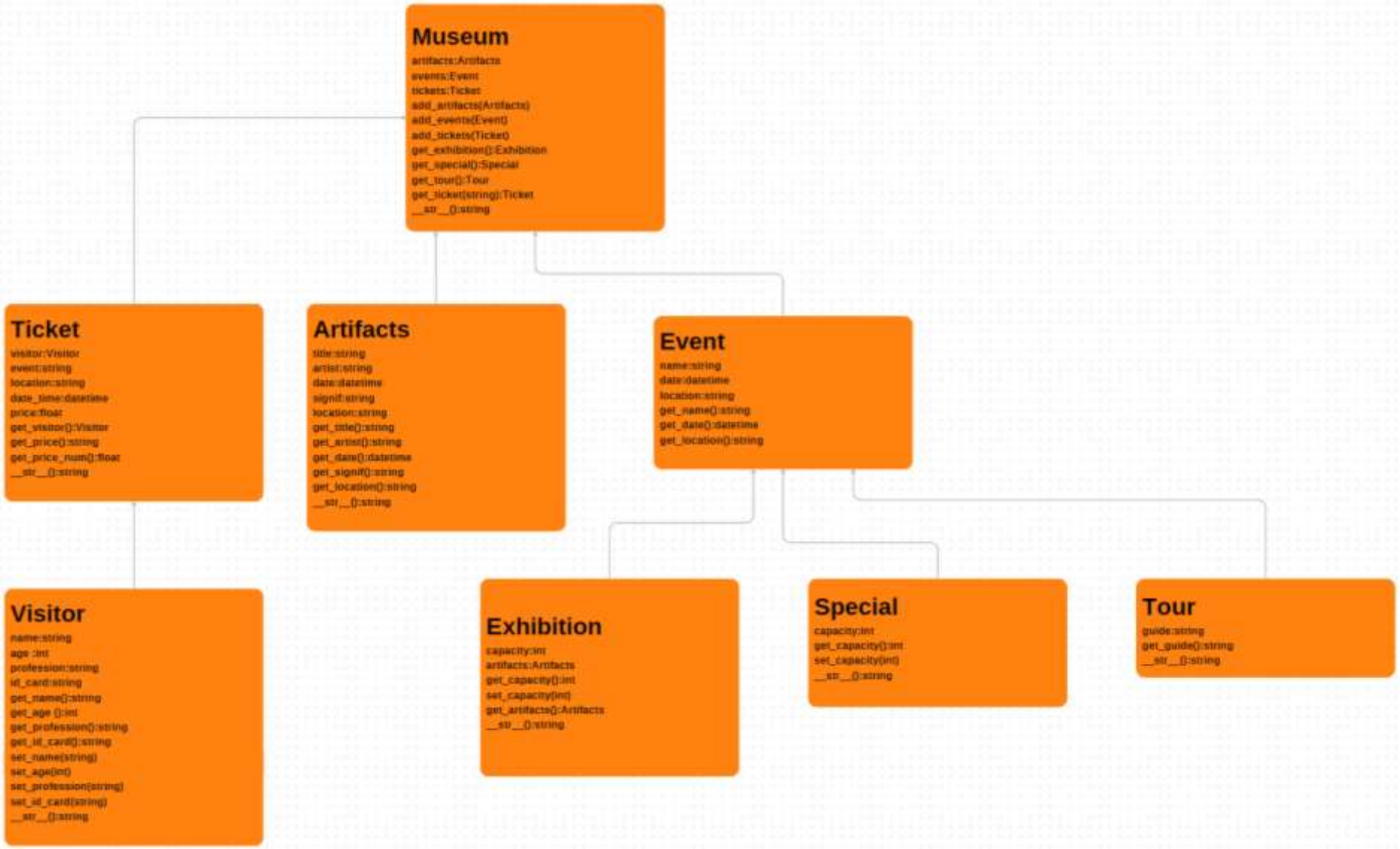
UML Diagram:


Museum
artifacts:Artifacts
events:Event
tickets:Ticket
add_artifacts(Artifacts)
add_events(Event)
add_tickets(Ticket)
get_exhibition():Exhibition
get_special():Special
get_tour():Tour
get_ticket(string):Ticket
__str__():string
Ticket
visitor:Visitor
event:string
location:string
date_time:datetime
price:float
get_visitor():Visitor
get_price():string
get_price_num():float
__str__():string
Artifacts
title:string
artist:string
date:datetime
signif:string
location:string
get_title():string
get_artist():string
get_date():datetime
get_signif():string
get_location():string
__str__():string
Event
name:string
date:datetime
location:string
get_name():string
get_date():datetime
get_location():string
Visitor
name:string
age :int
profession:string
id_card:string
get_name():string
get_age ():int
get_profession():string
get_id_card():string
set_name(string)
set_age(int)
set_profession(string)
set_id_card(string)
__str__():string
Exhibition
capacity:int
artifacts:Artifacts
get_capacity():int
set_capacity(int)
get_artifacts():Artifacts
__str__():string
Special
capacity:int
get_capacity():int
set_capacity(int)
__str__():string
Tour
guide:string
get_guide():string
__str__():string

Visitor:

This class holds information about the individual visitor such as name age profession etc.

```
class Visitor:
    def __init__(self,name,age,profession,id_card=None):
        self.name=name
        self.age=age
        self.profession=profession
        self.id_card=id_card

    def get_name(self):
        return self.name

    def get_age(self):
        return self.age

    def get_profession(self):
        return self.profession

    def get_id_card(self):
        return self.id_card

    def set_name(self,name):
        self.name=name

    def set_age(self,age):
        self.age=age

    def set_profession(self,profession):
```

```
        self.profession=profession


    def set_id_card(self,id_card):
        self.id_card=id_card


    def __str__(self):
        ret = f"Name:{self.name},Age:{self.age},Profession:{self.profession}"
        if(self.id_card):
            ret+=f",ID card={self.id_card}"
        return ret
```

Tickets:

This class holds information about the ticket of an individual visitor and holds the visitor information using an instance of visitor class.

```
from Visitor import Visitor
import datetime


class Ticket:
    def __init__(self,visitor:Visitor,event,location,date_time:datetime.datetime,group=False,price=None):
        self.visitor=visitor
        self.event=event
        self.location=location
        self.date_time=date_time
        if not price:
            price=(63+(63*0.1))
        if group:
            self.price=price*0.5
        else:
            if visitor.get_age()>=18 and visitor.get_age()<=60:
                self.price=price
```

```
        else:
            if visitor.get_id_card():
                self.price=0
            else:
                self.price=price

    def get_visitor(self):
        return self.visitor

    def get_price(self):
        return f"{self.price} AED"

    def get_price_num(self):
        return self.price

    def __str__(self):
        return
f"{self.visitor.__str__()},Event:{self.event},Location:{self.location},Date:{self.date_time.date()},Time:{self
.date_time.time()}"
```

Artifacts:

This class holds information related to the artifacts present in the Museum that includes its name ,artist date etc.

```
import datetime


class Artifacts:
    def __init__(self,title,artist,date:datetime.datetime,signif,location):
        self.title=title
        self.artist=artist
        self.date=date
```

```
        self.signif=signif
        self.location=location

    def get_title(self):
        return self.title

    def get_artist(self):
        return self.artist

    def get_signif(self):
        return self.signif

    def get_date(self):
        return self.date

    def get_location(self):
        return self.location

    def __str__(self):
        return f"Title:{self.title},Artist:{self.artist},Historical
significance:{self.signif},Date:{self.date.date()},Location:{self.location}"
```

Event:

This class acts as a base class for all the events that happen in the museum. This includes the event name date and location.

Exhibition:

This class inherits from Event class and includes additional information such as the artifacts present in the exhibition for the visitors.

Special:

This class also inherits from Event class and includes additional information such as the maximum number of visitors for the event.

Tour:

This class also inherits from Event class and includes additional information such as the guide name who will guide the group to the museum.

```
import datetime


class Event:
    def __init__(self,name,location,date_time:datetime.datetime):
        self.name=name
        self.location=location
        self.date_time=date_time

    def get_name(self):
        return self.name

    def get_location(self):
        return self.location

    def get_date_time(self):
        return self.date_time

class Exhibition(Event):
    def __init__(self, name, location, date_time: datetime.datetime,artifacts,capacity):
        super().__init__(name, location, date_time)
        self.artifacts=[i for i in artifacts if location==i.get_location()]
        self.capacity=capacity
```

```
    def get_capacity(self):
        return self.capacity

    def set_capacity(self,capacity):
        self.capacity=capacity

    def set_artifacts(self,artifacts):
        del self.artifacts
        self.artifacts=[i for i in artifacts if self.location==i.get_location()]

    def __str__(self):
        artifacts=""
        for i in self.artifacts:
            artifacts+=i.get_title()
        return f"Event
name:{self.name},Location:{self.location},Date:{self.date_time.date()},Time:{self.date_time.time()},Ticke
ts left={self.capacity}\nArtifacts={artifacts}"

class Tour(Event):
    def __init__(self, name, date_time: datetime.datetime,guide):
        self.location="Museum"
        self.guide=guide
        super().__init__(name, self.location, date_time)

    def get_guide(self):
        return self.guide

    def __str__(self) -> str:
```

```
        return f"Event
name:{self.name},Location:{self.location},Date:{self.date_time.date()},Time:{self.date_time.time()},Guid
e:{self.guide}"


class Special(Event):
    def __init__(self, name, location, date_time: datetime.datetime,capacity):
        super().__init__(name, location, date_time)
        self.capacity=capacity


    def get_capacity(self):
        return self.capacity


    def set_capacity(self,capacity):
        self.capacity=capacity


    def __str__(self) -> str:
        return f"Event
name:{self.name},Location:{self.location},Date:{self.date_time.date()},Time:{self.date_time.time()},Ticke
ts left={self.capacity}"
```

Museum:

This class holds all the information related to the museum such as artifacts, tickets sold and the events that will happen at the museum and provide function to add values to different instances of classes.

```
from Ticket import Ticket
from Visitor import Visitor
from Artifacts import Artifacts
import Event


min_group_num=3


class Museum:
```

```
    def __init__(self):
        self.artifacts=[]
        self.events=[]
        self.tickets=[]

    def add_artifacts(self,artifact:Artifacts):
        fil=[i for i in self.artifacts if i.get_title()==artifact.get_title() and i.get_artist()==artifact.location()]
        if(len(fil)>0):
            return "Cannot add Artifact as it is already present"
        else:
            self.artifacts.append(artifact)
            for i in self.get_exhibition():
                i.set_artifacts(self.artifacts)
            return "Artifact successfully added"

    def add_events(self,event):
        fil=[i for i in self.events if (i.get_location()==event.get_location() and
i.get_date_time()==event.get_date_time() )]
        if(len(fil)>0):
            return "Cannot add Artifact as it is already present"
        else:
            self.events.append(event)
            return "Artifact successfully added"

    def add_tickets(self,ticket:Ticket):
        self.tickets.append(ticket)

    def get_exhibition(self):
        fil=[ i for i in self.events if isinstance(i,Event.Exhibition) ]
```

```
        return fil


    def get_artifacts(self):
        return self.artifacts


    def get_tour(self):
        fil=[ i for i in self.events if isinstance(i,Event.Tour) ]
        return fil


    def get_special(self):
        fil=[ i for i in self.events if isinstance(i,Event.Special) ]
        return fil


    def get_ticket(self,name):
        fil=[i for i in self.tickets if i.get_visitor().get_name()==name]
        return fil


def buy_ticket(name,age,profession,event,museum:Museum,id_card=None):


ticket=Ticket(Visitor(name,age,profession,id_card),event.get_name(),event.get_location(),event.get_dat
e_time())
    museum.add_tickets(ticket)
```

Main function/file:

```
from tkinter import *
from tkinter import messagebox
import Museum
from Artifacts import Artifacts
from Visitor import Visitor
from Ticket import Ticket
```

```
import datetime
import Event

tour_min=2
tour_max=5

class main_menu:
    def __init__(self):
        self.root=Tk()
        self.root.title("Museum")
        self.root.geometry("852x560")
        self.museum=Museum.Museum()
        self.museum.add_artifacts(Artifacts("Pyrmaid","James",datetime.datetime(1850,5,12),"lorem ahjsdhasid","Outdoor spaces"))
        self.museum.add_events(Event.Exhibition("Mona Lisa","Exhibition halls",datetime.datetime(2000,3,4),self.museum.artifacts,12))
        self.museum.add_events(Event.Tour("Mona Lisa",datetime.datetime(2000,3,4),"Sam Smith"))

    def menu(self):
        self.clear()
        Label(self.root,text="The Louvre Museum",font=('Arial',20,'bold')).pack(pady=10)
        Button(self.root, padx=10,pady=5 , text="View artifacts",command= self.view_artifacts_gui ).pack(padx=50,pady=5)
        Button(self.root, padx=10,pady=5 , text="Add artifacts",command= self.add_artifacts_gui ).pack(padx=50,pady=5)
        Button(self.root, padx=10,pady=5 , text="View Exhibition events",command= self.view_exhibition_event_gui ).pack(padx=50,pady=5)
        Button(self.root, padx=10,pady=5 , text="Add Exhibition Event",command= self.add_exhibition_event_gui ).pack(padx=50,pady=5)
        Button(self.root, padx=10,pady=5 , text="View Special Event",command= self.view_special_event_gui ).pack(padx=50,pady=5)
```

```
    Button(self.root, padx=10,pady=5 , text="Add Special Event",command= self.add_special_event_gui
).pack(padx=50,pady=5)

    Button(self.root, padx=10,pady=5 , text="View Tour Event",command= self.view_tour_event_gui
).pack(padx=50,pady=5)

    Button(self.root, padx=10,pady=5 , text="Add Tour Event",command= self.add_tour_event_gui
).pack(padx=50,pady=5)

    Button(self.root, padx=10,pady=5 , text="Buy exhibition events ticket",command=
self.view_exhibition_ticket ).pack(padx=50,pady=5)

    Button(self.root, padx=10,pady=5 , text="Buy tour ticket",command= self.view_tour_ticket
).pack(padx=50,pady=5)

    Button(self.root, padx=10,pady=5 , text="Buy special events ticket",command=
self.view_special_ticket ).pack(padx=50,pady=5)


# exhibition event functions


  def view_exhibition_event_gui(self):

    self.clear()

    for i in self.museum.get_exhibition():

      Label(self.root,text=f"-> {i.__str__()}").pack(pady=5)

    Button(self.root, padx=10,pady=5 , text="Back",command= self.menu ).pack(padx=50,pady=5)


  def add_exhibition_event_gui(self):

    self.clear()


    # event name

    Label(self.root,text="Event name:").pack()

    self.exhibition_event_name=Entry(self.root,width=50,highlightthickness=2,borderwidth=0)

    self.exhibition_event_name.pack(padx=50,pady=5,ipadx=5,ipady=3)


    # date

    Label(self.root,text="Eevnt Date(Year-Month-Day):").pack()
```

```
        self.exhibition_event_date=Entry(self.root,width=50,highlightthickness=2,borderwidth=0)
        self.exhibition_event_date.pack(padx=50,pady=5,ipadx=5,ipady=3)

        #  capacity
        Label(self.root,text="Event maximum capacity:").pack()
        self.exhibition_event_capacity=Entry(self.root,width=50,highlightthickness=2,borderwidth=0)
        self.exhibition_event_capacity.pack(padx=50,pady=5,ipadx=5,ipady=3)

        # location
        location_options=[
            "Permanent galleries",
            "Exhibition halls",
            "Outdoor spaces"
        ]
        self.exhibition_event_location=StringVar()
        self.exhibition_event_location.set(None)
        Label(self.root,text="Event Location:").pack()
        for i in location_options:
            Radiobutton(self.root, text=f"{i}", variable=self.exhibition_event_location, value=f"{i}" ).pack()

        # add button
        Button(self.root, padx=10,pady=5 , text="Add event",command=
    self.add_exhibition_event).pack(padx=20,pady=5)

    def add_exhibition_event(self):
        artifacts=[i for i in self.museum.artifacts if i.get_location()==self.exhibition_event_location.get()]
        date=self.exhibition_event_date.get().split("-")
        if(len(date)<3):
            messagebox.showerror("Error","Date format in incorrect.Please try again")
```

```
        else:
            self.museum.add_events(Event.Exhibition(
                self.exhibition_event_name.get()
                ,self.exhibition_event_location.get(),
                datetime.datetime( int(date[0]) , int(date[1]) , int(date[2]) ) ,
                artifacts ,
                int(self.exhibition_event_capacity.get())
                ) )
        self.menu()

# special event functions

    def view_special_event_gui(self):
        self.clear()
        for i in self.museum.get_special():
            Label(self.root,text=f"-> {i.__str__()}").pack(pady=5)
        Button(self.root, padx=10,pady=5 , text="Back",command= self.menu ).pack(padx=50,pady=5)

    def add_special_event_gui(self):
        self.clear()

        # event name
        Label(self.root,text="Event name:").pack()
        self.special_event_name=Entry(self.root,width=50,highlightthickness=2,borderwidth=0)
        self.special_event_name.pack(padx=50,pady=5,ipadx=5,ipady=3)

        # date
        Label(self.root,text="Eevnt Date(Year-Month-Day):").pack()
        self.special_event_date=Entry(self.root,width=50,highlightthickness=2,borderwidth=0)
```

```
        self.special_event_date.pack(padx=50,pady=5,ipadx=5,ipady=3)

        #  capacity
        Label(self.root,text="Event maximum capacity:").pack()
        self.special_event_capacity=Entry(self.root,width=50,highlightthickness=2,borderwidth=0)
        self.special_event_capacity.pack(padx=50,pady=5,ipadx=5,ipady=3)

        # location
        location_options=[
            "Permanent galleries",
            "special halls",
            "Outdoor spaces"
        ]
        self.special_event_location=StringVar()
        self.special_event_location.set(None)
        Label(self.root,text="Event Location:").pack()
        for i in location_options:
            Radiobutton(self.root, text=f"{i}", variable=self.special_event_location, value=f"{i}" ).pack()

        # add button
        Button(self.root, padx=10,pady=5 , text="Add event",command=
self.add_special_event).pack(padx=20,pady=5)

    def add_special_event(self):
        date=self.special_event_date.get().split("-")
        if(len(date)<3):
            messagebox.showerror("Error","Date format in incorrect.Please try again")
        else:
            self.museum.add_events(Event.Special(
```

```
            self.special_event_name.get()
            ,self.special_event_location.get(),
            datetime.datetime( int(date[0]) , int(date[1]) , int(date[2]) ) ,
            int(self.special_event_capacity.get())
            ) )
        self.menu()

# tour event functions

    def view_tour_event_gui(self):
        self.clear()
        for i in self.museum.get_tour():
            Label(self.root,text=f"-> {i.__str__()}").pack(pady=5)
        Button(self.root, padx=10,pady=5 , text="Back",command= self.menu ).pack(padx=50,pady=5)

    def add_tour_event_gui(self):
        self.clear()

        # event name
        Label(self.root,text="Event name:").pack()
        self.tour_event_name=Entry(self.root,width=50,highlightthickness=2,borderwidth=0)
        self.tour_event_name.pack(padx=50,pady=5,ipadx=5,ipady=3)

        # date
        Label(self.root,text="Eevnt Date(Year-Month-Day):").pack()
        self.tour_event_date=Entry(self.root,width=50,highlightthickness=2,borderwidth=0)
        self.tour_event_date.pack(padx=50,pady=5,ipadx=5,ipady=3)

        #  guide
```

```
        Label(self.root,text="Event guide name:").pack()
        self.tour_event_guide=Entry(self.root,width=50,highlightthickness=2,borderwidth=0)
        self.tour_event_guide.pack(padx=50,pady=5,ipadx=5,ipady=3)

        # add button
        Button(self.root, padx=10,pady=5 , text="Add event",command=
self.add_tour_event).pack(padx=20,pady=5)

    def add_tour_event(self):
        date=self.tour_event_date.get().split("-")
        if(len(date)<3):
            messagebox.showerror("Error","Date format in incorrect.Please try again")
        else:
            self.museum.add_events(Event.Tour(
                self.tour_event_name.get(),
                datetime.datetime( int(date[0]) , int(date[1]) , int(date[2]) ) ,
                self.tour_event_guide.get()
                ) )
        self.menu()

# tour tickets functions

    def view_tour_ticket(self):
        self.clear()

        self.visitor=None
        self.visitors=[]
        tour=[i for i in self.museum.events if isinstance(i,Event.Tour)]
        if len(tour)==0:
```

```
            messagebox.showinfo("Sorry","There are currently no Group Tours")
            self.menu()
        else:
            event=IntVar()
            event.set(0)
            cnt=0
            for value in tour:
                Radiobutton(self.root, text=f"{value.__str__()}", variable=event, value=f"{cnt}" ).pack()
                cnt+=1
            self.total_visitors=Label(self.root,text=f"Total
visitors:{len(self.visitors)}(Min={tour_min},Max={tour_max})")
            self.total_visitors.pack()
            Button(self.root, padx=10,pady=5 , text="Enter Visitors Information",command= (
                lambda: self.get_visitor( 1 )
            ) ).pack(padx=20,pady=5)
            Button(self.root, padx=10,pady=5 , text="Buy ticket",command= (lambda:
self.buy_tour_ticket(tour[event.get()])) ).pack(padx=20,pady=5)


    def buy_tour_ticket(self,event):
        if len(self.visitors)<tour_min or len(self.visitors)>tour_max:
            messagebox.showerror("Error","Incorrect number of visitors.Please try again")
            self.menu()
        else:
            price=0
            out=""
            for i in self.visitors:
                ticket=Ticket(i,event.get_name(),event.get_location(),event.get_date_time(),True)
                self.museum.add_tickets(ticket)
                messagebox.showinfo("Ticket Successfull",f"{ticket.__str__()}")
```

```
            price+=ticket.get_price_num()
        messagebox.showinfo("Ticket Successfull",f"Total Bill:{price} AED")
        self.menu()


# exhibition tickets functions


    def view_exhibition_ticket(self):
        self.clear()


        self.visitor=None
        exhibition=self.museum.get_exhibition()
        if len(exhibition)==0:
            messagebox.showinfo("Sorry","There are currently no Exhibition events")
            self.menu()
        else:
            event=IntVar()
            event.set(0)
            cnt=0
            for value in exhibition:
                Radiobutton(self.root, text=f"{value.__str__()}", variable=event, value=f"{cnt}" ).pack()
                cnt+=1
            Button(self.root, padx=10,pady=5 , text="Enter Visitor Information",command=
self.get_visitor).pack(padx=20,pady=5)
            Button(self.root, padx=10,pady=5 , text="Buy ticket",command= (lambda:
self.buy_exhibition_ticket(exhibition[event.get()])) ).pack(padx=20,pady=5)


    def buy_exhibition_ticket(self,event):
        if self.visitor==None and event.get_capacity()>0:
```

```
            messagebox.showerror("Error","Please enter visitor information")
            self.menu()
        else:
            ticket=Ticket(self.visitor,event.get_name(),event.get_location(),event.get_date_time())
            self.museum.add_tickets(ticket)
            event.set_capacity(event.get_capacity()-1)
            messagebox.showinfo("Ticket Successfull",f"{ticket.__str__()},Total Bill:{ticket.get_price()}")
            self.menu()

# special tickets functions

    def view_special_ticket(self):
        self.clear()

        self.visitor=None
        special=self.museum.get_special()
        if len(special)==0:
            messagebox.showinfo("Sorry","There are currently no Special events")
            self.menu()
        else:
            event=IntVar()
            event.set(0)
            cnt=0
            for value in special:
                Radiobutton(self.root, text=f"{value.__str__()}", variable=event, value=f"{cnt}" ).pack()
                cnt+=1
            Button(self.root, padx=10,pady=5 , text="Enter Visitor Information",command=
self.get_visitor).pack(padx=20,pady=5)
```

```
        Button(self.root, padx=10,pady=5 , text="Buy ticket",command= (lambda:
self.buy_special_ticket(special[event.get()])) ).pack(padx=20,pady=5)


    def buy_special_ticket(self,event):
        if self.visitor==None and event.get_capacity()>0:
            messagebox.showerror("Error","Please enter visitor information")
            self.menu()
        else:
            ticket=Ticket(self.visitor,event.get_name(),event.get_location(),event.get_date_time())
            self.museum.add_tickets(ticket)
            event.set_capacity(event.get_capacity()-1)
            messagebox.showinfo("Ticket Successfull",f"{ticket.__str__()},Total Bill:{ticket.get_price()}")
            self.menu()

# tickets helper functions


    def get_visitor(self,tour=None):
        new_window=Toplevel(self.root)
        new_window.title("Visitor Information")
        new_window.geometry("852x520")


        # name
        Label(new_window,text="Visitor name:").pack()
        self.visitor_name=Entry(new_window,width=50,highlightthickness=2,borderwidth=0)
        self.visitor_name.pack(padx=50,pady=5,ipadx=5,ipady=3)


        # age
        Label(new_window,text="Visitor age:").pack()
        self.visitor_age=Entry(new_window,width=50,highlightthickness=2,borderwidth=0)
```

```
        self.visitor_age.pack(padx=50,pady=5,ipadx=5,ipady=3)


        # profession
        Label(new_window,text="Visitor profession:").pack()
        self.visitor_profession=Entry(new_window,width=50,highlightthickness=2,borderwidth=0)
        self.visitor_profession.pack(padx=50,pady=5,ipadx=5,ipady=3)


        # id_card
        Label(new_window,text="Visitor ID card(Optional):").pack()
        self.visitor_id_card=Entry(new_window,width=50,highlightthickness=2,borderwidth=0)
        self.visitor_id_card.pack(padx=50,pady=5,ipadx=5,ipady=3)


        Button(new_window, padx=10,pady=5 , text="Next",command= (lambda:
self.assign_visitor(new_window,tour))).pack(padx=20,pady=5)
        new_window.mainloop()


    def assign_visitor(self,new_window,tour=None):
        if self.visitor_id_card.get()=="":
            self.visitor_id_card=None
        else:
            self.visitor_id_card=self.visitor_id_card.get()

self.visitor=Visitor(self.visitor_name.get(),int(self.visitor_age.get()),self.visitor_profession.get(),self.visito
r_id_card)
        new_window.destroy()
        if tour:
            self.visitors.append(self.visitor)
            self.total_visitors.config(text=f"Total
visitors:{len(self.visitors)}(Min={tour_min},Max={tour_max})")
```

```
# artifacts functions
    def view_artifacts_gui(self):
        self.clear()
        for i in self.museum.get_artifacts():
            Label(self.root,text=f"-> {i.__str__()}").pack(pady=5)
        Button(self.root, padx=10,pady=5 , text="Back",command= self.menu ).pack(padx=50,pady=5)

    def add_artifacts_gui(self):
        self.clear()

        # title
        Label(self.root,text="Artifact Title:").pack()
        self.artifact_name=Entry(self.root,width=50,highlightthickness=2,borderwidth=0)
        self.artifact_name.pack(padx=50,pady=5,ipadx=5,ipady=3)

        # artist
        Label(self.root,text="Artifact Artist:").pack()
        self.artifact_artist=Entry(self.root,width=50,highlightthickness=2,borderwidth=0)
        self.artifact_artist.pack(padx=50,pady=5,ipadx=5,ipady=3)

        # date
        Label(self.root,text="Artifact Date(Year-Month-Day):").pack()
        self.artifact_date=Entry(self.root,width=50,highlightthickness=2,borderwidth=0)
        self.artifact_date.pack(padx=50,pady=5,ipadx=5,ipady=3)

        #  historical significance
        Label(self.root,text="Artifact historical significance:").pack()
        self.artifact_signif=Entry(self.root,width=50,highlightthickness=2,borderwidth=0)
        self.artifact_signif.pack(padx=50,pady=5,ipadx=5,ipady=3)
```

```
        # location
        location_options=[
            "Permanent galleries",
            "Exhibition halls",
            "Outdoor spaces"
        ]
        self.artifact_location=StringVar()
        self.artifact_location.set(None)
        Label(self.root,text="Artifact Location:").pack()
        for i in location_options:
            Radiobutton(self.root, text=f"{i}", variable=self.artifact_location, value=f"{i}" ).pack()

        # add button
        Button(self.root, padx=10,pady=5 , text="Add artifacts",command=
self.add_artifact).pack(padx=20,pady=5)

    def add_artifact(self):
        date=self.artifact_date.get().split("-")
        if(len(date)<3):
            messagebox.showerror("Error","Date format in incorrect.Please try again")
        else:

self.museum.add_artifacts(Artifacts(self.artifact_name.get(),self.artifact_artist.get(),datetime.datetime(
int(date[0]) , int(date[1]) , int(date[2]) ) , self.artifact_signif.get() , self.artifact_location.get() ) )
        self.menu()

    def clear(self):
        for i in self.root.pack_slaves():
            i.destroy()
```

```
    def run(self):
        self.root.mainloop()


menu=main_menu()
menu.menu()
menu.run()
```