# Exercise: Class and Static Methods

Please, submit your source code solutions for the described problems to the [Judge System](https://alpha.judge.softuni.org/Contests/Static-and-Class-Methods-Exercise/2431).

**Ask** **your** **questions** here: [https://www.slido.com](https://www.slido.com/) by entering the course code **#python-advanced**

## 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** |
| 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()) |
| **Output** |
| 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  -----------  [] [] [] []  -----------  [] []  ----------- |

## 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 is not exceeded
* **add\_dvd(dvd: DVD)** - add the DVD if capacity is 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** |
| 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) |
| **Output** |
| 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 |

## Document Management [Solve with AI]

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), and **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** |
| 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) |
| **Output** |
| 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 |

## Gym [Solve with AI]

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** to 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 |