**Name:** Keerthika Nagarajan **Superset ID:** 5370583

**College:** Saveetha Engineering College

# Coding Challenges

## CareerHub, The Job Board

**entity:**

**Applicant.py:**

```
class Applicant:
    def __init__(self, applicant_id=None, first_name=None, last_name=None, email=None,
                 phone=None, resume=None, experience_years=0, city=None, state=None):
        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.__experience_years = experience_years
        self.__city = city
        self.__state = state

    # Getters and Setters
    @property
    def applicant_id(self):
        return self.__applicant_id

    @applicant_id.setter
    def applicant_id(self, value):
        self.__applicant_id = value

    @property
    def first_name(self):
        return self.__first_name

    @first_name.setter
    def first_name(self, value):
        self.__first_name = value

    @property
    def last_name(self):
        return self.__last_name

    @last_name.setter
    def last_name(self, value):
        self.__last_name = value

    @property
    def email(self):
        return self.__email
```

```
@email.setter
def email(self, value):
    if '@' not in value or '.' not in value:
        raise ValueError("Invalid email format")
    self.__email = value

@property
def phone(self):
    return self.__phone

@phone.setter
def phone(self, value):
    self.__phone = value

@property
def resume(self):
    return self.__resume

@resume.setter
def resume(self, value):
    self.__resume = value

@property
def experience_years(self):
    return self.__experience_years

@experience_years.setter
def experience_years(self, value):
    if value < 0:
        raise ValueError("Experience years cannot be negative")
    self.__experience_years = value

@property
def city(self):
    return self.__city

@city.setter
def city(self, value):
    self.__city = value

@property
def state(self):
    return self.__state

@state.setter
def state(self, value):
    self.__state = value

def __str__(self):
```

```
        return f"Applicant(applicant_id={self.applicant_id}, name={self.first_name} {self.last_name},
email={self.email})"
```

**Company.py:**

```
class Company:
    def __init__(self, company_id=None, company_name=None, location=None):
        self.__company_id = company_id
        self.__company_name = company_name
        self.__location = location

    # Getters and Setters
    @property
    def company_id(self):
        return self.__company_id

    @company_id.setter
    def company_id(self, value):
        self.__company_id = value

    @property
    def company_name(self):
        return self.__company_name

    @company_name.setter
    def company_name(self, value):
        self.__company_name = value

    @property
    def location(self):
        return self.__location

    @location.setter
    def location(self, value):
        self.__location = value

    def __str__(self):
        return f"Company(company_id={self.company_id}, name={self.company_name},
location={self.location})"
```

**JobApplication.py:**

```
from datetime import datetime


class JobApplication:
    def __init__(self, application_id=None, job_id=None, applicant_id=None,
                 application_date=None, cover_letter=None):
        self.__application_id = application_id
        self.__job_id = job_id
```

```
        self.__applicant_id = applicant_id
        self.__application_date = application_date if application_date else datetime.now()
        self.__cover_letter = cover_letter

    # Getters and Setters
    @property
    def application_id(self):
        return self.__application_id

    @application_id.setter
    def application_id(self, value):
        self.__application_id = value

    @property
    def job_id(self):
        return self.__job_id

    @job_id.setter
    def job_id(self, value):
        self.__job_id = value

    @property
    def applicant_id(self):
        return self.__applicant_id

    @applicant_id.setter
    def applicant_id(self, value):
        self.__applicant_id = value

    @property
    def application_date(self):
        return self.__application_date

    @application_date.setter
    def application_date(self, value):
        self.__application_date = value

    @property
    def cover_letter(self):
        return self.__cover_letter

    @cover_letter.setter
    def cover_letter(self, value):
        self.__cover_letter = value

    def __str__(self):
        return f"JobApplication(application_id={self.application_id}, job_id={self.job_id},
applicant_id={self.applicant_id})"
```

**JobListing.py:**

```
from datetime import datetime


class JobListing:
    def __init__(self, job_id=None, company_id=None, job_title=None, job_description=None,
                 job_location=None, salary=None, job_type=None, posted_date=None):
        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 if posted_date else datetime.now()

    # Getters and Setters
    @property
    def job_id(self):
        return self.__job_id

    @job_id.setter
    def job_id(self, value):
        self.__job_id = value

    @property
    def company_id(self):
        return self.__company_id

    @company_id.setter
    def company_id(self, value):
        self.__company_id = value

    @property
    def job_title(self):
        return self.__job_title

    @job_title.setter
    def job_title(self, value):
        self.__job_title = value

    @property
    def job_description(self):
        return self.__job_description

    @job_description.setter
    def job_description(self, value):
        self.__job_description = value

    @property
    def job_location(self):
```

```
        return self.__job_location

    @job_location.setter
    def job_location(self, value):
        self.__job_location = value

    @property
    def salary(self):
        return self.__salary

    @salary.setter
    def salary(self, value):
        if value < 0:
            raise ValueError("Salary cannot be negative")
        self.__salary = value

    @property
    def job_type(self):
        return self.__job_type

    @job_type.setter
    def job_type(self, value):
        self.__job_type = value

    @property
    def posted_date(self):
        return self.__posted_date

    @posted_date.setter
    def posted_date(self, value):
        self.__posted_date = value

    def __str__(self):
        return f"JobListing(job_id={self.job_id}, title={self.job_title}, company_id={self.company_id},
salary={self.salary})"
```

**dao:**

**ICareerHubService.py:**

```
from abc import ABC, abstractmethod
from entity.Applicant import Applicant
from entity.Company import Company
from entity.JobApplication import JobApplication
from entity.JobListing import JobListing


class ICareerHubService(ABC):
    @abstractmethod
    def insert_job_listing(self, job_listing):
        pass
```

```
@abstractmethod
def insert_company(self, company):
    pass

@abstractmethod
def insert_applicant(self, applicant):
    pass

@abstractmethod
def insert_job_application(self, job_application):
    pass

@abstractmethod
def get_job_listings(self):
    pass

@abstractmethod
def get_companies(self):
    pass

@abstractmethod
def get_applicants(self):
    pass

@abstractmethod
def get_applications_for_job(self, job_id):
    pass

@abstractmethod
def get_jobs_by_salary_range(self, min_salary, max_salary):
    pass

@abstractmethod
def get_applications_by_applicant(self, applicant_id):
    pass

@abstractmethod
def get_companies_with_most_jobs(self):
    pass

@abstractmethod
def get_jobs_with_no_applications(self):
    pass

@abstractmethod
def get_jobs_by_title(self, title_keyword):
    pass
```

**CareerHubServiceImpl.py:**

```
from dao.ICareerHubService import ICareerHubService
from entity.Applicant import Applicant
from entity.Company import Company
from entity.JobApplication import JobApplication
from entity.JobListing import JobListing
from util.DBConnUtil import DBConnUtil
from exception.DatabaseConnectionException import DatabaseConnectionException
from exception.SalaryCalculationException import SalaryCalculationException
from datetime import datetime
import mysql.connector


class CareerHubServiceImpl(ICareerHubService):
    def __init__(self):
        self.connection = DBConnUtil.get_connection()

    def __del__(self):
        if self.connection and self.connection.is_connected():
            self.connection.close()

    def insert_job_listing(self, job_listing):
        try:
            cursor = self.connection.cursor()
            query = """
                INSERT INTO Jobs (CompanyID, JobTitle, JobDescription, JobLocation, Salary,
JobType, PostedDate)
                VALUES (%s, %s, %s, %s, %s, %s, %s)
            """
            values = (
                job_listing.company_id,
                job_listing.job_title,
                job_listing.job_description,
                job_listing.job_location,
                job_listing.salary,
                job_listing.job_type,
                job_listing.posted_date
            )
            cursor.execute(query, values)
            self.connection.commit()
            return cursor.lastrowid
        except mysql.connector.Error as e:
            raise DatabaseConnectionException(f"Error inserting job listing: {e}")
        except Exception as e:
            raise e

    def insert_company(self, company):
        try:
            cursor = self.connection.cursor()
            query = """
                INSERT INTO Companies (CompanyName, Location)
```

```
            VALUES (%s, %s)
        """
        values = (company.company_name, company.location)
        cursor.execute(query, values)
        self.connection.commit()
        return cursor.lastrowid
    except mysql.connector.Error as e:
        raise DatabaseConnectionException(f"Error inserting company: {e}")
    except Exception as e:
        raise e

def insert_applicant(self, applicant):
    try:
        cursor = self.connection.cursor()
        query = """
            INSERT INTO Applicants (FirstName, LastName, Email, Phone, Resume,
ExperienceYears, City, State)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
        """
        values = (
            applicant.first_name,
            applicant.last_name,
            applicant.email,
            applicant.phone,
            applicant.resume,
            applicant.experience_years,
            applicant.city,
            applicant.state
        )
        cursor.execute(query, values)
        self.connection.commit()
        return cursor.lastrowid
    except mysql.connector.Error as e:
        raise DatabaseConnectionException(f"Error inserting applicant: {e}")
    except Exception as e:
        raise e

def insert_job_application(self, job_application):
    try:
        cursor = self.connection.cursor()
        query = """
            INSERT INTO Applications (JobID, ApplicantID, CoverLetter)
            VALUES (%s, %s, %s)
        """
        values = (
            job_application.job_id,
            job_application.applicant_id,
            job_application.cover_letter
        )
        cursor.execute(query, values)
        self.connection.commit()
```

```
            return cursor.lastrowid
        except mysql.connector.Error as e:
            raise DatabaseConnectionException(f"Error inserting job application: {e}")
        except Exception as e:
            raise e

    def get_job_listings(self):
        try:
            cursor = self.connection.cursor(dictionary=True)
            query = "SELECT * FROM Jobs"
            cursor.execute(query)
            results = cursor.fetchall()

            job_listings = []
            for row in results:
                job = JobListing(
                    job_id=row['JobID'],
                    company_id=row['CompanyID'],
                    job_title=row['JobTitle'],
                    job_description=row['JobDescription'],
                    job_location=row['JobLocation'],
                    salary=row['Salary'],
                    job_type=row['JobType'],
                    posted_date=row['PostedDate']
                )
                job_listings.append(job)
            return job_listings
        except mysql.connector.Error as e:
            raise DatabaseConnectionException(f"Error retrieving job listings: {e}")
        except Exception as e:
            raise e

    def get_companies(self):
        try:
            cursor = self.connection.cursor(dictionary=True)
            query = "SELECT * FROM Companies"
            cursor.execute(query)
            results = cursor.fetchall()

            companies = []
            for row in results:
                company = Company(
                    company_id=row['CompanyID'],
                    company_name=row['CompanyName'],
                    location=row['Location']
                )
                companies.append(company)
            return companies
        except mysql.connector.Error as e:
            raise DatabaseConnectionException(f"Error retrieving companies: {e}")
        except Exception as e:
```

```
            raise e

    def get_applicants(self):
        try:
            cursor = self.connection.cursor(dictionary=True)
            query = "SELECT * FROM Applicants"
            cursor.execute(query)
            results = cursor.fetchall()

            applicants = []
            for row in results:
                applicant = Applicant(
                    applicant_id=row['ApplicantID'],
                    first_name=row['FirstName'],
                    last_name=row['LastName'],
                    email=row['Email'],
                    phone=row['Phone'],
                    resume=row['Resume'],
                    experience_years=row['ExperienceYears'],
                    city=row['City'],
                    state=row['State']
                )
                applicants.append(applicant)
            return applicants
        except mysql.connector.Error as e:
            raise DatabaseConnectionException(f"Error retrieving applicants: {e}")
        except Exception as e:
            raise e

    def get_applications_for_job(self, job_id):
        try:
            cursor = self.connection.cursor(dictionary=True)
            query = """
                SELECT a.*, app.ApplicationDate, app.CoverLetter
                FROM Applicants a
                JOIN Applications app ON a.ApplicantID = app.ApplicantID
                WHERE app.JobID = %s
            """
            cursor.execute(query, (job_id,))
            results = cursor.fetchall()

            applications = []
            for row in results:
                applicant = Applicant(
                    applicant_id=row['ApplicantID'],
                    first_name=row['FirstName'],
                    last_name=row['LastName'],
                    email=row['Email'],
                    phone=row['Phone'],
                    resume=row['Resume'],
                    experience_years=row['ExperienceYears'],
```

```
                city=row['City'],
                state=row['State']
            )

            application = JobApplication(
                job_id=job_id,
                applicant_id=row['ApplicantID'],
                application_date=row['ApplicationDate'],
                cover_letter=row['CoverLetter']
            )

            applications.append((applicant, application))
        return applications
    except mysql.connector.Error as e:
        raise DatabaseConnectionException(f"Error retrieving applications for job: {e}")
    except Exception as e:
        raise e

def get_jobs_by_salary_range(self, min_salary, max_salary):
    try:
        cursor = self.connection.cursor(dictionary=True)
        query = """
            SELECT j.*, c.CompanyName
            FROM Jobs j
            JOIN Companies c ON j.CompanyID = c.CompanyID
            WHERE j.Salary BETWEEN %s AND %s
            ORDER BY j.Salary DESC
        """
        cursor.execute(query, (min_salary, max_salary))
        results = cursor.fetchall()

        jobs = []
        for row in results:
            job = JobListing(
                job_id=row['JobID'],
                company_id=row['CompanyID'],
                job_title=row['JobTitle'],
                job_description=row['JobDescription'],
                job_location=row['JobLocation'],
                salary=row['Salary'],
                job_type=row['JobType'],
                posted_date=row['PostedDate']
            )
            company_name = row['CompanyName']
            jobs.append((job, company_name))
        return jobs
    except mysql.connector.Error as e:
        raise DatabaseConnectionException(f"Error retrieving jobs by salary range: {e}")
    except Exception as e:
        raise e
```

```
    def get_applications_by_applicant(self, applicant_id):
        try:
            cursor = self.connection.cursor(dictionary=True)
            query = """
                SELECT j.*, c.CompanyName, app.ApplicationDate, app.CoverLetter
                FROM Jobs j
                JOIN Applications app ON j.JobID = app.JobID
                JOIN Companies c ON j.CompanyID = c.CompanyID
                WHERE app.ApplicantID = %s
                ORDER BY app.ApplicationDate DESC
            """
            cursor.execute(query, (applicant_id,))
            results = cursor.fetchall()

            applications = []
            for row in results:
                job = JobListing(
                    job_id=row['JobID'],
                    company_id=row['CompanyID'],
                    job_title=row['JobTitle'],
                    job_description=row['JobDescription'],
                    job_location=row['JobLocation'],
                    salary=row['Salary'],
                    job_type=row['JobType'],
                    posted_date=row['PostedDate']
                )

                application = {
                    'job': job,
                    'company_name': row['CompanyName'],
                    'application_date': row['ApplicationDate'],
                    'cover_letter': row['CoverLetter']
                }
                applications.append(application)
            return applications
        except mysql.connector.Error as e:
            raise DatabaseConnectionException(f"Error retrieving applications by applicant: {e}")
        except Exception as e:
            raise e

    def get_companies_with_most_jobs(self):
        try:
            cursor = self.connection.cursor(dictionary=True)
            query = """
                SELECT c.CompanyName, COUNT(j.JobID) AS JobCount
                FROM Companies c
                JOIN Jobs j ON c.CompanyID = j.CompanyID
                GROUP BY c.CompanyID, c.CompanyName
                HAVING COUNT(j.JobID) = (
                    SELECT COUNT(JobID)
                    FROM Jobs
```

```
                GROUP BY CompanyID
                ORDER BY COUNT(JobID) DESC
                LIMIT 1
            )
        """
        cursor.execute(query)
        results = cursor.fetchall()
        return results
    except mysql.connector.Error as e:
        raise DatabaseConnectionException(f"Error retrieving companies with most jobs: {e}")
    except Exception as e:
        raise e

def get_jobs_with_no_applications(self):
    try:
        cursor = self.connection.cursor(dictionary=True)
        query = """
            SELECT j.*, c.CompanyName
            FROM Jobs j
            JOIN Companies c ON j.CompanyID = c.CompanyID
            LEFT JOIN Applications a ON j.JobID = a.JobID
            WHERE a.ApplicationID IS NULL
        """
        cursor.execute(query)
        results = cursor.fetchall()

        jobs = []
        for row in results:
            job = JobListing(
                job_id=row['JobID'],
                company_id=row['CompanyID'],
                job_title=row['JobTitle'],
                job_description=row['JobDescription'],
                job_location=row['JobLocation'],
                salary=row['Salary'],
                job_type=row['JobType'],
                posted_date=row['PostedDate']
            )
            company_name = row['CompanyName']
            jobs.append((job, company_name))
        return jobs
    except mysql.connector.Error as e:
        raise DatabaseConnectionException(f"Error retrieving jobs with no applications: {e}")
    except Exception as e:
        raise e

def get_jobs_by_title(self, title_keyword):
    try:
        cursor = self.connection.cursor(dictionary=True)
        query = """
            SELECT j.*, c.CompanyName
```

```
            FROM Jobs j
            JOIN Companies c ON j.CompanyID = c.CompanyID
            WHERE j.JobTitle LIKE %s
        """
        cursor.execute(query, (f"%{title_keyword}%",))
        results = cursor.fetchall()

        jobs = []
        for row in results:
            job = JobListing(
                job_id=row['JobID'],
                company_id=row['CompanyID'],
                job_title=row['JobTitle'],
                job_description=row['JobDescription'],
                job_location=row['JobLocation'],
                salary=row['Salary'],
                job_type=row['JobType'],
                posted_date=row['PostedDate']
            )
            company_name = row['CompanyName']
            jobs.append((job, company_name))
        return jobs
    except mysql.connector.Error as e:
        raise DatabaseConnectionException(f"Error retrieving jobs by title: {e}")
    except Exception as e:
        raise e

def get_average_salary(self):
    try:
        cursor = self.connection.cursor()
        query = "SELECT ROUND(AVG(Salary), 2) AS AverageSalary FROM Jobs WHERE Salary
> 0"
        cursor.execute(query)
        result = cursor.fetchone()
        return result[0] if result else 0
    except mysql.connector.Error as e:
        raise DatabaseConnectionException(f"Error calculating average salary: {e}")
    except Exception as e:
        raise e

def __del__(self):
    if hasattr(self, 'connection') and self.connection and self.connection.is_connected():
        self.connection.close()

def get_applicants_in_city_with_experience(self, city, min_experience):
    try:
        cursor = self.connection.cursor(dictionary=True)
        query = """
            SELECT a.*
            FROM Applicants a
            JOIN Applications app ON a.ApplicantID = app.ApplicantID
```

```
            JOIN Jobs j ON app.JobID = j.JobID
            JOIN Companies c ON j.CompanyID = c.CompanyID
            WHERE c.Location = %s AND a.ExperienceYears >= %s
        """
        cursor.execute(query, (city, min_experience))
        results = cursor.fetchall()

        applicants = []
        for row in results:
            applicant = Applicant(
                applicant_id=row['ApplicantID'],
                first_name=row['FirstName'],
                last_name=row['LastName'],
                email=row['Email'],
                phone=row['Phone'],
                resume=row['Resume'],
                experience_years=row['ExperienceYears'],
                city=row['City'],
                state=row['State']
            )
            applicants.append(applicant)
        return applicants
    except mysql.connector.Error as e:
        raise DatabaseConnectionException(f"Error retrieving applicants: {e}")
    except Exception as e:
        raise e
```

**exception:**

**ApplicationDeadlineException.py:**

```
class ApplicationDeadlineException(Exception):
    def __init__(self, message="Application deadline has passed"):
        self.message = message
        super().__init__(self.message)
```

**DatabaseConnectionException.py:**

```
class DatabaseConnectionException(Exception):
    def __init__(self, message="Database connection error"):
        self.message = message
        super().__init__(self.message)
```

**FileUploadException.py:**

```
class FileUploadException(Exception):
    def __init__(self, message="File upload error"):
        self.message = message
        super().__init__(self.message)
```

**InvalidEmailException.py:**

```
class InvalidEmailException(Exception):
    def __init__(self, message="Invalid email format"):
        self.message = message
        super().__init__(self.message)
```

**SalaryCalculationException.py:**

```
class SalaryCalculationException(Exception):
    def __init__(self, message="Invalid salary value encountered"):
        self.message = message
        super().__init__(self.message)
```

**util:**

**DBConnUtil.py:**

```
import mysql.connector
from mysql.connector import Error
from util.DBPropertyUtil import DBPropertyUtil


class DBConnUtil:
    @staticmethod
    def get_connection(connection_string=None):
        try:
            if connection_string is None:
                connection_string = DBPropertyUtil.get_connection_string("database.properties")

            if not connection_string:
                raise Exception("Could not get connection string")

            # Parse connection string into dictionary
            params = {}
            for item in connection_string.split():
                key, value = item.split('=', 1)
                params[key] = value

            connection = mysql.connector.connect(
                host=params['host'],
                database=params['dbname'],
                user=params['user'],
                password=params['password']
            )

            if connection.is_connected():
                print("Connected to MySQL database")
                return connection

        except Error as e:
```

```
            print(f"Error while connecting to MySQL: {e}")
            return None
```

**DBPropertyUtil.py:**

```
import configparser
import os


class DBPropertyUtil:
    @staticmethod
    def get_connection_string(property_file):
        try:
            if not os.path.exists(property_file):
                raise FileNotFoundError(f"Property file '{property_file}' not found")

            config = configparser.ConfigParser()
            config.read(property_file)

            if not config.has_section('Database'):
                raise ValueError("Database section not found in property file")

            host = config.get('Database', 'host')
            database = config.get('Database', 'database')
            user = config.get('Database', 'user')
            password = config.get('Database', 'password')

            return f"host={host} dbname={database} user={user} password={password}"

        except Exception as e:
            print(f"Error reading property file: {e}")
            return None
```

**main.py:**

```
from dao.CareerHubServiceImpl import CareerHubServiceImpl
from entity.Applicant import Applicant
from entity.Company import Company
from entity.JobApplication import JobApplication
from entity.JobListing import JobListing
from exception.InvalidEmailException import InvalidEmailException
from exception.SalaryCalculationException import SalaryCalculationException
from exception.FileUploadException import FileUploadException
from exception.ApplicationDeadlineException import ApplicationDeadlineException
from exception.DatabaseConnectionException import DatabaseConnectionException
from datetime import datetime, timedelta
import re


class MainModule:
```

```
def __init__(self):
    self.service = CareerHubServiceImpl()

def display_menu(self):
    print("\nCareerHub Job Board System")
    print("1. Company Operations")
    print("2. Applicant Operations")
    print("3. Job Listing Operations")
    print("4. Application Operations")
    print("5. Reports and Analytics")
    print("6. Exit")

def company_menu(self):
    while True:
        print("\nCompany Operations")
        print("1. Add New Company")
        print("2. View All Companies")
        print("3. Post New Job")
        print("4. Back to Main Menu")

        choice = input("Enter your choice: ")

        if choice == '1':
            self.add_company()
        elif choice == '2':
            self.view_companies()
        elif choice == '3':
            self.post_job()
        elif choice == '4':
            break
        else:
            print("Invalid choice. Please try again.")

def applicant_menu(self):
    while True:
        print("\nApplicant Operations")
        print("1. Create Applicant Profile")
        print("2. View All Applicants")
        print("3. Apply for Job")
        print("4. View Application History")
        print("5. Back to Main Menu")

        choice = input("Enter your choice: ")

        if choice == '1':
            self.create_applicant_profile()
        elif choice == '2':
            self.view_applicants()
        elif choice == '3':
            self.apply_for_job()
        elif choice == '4':
```

```
            self.view_application_history()
        elif choice == '5':
            break
        else:
            print("Invalid choice. Please try again.")

    def job_menu(self):
        while True:
            print("\nJob Listing Operations")
            print("1. View All Job Listings")
            print("2. Search Jobs by Salary Range")
            print("3. Search Jobs by Title")
            print("4. View Jobs with No Applications")
            print("5. Back to Main Menu")

            choice = input("Enter your choice: ")

            if choice == '1':
                self.view_job_listings()
            elif choice == '2':
                self.search_jobs_by_salary()
            elif choice == '3':
                self.search_jobs_by_title()
            elif choice == '4':
                self.view_jobs_with_no_applications()
            elif choice == '5':
                break
            else:
                print("Invalid choice. Please try again.")

    def application_menu(self):
        while True:
            print("\nApplication Operations")
            print("1. View Applications for Job")
            print("2. Back to Main Menu")

            choice = input("Enter your choice: ")

            if choice == '1':
                self.view_applications_for_job()
            elif choice == '2':
                break
            else:
                print("Invalid choice. Please try again.")

    def reports_menu(self):
        while True:
            print("\nReports and Analytics")
            print("1. Companies with Most Jobs")
            print("2. Average Salary of Jobs")
            print("3. Applicants in City with Experience")
```

```
            print("4. Back to Main Menu")

            choice = input("Enter your choice: ")

            if choice == '1':
                self.view_companies_with_most_jobs()
            elif choice == '2':
                self.view_average_salary()
            elif choice == '3':
                self.view_applicants_in_city_with_experience()
            elif choice == '4':
                break
            else:
                print("Invalid choice. Please try again.")

    def add_company(self):
        try:
            print("\nAdd New Company")
            company_name = input("Enter company name: ")
            location = input("Enter company location: ")

            company = Company(company_name=company_name, location=location)
            company_id = self.service.insert_company(company)
            print(f"Company added successfully with ID: {company_id}")
        except DatabaseConnectionException as e:
            print(f"Database error: {e}")
        except Exception as e:
            print(f"Error: {e}")

    def view_companies(self):
        try:
            print("\nList of Companies")
            companies = self.service.get_companies()
            for company in companies:
                print(f"ID: {company.company_id}, Name: {company.company_name}, Location:
{company.location}")
        except DatabaseConnectionException as e:
            print(f"Database error: {e}")
        except Exception as e:
            print(f"Error: {e}")

    def post_job(self):
        try:
            print("\nPost New Job")
            company_id = int(input("Enter company ID: "))
            job_title = input("Enter job title: ")
            job_description = input("Enter job description: ")
            job_location = input("Enter job location: ")
            salary = float(input("Enter salary: "))
            job_type = input("Enter job type (Full-time/Part-time/Contract/Internship): ")
```

```
            job = JobListing(
                company_id=company_id,
                job_title=job_title,
                job_description=job_description,
                job_location=job_location,
                salary=salary,
                job_type=job_type
            )

            job_id = self.service.insert_job_listing(job)
            print(f"Job posted successfully with ID: {job_id}")
        except ValueError as e:
            print(f"Invalid input: {e}")
        except SalaryCalculationException as e:
            print(f"Salary error: {e}")
        except DatabaseConnectionException as e:
            print(f"Database error: {e}")
        except Exception as e:
            print(f"Error: {e}")

    def create_applicant_profile(self):
        try:
            print("\nCreate Applicant Profile")
            first_name = input("Enter first name: ")
            last_name = input("Enter last name: ")
            email = input("Enter email: ")

            # Validate email format
            if not re.match(r"[^@]+@[^@]+\.[^@]+", email):
                raise InvalidEmailException()

            phone = input("Enter phone number: ")
            resume = input("Enter resume details: ")
            experience_years = int(input("Enter years of experience: "))
            city = input("Enter city: ")
            state = input("Enter state: ")

            applicant = Applicant(
                first_name=first_name,
                last_name=last_name,
                email=email,
                phone=phone,
                resume=resume,
                experience_years=experience_years,
                city=city,
                state=state
            )

            applicant_id = self.service.insert_applicant(applicant)
            print(f"Applicant profile created successfully with ID: {applicant_id}")
        except InvalidEmailException as e:
```

```
            print(f"Invalid email: {e}")
        except ValueError as e:
            print(f"Invalid input: {e}")
        except DatabaseConnectionException as e:
            print(f"Database error: {e}")
        except Exception as e:
            print(f"Error: {e}")

    def view_applicants(self):
        try:
            print("\nList of Applicants")
            applicants = self.service.get_applicants()
            for applicant in applicants:
                print(
                    f"ID: {applicant.applicant_id}, Name: {applicant.first_name} {applicant.last_name},
Email: {applicant.email}")
        except DatabaseConnectionException as e:
            print(f"Database error: {e}")
        except Exception as e:
            print(f"Error: {e}")

    def apply_for_job(self):
        try:
            print("\nApply for Job")
            applicant_id = int(input("Enter your applicant ID: "))
            job_id = int(input("Enter job ID to apply for: "))
            cover_letter = input("Enter cover letter: ")

            # Check if application deadline has passed (example: 30 days after posting)
            jobs = self.service.get_job_listings()
            job = next((j for j in jobs if j.job_id == job_id), None)

            if not job:
                print("Job not found")
                return

            deadline = job.posted_date + timedelta(days=30)
            if datetime.now() > deadline:
                raise ApplicationDeadlineException()

            application = JobApplication(
                job_id=job_id,
                applicant_id=applicant_id,
                cover_letter=cover_letter
            )

            application_id = self.service.insert_job_application(application)
            print(f"Application submitted successfully with ID: {application_id}")
        except ApplicationDeadlineException as e:
            print(f"Application error: {e}")
        except ValueError as e:
```

```
            print(f"Invalid input: {e}")
        except DatabaseConnectionException as e:
            print(f"Database error: {e}")
        except Exception as e:
            print(f"Error: {e}")

    def view_application_history(self):
        try:
            print("\nApplication History")
            applicant_id = int(input("Enter applicant ID: "))

            applications = self.service.get_applications_by_applicant(applicant_id)
            if not applications:
                print("No applications found")
                return

            print(f"\nApplication History for Applicant ID: {applicant_id}")
            for app in applications:
                print(f"\nJob Title: {app['job'].job_title}")
                print(f"Company: {app['company_name']}")
                print(f"Salary: {app['job'].salary}")
                print(f"Application Date: {app['application_date']}")
                print(f"Cover Letter: {app['cover_letter'][:50]}...")  # Show first 50 chars
        except ValueError as e:
            print(f"Invalid input: {e}")
        except DatabaseConnectionException as e:
            print(f"Database error: {e}")
        except Exception as e:
            print(f"Error: {e}")

    def view_job_listings(self):
        try:
            print("\nJob Listings")
            jobs = self.service.get_job_listings()
            companies = {c.company_id: c.company_name for c in self.service.get_companies()}

            for job in jobs:
                company_name = companies.get(job.company_id, "Unknown")
                print(f"\nID: {job.job_id}, Title: {job.job_title}")
                print(f"Company: {company_name}, Location: {job.job_location}")
                print(f"Salary: {job.salary}, Type: {job.job_type}")
                print(f"Posted: {job.posted_date}")
                print(f"Description: {job.job_description[:50]}...")  # Show first 50 chars
        except DatabaseConnectionException as e:
            print(f"Database error: {e}")
        except Exception as e:
            print(f"Error: {e}")

    def search_jobs_by_salary(self):
        try:
            print("\nSearch Jobs by Salary Range")
```

```
            min_salary = float(input("Enter minimum salary: "))
            max_salary = float(input("Enter maximum salary: "))

            jobs = self.service.get_jobs_by_salary_range(min_salary, max_salary)
            if not jobs:
                print("No jobs found in this salary range")
                return

            print(f"\nJobs between {min_salary} and {max_salary}:")
            for job, company_name in jobs:
                print(f"\nTitle: {job.job_title}")
                print(f"Company: {company_name}, Location: {job.job_location}")
                print(f"Salary: {job.salary}, Type: {job.job_type}")
        except ValueError as e:
            print(f"Invalid input: {e}")
        except DatabaseConnectionException as e:
            print(f"Database error: {e}")
        except Exception as e:
            print(f"Error: {e}")

    def search_jobs_by_title(self):
        try:
            print("\nSearch Jobs by Title")
            keyword = input("Enter job title keyword: ")

            jobs = self.service.get_jobs_by_title(keyword)
            if not jobs:
                print(f"No jobs found with '{keyword}' in title")
                return

            print(f"\nJobs with '{keyword}' in title:")
            for job, company_name in jobs:
                print(f"\nTitle: {job.job_title}")
                print(f"Company: {company_name}, Location: {job.job_location}")
                print(f"Salary: {job.salary}, Type: {job.job_type}")
        except DatabaseConnectionException as e:
            print(f"Database error: {e}")
        except Exception as e:
            print(f"Error: {e}")

    def view_jobs_with_no_applications(self):
        try:
            print("\nJobs with No Applications")
            jobs = self.service.get_jobs_with_no_applications()
            if not jobs:
                print("All jobs have at least one application")
                return

            for job, company_name in jobs:
                print(f"\nTitle: {job.job_title}")
                print(f"Company: {company_name}, Location: {job.job_location}")
```

```
            print(f"Salary: {job.salary}, Type: {job.job_type}")
            print(f"Posted: {job.posted_date}")
    except DatabaseConnectionException as e:
        print(f"Database error: {e}")
    except Exception as e:
        print(f"Error: {e}")

def view_applications_for_job(self):
    try:
        print("\nApplications for Job")
        job_id = int(input("Enter job ID: "))

        applications = self.service.get_applications_for_job(job_id)
        if not applications:
            print("No applications found for this job")
            return

        print(f"\nApplications for Job ID: {job_id}")
        for applicant, application in applications:
            print(f"\nApplicant: {applicant.first_name} {applicant.last_name}")
            print(f"Email: {applicant.email}, Phone: {applicant.phone}")
            print(f"Experience: {applicant.experience_years} years")
            print(f"Application Date: {application.application_date}")
            print(f"Cover Letter: {application.cover_letter[:50]}...")  # Show first 50 chars
    except ValueError as e:
        print(f"Invalid input: {e}")
    except DatabaseConnectionException as e:
        print(f"Database error: {e}")
    except Exception as e:
        print(f"Error: {e}")

def view_companies_with_most_jobs(self):
    try:
        print("\nCompanies with Most Job Postings")
        companies = self.service.get_companies_with_most_jobs()
        if not companies:
            print("No companies found")
            return

        for company in companies:
            print(f"\nCompany: {company['CompanyName']}")
            print(f"Number of Jobs: {company['JobCount']}")
    except DatabaseConnectionException as e:
        print(f"Database error: {e}")
    except Exception as e:
        print(f"Error: {e}")

def view_average_salary(self):
    try:
        print("\nAverage Salary of Jobs")
        avg_salary = self.service.get_average_salary()
```

```
            print(f"The average salary of all jobs is: {avg_salary}")
        except DatabaseConnectionException as e:
            print(f"Database error: {e}")
        except Exception as e:
            print(f"Error: {e}")

    def view_applicants_in_city_with_experience(self):
        try:
            print("\nApplicants in City with Experience")
            city = input("Enter city: ")
            min_experience = int(input("Enter minimum years of experience: "))

            applicants = self.service.get_applicants_in_city_with_experience(city, min_experience)
            if not applicants:
                print(f"No applicants found in {city} with {min_experience}+ years experience")
                return

            print(f"\nApplicants in {city} with {min_experience}+ years experience:")
            for applicant in applicants:
                print(f"\nName: {applicant.first_name} {applicant.last_name}")
                print(f"Email: {applicant.email}, Phone: {applicant.phone}")
                print(f"Experience: {applicant.experience_years} years")
                print(f"Location: {applicant.city}, {applicant.state}")
        except ValueError as e:
            print(f"Invalid input: {e}")
        except DatabaseConnectionException as e:
            print(f"Database error: {e}")
        except Exception as e:
            print(f"Error: {e}")

    def run(self):
        while True:
            self.display_menu()
            choice = input("Enter your choice: ")

            if choice == '1':
                self.company_menu()
            elif choice == '2':
                self.applicant_menu()
            elif choice == '3':
                self.job_menu()
            elif choice == '4':
                self.application_menu()
            elif choice == '5':
                self.reports_menu()
            elif choice == '6':
                print("Exiting CareerHub Job Board System. Goodbye!")
                break
            else:
                print("Invalid choice. Please try again.")
```

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

database.properties:

```
[Database]
host=localhost
database=CareerHub
user=root
password=root
```

## OUTPUT:

## Company Operations

### 1. Add New Company

```
Add New Company
Enter company name: Google
Enter company location: Bangalore
Company added successfully with ID: 12
```

### 2. View All Companies

```
List of Companies
ID: 1, Name: Hexaware Technologies, Location: Mumbai
ID: 2, Name: Infosys, Location: Bangalore
ID: 3, Name: TCS, Location: Chennai
ID: 4, Name: Wipro, Location: Pune
ID: 5, Name: Tech Mahindra, Location: Hyderabad
ID: 6, Name: HCL Technologies, Location: Noida
ID: 7, Name: LTI Mindtree, Location: Mumbai
ID: 8, Name: Mphasis, Location: Bangalore
ID: 9, Name: Capgemini, Location: Gurgaon
ID: 10, Name: Accenture, Location: Pune
ID: 11, Name: TechComp, Location: Chennai
ID: 12, Name: Google, Location: Bangalore
```

### 3. Post New Job

```
Post New Job
Enter company ID: 12
Enter job title: Data Scientist
Enter job description: ML and Python skills needed
Enter job location: Hybrid
Enter salary: 1500000
Enter job type (Full-time/Part-time/Contract/Internship): Full-time
Job posted successfully with ID: 13
```

## Applicant Operations

### 1. Create Applicant Profile

```
Create Applicant Profile
Enter first name: Alice
Enter last name: Johnson
Enter email: alice.j@email.com
Enter phone number: 9876543210
Enter resume details: 3 years ML experience
Enter years of experience: 3
Enter city: Bangalore
Enter state: Karnataka
Applicant profile created successfully with ID: 12
```

### 2. View All Applicants

```
List of Applicants
ID: 1, Name: Rahul Sharma, Email: rahul.sharma@email.com
ID: 2, Name: Priya Patel, Email: priya.patel@email.com
ID: 3, Name: Amit Singh, Email: amit.singh@email.com
ID: 4, Name: Neha Gupta, Email: neha.gupta@email.com
ID: 5, Name: Raj Verma, Email: raj.verma@email.com
ID: 6, Name: Ananya Joshi, Email: ananya.joshi@email.com
ID: 7, Name: Vikram Reddy, Email: vikram.reddy@email.com
ID: 8, Name: Divya Malhotra, Email: divya.malhotra@email.com
ID: 9, Name: Arjun Kapoor, Email: arjun.kapoor@email.com
ID: 10, Name: Pooja Mehta, Email: pooja.mehta@email.com
ID: 11, Name: Ravi Kumar, Email: ravi.kumar@email.com
ID: 12, Name: Alice Johnson, Email: alice.j@email.com
```

### 3. Apply for Job

```
Apply for Job
Enter your applicant ID: 12
Enter job ID to apply for: 13
Enter cover letter: I have TensorFlow certification...
Application submitted successfully with ID: 11
```

### 4. View Application History

```
Application History
Enter applicant ID: 12

Application History for Applicant ID: 12

Job Title: Data Scientist
Company: Google
Salary: 1500000.00
Application Date: 2025-04-09 16:23:47
Cover Letter: I have TensorFlow certification......
```

# Job Listing Operations

## 1. View All Job Listings

```
Job Listings

ID: 1, Title: Java Developer
Company: Hexaware Technologies, Location: Mumbai
Salary: 900000.00, Type: Full-time
Posted: 2025-04-09 14:28:26
Description: Spring Boot development for banking solutions...

ID: 2, Title: Automation Tester
Company: Hexaware Technologies, Location: Mumbai
Salary: 750000.00, Type: Full-time
Posted: 2025-04-09 14:28:26
Description: Selenium with Java testing framework...

ID: 3, Title: Data Scientist
Company: Infosys, Location: Bangalore
Salary: 1200000.00, Type: Full-time
Posted: 2025-04-09 14:28:26
Description: Build ML models for predictive analytics...

ID: 4, Title: .NET Developer
Company: TCS, Location: Chennai
Salary: 850000.00, Type: Full-time
Posted: 2025-04-09 14:28:26
Description: ASP.NET Core development...
```

```
ID: 5, Title: DevOps Engineer
Company: Wipro, Location: Pune
Salary: 1100000.00, Type: Full-time
Posted: 2025-04-09 14:28:26
Description: AWS, Docker and Kubernetes implementation...

ID: 6, Title: SAP Consultant
Company: Tech Mahindra, Location: Hyderabad
Salary: 1000000.00, Type: Contract
Posted: 2025-04-09 14:28:26
Description: SAP FICO module implementation...

ID: 7, Title: Python Developer
Company: HCL Technologies, Location: Noida
Salary: 800000.00, Type: Full-time
Posted: 2025-04-09 14:28:26
Description: Django and Flask development...

ID: 8, Title: UI/UX Designer
Company: LTI Mindtree, Location: Mumbai
Salary: 700000.00, Type: Full-time
Posted: 2025-04-09 14:28:26
Description: Figma and Adobe XD prototyping...
```

```
ID: 9, Title: Business Analyst
Company: Mphasis, Location: Bangalore
Salary: 950000.00, Type: Full-time
Posted: 2025-04-09 14:28:26
Description: Requirement gathering and analysis...

ID: 10, Title: Cloud Architect
Company: Capgemini, Location: Gurgaon
Salary: 1300000.00, Type: Full-time
Posted: 2025-04-09 14:28:26
Description: Azure cloud solutions design...

ID: 11, Title: HR Recruiter
Company: Accenture, Location: Pune
Salary: 600000.00, Type: Full-time
Posted: 2025-04-09 14:28:26
Description: Tech recruitment for IT positions...

ID: 12, Title: DevOps Engineer
Company: TechComp, Location: Remote
Salary: 1200000.00, Type: Full-time
Posted: 2025-04-09 16:12:56
Description: AWS and Kubernetes experience required...
```

```
ID: 13, Title: Data Scientist
Company: Google, Location: Hybrid
Salary: 1500000.00, Type: Full-time
Posted: 2025-04-09 16:22:33
Description: ML and Python skills needed...
```

## 2. Search Jobs by Salary Range

```
Search Jobs by Salary Range
Enter minimum salary: 1000000
Enter maximum salary: 1500000

Jobs between 1000000.0 and 1500000.0:

Title: Data Scientist
Company: Google, Location: Hybrid
Salary: 1500000.00, Type: Full-time

Title: Cloud Architect
Company: Capgemini, Location: Gurgaon
Salary: 1300000.00, Type: Full-time

Title: Data Scientist
Company: Infosys, Location: Bangalore
Salary: 1200000.00, Type: Full-time

Title: DevOps Engineer
Company: TechComp, Location: Remote
Salary: 1200000.00, Type: Full-time

Title: DevOps Engineer
Company: Wipro, Location: Pune
Salary: 1100000.00, Type: Full-time
Title: SAP Consultant
Company: Tech Mahindra, Location: Hyderabad
Salary: 1000000.00, Type: Contract
```

## 3. Search Jobs by Title

```
Search Jobs by Title
Enter job title keyword: Dev

Jobs with 'Dev' in title:

Title: Java Developer
Company: Hexaware Technologies, Location: Mumbai
Salary: 900000.00, Type: Full-time

Title: .NET Developer
Company: TCS, Location: Chennai
Salary: 850000.00, Type: Full-time

Title: DevOps Engineer
Company: Wipro, Location: Pune
Salary: 1100000.00, Type: Full-time

Title: Python Developer
Company: HCL Technologies, Location: Noida
Salary: 800000.00, Type: Full-time

Title: DevOps Engineer
Company: TechComp, Location: Remote
Salary: 1200000.00, Type: Full-time
```

### 4. View Jobs with No Applications

```
Jobs with No Applications

Title: HR Recruiter
Company: Accenture, Location: Pune
Salary: 600000.00, Type: Full-time
Posted: 2025-04-09 14:28:26

Title: DevOps Engineer
Company: TechComp, Location: Remote
Salary: 1200000.00, Type: Full-time
Posted: 2025-04-09 16:12:56
```

## Application Operations

### 1. View Applications for Job

```
Applications for Job
Enter job ID: 13

Applications for Job ID: 13

Applicant: Alice Johnson
Email: alice.j@email.com, Phone: 9876543210
Experience: 3 years
Application Date: 2025-04-09 16:23:47
Cover Letter: I have TensorFlow certification......
```

## Reports and Analytics

### 1. Companies with Most Jobs

```
Companies with Most Job Postings

Company: Hexaware Technologies
Number of Jobs: 2
```

### 2. Average Salary of Jobs

```
Average Salary of Jobs
The average salary of all jobs is: 988461.54
```

### 3. Applicants in City with Experience

```
Applicants in City with Experience
Enter city: Bangalore

Name: Alice Johnson
Email: alice.j@email.com, Phone: 9876543210
Experience: 3 years
Location: Bangalore, Karnataka
```