Социальная сеть

Простой вариант

Ваша задача состоит в том, чтобы создать классы для системы управления социальной сетью, используя принципы ООП и миксины. Иерархия классов должна включать в себя следующие классы:

- `User` (пользователь)
- `Post` (пост)
- `SocialMedia` (социальная сеть)

Для каждого класса реализуйте необходимые методы и свойства. Например, для класса `User` можно реализовать методы `create_post()` и `like_post()`, а для класса `SocialMedia` можно реализовать методы `add_user()` и `get_most_liked_posts()`.

Кроме того, необходимо реализовать миксины для расширения функциональности классов. Например, миксин `Likable` может добавить функциональность лайков к постам, а миксин `Commentable` может добавить функциональность комментирования к постам.

Решение:

Иерархия классов может быть реализована следующим образом:

```
from collections import Counter


class Likable:
    def __init__(self):
        self.likes = 0

    def like(self):
        self.likes += 1


class Commentable:
    def __init__(self):
        self.comments = []

    def comment(self, text):
        self.comments.append(text)


class User:
    def __init__(self, name):
        self.name = name
        self.posts = []

    def create_post(self, text):
        post = Post(text, self)
        self.posts.append(post)
        return post

    def like_post(self, post):
        post.like()


class Post(Likable, Commentable):
    def __init__(self, text, author):
        Likable.__init__(self)
        Commentable.__init__(self)
        self.text = text
```

```
        self.author = author

    def __str__(self):
        return f'{self.text} by {self.author.name} - {self.likes} likes'


class SocialMedia:
    def __init__(self):
        self.users = []
        self.posts = []

    def add_user(self, user):
        self.users.append(user)

    def get_most_liked_posts(self, n=5):
        likes_counter = Counter(post.likes for post in self.posts)
        most_liked_likes = likes_counter.most_common(n)
        most_liked_posts = [
            post for post in self.posts
            if post.likes in [likes for likes, _ in most_liked_likes]
        ]
        return most_liked_posts

    def __str__(self):
        posts_str = '\n'.join(str(post) for post in self.posts)
        return f'Social media posts:\n{posts_str}'
```

Пример работы кода:

```
# Создание социальной сети и пользователей
social_media = SocialMedia()
user1 = User('John')
user2 = User('Jane')
social_media.add_user(user1)
social_media.add_user(user2)

# Создание постов
post1 = user1.create_post('Hello, world!')
post2 = user1.create_post('I love Python!')
post3 = user2.create_post('Hello, everyone!')
post4 = user2.create_post('I love Java!')
social_media.posts.extend([post1, post2, post3, post4])

# Лайки постов
user1.like_post(post1)
user1.like_post(post2)
user1.like_post(post2)
user2.like_post(post3)
user2.like_post(post4)

# Комментарии к постам
post1.comment('Nice post!')
post2.comment('Python is great!')
post3.comment('Welcome to the community!')
post4.comment('Java is also good!')

# Вывод информации о постах
print(social_media)
# Social media posts:
# Hello, world! by John - 1 likes
# I love Python! by John - 2 likes
# Hello, everyone! by Jane - 1 likes
# I love Java! by Jane - 1 likes

# Получение самых популярных постов
```

```
most_liked_posts = social_media.get_most_liked_posts()
print(most_liked_posts)
# [<__main__.Post object at 0x000000000000000>, <__main__.Post object at 0x0000000000000000>]
```

В данном решении использованы принципы ООП для моделирования пользователей, постов и социальной сети. Кроме того, используются миксины `Likable` и `Commentable` для расширения функциональности класса `Post`. Миксины позволяют избежать дублирования кода и упрощают структуру классов.

В примере работы кода демонстрируется использование созданных классов и методов для решения практических задач, связанных с управлением социальной сетью. Примеры показывают, как можно использовать принципы ООП и миксины для моделирования реальных объектов и процессов, а также для упрощения и структурирования кода.

Расширенный вариант

Ваша задача состоит в том, чтобы создать классы для системы управления социальной сетью, используя принципы ООП и миксины. Иерархия классов должна включать в себя следующие классы:

- `User` (пользователь)
- `Post` (пост)
- `Comment` (комментарий)
- `Group` (группа)
- `Event` (событие)
- `SocialMedia` (социальная сеть)

Для каждого класса реализуйте необходимые методы и свойства. Например, для класса `User` можно реализовать методы `create_post()`, `like_post()`, `join_group()`, `create_event()`, а для класса `SocialMedia` можно реализовать методы `add_user()`, `get_most_liked_posts()`, `get_most_popular_groups()`, `get_upcoming_events()`.

Кроме того, необходимо реализовать миксины для расширения функциональности классов. Например, миксин `Likable` может добавить функциональность лайков к постам, а миксин `Commentable` может добавить функциональность комментирования к постам. Миксин `Joinable` может добавить функциональность вступления в группы, а миксин `Schedulable` может добавить функциональность создания и управления событиями.

Решение:

Иерархия классов может быть реализована следующим образом:

```
from collections import Counter
from datetime import datetime, timedelta

class Likable:
    def __init__(self):
        self.likes = 0

    def like(self):
        self.likes += 1

class Commentable:
    def __init__(self):
        self.comments = []

    def comment(self, text, author):
        comment = Comment(text, author)
        self.comments.append(comment)
        return comment

class Joinable:
    def __init__(self):
        self.members = []
```

```
    def join(self, member):
        self.members.append(member)

class Schedulable:
    def __init__(self, start_time, end_time):
        self.start_time = start_time
        self.end_time = end_time

    def is_upcoming(self):
        now = datetime.now()
        return self.start_time > now and self.start_time < now + timedelta(days=7)

class User:
    def __init__(self, name):
        self.name = name
        self.posts = []
        self.groups = []
        self.events = []

    def create_post(self, text):
        post = Post(text, self)
        self.posts.append(post)
        return post

    def like_post(self, post):
        post.like()

    def join_group(self, group):
        group.join(self)
        self.groups.append(group)

    def create_event(self, name, start_time, end_time):
        event = Event(name, start_time, end_time, self)
        self.events.append(event)
        return event

class Post(Likable, Commentable):
    def __init__(self, text, author):
        Likable.__init__(self)
        Commentable.__init__(self)
        self.text = text
        self.author = author

    def __str__(self):
        return f'{self.text} by {self.author.name} - {self.likes} likes'

class Comment:
    def __init__(self, text, author):
        self.text = text
        self.author = author

    def __str__(self):
        return f'{self.text} by {self.author.name}'

class Group(Joinable):
    def __init__(self, name):
        super().__init__()
        self.name = name

    def __str__(self):
        return f'{self.name} - {len(self.members)} members'

class Event(Schedulable):
    def __init__(self, name, start_time, end_time, author):
        super().__init__(start_time, end_time)
        self.name = name
```

```
        self.author = author

    def __str__(self):
        return f'{self.name} by {self.author.name} - {self.start_time} - {self.end_time}'

class SocialMedia:
    def __init__(self):
        self.users = []
        self.posts = []
        self.groups = []
        self.events = []

    def add_user(self, user):
        self.users.append(user)

    def get_most_liked_posts(self, n=5):
        likes_counter = Counter(post.likes for post in self.posts)
        most_liked_likes = likes_counter.most_common(n)
        most_liked_posts = [
            post for post in self.posts
            if post.likes in [likes for likes, _ in most_liked_likes]
        ]
        return most_liked_posts

    def get_most_popular_groups(self, n=5):
        members_counter = Counter(len(group.members) for group in self.groups)
        most_popular_members = members_counter.most_common(n)
        most_popular_groups = [
            group for group in self.groups
            if len(group.members) in [members for members, _ in most_popular_members]
        ]
        return most_popular_groups

    def get_upcoming_events(self):
        return [event for event in self.events if event.is_upcoming()]

    def __str__(self):
        posts_str = '\n'.join(str(post) for post in self.posts)
        groups_str = '\n'.join(str(group) for group in self.groups)
        events_str = '\n'.join(str(event) for event in self.events)
        return f'Social media posts:\n{posts_str}\nGroups:\n{groups_str}\nEvents:\n{events_str}'
```

Пример работы кода:

```
# Создание социальной сети и пользователей
social_media = SocialMedia()
user1 = User('John')
user2 = User('Jane')
user3 = User('Alice')
social_media.add_user(user1)
social_media.add_user(user2)
social_media.add_user(user3)

# Создание постов
post1 = user1.create_post('Hello, world!')
post2 = user1.create_post('I love Python!')
post3 = user2.create_post('Hello, everyone!')
post4 = user2.create_post('I love Java!')
social_media.posts.extend([post1, post2, post3, post4])

# Лайки постов
user1.like_post(post1)
user1.like_post(post2)
user1.like_post(post2)
user2.like_post(post3)
```

```
user2.like_post(post4)
user3.like_post(post1)
user3.like_post(post3)

# Комментарии к постам
post1.comment('Nice post!', user2)
post2.comment('Python is great!', user3)
post3.comment('Welcome to the community!', user1)
post4.comment('Java is also good!', user3)

# Создание групп
group1 = Group('Python developers')
group2 = Group('Java developers')
group3 = Group('Web developers')
social_media.groups.extend([group1, group2, group3])

# Вступление в группы
user1.join_group(group1)
user1.join_group(group3)
user2.join_group(group2)
user2.join_group(group3)
user3.join_group(group1)
user3.join_group(group2)

# Создание событий
event1 = user1.create_event('Python meetup', datetime(2023, 3, 1, 18, 0), datetime(2023, 3, 1, 20, 0))
event2 = user2.create_event('Java meetup', datetime(2023, 3, 2, 18, 0), datetime(2023, 3, 2, 20, 0))
event3 = user3.create_event('Web meetup', datetime(2023, 3, 3, 18, 0), datetime(2023, 3, 3, 20, 0))
social_media.events.extend([event1, event2, event3])

# Вывод информации о популярных постах, группах и событиях
print(social_media.get_most_liked_posts())
# [<__main__.Post object at 0×0000000000000000>, <__main__.Post object at 0×0000000000000000>]
print(social_media.get_most_popular_groups())
# [<__main__.Web developers object at 0×0000000000000000>, <__main__.Python developers object at
0×0000000000000000>]
print(social_media.get_upcoming_events())
# [<__main__.Python meetup object at 0×0000000000000000>, <__main__.Java meetup object at
0×0000000000000000>, <__main__.Web meetup object at 0×0000000000000000>]
```

В данном решении использованы принципы ООП для моделирования пользователей, постов, комментариев, групп, событий и социальной сети. Кроме того, используются миксины `Likable`, `Commentable`, `Joinable` и `Schedulable` для расширения функциональности классов. Миксины позволяют избежать дублирования кода и упрощают структуру классов.

В примере работы кода демонстрируется использование созданных классов и методов для решения практических задач, связанных с управлением социальной сетью. Примеры показывают, как можно использовать принципы ООП и миксины для моделирования реальных объектов и процессов, а также для упрощения и структурирования кода.