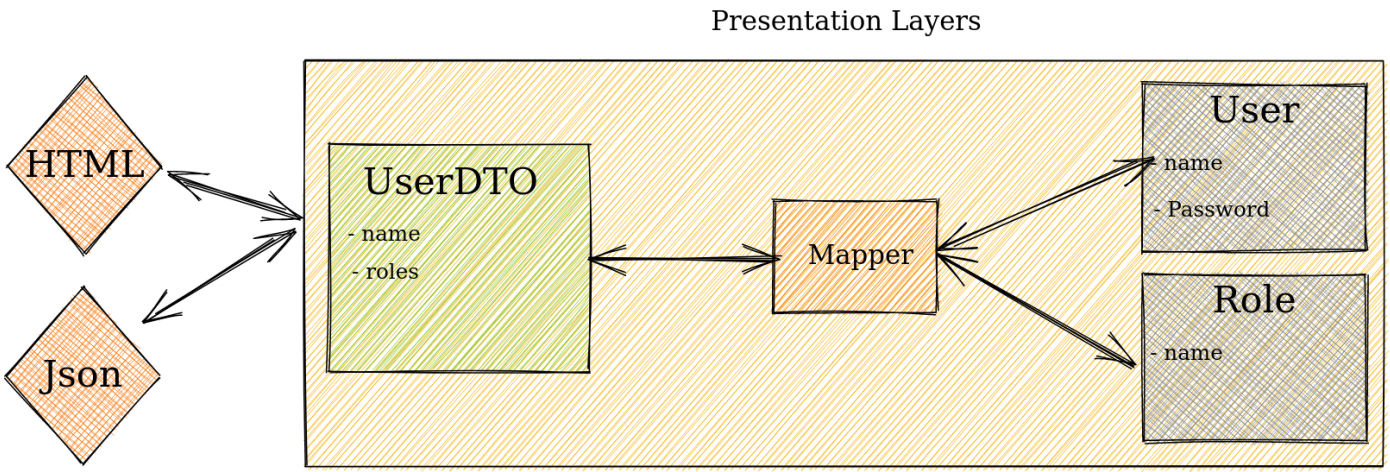
Par exemple entre la base de donnée et le repository ou encore entre le présentateur et la vue.

Voci un exemple où le DTO est un objet de transfère entre la base de donnée et le repository :



Ici on veut que le repository n'ia pas accès au secret, on peut donc créer un productDTO qui n'a pas accès à ce secret, si l'inverse aurait été vrai, le ProductDTO et le Product aurait dû être inversé

Mais on peut aussi utiliser un DTO pour transférer les données entre le présentateur et la vue.



Par exemple ici, on veut que la vue conaisse le nom et le role mais pas le mot de passe, alors on crée un UserDTO qui va contenir uniquement les informations nécessaires.

Revision #1
Created 4 October 2023 07:59:46 by SnowCode
Updated 4 October 2023 08:03:12 by SnowCode