Coding Challenges: CareerHub, The Job Board

By-Kunal Sudhir Mishra

Problem Statement: A Job Board scenario is a digital platform or system that facilitates the process of job searching and recruitment. In this scenario, various stakeholders, such as job seekers, companies, and recruiters, use the platform to post, search for, and apply to job opportunities.

Task- 1. Create and implement the mentioned class and the structure in your application.

Source Code:

Joblisting.py

```python
from datetime import datetime

5 usages
class JobListing:
    def __init__(self, job_id, company_id, job_title, job_description, job_location, salary, job_type, posted_date):
        self.job_id = job_id
        self.company_id = company_id
        self.job_title = job_title
        self.job_description = job_description
        self.job_location = job_location
        self.salary = salary
        self.job_type = job_type
        self.posted_date = posted_date
        self.applicants = []  # List to store applicants

    2 usages
    def apply(self, applicant_id, cover_letter):
        application_date = datetime.now()
        application = {"applicant_id": applicant_id, "cover_letter": cover_letter, "application_date": application_date}
        self.applicants.append(application)
```

```python
    2 usages
    def get_applicants(self):
        return self.applicants

    1 usage
    def get_job_details(self):
        return {
            "JobID": self.job_id,
            "CompanyID": self.company_id,
            "JobTitle": self.job_title,
            "JobDescription": self.job_description,
            "JobLocation": self.job_location,
            "Salary": self.salary,
            "JobType": self.job_type,
            "PostedDate": self.posted_date
        }
```

Company.py

main.py | joblisting.py | company.py | applicant.py | jobapplication.py

```python
from datetime import datetime
from joblisting import JobListing

3 usages
class Company:

    def __init__(self, company_id, company_name, location):
        self.company_id = company_id
        self.company_name = company_name
        self.location = location
        self.job_listings = []  # List to store job listings

    2 usages
    def post_job(self, job_title, job_description, job_location, salary, job_type):
        job_id = len(self.job_listings) + 1
        posted_date = datetime.now()
        job_listing = JobListing(job_id, self.company_id, job_title, job_description, job_location, salary,
                                 job_type, posted_date)
        self.job_listings.append(job_listing)
        return job_listing

```

```python
2 usages
def get_jobs(self):
    return self.job_listings


1 usage
def get_company_details(self):
    return {
        "CompanyID": self.company_id,
        "CompanyName": self.company_name,
        "Location": self.location
    }
```

Applicant.py

```python
from datetime import datetime
from jobapplication import JobApplication

3 usages
class Applicant:
    def __init__(self, applicant_id, first_name, last_name, email, phone, resume):
        self.applicant_id = applicant_id
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.phone = phone
        self.resume = resume
        self.applied_jobs = []  # List to store applied jobs

    2 usages
    def create_profile(self, email, first_name, last_name, phone):
        self.email = email
        self.first_name = first_name
        self.last_name = last_name
        self.phone = phone
```

```python
    2 usages                                                                    ⚠ 3
    def apply_for_job(self, job_id, cover_letter):
        application_id = len(self.applied_jobs) + 1
        application_date = datetime.now()
        job_application = JobApplication(application_id, job_id, self.applicant_id, application_date, cover_letter)
        self.applied_jobs.append(job_application)
        return job_application

    1 usage
    def get_applied_jobs(self):
        return self.applied_jobs

    1 usage
    def get_applicant_details(self):
        return {
            "ApplicantID": self.applicant_id,
            "FirstName": self.first_name,
            "LastName": self.last_name,
            "Email": self.email,
            "Phone": self.phone,
            "Resume": self.resume
        }
```

Jobapplication.py

```python
from datetime import datetime

5 usages
class JobApplication:
    def __init__(self, application_id, job_id, applicant_id, application_date, cover_letter):
        self.application_id = application_id
        self.job_id = job_id
        self.applicant_id = applicant_id
        self.application_date = application_date
        self.cover_letter = cover_letter

    2 usages
    def get_application_details(self):
        return {
            "ApplicationID": self.application_id,
            "JobID": self.job_id,
            "ApplicantID": self.applicant_id,
            "ApplicationDate": self.application_date,
            "CoverLetter": self.cover_letter
        }
```

Driver code

Main.py

```python
from datetime import datetime
from jobapplication import JobApplication
from applicant import Applicant
from company import Company
from joblisting import JobListing

# Example usage of JobApplication
job_application = JobApplication(
    application_id=1,
    job_id=1,
    applicant_id=1,
    application_date=datetime.now(),
    cover_letter="I am interested in this position."
)

application_details = job_application.get_application_details()
print("Job Application Details:", application_details)
print()

# Example usage of Applicant
applicant = Applicant(
    applicant_id=1,
    first_name="kunal",
```

```
    last_name="Mishra",
    email="kunal.22@gmail.com",
    phone="9823567128",
    resume="path/to/resume.pdf"
)

applicant.create_profile(
    email="akash.ded@gmail.com",
    first_name="akash",
    last_name="dedhi",
    phone="9234567810"
)

applicant.apply_for_job(
    job_id=1,
    cover_letter="I am applying for the job."
)

print()

# Example usage of Company
company = Company(
    company_id=1,
    company_name="Tech Solutions",
    location="City Center"
)

company.post_job(
    job_title="Software Engineer",
    job_description="Developing innovative software solutions",
    job_location="Tech Park",
    salary=80000.0,
    job_type="Full-time"
)

jobs_posted = company.get_jobs()
print("Jobs Posted by the Company:", jobs_posted)
print()

# Example usage of JobListing
job_listing = JobListing(
    job_id=1,
    company_id=1,
    job_title="Data Analyst",
    job_description="Analyzing and interpreting complex data sets",
    job_location="Downtown",
    salary=60000.0,
    job_type="Part-time",
    posted_date=datetime.now()
)

job_listing.apply(applicant_id=1, cover_letter="I have relevant experience.")
applicants_list = job_listing.get_applicants()
print("Applicants for the Job Listing:", applicants_list)
```

Output:

```
C:\Users\Kunal\PycharmProjects1\CareerHub\.venv\Scripts\python.exe C:\Users\Kunal\PycharmProjects1\CareerHub\main.py
Job Application Details: {'ApplicationID': 1, 'JobID': 1, 'ApplicantID': 1, 'ApplicationDate': datetime.datetime(2024, 2, 6, 11, 7, 32, 74440),
Applied Jobs: [<jobapplication.JobApplication object at 0x000001878234EA20>]
Applicant Details: {'ApplicantID': 1, 'FirstName': 'NewJohn', 'LastName': 'NewDoe', 'Email': 'newemail@example.com', 'Phone': '987654321', 'Res
Applicants: [{'applicant_id': 123, 'cover_letter': 'I am excited about this opportunity!', 'application_date': datetime.datetime(2024, 2, 6, 11
Job Details: {'JobID': 1, 'CompanyID': 1, 'JobTitle': 'Software Engineer', 'JobDescription': 'Developing awesome software', 'JobLocation': 'Tec
Posted Jobs: [<joblisting.JobListing object at 0x000001878234E2D0>]
Company Details: {'CompanyID': 1, 'CompanyName': 'Tech Innovators', 'Location': 'Tech City'}
Job Application Details: {'ApplicationID': 1, 'JobID': 1, 'ApplicantID': 1, 'ApplicationDate': datetime.datetime(2024, 2, 6, 11, 7, 32, 79050),


Jobs Posted by the Company: [<joblisting.JobListing object at 0x000001878234E3C0>]

Applicants for the Job Listing: [{'applicant_id': 1, 'cover_letter': 'I have relevant experience.', 'application_date': datetime.datetime(2024,

Process finished with exit code 0
```

```
74440), 'CoverLetter': 'I am interested in this position.'}

1', 'Resume': 'Resume.pdf'}
2, 6, 11, 7, 32, 79050)}]
n': 'Tech City', 'Salary': 80000.0, 'JobType': 'Full-time', 'PostedDate': datetime.datetime(2024, 2, 6, 11, 7, 32, 79050)}


79050), 'CoverLetter': 'I am interested in this position.'}




me(2024, 2, 6, 11, 7, 32, 79050)}]
```

Task-2 DatabaseManager Class

Methods:

• InitializeDatabase(): Initializes the database schema and tables.

• InsertJobListing(job: JobListing): Inserts a new job listing into the "Jobs" table.

• InsertCompany(company: Company): Inserts a new company into the "Companies" table. •

InsertApplicant(applicant: Applicant): Inserts a new applicant into the "Applicants" table.

• InsertJobApplication(application: JobApplication): Inserts a new job application into the "Applications" table.

• GetJobListings(): List: Retrieves a list of all job listings.

• GetCompanies(): List: Retrieves a list of all companies.

• GetApplicants(): List: Retrieves a list of all applicants.

• GetApplicationsForJob(jobID: int): List: Retrieves a list of job applications for a specific job listing.

Databasemanager.py

```python
import mysql.connector
from joblisting import JobListing
from company import Company
from applicant import Applicant
from jobapplication import JobApplication
from datetime import datetime
2 usages
class DatabaseConnector:
    def __init__(self):
        self.connection = None
    1 usage
    def create_tables(self):
        cursor = self.connection.cursor()

    1 usage
    def open_connection(self):
        self.connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root"
        )
        self.create_database()
        self.connection.database = "careerhub"
        self.create_tables()

    1 usage
    def close_connection(self):
        if self.connection:
            self.connection.close()

    1 usage
    def create_database(self):
        cursor = self.connection.cursor()
        try:
            cursor.execute("CREATE DATABASE IF NOT EXISTS careerhub")
        except Exception as e:
            print(f"Error creating database: {e}")
        finally:
```

```
        finally:
            cursor.close()


    def initialize_database(self):



        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS Jobs (
                JobID INTEGER PRIMARY KEY,
                CompanyID INTEGER,
                JobTitle TEXT,
                JobDescription TEXT,
                JobLocation TEXT,
                Salary REAL,
                JobType TEXT,
                PostedDate DATETIME
            )
```

```
        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS Companies (
                CompanyID INTEGER PRIMARY KEY,
                CompanyName TEXT,
                Location TEXT
            )
        ''')

        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS Applicants (
                ApplicantID INTEGER PRIMARY KEY,
                FirstName TEXT,
                LastName TEXT,
                Email TEXT,
                Phone TEXT,
                Resume TEXT
            )
        ''')
```

```
        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS Applications (
                ApplicationID INTEGER PRIMARY KEY,
                JobID INTEGER,
                ApplicantID INTEGER,
                ApplicationDate DATETIME,
                CoverLetter TEXT,
                FOREIGN KEY (JobID) REFERENCES Jobs(JobID),
                FOREIGN KEY (ApplicantID) REFERENCES Applicants(ApplicantID)
            )
        ''')

        self.connection.commit()

    def insert_job_listing(self, job):
        # Implement the insert_job_listing method
        self.cursor.execute('''
            INSERT INTO Jobs (CompanyID, JobTitle, JobDescription, JobLocation, Salary, JobType, PostedDate)
```

```
        ''', (job.company_id, job.job_title, job.job_description, job.job_location, job.salary,
              job.job_type, job.posted_date))
        self.connection.commit()

    def insert_company(self, company):
        # Implement the insert_company method
        self.cursor.execute('''
            INSERT INTO Companies (CompanyName, Location)
            VALUES (?, ?)
        ''', (company.company_name, company.location))
        self.connection.commit()

    def insert_applicant(self, applicant):
        # Implement the insert_applicant method
        self.cursor.execute('''
            INSERT INTO Applicants (FirstName, LastName, Email, Phone, Resume)
            VALUES (?, ?, ?, ?, ?)
```

```
        ''', (application.job_id, application.applicant_id, application.application_date, application.cover_letter))
        self.connection.commit()

    def get_job_listings(self):
        # Implement the get_job_listings method
        self.cursor.execute('SELECT * FROM Jobs')
        rows = self.cursor.fetchall()
        job_listings = [JobListing(*row) for row in rows]
        return job_listings

    def get_companies(self):
        self.cursor.execute('SELECT * FROM Companies')
        rows = self.cursor.fetchall()
        companies = [Company(*row) for row in rows]
        return companies

    def get_applicants(self):
        self.cursor.execute('SELECT * FROM Applicants')
        rows = self.cursor.fetchall()
```

```
    def get_applicants(self):
        self.cursor.execute('SELECT * FROM Applicants')
        rows = self.cursor.fetchall()
        applicants = [Applicant(*row) for row in rows]
        return applicants

    def get_applications_for_job(self, job_id):
        self.cursor.execute('SELECT * FROM Applications WHERE JobID = ?', (job_id,))
        rows = self.cursor.fetchall()
        applications = [JobApplication(*row) for row in rows]
        return applications
```

Main.py

```
from databasemanager import DatabaseConnector

try:

    db_connector = DatabaseConnector()


    db_connector.open_connection()

    db_connector.close_connection()

except Exception as e:
    print(f"Error: {e}")
```

Task-3. Exceptions handling

Create and implement the following exceptions in your application.

Applicant.py

```python
from exceptions import InvalidEmailError

4 usages
class Applicant:
    def __init__(self, applicant_id, first_name, last_name, email, phone, resume):
        self.applicant_id = applicant_id
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.phone = phone
        self.resume = resume

    def create_profile(self, email, first_name, last_name, phone):
        try:

            self.validate_email(email)
            self.validate_email(email)
        except InvalidEmailError as e:
            print(f"Error: {e}")

    1 usage
    def validate_email(self, email):

        if "@" not in email or "." not in email:
            raise InvalidEmailError()
```

Databasemanager.py

```python
from exceptions import NegativeSalaryError

class DatabaseManager:
    def __init__(self, database_name):
        self.connection = mysql.connect(database_name)
        self.cursor = self.connection.cursor()



    def calculate_average_salary(self):
        try:
            self.cursor.execute('SELECT Salary FROM Jobs')
            salaries = self.cursor.fetchall()
            valid_salaries = [salary[0] for salary in salaries if salary[0] >= 0]
            if not valid_salaries:
                raise NegativeSalaryError("No valid salaries found.")
            average_salary = sum(valid_salaries) / len(valid_salaries)
            return average_salary
```

```python
            return average_salary
        except NegativeSalaryError as e:
            print(f"Error calculating average salary: {e}")
        except Exception as e:
            print(f"Error calculating average salary: {e}")
```

Applicant.py

```python
from exceptions import FileUploadError
4 usages
class Applicant:
    def __init__(self):



        def upload_resume(self, resume_file):
            try:

                with open(resume_file, 'rb') as file:

                    pass
            except FileNotFoundError:
                raise FileUploadError("File not found. Please check the file path.")
            except IOError as e:
                raise FileUploadError(f"Error uploading file: {e}")
            except Exception as e:
                raise FileUploadError(f"Unknown error occurred: {e}")
```

Jobapplication.py

```python
application_details = job_application.get_application_details()
print("Job Application Details:", application_details)
from datetime import datetime
from exceptions import DeadlineExceededError

6 usages
class JobApplication:
    def __init__(self, application_id, job_id, applicant_id, application_date, cover_letter, deadline):
        self.application_id = application_id
        self.job_id = job_id
        self.applicant_id = applicant_id
        self.application_date = application_date
        self.cover_letter = cover_letter
        self.deadline = deadline

    def submit_application(self):
        current_date = datetime.now()
        if current_date > self.deadline:
            raise DeadlineExceededError("Application deadline has passed. Applications are no longer accepted.")
```

```python
def submit_application(self):
    current_date = datetime.now()
    if current_date > self.deadline:
        raise DeadlineExceededError("Application deadline has passed. Applications are no longer accepted.")
    else:
        # Proceed with application submission
        pass
```

Databasemanager.py

```
def create_tables(self):
    try:
        cursor = self.connection.cursor()
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS Jobs (
                JobID INTEGER PRIMARY KEY,
                CompanyID INTEGER,
                JobTitle TEXT,
                JobDescription TEXT,
                JobLocation TEXT,
                Salary REAL,
                JobType TEXT,
                PostedDate DATETIME
            )
        ''')

        cursor.execute('''
            CREATE TABLE IF NOT EXISTS Companies (
                CompanyID INTEGER PRIMARY KEY,
                CompanyName TEXT,
```

```
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS Applicants (
                ApplicantID INTEGER PRIMARY KEY,
                FirstName TEXT,
                LastName TEXT,
                Email TEXT,
                Phone TEXT,
                Resume TEXT
            )
        ''')

        cursor.execute('''
            CREATE TABLE IF NOT EXISTS Applications (
                ApplicationID INTEGER PRIMARY KEY,
                JobID INTEGER,
                ApplicantID INTEGER,
                ApplicationDate DATETIME,
                CoverLetter TEXT,
                FOREIGN KEY (JobID) REFERENCES Jobs(JobID),
                FOREIGN KEY (ApplicantID) REFERENCES Applicants(ApplicantID)
```

```
            self.connection.commit()
            cursor.close()
        except mysql.connector.Error as e:
            raise DatabaseConnectionError(str(e))
```

Exceptions.py

```
import mysql.connector

3 usages
class InvalidEmailError(Exception):
    def __init__(self, message="Invalid email format. Please provide a valid email address."):
        self.message = message
        super().__init__(self.message)

3 usages
class NegativeSalaryError(Exception):
    def __init__(self, message="Salary must be a non-negative value."):
        self.message = message
        super().__init__(self.message)

4 usages
class FileUploadError(Exception):
    def __init__(self, message="Error uploading file."):
        self.message = message
        super().__init__(self.message)
```

```
2 usages
class DeadlineExceededError(Exception):
    def __init__(self, message="Application deadline has passed. Applications are no longer accepted."):
        self.message = message
        super().__init__(self.message)

3 usages
class DatabaseConnectionError(Exception):
    def __init__(self, message="Error establishing a database connection."):
        self.message = message
        super().__init__(self.message)

class DatabaseConnector:
    def __init__(self):
        self.connection = None

    def open_connection(self, host, user, password, database):
        try:
            self.connection = mysql.connector.connect(
                host=host,
```

```
            host=host,
            user=user,
            password=password,
            database=database
        )
    except mysql.connector.Error as e:
        raise DatabaseConnectionError(str(e))

def close_connection(self):
    if self.connection:
        self.connection.close()

def create_database(self):
    try:
        cursor = self.connection.cursor()
        cursor.execute("CREATE DATABASE IF NOT EXISTS careerhub")
        cursor.close()
        self.connection.database = "careerhub"
    except mysql.connector.Error as e:
```

```
        self.connection.commit()
        cursor.close()
    except mysql.connector.Error as e:
        raise DatabaseConnectionError(str(e))
```

Task- 4. Database Connectivity

Create and implement the following tasks in your application.

• Job Listing Retrieval:

```
def retrieve_job_listings(self):
    try:
        self.connect()
        self.cursor.execute("SELECT JobTitle, CompanyID, Salary FROM Jobs")
        job_listings = self.cursor.fetchall()
        return job_listings
    except DatabaseConnectionError as e:
        print(f"Database Connection Error: {e}")
    except DatabaseQueryError as e:
        print(f"Database Query Error: {e}")
    finally:
        self.disconnect()
```

• **Applicant Profile Creation:**

```
def create_applicant_profile(self, first_name, last_name, email, phone, resume):
    try:
        self.connect()
        self.execute_query(
            "INSERT INTO Applicants (FirstName, LastName, Email, Phone, Resume) VALUES (1, 2, 3, 4, 5)",
            (first_name, last_name, email, phone, resume))
        print("Applicant profile created successfully.")
    except DatabaseConnectionError as e:
        print(f"Database Connection Error: {e}")
    except DatabaseQueryError as e:
        print(f"Database Query Error: {e}")
    finally:
        self.disconnect()
```

- **Job Application Submission:**

```
def submit_job_application(self, job_id, applicant_id, application_date, cover_letter):
    try:
        self.connect()
        self.execute_query(
            "INSERT INTO Applications (JobID, ApplicantID, ApplicationDate, CoverLetter) VALUES (1, 3, 5, 6)",
            (job_id, applicant_id, application_date, cover_letter))
        print("Job application submitted successfully.")
    except DatabaseConnectionError as e:
        print(f"Database Connection Error: {e}")
    except DatabaseQueryError as e:
        print(f"Database Query Error: {e}")
    finally:
        self.disconnect()
```

- **Company Job Posting:**

```
def post_job_listing(self, company_id, job_title, job_description, job_location, salary, job_type, posted_date):
    try:
        self.connect()
        self.execute_query("INSERT INTO Jobs (CompanyID, JobTitle, JobDescription, JobLocation, "
                           "Salary, JobType, PostedDate) VALUES (1, 2, 3, 4, 5, 6, 7)",
                           (company_id, job_title, job_description, job_location, salary, job_type, posted_date))
        print("Job listing posted successfully.")
    except DatabaseConnectionError as e:
        print(f"Database Connection Error: {e}")
    except DatabaseQueryError as e:
        print(f"Database Query Error: {e}")
    finally:
        self.disconnect()
```

- **Salary Range Query:**

```python
def search_job_listings_by_salary_range(self, min_salary, max_salary):
    try:
        self.connect()
        self.cursor.execute("SELECT JobTitle, CompanyID, Salary FROM Jobs WHERE Salary BETWEEN ? AND ?",
                            (min_salary, max_salary))
        job_listings = self.cursor.fetchall()
        return job_listings
    except DatabaseConnectionError as e:
        print(f"Database Connection Error: {e}")
    except DatabaseQueryError as e:
        print(f"Database Query Error: {e}")
    finally:
        self.disconnect()
```

------*-------