GUI

The GUI is responsible for all the business logic implemented in our system

Imports

```
import tkinter as tk
import tkinter.ttk as ttk
import classes
from tkinter import simpledialog as sd,messagebox
from datetime import date
```

UI initialization function

```
class GUI:
    def __init__(self, master=None):
        # build ui
        self.main = tk.Tk() if master is None else tk.Toplevel(master)
        self.main.configure(
            height=320,
            highlightbackground="#686868",
            relief="flat",
            width=480)
        self.main.title("OOP")
        self.Employee = ttk.Labelframe(self.main)
        self.Employee.configure(height=241, text='Employee', width=480)
        label1 = ttk.Label(self.Employee)
        label1.configure(
            compound="center",
            relief="flat",
            takefocus=False,
            text='Name')
        label1.grid(column=0, ipadx=10, padx=20, row=0, sticky="w")
        self.name = ttk.Entry(self.Employee)
        self.name.grid(column=1, ipadx=30, ipady=5, row=0, sticky="w")
        label2 = ttk.Label(self.Employee)
        label2.configure(
            compound="center",
            relief="flat",
            takefocus=False,
            text='Age')
```

```
label2.grid(column=2, padx=20, row=0, sticky="w")
self.age = ttk.Entry(self.Employee)
self.age.grid(column=3, ipadx=30, ipady=5, row=0, sticky="w")
label3 = ttk.Label(self.Employee)
label3.configure(
   compound="center",
   relief="flat",
   takefocus=False,
   text='Salary')
label3.grid(column=2, padx=20, row=1, sticky="w")
label4 = ttk.Label(self.Employee)
label4.configure(
   compound="center",
   relief="flat",
   takefocus=False,
   text='Department')
label4.grid(column=0, padx=20, row=1, sticky="w")
self.depart = ttk.Entry(self.Employee)
self.depart.grid(
   column=1,
   ipadx=30,
   ipady=5,
   pady=10,
   row=1,
   sticky="w")
label5 = ttk.Label(self.Employee)
label5.configure(
   compound="center",
   relief="flat",
   takefocus=False,
   text='Passport No.')
label5.grid(column=0, padx=20, row=2, sticky="w")
self.passport = ttk.Entry(self.Employee)
self.passport.grid(column=1, ipadx=30, ipady=5, row=2, sticky="w")
self.salary = ttk.Entry(self.Employee)
self.salary.grid(column=3, ipadx=30, ipady=5, row=1, sticky="w")
label6 = ttk.Label(self.Employee)
label6.configure(
   compound="center",
            relief="flat",
            takefocus=False,
            text='Job Title')
        label6.grid(column=2, padx=20, row=2, sticky="w")
        self.job = ttk.Combobox(self.Employee)
        self.job.grid(column=3, ipadx=28, ipady=5, row=2)
        self.Employee.pack(fill="both", side="top")
        frame1 = ttk.Frame(self.main)
        frame1.configure(height=200, width=200)
        self.emp_add_btn = ttk.Button(frame1)
        self.emp_add_btn.configure(text='Add')
        self.emp_add_btn.grid(column=0, ipadx=10, padx=10, row=0, sticky="w")
        self.emp_search_btn = ttk.Button(frame1)
        self.emp_search_btn.configure(text='Search')
        self.emp_search_btn.grid(column=1, ipadx=10, padx=10, row=0)
        self.emp_delete_btn = ttk.Button(frame1)
        self.emp_delete_btn.configure(text='Delete')
        self.emp_delete_btn.grid(column=2, ipadx=10, padx=10, row=0)
        self.emp_modify_btn = ttk.Button(frame1)
        self.emp_modify_btn.configure(text='Modify')
        self.emp_modify_btn.grid(column=3, ipadx=10, padx=10, row=0)
        frame1.pack(fill="both", side="top")
        self.labelframe1 = ttk.Labelframe(self.main)
        self.labelframe1.configure(height=241, text='Car', width=250)
        label7 = ttk.Label(self.labelframe1)
        label7.configure(
            compound="center",
            relief="flat",
            takefocus=False,
            text='Name')
        label7.grid(column=0, padx=20, row=0, sticky="w")
        self.car_name = ttk.Entry(self.labelframe1)
        self.car_name.grid(
            column=1,
            ipadx=30,
            ipady=5,
            padx=20,
            row=0,
            sticky="w")
```

```
label8 = ttk.Label(self.labelframe1)
label8.configure(
    compound="center",
    relief="flat",
    takefocus=False,
    text='Model')
label8.grid(column=2, padx=20, row=0, sticky="w")
self.car_model = ttk.Entry(self.labelframe1)
self.car_model.grid(column=3, ipadx=30, ipady=5, row=0, sticky="w")
label10 = ttk.Label(self.labelframe1)
label10.configure(
    compound="center",
    relief="flat",
    takefocus=False,
    text='Price')
label10.grid(column=0, padx=20, row=1, sticky="w")
self.car_price = ttk.Entry(self.labelframe1)
self.car_price.grid(
    column=1,
    ipadx=30,
    ipady=5,
    padx=20,
    row=1,
    sticky="w")
label11 = ttk.Label(self.labelframe1)
label11.configure(
    compound="center",
    relief="flat",
    takefocus=False,
    text='ID No.')
label11.grid(column=0, padx=20, row=2, sticky="w")
self.car_id = ttk.Entry(self.labelframe1)
self.car_id.grid(
    column=1,
    ipadx=30,
    ipady=5,
    padx=20,
    row=2,
    sticky="w")
```

```
label12 = ttk.Label(self.labelframe1)
label12.configure(
    compound="center",
    relief="flat",
    takefocus=False,
    text='Type')
label12.grid(column=2, padx=20, row=1, sticky="w")
self.car_type = ttk.Combobox(self.labelframe1)
self.car_type.grid(column=3, ipadx=28, ipady=5, pady=10, row=1)
self.labelframe1.pack(fill="both", side="top")
frame2 = ttk.Frame(self.main)
frame2.configure(height=200, width=200)
self.car_add_btn = ttk.Button(frame2)
self.car_add_btn.configure(text='Add')
self.car_add_btn.grid(column=0, ipadx=10, padx=10, row=0, sticky="w")
self.car_search_btn = ttk.Button(frame2)
self.car_search_btn.configure(text='Search')
self.car_search_btn.grid(column=1, ipadx=10, padx=10, row=0)
self.car_delete_btn = ttk.Button(frame2)
self.car_delete_btn.configure(text='Delete')
self.car_delete_btn.grid(column=2, ipadx=10, padx=10, row=0)
self.car_modify_btn = ttk.Button(frame2)
self.car_modify_btn.configure(text='Modify')
self.car_modify_btn.grid(column=3, ipadx=10, padx=10, row=0)
frame2.pack(fill="both", side="top")
self.labelframe2 = ttk.Labelframe(self.main)
self.labelframe2.configure(height=241, text='Sale', width=250)
label17 = ttk.Label(self.labelframe2)
label17.configure(
    compound="center",
    relief="flat",
    takefocus=False,
    text='Employee Id')
label17.grid(column=0, padx=20, row=0, sticky="w")
label18 = ttk.Label(self.labelframe2)
label18.configure(
    compound="center",
    relief="flat",
    takefocus=False,
            text='car Id')
        label18.grid(column=2, padx=20, row=0, sticky="w")
        combobox4 = ttk.Combobox(self.labelframe2)
        combobox4.grid(column=1, ipadx=30, ipady=5, row=0, sticky="w")
        combobox5 = ttk.Combobox(self.labelframe2)
        combobox5.grid(column=3, ipadx=30, ipady=5, row=0, sticky="w")
        self.labelframe2.pack(fill="both", side="top")
        frame3 = ttk.Frame(self.main)
        frame3.configure(height=200, width=200)
        self.sale_add_btn = ttk.Button(frame3)
        self.sale_add_btn.configure(text='Add')
        self.sale_add_btn.grid(column=0, ipadx=10, padx=10, row=0, sticky="w")
        self.sale_search_btn = ttk.Button(frame3)
        self.sale_search_btn.configure(text='Search')
        self.sale_search_btn.grid(column=1, ipadx=10, padx=10, row=0)
        frame3.pack(fill="both", side="top")

        # Main widget
        self.mainwindow = self.main

    def run(self):
        self.mainwindow.mainloop()


if __name__ == "__main__":
    app = GUI()
    app.run()
```

Employee management function in GUI

```
##################################################### EMPLOYEE
######################################################
    #adding and modifying employee to the list using methods from classes.py
    def addEmployee(self):
        """
Adds employee to the dealer's list using data provided by the user.
```

```
This function extracts the employee's name, age, salary, passport, department, and job title from the
corresponding input fields. It validates the input data using methods from the Dealer class, and if the data is
valid, it adds the employee to the dealer's list. If the job title is 'Manager', a new manager is created and
added to the manager's list, otherwise, a salesperson is created and added to the salespeople list with a
manager ID selected by the user.

Parameters:
None

Returns:
None
"""
        emp_name=self.name.get()
        emp_age=self.age.get()
        emp_salary=self.salary.get()
        emp_pass=self.passport.get()
        emp_dep=self.depart.get()
        emp_job=self.jobTitle.get()
        if self.dealer.validate_string(emp_name) and self.dealer.is_valid_price(emp_pass) and
self.dealer.validate_string(emp_dep) and self.dealer.is_valid_price(emp_age) and
self.dealer.is_valid_price(emp_salary):

            if emp_job=="Manager":
                newId=len(self.dealer.managersList)+len(self.dealer.salesManList)+1

man=classes.Manager(name=emp_name,age=emp_age,salary=emp_salary,passport=emp_pass,department=
emp_dep,email="test@email.com",managerId=newId)
                self.dealer.addManager(manager=man)
                classes.append_data_to_file(self.dealer)
                self.show_confirmation_data_message("Your Employee ID is : "+str(newId))
            elif emp_job=="Sales Man":
                newId=len(self.dealer.managersList)+len(self.dealer.salesManList)+1
                items=[]
                for emp in self.dealer.managersList:
                    if emp.isDeleted==False:
                        items.append(str(emp.managerId))

                dialog= MyDialog(self.main, "Select Manager ID:", items)
            manId=dialog.result
            try:
               int(manId)
            except Exception:
               manId="None"
            if self.dealer.search_manager_by_id(manId):

emp=classes.SalesMan(name=emp_name,age=emp_age,salary=emp_salary,passport=emp_pass,department
=emp_dep,email="test@email.com",managerId=manId,empId=newId)
               self.dealer.addSalesMan(emp)
               classes.append_data_to_file(self.dealer)
               self.show_confirmation_data_message("Your Employee ID is : "+str(newId))
               self.populate()
            else:
               self.show_invalid_data_message("Manager not found!!!")
      else:
         self.show_invalid_data_message("Invalid Data!!!!!!\nPlease check your input!!!")

   def modifyEmployee(self):

      emp_name=self.name.get()
      emp_age=self.age.get()
      emp_salary=self.salary.get()
      emp_pass=self.passport.get()
      emp_dep=self.depart.get()
      emp_job=self.jobTitle.get()
      if emp_job=="Manager":
         if
self.dealer.modifyManager(managerId=self.empId_modify,newSalary=emp_salary,newAge=emp_age,newDe
p=emp_dep,newName=emp_name,newPass=emp_pass):
            self.show_confirmation_data_message("Data has been modified successfully!!")
         else:
            self.show_invalid_data_message("Unfotunately some error occured\nPlease try again")

      else:
         if
self.dealer.modifySalesMan(empId=self.empId_modify,newSalary=emp_salary,newAge=emp_age,newDep=e
mp_dep,newName=emp_name,newPass=emp_pass):
            self.show_confirmation_data_message("Data has been modified successfully!!")
         else:
            self.show_invalid_data_message("Unfotunately some error occured\nPlease try again")
      self.name.delete(0, tk.END)
      self.age.delete(0, tk.END)
      self.passport.delete(0, tk.END)
      self.salary.delete(0, tk.END)
      self.depart.delete(0, tk.END)
      self.jobTitle.current(0)
      self.emp_modify_btn.configure(state="disabled")
      self.jobTitle.configure(state="normal")
      self.emp_add_btn.configure(state="normal")



   #deleteing employee
   def delEmployee(self):
      """
   Deletes an employee from the system.

   Displays a list of active employees (salesmen and managers) and prompts the user to select an ID to delete.
   If the selected ID is found in the system, the corresponding employee is deleted and a success message is displayed.
   If the selected ID is not found, or if a manager is being deleted and there are still salesmen under them, an error message is displayed.
   Finally, clears the input fields and updates the system file.
   """
      items = []
      for emp in self.dealer.salesManList:
         if emp.isDeleted==False:
            items.append(str(emp.empId))
      for emp in self.dealer.managersList:
         if emp.isDeleted==False:
            items.append(str(emp.managerId))

      dialog = MyDialog(self.main, "Select an option", items)

      if self.dealer.deleteSalesMan(str(dialog.result)) or self.dealer.deleteManager(str(dialog.result)):
         self.show_confirmation_data_message("Employee has been deleted Successfully!!")
         classes.append_data_to_file(self.dealer)
         self.populate()
      else:
         self.show_invalid_data_message("Record not found!!\nIf you are trying to remove a manager\nplease
make sure that there is no sales man related to that manager.")
      self.name.delete(0, tk.END)
      self.age.delete(0, tk.END)
      self.passport.delete(0, tk.END)
      self.salary.delete(0, tk.END)
      self.depart.delete(0, tk.END)
      self.jobTitle.current(0)
      self.emp_add_btn.configure(state="normal")
      self.emp_modify_btn.configure(state="disabled")


   def searchEmployee(self):
      """
   Searches for an employee in the system.

   Displays a list of active employees (salesmen and managers) and prompts the user to select an ID to search
for.
   If the selected ID is found in the system, the employee's details are displayed and the input fields are
populated.
   If the selected ID is not found, an error message is displayed.
   If a manager is found, displays the manager's name, salary, ID number, department, profit, age and
passport.
   If a sales man is found, displays the sales man's name, salary, ID number, department, profit, age and
passport.
   """
      items = []
      for man in self.dealer.managersList:
         if man.isDeleted==False:
            items.append(str(man.managerId))
      for emp in self.dealer.salesManList:
         if emp.isDeleted==False:
            items.append(str(emp.empId))
      d=MyDialog(self.main, "Select an option", items)
      selected_item=d.result

      message=""
```

```
      if self.dealer.search_manager_by_id(selected_item) or self.dealer.searchSalesMan(selected_item) is not
None:
        if self.dealer.search_manager_by_id(selected_item):
          for man in self.dealer.managersList:
            if int(man.managerId)==int(selected_item):
              name=str(man.name)
              salary=float(man.salary)
              idNmbr=str(man.managerId)
              dep=str(man.department)
              profit=float(man.profit)
              job="Manager"
              age=man.age
              passport=man.passport
              self.jobTitle.current(0)

        elif self.dealer.searchSalesMan(selected_item) is not None:
          emp=self.dealer.searchSalesMan(selected_item)
          name=str(emp.name)
          salary=float(emp.salary)
          idNmbr=str(emp.empId)
          dep=str(emp.department)
          profit=float(emp.profit)
          job="Sales Man"
          age=emp.age
          passport=emp.passport
          self.jobTitle.current(1)
        message=f"Name : {name}\nID Number : {idNmbr}\nDepartment : {dep}\nJob : {job}\nBasic Salary :
{salary}\nTotal Salary : {round(salary+profit,2)}\nPassport No. : {passport}"
        self.empId_modify=idNmbr
        self.show_confirmation_data_message(message)

        self.name.insert(0,str(name))
        self.age.insert(0,str(age))
        self.passport.insert(0,str(passport))
        self.salary.insert(0,str(salary))
        self.depart.insert(0,str(dep))
        self.emp_modify_btn.configure(state="normal")
        self.jobTitle.configure(state="disabled")
        self.emp_add_btn.configure(state="disabled")
```

Sales management functions in GUI

```
#####################################  SALES
#####################################################
  #adding the sales
  def add_sale(self):
    """Add a new sale made by an employee to the dealer's records.

  The function retrieves the selected employee ID and car ID from the UI,
  creates a new Sales object with the current date, and attempts to add the
  sale to the dealer's records. If the sale is added successfully, a
  confirmation message is displayed. If there is an issue with adding the
  sale, an error message is displayed.

  Args:
    self: The object itself.

  Returns:
    None.
  """
    empId=self.combobox4.get()

    carId=self.combobox5.get()

    today = date.today()
    sale=classes.Sales(carId=carId,empId=empId,date=today)
    if self.dealer.addSales(sale=sale):
      self.show_confirmation_data_message("Sale has been made successfully!!")
      classes.append_data_to_file(self.dealer)
    else:
      self.show_invalid_data_message("There have been some issues in performing desired operation!!")

  #searching sales of employeee
  def sales_search(self):
    """Search for sales made by a specific employee in the dealer's records.

  The function retrieves a list of employee IDs from the dealer's records,
  displays them in a dropdown menu, and prompts the user to select an ID.
  If an ID is selected, the function retrieves the employee's name and all
  sales made by that employee from the dealer's records. It then looks up
```

```
the name and price of each car that was sold, and displays a message with
the employee's name, each car's name and price, and the total sales made
by the employee. If the employee does not exist in the system, or has no
sales records, an error message is displayed.

Args:
  self: The object itself.

Returns:
  None.
"""
  classes.load_from_pickle_file()
  items = []
  for emp in self.dealer.salesManList:
    if emp.isDeleted==False:
      items.append(str(emp.empId))
  d=MyDialog(self.main, "Select an option", items)
  selected_item=d.result

  message=""
  for emp in self.dealer.salesManList:
    if int(emp.empId)==int(selected_item):
      name=emp.name
      sales=self.dealer.searchSalesByEmpId(selected_item)

      for sale in sales:
        print(sale.empId)
        for car in self.dealer.carList:

          if car.carId==sale.carId:

            carId=car.name
            carprice=car.price
            break
        message=message+name+"\t"+carId+"\t"+carprice+"\n"
      self.show_confirmation_data_message(message=message)
      return
    self.show_invalid_data_message("Employee does not exist in system!!\nEmployee is either deleted from
system\nor was never part of the system")
```

Car management functions in the GUI

```
############################################ CAR Managemnt
##########################################################
  #method to delete a car
  def del_car(self):
    """
  Method to delete a car from the dealer's inventory.

  Prompts the user for the car ID and attempts to delete the car from the
  inventory using the dealer's deleteCar() method. If successful, updates
  the inventory file, clears the UI input fields, and displays a
  confirmation message. If the car is not found, displays an error message.

  Returns:
    None
  """
    carList=[]
    for car in self.dealer.carList:
      if car.isDeleted==False:
        carList.append(str(car.carId))

    d=MyDialog(self.main, "Select an option", carList)
    selected_item=d.result

    if self.dealer.deleteCar(carId=selected_item):
      self.show_confirmation_data_message("Car has been Deleted Successfully!!")
      classes.append_data_to_file(self.dealer)
      self.populate()

    else:
      self.show_invalid_data_message("Car from this Id not found in system!!")
    self.car_price.delete(0,tk.END)
    self.car_model.delete(0,tk.END)
    self.car_name.delete(0,tk.END)
    self.car_type.current(0)
```

```python
        self.car_id.delete(0,tk.END)
        self.car_add_btn.configure(state="normal")
        self.car_id.configure(state="normal")
        self.car_modify_btn.configure(state="disabled")

    #search car method
    def search_car(self):
        """
    Method to search for a car in the dealer's inventory.

    Prompts the user for the car ID and attempts to find the car in the
    inventory using the dealer's searchCar() method. If the car is found,
    displays its information in the UI input fields and sets the Add/Modify
    button to Modify. If the car is not found, displays an error message.

    Returns:
        None
    """
        carList=[]
        for car in self.dealer.carList:
            if car.isDeleted==False:
                carList.append(str(car.carId))

        d=MyDialog(self.main, "Select an option", carList)
        selected_item=d.result

        car=self.dealer.searchCar(carId=selected_item)
        if car is not None:
            self.car_price.delete(0,tk.END)
            self.car_model.delete(0,tk.END)
            self.car_name.delete(0,tk.END)
            self.car_type.current(0)
            self.car_id.delete(0,tk.END)
            self.car_price.insert(0,(str(car.price)))
            self.car_model.insert(0,(str(car.model)))
            self.car_name.insert(0,(str(car.name)))
            if car.type=="Sedan":
                self.car_type.current(0)
            elif car.type=="Hatch":
```

```
            self.car_type.current(1)
        else:
            self.car_type.current(2)
        self.car_id.insert(0,str(selected_item))
        self.car_modify_btn.configure(state="normal")
        self.car_add_btn.configure(state="disabled")
        self.car_id.configure(state="disabled")
    else:
        self.show_invalid_data_message("Car from this Id not found in system!!")


#adding cars to the list using methods from classes.py
def addCar_toList(self):
    """
Method to add a car to the dealer's inventory.

Collects information about the car from the UI input fields, validates
the data using the dealer's validation methods, and either adds the car
to the inventory If the data is invalid, displays an error message.
If the action is successful, updates the inventory file, clears the UI
input fields, and displays a confirmation message.

Returns:
    None
"""
    name=self.car_name.get()
    model=self.car_model.get()
    carId=self.car_id.get()
    carType=self.car_type.get()
    price=self.car_price.get()

    #validating the data
    if self.dealer.validate_string(name) and self.dealer.validate_string(model) and
self.dealer.is_valid_price(price) and self.dealer.validate_car_id(carId):
        if True:
            car=classes.Car(name=name,carId=carId,type=carType,model=model,price=price)
            self.dealer.addCar(car=car)
            classes.append_data_to_file(self.dealer)
            self.show_confirmation_data_message("Operation Successful!!!\nCar has been Added to System!!!")
```

```python
                self.populate()
        else:
            self.show_invalid_data_message("Please check the Input!!!")

    def modify_car(self):
        name=self.car_name.get()
        model=self.car_model.get()
        carId=self.car_id.get()
        carType=self.car_type.get()
        price=self.car_price.get()
        if self.dealer.validate_string(name) and self.dealer.validate_string(model) and
self.dealer.is_valid_price(price) :

self.dealer.modifyCar(carId=carId,newPrice=price,newName=name,newModel=model,newType=carType)
            self.show_confirmation_data_message("Data modified Successfully!!")
            self.car_price.delete(0,tk.END)
            self.car_model.delete(0,tk.END)
            self.car_name.delete(0,tk.END)
            self.car_type.current(0)
            self.car_id.delete(0,tk.END)
            self.car_add_btn.configure(state="normal",command=self.addCar_toList)
            self.car_id.configure(state="normal")
            self.car_modify_btn.configure(state="disabled")
        else:
            self.show_invalid_data_message("Please check the Input!!!")
```