PRACTICE ASSESSMENT 5

Task: Using the starter repository, implement an interface and several classes and functions in TypeScript. You will use inheritance with a parent and child class. You will aso create two classes that implement an interface. The functions work with these classes. You will write tests for all of the classes and functions, in which you will also demonstrate polymorphism.

Setup and Submission:
See readme.md for setup instructions. Commit and push to submit.

Build Specifications

- Below are details for implementing the interface, classes, and functions: `Job` interface, `SalaryJob` class, `HourlyJob` class, `Candidate` class, `RemoteCandidate` class, `findQualifiedCandidates` function, `getCombinedYearsExperience` function, and `findBestPayingJob` function.
- Each class and interface has its own file. There is also a test file for each one.
- All functions go in the functions.ts file, and their tests go in functions.test.ts.
- NOTE: This assessment does NOT ask for any code in index.ts. However, feel free to add code there if you like. It will not affect your grade.

Job Interface (1 point)
In the Job.ts file, create an interface named `Job` and export it.

- Properties:
  - `title` (a string)
  - `yearsRequired` (a number)
  - `remote` (a boolean)
- Methods:
  - `getAnnualPay()`: Has no parameters. It returns a number.
- Jest Tests: N/A

continued on next page...

SalaryJob Class (5 points: 1 per test case + 1 for implementing Job interface)
In the SalaryJob.ts file, create a class named `SalaryJob` and export it.

- Implements the `Job` interface.
- Properties:
  - `salary` (a number)
  - `title` (a string)
  - `yearsRequired` (a number)
  - `remote` (a boolean)
- Constructor Parameters:
  - `salary` (a number) sets the `salary` property.
  - `title` (a string) sets the `title` property.
  - `yearsRequired` (a string) sets the `yearsRequired` property. This parameter is optional and has a default value of **0**.
  - `remote` (a string) sets the `remote` property. This parameter is optional and has a default value of **false**.
- Methods:
  - `getAnnualPay()`: Has no parameters. It simply returns the `salary` property.
- Jest Tests:
  - The `salary`, `title`, `yearsRequired`, and `remote` properties are set from the constructor parameters.
  - `yearsRequired` defaults to **0**, when the third constructor parameter is omitted.
  - `remote` defaults to **false**, when the fourth constructor parameter is omitted.
  - The `getAnnualPay` method returns the `salary` property.

continued on next page...

HourlyJob Class (5 points: 1 per test case + 1 for implementing Job interface)
In the HourlyJob.ts file, create a class named `HourlyJob` and export it.

- Implements the `Job` interface.
- Properties:
    - `hourlyWage` (a number)
    - `title` (a string)
    - `yearsRequired` (a number)
    - `remote` (a boolean)
- Constructor Parameters:
    - `hourlyWage` (a number) sets the `hourlyWage` property.
    - `title` (a string) sets the `title` property.
    - `yearsRequired` (a string) sets the `yearsRequired` property. This parameter is optional and has a default value of **0**.
    - `remote` (a string) sets the `remote` property. This parameter is optional and has a default value of **false**.
- Methods:
    - `getAnnualPay()`: Has no parameters. It returns the `hourlyWage` property times **2000** (the number of hours worked per year).
- Jest Tests:
    - The `hourlyWage`, `title`, `yearsRequired`, and `remote` properties are set from the constructor parameters.
    - `yearsRequired` defaults to **0**, when the third constructor parameter is omitted.
    - `remote` defaults to **false**, when the fourth constructor parameter is omitted.
    - The `getAnnualPay` method returns the `hourlyWage` property times **2000**.

continued on next page...

Candidate Class (7 points: 1 per test case)

In the Candidate.ts file, create a class named `Candidate` and export it.

- Properties:
  - `name` (a string)
  - `yearsExperience` (a number) - This property **ALWAYS STARTS at 0**.
- Constructor Parameters:
  - `name` (a string) sets the `name` property.
- Methods:
  - `addExperience`
    - It takes one parameter: `years` (a number).
    - It returns nothing.
    - The function increases `yearsExperience` by the given `years` argument.
  - `fitsJob`
    - It takes one parameter: `job` (a `Job`).
    - It returns a boolean.
    - The function determines if this candidate's `yearsExperience` is greater than or equal to the job's `yearsRequired`. If so, return **true**; otherwise, return **false**.
- Jest Tests:
  - The `name` property is set from the constructor parameter.
  - Confirm that a new instance of `Candidate` has `yearsExperience` set to **0**.
  - Calling `addExperience` once, increases yearsExperience by the given number of years.
  - Calling `addExperience` twice, increases yearsExperience by the combined given number of years.
  - For the following three test cases, try some with `SalaryJob` and some with `HourlyJob`. Also use different numbers for `yearsExperience`. (Hint: the correct way to set yearsExperience is to call `addExperience`.)
    - `fitsJob` returns **true** when `yearsExperience` is greater than `yearsRequired`.
    - `fitsJob` returns **true** when `yearsExperience` is equal to `yearsRequired`.
    - `fitsJob` returns **false** when `yearsExperience` is less than `yearsRequired`.

continued on next page...

RemoteCandidate Class (7 points: 1 per test case + 1 for extending Candidate class)
In the RemoteCandidate.ts file, create a class named `RemoteCandidate` and export it.

- `RemoteCandidate` is a subclass of `Candidate`.
- Properties:
  - `timezone` (a string)
  - (NOTE: `RemoteCandidate` will inherit `name` and `yearsExperience`.)
- Constructor Parameters:
  - `name` (a string) sets the `name` property using a call to `super`.
  - `timezone` (a string) sets the `timezone` property
- Methods:
  - Override `fitsJob`
    - It takes one parameter: `job` (a `Job`).
    - It returns a boolean.
    - The function determines if this candidate's `yearsExperience` is greater than or equal to the job's `yearsRequired` AND `remote` is **`true`**. If so, return **`true`**; otherwise, return **`false`**.
  - (NOTE: `RemoteCandidate` will inherit `addExperience`.)
- Jest Tests:
  - The `name` and `timezone` properties are set from the constructor parameter.
  - Confirm that a new instance of `RemoteCandidate` has `yearsExperience` set to **`0`**.
  - For the following four test cases, try some with `SalaryJob` and some with `HourlyJob`. Also use different numbers for `yearsExperience`. (Hint: the correct way to set yearsExperience is to call `addExperience`.)
    - `fitsJob` returns **`true`** when `yearsExperience` is greater than `yearsRequired` and `remote` is **`true`**.
    - `fitsJob` returns **`true`** when `yearsExperience` is equal to `yearsRequired` and `remote` is **`true`**.
    - `fitsJob` returns **`false`** when `yearsExperience` is less than `yearsRequired` and `remote` is **`true`**.
    - `fitsJob` returns **`false`** when `yearsExperience` is greater than `yearsRequired` and `remote` is **`false`**.

continued on next page...

findQualifiedCandidates Function (3 points: 1 for function + 1 per test case)
In the functions.ts file, create a function named `findQualifiedCandidates` and export it.

- Parameters:
  - `job` (a `Job`)
  - `candidates` (an array of `Candidate`)
- Returns: an array of `Candidate`
- Functionality: Filter the `candidates` array to find only the candidates that return for `jobFits` with the given job. Return a new array of those matching candidates.
- Jest Tests: For each test case, create an array of `Candidate`. Call `findQualifiedCandidates` with this array and a job and confirm the correct result.
  - Do a test case with an array that has a mix of `Candidate` and `RemoteCandidate`.
  - [Optional] Do a test case where none of the candidates match the job. (Expect an empty array as the result.)
  - Do a test case with an empty array. (Expect an empty array as the result.)

getCombinedYearsExperience Function (3 points: 1 for function + 1 per test case)
In the functions.ts file, create a function named `getCombinedYearsExperience` and export it.

- Parameters:
  - `candidates` (an array of `Candidate`)
- Returns: a number
- Functionality: Add together the yearsExperience for each candidate in the array to get the sum. Return the result.
- Jest Tests:
  - Create an array of any three candidates. The array **must include a mix** of `Candidate` and `RemoteCandidate`. Call `getCombinedYearsExperience` with this array and confirm the correct result.
  - Call `getCombinedYearsExperience` with an empty array and confirm the result is **0**.

continued on next page...

findBestPayingJob Function (4 points: 1 for function + 1 for test case)
In the functions.ts file, create a function named `findBestPayingJob` and export it.

- Parameters:
  - `jobs` (an array of `Job`)
- Returns: a `Job` or **null**
- Functionality: Use the `getAnnualPay` method of the jobs in the array to find the job with the highest annual pay. If the array is empty, return null. Don't worry about ties–assume there will not be multiple jobs that tie for the best paying job.
- Jest Tests:
  - For each of the following test cases, create an array of any three or more jobs. The array **must include a mix** of `SalaryJob` and `HourlyJob`. Call `findBestPayingJob` with this array and confirm the correct result.
    - Do a test case where a `SalaryJob` has the best pay.
    - Do a test case where an `HourlyJob` has the best pay.
    - [Optional] Do a test case where the first item in the array has the best pay.
    - [Optional] Do a test case where the last item in the array has the best pay.
  - Call `findBestPayingJob` with an empty array and confirm the result is **null**.

Grading:
There are a total of 35 points, as indicated above.

- 28 points - You will receive 1 point for each of the required test cases if that test is correct and is passing. The feature being tested must also be correctly implemented.
- 1 point - `Job` interface is correctly defined.
- 1 point - `SalaryJob` class implements `Job`.
- 1 point - `HourlyJob` class implements `Job`.
- 1 point - `RemoteCandidate` is a subclass of `Candidate`.
- 1 point - `getOrderTotal` has all correct TypeScript annotations (parameters and return type) and is implemented correctly.
- 1 point - `getCombinedYearsExperience` has all correct TypeScript annotations (parameters and return type) and is implemented correctly.
- 1 point - `findBestPayingJob` has all correct TypeScript annotations (parameters and return type) and is implemented correctly.