# Lab: Models Inheritance and Customization

This document defines the problems for the in-class lab for the [Python ORM course @ Software University](https://softuni.bg/trainings/4253/python-orm-october-2023).

Submit your solutions in the SoftUni [Judge system](https://judge.softuni.org/Contests/4329/Models-Inheritance-and-Customization-Lab).

For this lab, you are given an empty **ORM project skeleton** (you can download it from [here](https://softuni.bg/downloads/svn/python-db/trunk/Sep-2023/Python-ORM/05-Working-with-Queries-in-Django/05-lab-orm-skeleton.zip)). Your task for today is to **create a Zoo Management System** with **animals**, **employees**, and an animal **display system** for the visitors.

## Zoo Animals

In the **main\_app** create a model called **"Animal"** with the following fields:

* **name**
  + A **character** field.
  + It has a **maximum length** of **100** characters.
  + Represents the name of the animal.
* **species**
  + A **character** field.
  + It has a **maximum length** of **100** characters.
  + Represents the species of the animal.
* **birth\_date**
  + A **date** field.
  + Represents the date of birth of the animal.
* **sound**
  + A **character** field.
  + It has a **maximum length** of **100** characters.
  + Represents the sound that the animal makes.

Then, **create 3 more models: "Mammal"**, **"Bird"**, and **"Reptile"** - all of them are **types of animals**:

* The mammal has **an additional character field called "fur\_color"** with a **maximum length of 50** chars.
* The bird has **an additional decimal field called "wing\_span"** that can **store up to 5 digits** andhasexactly **2 decimal places**.
* The reptile has **an additional character field called "scale\_type"** with a **maximum length of 50** chars.

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| from main\_app.models import Animal, Mammal, Bird, Reptile  Animal.objects.create(name="Nemo", species="Clownfish", birth\_date="2019-04-10", sound="Bubbles")  Mammal.objects.create(name="Fluffy", species="Orangutan", birth\_date="2018-02-10", sound="Chomps", fur\_color="Reddish-brown")  Bird.objects.create(name="Robby", species="American Robin", birth\_date="2021-03-20", sound="Chirp", wing\_span=28.50)  Reptile.objects.create(name="Python", species="Ball Python", birth\_date="2019-07-01", sound="Hiss", scale\_type="Smooth")  animals = Animal.objects.all()  for a in animals:  print(f"{a.name} is {a.species}.") |
| **Output** |
| Nemo is Clownfish.  Fluffy is Orangutan.  Robby is American Robin.  Python is Ball Python. |

## Zoo Employees

In the **main\_app** create an **additional** **model** called **"Employee"**. It is a **base class** for any **type of employee in the zoo,** and it **is NOT meant to create a database table on its own**. The model has the following fields:

* **first\_name**
  + A **character** field.
  + It has a **maximum length** of **50** characters.
  + Represents the first name of the employee.
* **last\_name**
  + A **character** field.
  + It has a **maximum length** of **50** characters.
  + Represents the last name of the employee.
* **phone\_number**
  + A **character** field.
  + It has a **maximum length** of **10** characters.
  + Represents the phone number of the employee.

Then, **create 2 more models: "ZooKeeper"** and **"Veterinarian"** - they are **types of employees**:

* **ZooKeeper** includes **2 additional fields**:
  + **specialty**
    - A **character** field.
    - It has a **maximum length** of **10** characters.
    - Holds **predefined choices** of each specialization: **"Mammals"**, **"Birds"**, **"Reptiles"**, and **"Others"**.
    - Represents the area of expertise of a zookeeper.
  + **managed\_animals**
    - A **many-to-many** field.
    - Establishes a **many-to-many relationship** with the **"Animals" model**.
    - It allows each zookeeper to be **connected with one or more animals**, indicating the animals they are responsible for or manage within the zoo.
* **Veterinarian** includes **1 additional field**:
  + **license\_number**
    - A **character** field.
    - It has a **maximum length** of **10** characters.
    - Represents the license number of a veterinarian.

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| from main\_app.models import ZooKeeper, Veterinarian  *# keep the data from the previous exercise, so you can reuse it*  zookeeper = ZooKeeper.objects.create(first\_name="Peter", last\_name="Johnson", phone\_number="0899524265", specialty="Mammals")  mammal = Mammal.objects.get(name="Fluffy")  zookeeper.managed\_animals.add(mammal)  veterinarian = Veterinarian.objects.create(first\_name="Dr. Michael", last\_name="Smith", phone\_number="9876543210", license\_number="VET12345")  zookeeper\_from\_db = ZooKeeper.objects.first()  print(f"{zookeeper\_from\_db.first\_name} {zookeeper\_from\_db.last\_name} is a ZooKeeper.")  veterinarian\_from\_db = Veterinarian.objects.first()  print(f"{veterinarian\_from\_db.first\_name} {veterinarian\_from\_db.last\_name} is a Veterinarian.") |
| **Output** |
| Peter Johnson is a ZooKeeper.  Dr. Michael Smith is a Veterinarian. |

## Animal Display System

In the **main\_app** create one **additional** **model** called **"ZooDisplayAnimal"**. It inherits from the **"Animal"** model but does **NOT** have its **own database table**. Its primary purpose is to **extend** the **"Animal"** model behavior. Currently, it is **NOT** **needed to** **add additional logic** to the model.

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| from main\_app.models import ZooDisplayAnimal  is\_proxy = ZooDisplayAnimal.\_meta.proxy  if is\_proxy:  print("ZooDisplayAnimal is a proxy model.")  else:  print("ZooDisplayAnimal is not a proxy model.") |
| **Output** |
| ZooDisplayAnimal is a proxy model. |

## Zookeeper's Specialty

In the **"ZooKeeper"** **model add a custom validation logic** before each zookeeper object is saved. Create a **validation** to ensure that the object **is checked against the given list of valid choices (**"**SPECIALITIES**"**). If the specialty is not a valid choice,** a **ValidationError** should be raised with the message: **"Specialty must be a valid choice."**.

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| from main\_app.models import ZooKeeper  zookeeper = ZooKeeper(first\_name="John", last\_name="Doe", phone\_number="0123456789", specialty="Fishes")  zookeeper.full\_clean()  zookeeper.save() |
| **Output** |
| django.core.exceptions.ValidationError: {'specialty': ["Value 'Fishes' is not a valid choice."], '\_\_all\_\_': ['Specialty must be a valid choice.']} |

## Animal Display System Logic

It is time to add logic to the **"ZooDisplayAnimal"** model. This logic is designed to create a customized view of animal data exclusively for visitors, allowing them to access animal information without making any changes to the original **"Animal"** model. Your task is to **implement two custom methods**:

* **display\_info**
  + It returns information for the animal in the following format: **"Meet {name}! It's {species} and it's born {birth\_date}. It makes a noise like '{sound}'!{extra\_info}"**
  + The extra\_info is an **additional string** **added to the string above**, depending on the animal's type:
    - For **mammals** it is **" Its fur color is {fur\_color}."**
    - For **birds** it is **" Its wingspan is {wing\_span} cm."**
    - For **reptiles** it is **" Its scale type is {scale\_type}."**
    - If **none** of theabove types are given, the **string should be ignored**.
* **is\_endangered**
  + The method returns information if the population of that species is at significant risk of becoming extinct in the near future.
  + If the **species** of the animal is **"Cross River Gorilla"**, **"Orangutan"**, or **"Green Turtle"** the method should **return** **True**. Otherwise, it should **return** **False**.

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| from main\_app.models import ZooDisplayAnimal  *# keep the data from the previous exercises, so you can reuse it*  all\_animals\_info = ZooDisplayAnimal.objects.all()  for a in all\_animals\_info:  print(a.display\_info())  print(a.is\_endangered()) |
| **Output** |
| Meet Nemo! It's Clownfish and it's born 2019-04-10. It makes a noise like 'Bubbles'!  False  Meet Fluffy! It's Orangutan and it's born 2018-02-10. It makes a noise like 'Chomps'! Its fur color is Reddish-brown.  True  Meet Robby! It's American Robin and it's born 2021-03-20. It makes a noise like 'Chirp'! Its wingspan is 28.50 cm.  False  Meet Python! It's Ball Python and it's born 2019-07-01. It makes a noise like 'Hiss'! Its scale type is Smooth.  False |

## Animal's Age

In the **"Animal"** model implement **one** **property** that **calculates** **and returns the age** of an animal based on its birth date. The age is dynamically calculated each time ensuring that it remains accurate over time.

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| from datetime import date, timedelta  from main\_app.models import Mammal, Reptile  lion\_birth\_date = date.today() - timedelta(days=731)  lion = Mammal.objects.create(name="Simba", species="Lion", birth\_date=lion\_birth\_date, sound="Roar", fur\_color="Golden")  print(f"The lion's age is {lion.age}.")  snake\_birth\_date = date.today() - timedelta(days=30)  snake = Reptile.objects.create(name="Kaa", species="Python", birth\_date=snake\_birth\_date, sound="Hiss", scale\_type="Scales")  print(f"The snake's age is {snake.age}.") |
| **Output** |
| The lion's age is 2.  The snake's age is 0. |

## Veterinarian Availability

In the **"Veterinarian"** model implement a new **field** called **"availability" with a custom model field type called "BooleanChoiceField"** to represent the availability of a veterinarian. It should **behave** **like a** **Boolean field** but has **custom choices** and a **default value**:

* Define **2 choices**: **"Available"** (**True**) and **"Not Available"** (**False**).
* Set the **default value** for the field to **True**, indicating that a veterinarian is initially considered available.

Next, create an additional **method** called **"is\_available"** to **check the veterinarian availability** based on the field.

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| from main\_app.models import Veterinarian  v1 = Veterinarian.objects.create(first\_name="John", last\_name="Doe", phone\_number="0896625120", license\_number="VET123", availability=False)  print(v1.is\_available())  v2 = Veterinarian.objects.create(first\_name="Alice", last\_name="Johnson", phone\_number="0896529728", license\_number="VET789")  print(v2.is\_available()) |
| **Output** |
| False  True |