# 3 LAYERED ARCHITECTURE

Slides by Björgvin B. Björgvinsson

## 3 LAYERED ARCHITECTURE

- A common approach when building software is to divide the project into layers
- A 3 layered architecture is very common
- The 3 layers are the following
  - Presentation layer (UI layer)
  - Domain layer (Business layer)
  - Data access layer (Persistence layer, Repository layer, Data layer, Data access layer)

## 3 LAYERED ARCHITECTURE - PRESENTATION LAYER

- **Presentation layer**
  - Often called UI layer (UI = user interface)
  - Contains classes that take care of the interaction with the user
  - Contains minimal logic
  - Interacts with classes from the layer below (domain layer)

## 3 LAYERED ARCHITECTURE - DOMAIN LAYER

- **Domain layer**
  - Often called business layer / service layer
  - Contains classes that take care of business logic
    - Takes care of calculations and validation
    - Classes that belong to the domain layer are commonly referred to as services
  - Interacts with classes from the layer below (data access layer)

## 3 LAYERED ARCHITECTURE - DATA ACCESS LAYER

- **Data layer**
  - Often called persistence layer and sometimes repository layer
  - Contains classes that interact with a datasource
    - The data source can be a database, a text or binary file etc.
  - Classes that belong to the data access layer are commonly referred to as repositories

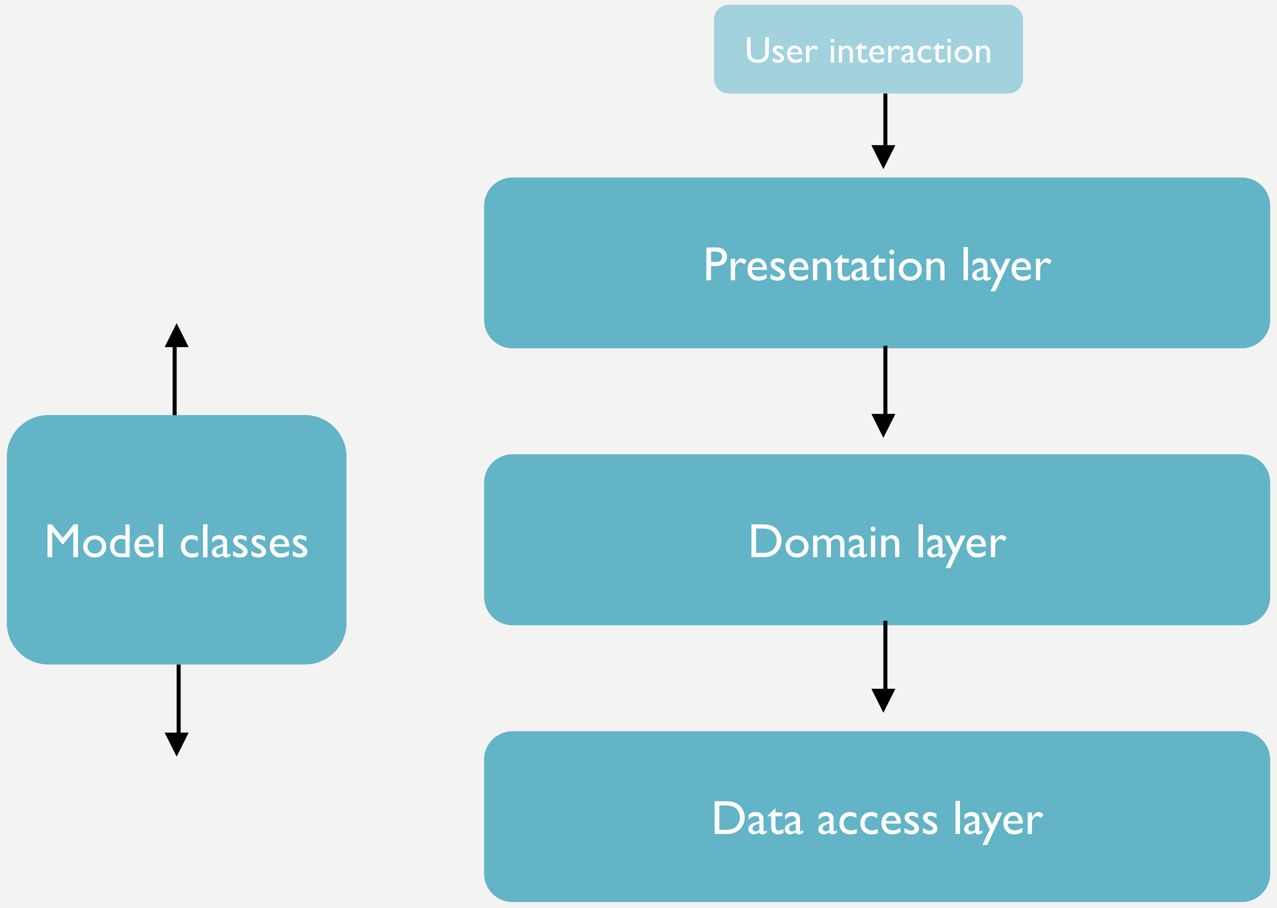
- **Repository meaning**
  - a central location in which data is stored and managed
  - a place where things are or may be stored

## 3 LAYERED ARCHITECTURE - MODELS

- When creating software that follows this 3 layered architecture we will need to have some classes that consist of the actual objects we are working with
  - As an example, when creating a software system for a pizza place we will need a class that represents a pizza
  - If we were to create a software system for a car sale we would need a class to represent a car
- These classes don’t really fall into any of the layers in the 3 layer architecture
- These classes are often called model classes or data classes (POCO classes, plain old class objects)
  - They don’t contain any logic. They hold information about the object they represent
- These classes travel between layers, often as parameters in functions / methods

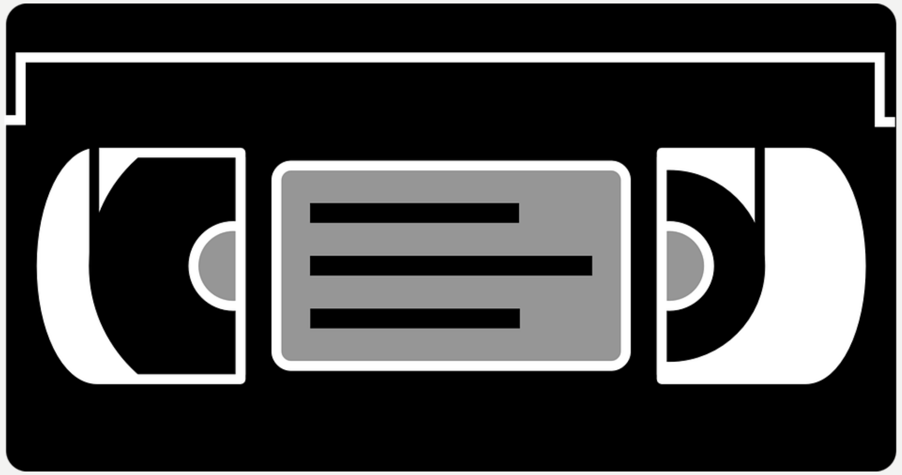
# 3 LAYERED ARCHITECTURE

- This is a visual representation of the 3 layered architecture
- Each layer should only communicate with the layer that is one layer lower



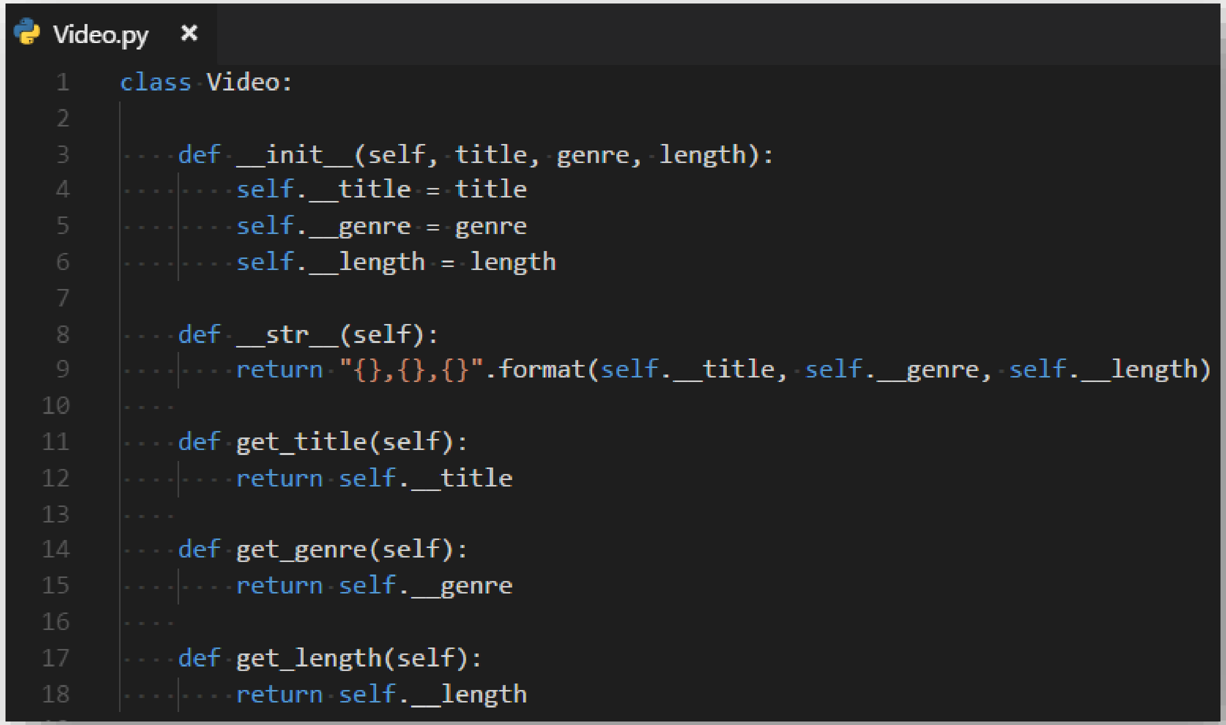
## 3 LAYERED ARCHITECTURE

- The best way to understand this is probably via an example
- Lets say that we are to create an software system for a video rent
- For now it should not be able to do much but it should be able to add a videotape to the data store



# 3 LAYERED ARCHITECTURE

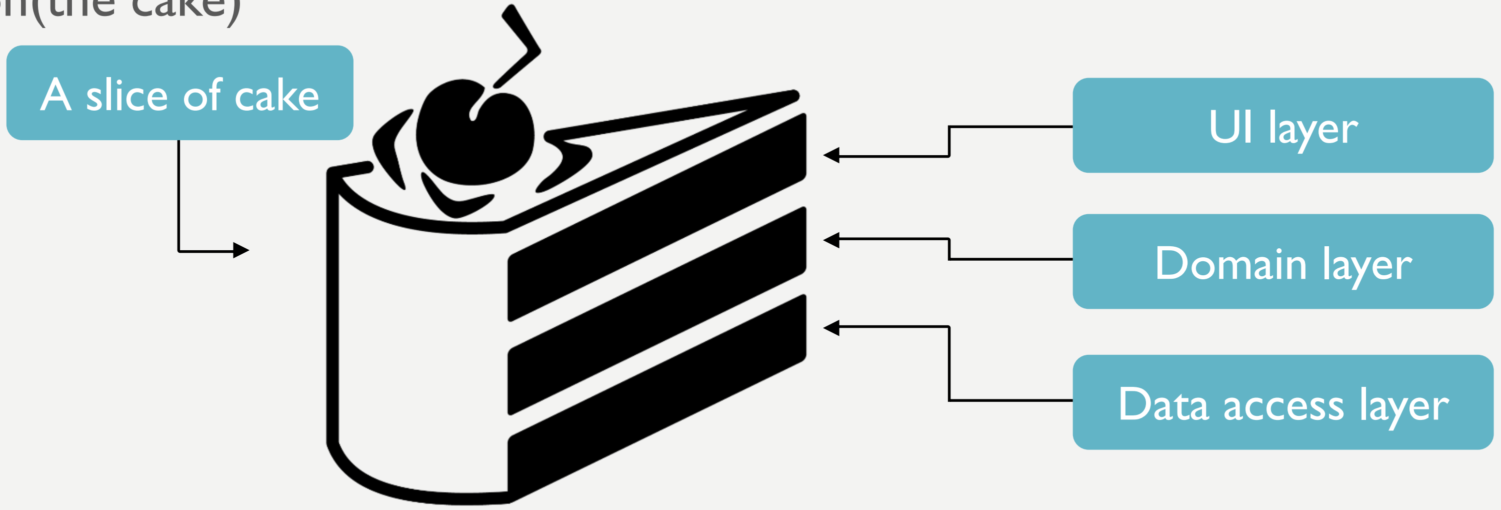
- We will need a class that represents a video!
- Here is a class that represents a single video
- It has a single constructor, get methods and a __str__() method
- This class does not fall into one of the 3 layers.
  - It is a model class(data class)



```python
class Video:

    def __init__(self, title, genre, length):
        self.__title = title
        self.__genre = genre
        self.__length = length

    def __str__(self):
        return "{},{},{}".format(self.__title, self.__genre, self.__length)

    def get_title(self):
        return self.__title

    def get_genre(self):
        return self.__genre

    def get_length(self):
        return self.__length
```

# 3 LAYERED ARCHITECTURE

- Now that we have a data class that we can use, let's create an operation that uses all three layers!
- When implementing an operation from the UI down to the data access layer it is often referred to as **slicing the cake** because we are doing something in each layer of the application(the cake)



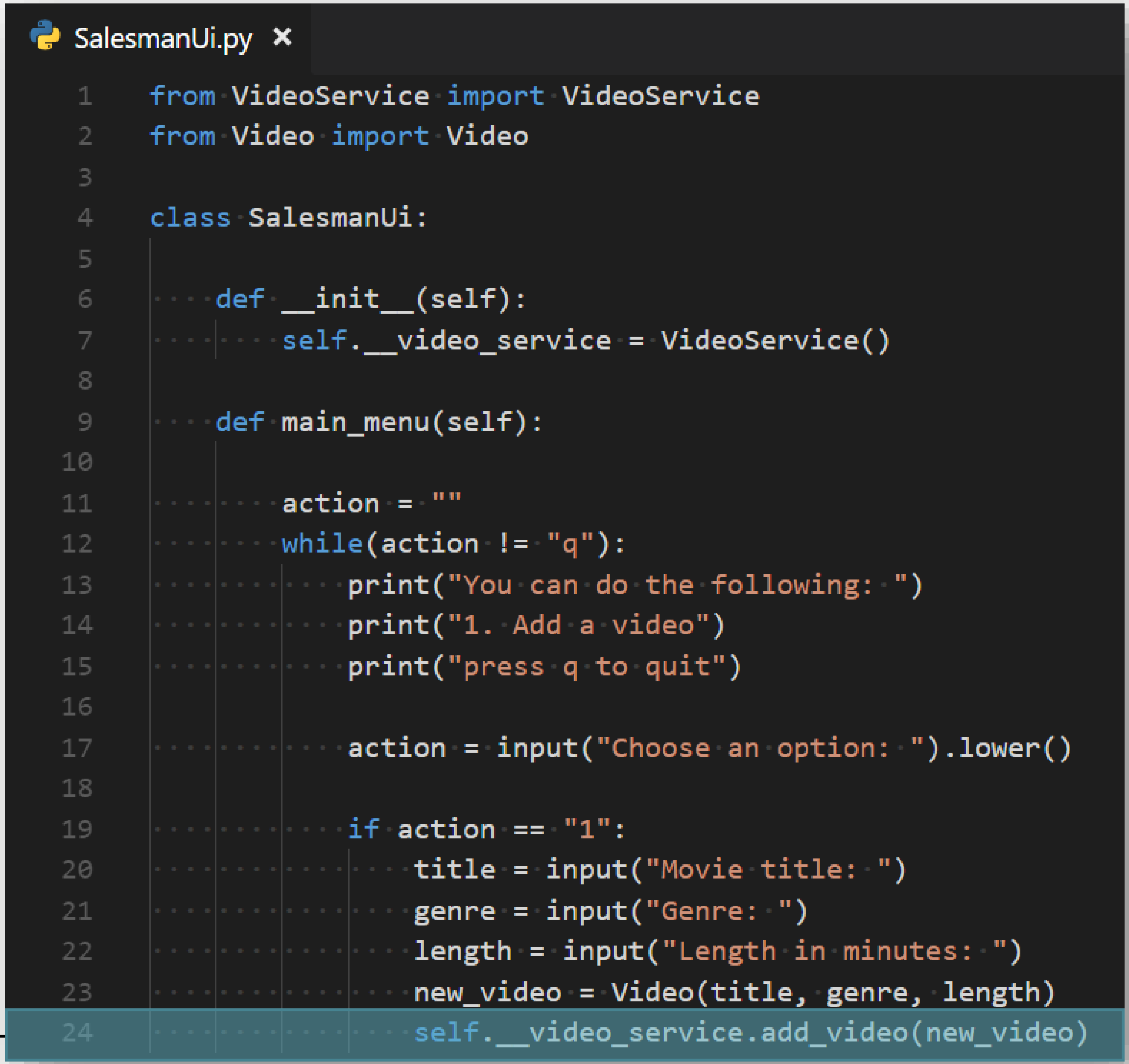
## 3 LAYERED ARCHITECTURE

- We need to create a class for the UI(user interface)
- That class should own an instance of a service class that belongs to the layer below(domain layer)
- Remember, classes that belong to the UI layer should contain minimal logic. They handle the interaction with the user such as
  - Displaying messages to the user
  - Taking input from the user
  - Guiding the user through the program

# 3 LAYERED ARCHITECTURE

- This is the class SalesmanUI
  - It has a single method called main_menu which displays the main menu to the screen
- It also has a private attribute of type VideoService

Once the user is done inputting the data about the video that is to be added the UI's class work is done and the control is passed to the service layer. This is done by calling the add_video method in the video_service

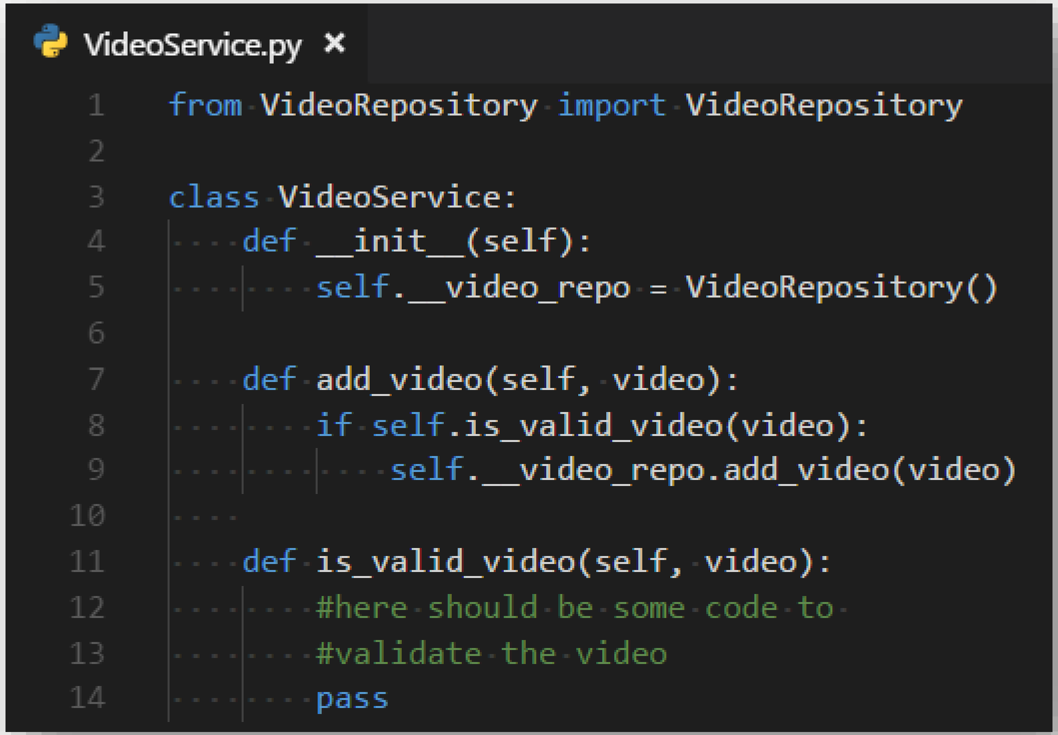


SalesmanUi.py

```python
from VideoService import VideoService
from Video import Video

class SalesmanUi:

    def __init__(self):
        self.__video_service = VideoService()

    def main_menu(self):

        action = ""
        while(action != "q"):
            print("You can do the following: ")
            print("1. Add a video")
            print("press q to quit")

            action = input("Choose an option: ").lower()

            if action == "1":
                title = input("Movie title: ")
                genre = input("Genre: ")
                length = input("Length in minutes: ")
                new_video = Video(title, genre, length)
                self.__video_service.add_video(new_video)
```

# 3 LAYERED ARCHITECTURE

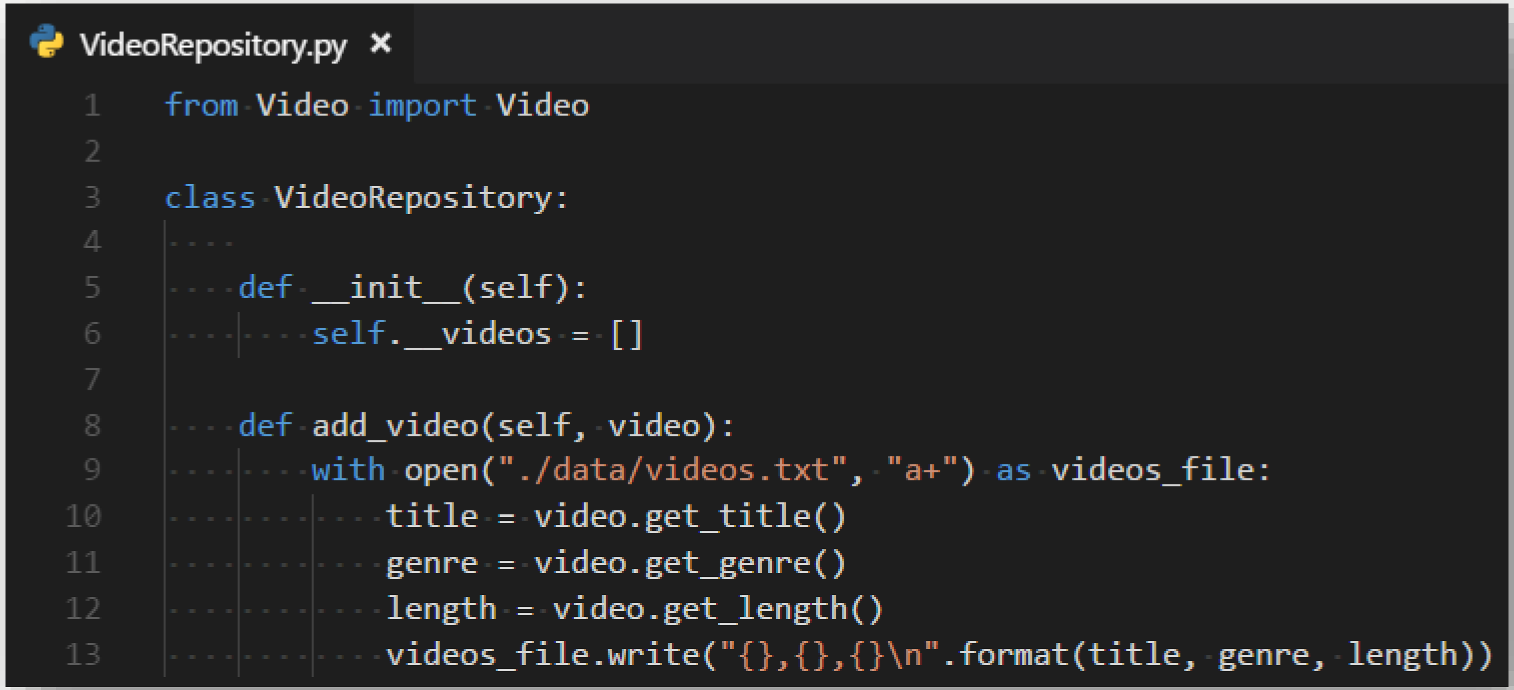
- This is the class VideoService
- It has a private attribute of type VideoRepository
- The is_valid_video method is a method that gets called by the add_video method as soon as the class from the UI layer tries to add a video
- If the video is valid it is safe to add the video to our datastore so a repository class instance is called and the video is passed to it


VideoService.py

```python
from VideoRepository import VideoRepository

class VideoService:
    def __init__(self):
        self.__video_repo = VideoRepository()

    def add_video(self, video):
        if self.is_valid_video(video):
            self.__video_repo.add_video(video)
    
    def is_valid_video(self, video):
        #here should be some code to 
        #validate the video
        pass
```

# 3 LAYERED ARCHITECTURE

- This is the VideoRepository class
- It has a a constructor and method that knows how to write a Video object to a file!

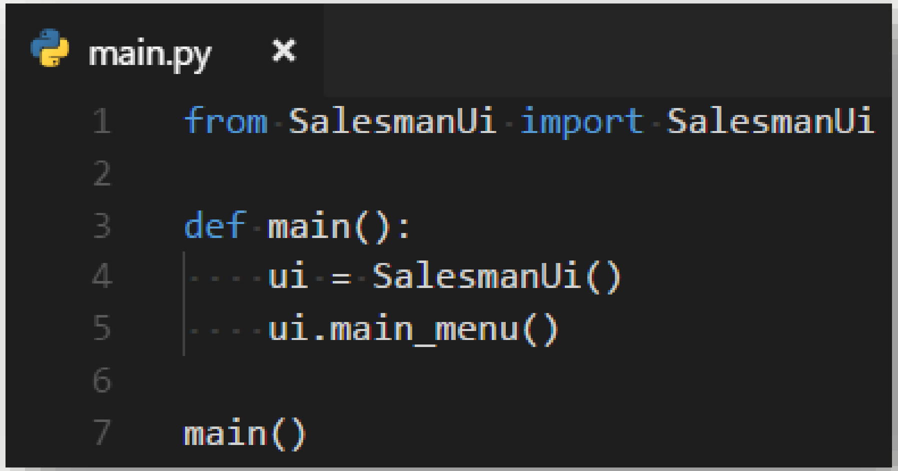


VideoRepository.py

```python
from Video import Video

class VideoRepository:
    
    def __init__(self):
        self.__videos = []

    def add_video(self, video):
        with open("./data/videos.txt", "a+") as videos_file:
            title = video.get_title()
            genre = video.get_genre()
            length = video.get_length()
            videos_file.write("{},{},{}\n".format(title, genre, length))
```

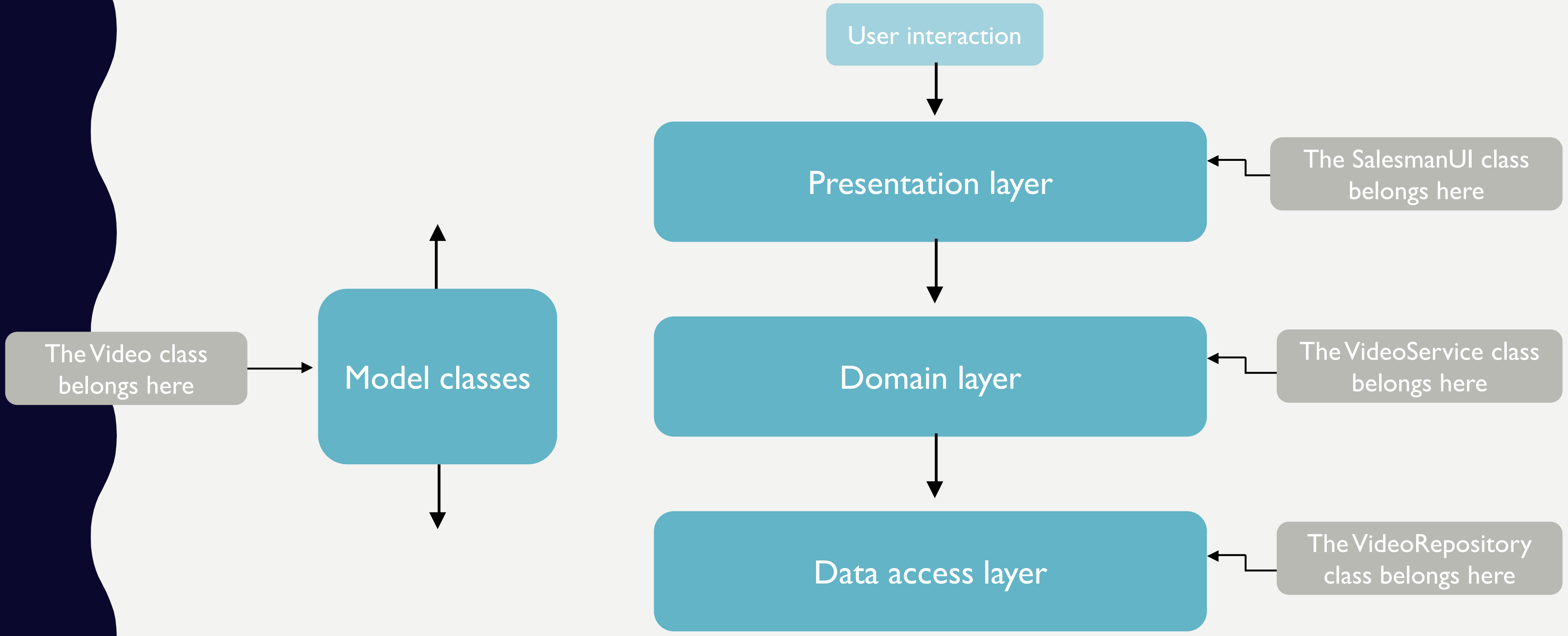
# 3 LAYERED ARCHITECTURE

- Now that we have created the add_video functionality in all layers from the UI down to the data access layer we can register our main menu in the main function
- This is all the main function will need to do!
- All the functionality is in our 3 layered project



```python
from SalesmanUi import SalesmanUi

def main():
    ui = SalesmanUi()
    ui.main_menu()

main()
```

# 3 LAYERED ARCHITECTURE


User interaction
Presentation layer
Domain layer
Data access layer
Model classes
The SalesmanUI class belongs here
The VideoService class belongs here
The VideoRepository class belongs here
The Video class belongs here

# 3 LAYERED ARCHITECTURE

1

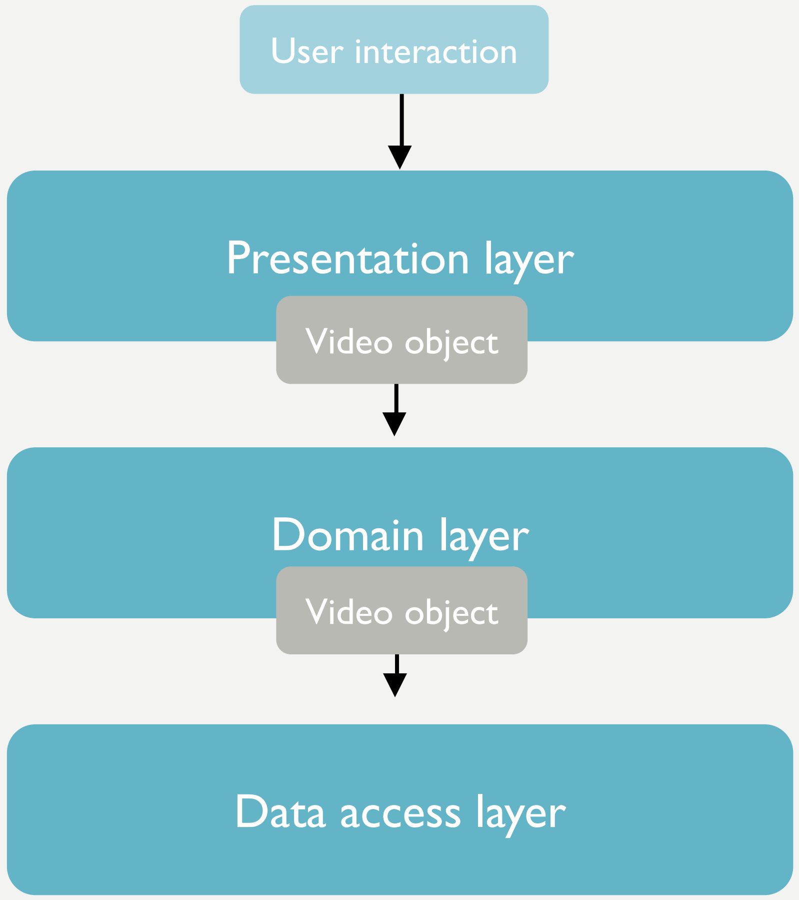
The user wants to add a video. The user inputs information about a  video in the UI layer. A video object is created and is sent down to the domain layer

2

Once the video object hits the domain layer it is validated. If it is a valid video it is sent down to the data access layer

3

In the data access layer the video object is written to the datastore. Whether it is a text file or a database



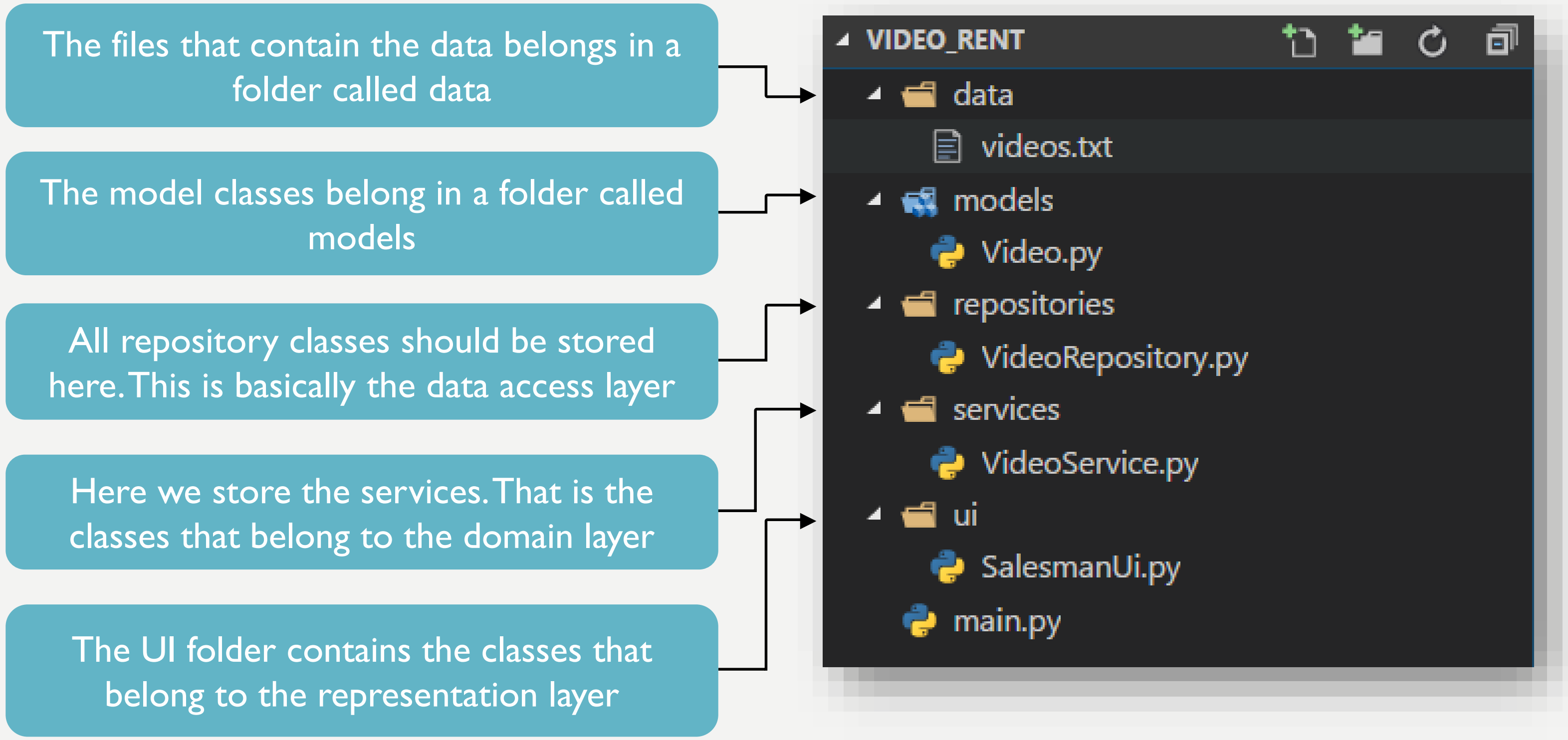
Slides by Björgvin B. Björgvinsson

# 3 LAYERED ARCHITECTURE

- 3 layered architecture is an abstract concept and can be hard to grasp to begin with
- It is worth noting that there are **no actual layers**, you will need to visualize them by yourself
- However, a good way to get used to this idea is to create a directory(folder) structure that emphasises the layering structure of the project
- Lets see an example on the next slide

# 3 LAYERED ARCHITECTURE

- Here is an image that shows a proper directory (folder) structure

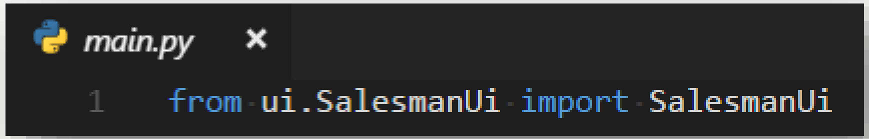


The files that contain the data belongs in a folder called data
The model classes belong in a folder called models
All repository classes should be stored here. This is basically the data access layer
Here we store the services. That is the classes that belong to the domain layer
The UI folder contains the classes that belong to the representation layer

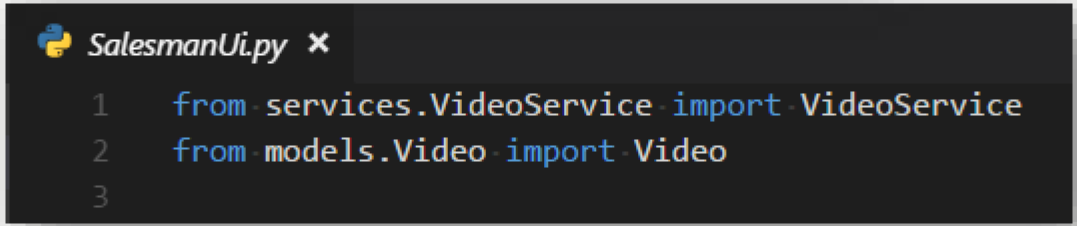
```
VIDEO_RENT
  data
    videos.txt
  models
    Video.py
  repositories
    VideoRepository.py
  services
    VideoService.py
  ui
    SalesmanUi.py
  main.py
```

## 3 LAYERED ARCHITECTURE

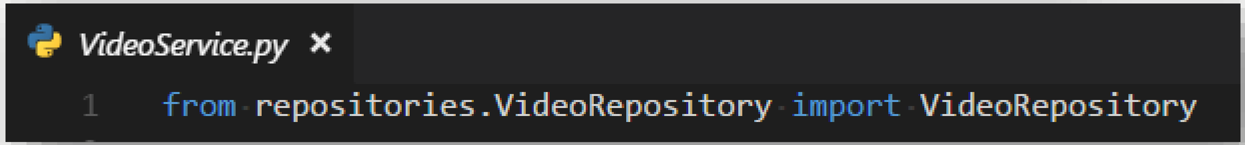
- But remember that with this folder structure you must change the import statements



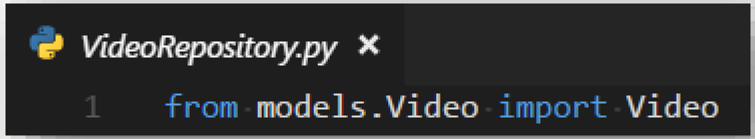
```python
from ui.SalesmanUi import SalesmanUi
```



```python
from services.VideoService import VideoService
from models.Video import Video

```



```python
from repositories.VideoRepository import VideoRepository
```



```python
from models.Video import Video
```

## 3 LAYERED ARCHITECTURE

- Remember that these code examples‘ main purpose is to demonstrate a 3 layered architecture
  - In your project you must think about error handling and validation