

Exercise: Class and Static Methods

Problems for exercise and homework for the [Python OOP Course @SoftUni](#).

Submit your solutions in the SoftUni judge system at <https://judge.softuni.org/Contests/2431>.

1. Photo Album

Create a class called **PhotoAlbum**. Upon initialization, it should receive **pages (int)**. It should also have **one more attribute: photos** (empty matrix) representing the album with its pages where you should put the photos. Each page can contain only **4 photos**. The class should also have **3 more methods**:

- **from_photos_count(photos_count: int)** - creates a **new instance** of **PhotoAlbum**. Note: Each page can contain **4 photos**.
- **add_photo(label:str)** - adds the photo in the **first possible page** and **slot** and return **"{label} photo added successfully on page {page_number(starting from 1)} slot {slot_number(starting from 1)}"**. If there are **no free slots** left, return **"No more free slots"**
- **display()** - returns a **string representation** of **each page** and the **photos** in it. Each photo is marked with **"[]"**, and the **page separation** is made using **11 dashes (-)**. For example, if we have **1 page** and **2 photos**:

```
-----  
[] []  
-----
```

and if we have **2 empty pages**:

```
-----  
  
-----  
  
-----
```

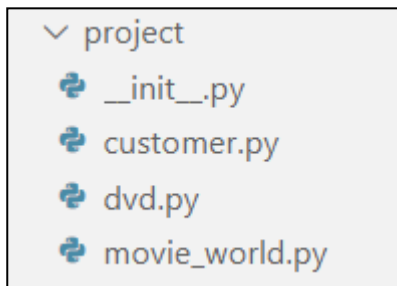
Examples

Test Code
<pre>album = PhotoAlbum(2) print(album.add_photo("baby")) print(album.add_photo("first grade")) print(album.add_photo("eight grade")) print(album.add_photo("party with friends")) print(album.photos) print(album.add_photo("prom")) print(album.add_photo("wedding")) print(album.display())</pre>
Output
<pre>baby photo added successfully on page 1 slot 1 first grade photo added successfully on page 1 slot 2 eight grade photo added successfully on page 1 slot 3 party with friends photo added successfully on page 1 slot 4 [['baby', 'first grade', 'eight grade', 'party with friends'], []] prom photo added successfully on page 2 slot 1 wedding photo added successfully on page 2 slot 2</pre>

```
-----  
[] [] [] []  
-----  
[] []  
-----
```

2. Movie World

Create the following project structure



Class Customer

Upon initialization, the **Customer** class should receive the following parameters: **name: str, age: int, id: int**. Each customer should also have an instance **attribute** called **rented_dvds** (empty list with **DVD instances**).

Implement the **__repr__** method, so it **returns** the following string: **"{id}: {name} of age {age} has {count_rented_dvds} rented DVD's ({dvd_names joined by comma and space})"**

Class DVD

Upon initialization, the **DVD class** should receive the following parameters: **name: str, id: int, creation_year: int, creation_month: str, age_restriction: int**. Each DVD should also have an **attribute** called **is_rented** (**False** by default)

Create a method called **from_date(id: int, name: str, date: str, age_restriction: int)** - it should create a **new instance** using the provided data. The **date** will be in the format **"day.month.year"** - **all of them should be numbers**.

Implement the **__repr__** method so it returns the following string: **"{id}: {name} ({creation_month} {creation_year}) has age restriction {age_restriction}. Status: {rented/not rented}"**

Class MovieWorld

The **MovieWorld** class should receive **one parameter** upon initialization: **name: str**. Each **MovieWorld** instance should also have **2 more attributes**: **customers** (empty list of **Customer objects**), **dvds** (empty list of **DVD objects**). The class should also have the following **methods**:

- **dvd_capacity()** - returns **15** - the **DVD capacity** of a movie world
- **customer_capacity()** - returns **10** - the **customer capacity** of a movie world
- **add_customer(customer: Customer)** - add the customer if capacity not exceeded
- **add_dvd(dvd: DVD)** - add the DVD if capacity not exceeded
- **rent_dvd(customer_id: int, dvd_id: int)**
 - If the customer has **already rented** that DVD return **"{customer_name} has already rented {dvd_name}"**

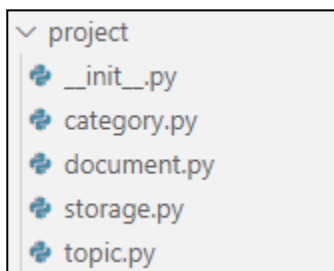
- If the DVD is **rented by someone else**, return **"DVD is already rented"**
- If the customer is **not allowed** to rent the DVD, return **"{customer_name} should be at least {dvd_age_restriction} to rent this movie"**
- Otherwise, the rent is **successful** (the DVD is rented and added to the customer's DVDs). Return **"{customer_name} has successfully rented {dvd_name}"**
- **return_dvd(customer_id, dvd_id)** - if the DVD is in the customer, he/she should **return** it and the method should return the message **"{customer_name} has successfully returned {dvd_name}"**. Otherwise, return **"{customer_name} does not have that DVD"**
- **__repr__()** - return the **string representation** of each customer and each DVD on separate lines

Examples

Test Code
<pre> from project.customer import Customer from project.dvd import DVD from project.movie_world import MovieWorld c1 = Customer("John", 16, 1) c2 = Customer("Anna", 55, 2) d1 = DVD("Black Widow", 1, 2020, "April", 18) d2 = DVD.from_date(2, "The Croods 2", "23.12.2020", 3) movie_world = MovieWorld("The Best Movie Shop") movie_world.add_customer(c1) movie_world.add_customer(c2) movie_world.add_dvd(d1) movie_world.add_dvd(d2) print(movie_world.rent_dvd(1, 1)) print(movie_world.rent_dvd(2, 1)) print(movie_world.rent_dvd(1, 2)) print(movie_world) </pre>
Output
<pre> John should be at least 18 to rent this movie Anna has successfully rented Black Widow John has successfully rented The Croods 2 1: John of age 16 has 1 rented DVD's (The Croods 2) 2: Anna of age 55 has 1 rented DVD's (Black Widow) 1: Black Widow (April 2020) has age restriction 18. Status: rented 2: The Croods 2 (December 2020) has age restriction 3. Status: rented </pre>

3. Document Management

Create the following project structure



Class Topic

The **Topic** class should receive the following **parameters** upon initialization: **id: int, topic: str, storage_folder: str**. It should have **two methods**:

- **edit(new_topic: str, new_storage_folder: str)** - change the **topic** and the **storage folder**
- **__repr__()** - returns a **string representation** of the topic in the format: **"Topic {id}: {topic} in {storage_folder}"**

Class Category

The **Category** class should receive the following **parameters** upon initialization: **id: int, name: str**. The class should have **two methods**:

- **edit(new_name: str)** - edit the **name** of the category
- **__repr__()** - returns a **string representation** of the category in the following format: **"Category {id}: {name}"**

Class Document

The **Document** class should receive the following **parameters** upon initialization: **id: int, category_id: int, topic_id: int, file_name: str**. The class should also have **one more attribute** called **tags** (empty list). The class should also have **4 methods**:

- **from_instances(id: int, category: Category, topic: Topic, file_name: str)** - create a **new instance** using the provided **category** and **topic** instances
- **add_tag(tag_content: str)** - if the **tag** is **not** already in the tags list, **add** it to the tags list
- **remove_tag(tag_content: str)** - if the tag is **in** the tags list, **delete** it
- **edit(file_name: str)** - **change** the **file name** with the given one
- **__repr__()** - returns a string representation of a document in the format: **"Document {id}: {file_name}; category {category_id}, topic {topic_id}, tags: {tags joined by comma and space}"**

Class Storage

Upon initialization, the class **Storage** will **not receive any parameters**. It should have **3 instance attributes**: **categories** (empty list), **topics** (empty list), **documents** (empty list). The class should have the following **methods**:

- **add_category(category: Category)** - add the category if it **is not in the list**
- **add_topic(topic: Topic)** - add the topic if it **does not exist**
- **add_document(document: Document)** - add the document if it **does not exist**

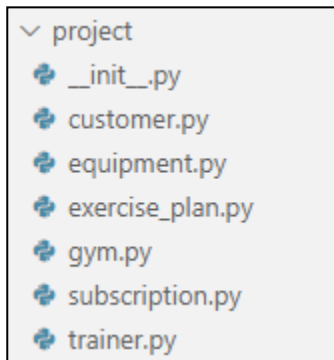
- `edit_category(category_id: int, new_name:str)` - edit the **name** of the category with the provided **id**
- `edit_topic(topic_id: int, new_topic: str, new_storage_folder: str)` - edit the **topic** with the given **id**
- `edit_document(document_id: int, new_file_name: str)` - edit the **document** with the given **id**
- `delete_category(category_id)` - delete the **category** with the provided **id**
- `delete_topic(topic_id)` - delete the **topic** with the provided **id**
- `delete_document(document_id)` - delete the **document** with the provided **id**
- `get_document(document_id)` - return the **document** with the provided **id**
- `__repr__()` - returns a **string representation** of each document on **separate lines**

Examples

Test Code
<pre> from project.category import Category from project.document import Document from project.storage import Storage from project.topic import Topic c1 = Category(1, "work") t1 = Topic(1, "daily tasks", "C:\\\\work_documents") d1 = Document(1, 1, 1, "finilize project") d1.add_tag("urgent") d1.add_tag("work") storage = Storage() storage.add_category(c1) storage.add_topic(t1) storage.add_document(d1) print(c1) print(t1) print(storage.get_document(1)) print(storage) </pre>
Output
<pre> Category 1: work Topic 1: daily tasks in C:\\work_documents Document 1: finilize project; category 1, topic 1, tags: urgent, work Document 1: finilize project; category 1, topic 1, tags: urgent, work </pre>

4. Gym

Create the following project structure:



Class Customer

Upon initialization, each customer will receive the following **parameters**: **name: str, address: str, email: str**. Each customer should also have a personal **id (autoincremented)**, starting from 1). To do the incrementation, you should create a **class attribute id** equal to 1, which will keep the value of the **id for the upcoming customer**. For example, if there are **no customers**, the class **id** should be equal to 1. When there is **one customer** - the class **id** should be equal to 2.

Create a method called **get_next_id**, which returns the **id** that will be given to the **next customer**.

Implement the **__repr__** method so it returns the **info** about the customer in the following format: **"Customer <{id}> {name}; Address: {address}; Email: {email}"**

Class Equipment

Upon initialization, the class will receive the following **parameters**: **name: str**. Each equipment should also have an **id (autoincremented)**, starting from 1). To do the incrementation, you should create a **class attribute id** equal to 1, which will keep the value of the **id for the following equipment's id**.

Create a **method** called **get_next_id**, which returns the **id** that will be given to the **following equipment**.

Implement the **__repr__** method so it returns the **info** about the equipment in the following format: **"Equipment <{id}> {name}"**

Create a **static method** called **get_next_id**, which returns the **id** that will be given to the **following equipment**.

Class ExercisePlan

Upon initialization, the class will receive the following **parameters**: **trainer_id: int, equipment_id: int, duration: int** (in minutes). Each plan should also have an **id (autoincremented)**, starting from 1). To do the incrementation, you should create a **class attribute id** equal to 1, which will keep the value of the **id for the next plan's id**. Create the following **methods**:

- **from_hours(trainer_id:int, equipment_id:int, hours:int)** - creates **new instance** using the provided information
- **get_next_id()** - **static method** that returns the **id** that will be given to the **next plan**
- **__repr__()** - returns the **information** about the plan in the following format: **"Plan <{id}> with duration {duration} minutes"**

Class Subscription

Upon initialization, the class will receive the following parameters: **date: str, customer_id: int, trainer_id: int, exercise_id: int**. The class should also have an **id** (autoincremented starting from 1). To do the incrementation, you should create a **class attribute id** equal to 1, which will keep the value of the **id** for the next subscription's id.

Implement the **__repr__** method so it returns the **info** about the subscription in the following format: "Subscription <{id}> on {date}"

Create a **static method** called **get_next_id** which returns the **id** that will be given to the **next subscription**

Class Trainer

Upon initialization, the class will receive the following parameters: **name: str**. The class should also have an **id** (autoincremented starting from 1). To do the incrementation, you should create a **class attribute id** equal to 1, which will keep the value of the **id** for the next trainer's id.

Implement the **__repr__** method so it returns the **info** about the trainer in the following format: "Trainer <{id}> {name}"

Create a **static method** called **get_next_id**, which returns the **id** that will be given to the **next trainer**.

Class Gym

Upon initialization, the class will **not receive** any parameters. However, it should have the following attributes: **customers** (empty list of customer objects), **trainers** (empty list of trainer objects), **equipment** (empty list of equipment objects), **plans** (empty list of plan objects), **subscriptions** (empty list of subscription objects)

Create the following methods:

- **add_customer(customer: Customer)** - add the customer in the customer list if the customer is **not** already in it
- **add_trainer(trainer: Trainer)** - add the trainer to the trainers' list, if the trainer is **not** already in it
- **add_equipment(equipment: Equipment)** - add the equipment to the equipment list, if the equipment is **not** already in it
- **add_plan(plan: ExercisePlan)** - add the plan to the plans' list, if the plan is **not** already in it
- **add_subscription(subscription: Subscription)** - add the subscription in the subscriptions list if the subscription is **not** already in it
- **subscription_info(subscription_id: int)** - get the **subscription**, the **customer**, the **trainer**, the **equipment**, and the **plan**. Then return their **string representations** each on a **new line**.

Examples

Test Code

```
from project.customer import Customer
from project.equipment import Equipment
from project.exercise_plan import ExercisePlan
from project.gym import Gym
from project.subscription import Subscription
from project.trainer import Trainer

customer = Customer("John", "Maple Street", "john.smith@gmail.com")
equipment = Equipment("Treadmill")
trainer = Trainer("Peter")
```

```

subscription = Subscription("14.05.2020", 1, 1, 1)
plan = ExercisePlan(1, 1, 20)

gym = Gym()

gym.add_customer(customer)
gym.add_equipment(equipment)
gym.add_trainer(trainer)
gym.add_plan(plan)
gym.add_subscription(subscription)

print(Customer.get_next_id())

print(gym.subscription_info(1))

```

Output

```

2
Subscription <1> on 14.05.2020
Customer <1> John; Address: Maple Street; Email: john.smith@gmail.com
Trainer <1> Peter
Equipment <1> Treadmill
Plan <1> with duration 20 minutes

```