



Pairing Interview

| Focus on Empathy, Aptitude, and Communication

Our Pairing interview is the first gate to working with Focused Labs. Right now we use an almost exact replica of an interview from a previous company. It's important that we understand why we replicated this interview and what the goal is.

We use this exercise as our first interview because it is objective, repeatable, canned, and has a consistent scoring system. The exercise tests a candidate's ability to implement an object oriented solution, test drive that solution, communicate about the code in that solution, and be empathetic to the boundaries we set. This interview is a great way to get a quick leading indicator of success and is a lot of fun!



This interview *is* a leading indicator of success



This interview *is not* a 1 to 1 evaluation of the candidates skill

[Interviewer Overview](#)

[Scoring](#)

[Scoring Rubric](#)

[Losing Points](#)

[Earning Points](#)

[Environment](#)

[Before an interview](#)

[Rules](#)

[Script](#)

[Opening](#)

[isEmpty](#)

[size](#)

[Refactor - Remove empty state](#)

[Refactor - Dry up tests](#)

[contains](#)

[remove](#)

[Refactor - extract indexOf](#)

[ignoresDuplicates](#)

[grows](#)

[Ending the Interview](#)

[Other Notes](#)

Interviewer Overview

The interview takes somewhere between 30 minutes - 1 hour to complete. The time it takes to finish is not an indicator of the competency or quality of the candidate.

During the hour the interviewer will guide the candidate through developing a `Set` class in Java through test driven development.

It's the interviewer's job to finish the entire exercise in the allotted time no-matter the candidate's performance.

The job of the interviewer is to make the time a productive, friendly, and kind experience with the candidate. Make sure you evaluate fairly.

You should avoid using your mouse at all costs and know the key bindings like the back of your hand.

Remember every interview should end exactly the same.

Scoring

Scoring Rubric

This interview is scored from 100 - 0 (although it's rare to keep a score lower than 85).

When the interview starts the candidate is given 100 points. During the interview as we ask questions of the candidate the interviewer will either add or remove points from a candidate. It is possible, although very rare, for a candidate to score over 100.

Losing Points

A candidate can lose points for the following:



It's at the discretion of the interviewer when to remove a point and how many to remove. Your judgment is based on if they are discussing a solution with you, or need you to give them an answer. That said, there are consistent places throughout the exercise where points are deducted.

- Getting a question wrong
- Taking too long to answer
- Sitting silent and not communicating
- Insistent that an approach is correct after the guidance of the interviewer
- Generally not being nice or reflecting our values

Earning Points

We often ask similar questions more than once or coach a candidate to a solution. As a test for aptitude the interviewer should give points back to a candidate if they previously missed something that they then got correct.

Candidates can also earn points by spotting bugs, being exceptionally clever or smart, or having elevated conversations about the **why** of how the code is being written or structured

Environment

- IntelliJ should be set up with two panes side by side. On the left the implementation. On the right the test class.
- Colors should be accessible and color blind friendly
- Font sizes should be greater than 15

Before an interview

- Open the interview template and with a completed test and implementation class run your test to make sure packages are updated and code can run
- Clear out the entire contents of the implementation class leaving only the skeleton of the `Set`

```
package interview;

public class Set {
}
```

- Clear out the contents of the test class leaving only the imports and skeleton.

```
package interview;

import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.*;

public class SetTest {
}
```

- Run your tests to ensure that the empty classes run

Rules

- The exercise will be done in Java.
- The **implementation** is restricted to Java primitives and no imports (this means no collections classes)
- The candidate is not allowed to type



We have found asking candidates to type when they aren't experienced in the stack or are not comfortable typing in front of others creates a negative experience that can cause a high quality candidate to perform poorly. By typing ourselves we focus on empathy, aptitude, and communication. It also forces the candidate to articulate precisely about the code they wish to drive out, which is a fundamental skill of any good pair.

- The candidate is not expected to know Java but instead expected to articulate intent clearly and accurately about the code they are proposing

Script

!! It is important that anything in a quote block should be said exactly as is written. This guarantees a fair playing field and consistent results between candidates.

Opening

The opening is not scored.

Open by explaining the above rules to the candidate. After they have a clear understanding of the rules ask them

| Do you know the characteristics of a set?

The answer is a good indicator of the candidates experience and can help the interviewer understand if the candidate has a basic understanding of data structures.

No matter the answer of the candidate reiterate that a set is

| An unordered collection of unique elements

You should make sure you say this three times



Always run the tests to show them fail then pass. Never refactor while tests are failing.

isEmpty

Start by implementing a failing test that asserts that all sets are empty and ask the candidate

| What's the simplest thing I can do to make this test pass?

After getting past that test introduce the main three Sets `empty` `one` and `many`

Add the string `"1"` to `one` and the strings `"1"` and `"2"` to `many`.

Implement a three assertions, asserting `empty` is empty while `one` and `many` are not empty.

You will need to add an `add` method to the implementation. This method is never explicitly tested.

| Now, what's the simplest thing I can do to make this test pass?



This should be implemented using a boolean field called `empty`

size

Copy and paste the test from `isEmpty` and remove the assertions while keeping the prep part of the test.



The copy and paste is important as it's a soft indicator that there is duplication that should be refactored later.

Implement a test for size asserting the size of `empty` is 0 and the size of `one` is 1.

| How do you think we should assert the size of `many` ?

The candidate should say assert greater than one, any other answer loses points. Feel free to ask why.

You can hint if you want

| Is there a better way to assert the size of `many`



This should be implemented using a counter called `size` .

Do not refactor `empty` away until tests are green, and do not allow for the introduction of an array... yet. Any persistence to do either loses points.

Refactor - Remove empty state

Do you see any refactorings

After green tests remove the empty field and refactor `isEmpty` to use a boolean return like so

```
return size == 0
```

Refactor - Dry up tests

Refactor your tests to contain a before block.

If the candidate suggests both refactors give them some extra credit.

contains

Write a test that asserts the full truth table of `empty`, `one` and `many`. `empty` should not contain the strings `"1"` and `"2"`, `one` should contain `"1"` but not `"2"`. `many` should contain both `"1"` and `"2"`

The candidate should suggest that we need to store state now. That's correct.

Implement the state as an `Object[]` named `elements` with a length of 10.

The candidate now needs to add the `elements` to the elements array and check that the array contains the elements. Either can be done first.

Finding the Elements



This should be implemented using a `for` loop.

If the candidate suggests any loop other than a `for` loop ask why then drive them to a `for` loop.

Use a generator to generate a for loop. The default generator in IntelliJ is `fori` as you are implementing the for loop as the candidate

| How far should we loop?

The candidate should suggest `size` any other answer is a loss of points.

As you iterate through the `elements` array check that the element at `i` equals the given value.

Adding the elements

The candidate should suggest inserting the element at an index, there is no push in Java. The correct index is `size` then increment `size`.

remove

The remove test should be implemented with a new `Set`. Add two strings `"1"` and `"2"` to the new `Set` and then remove `"1"`.

| How would you test remove?



This is the first time we ask the candidate to write a test

The candidate should assert the `size` is 1 and we no longer contain the `"1"`

After getting a successful red test ask the candidate to make the test pass. They should suggest looping again. Copy paste the contains code, this indicates there is duplication to clean up. Now delete the contents of the `if`



This is the hardest part of the interview

The correct answer is to move the last element into the spot of the found and then decrement the size by 1. No other solution is correct or more efficient. I generally do not remove points if a candidate suggests shifting the contents down, but will remove points if they require any more prompting than suggesting there's actually a cleaner solution.

We do introduce a memory leak if the candidate does not null out the last element of the array before decrementing size. Spotting the memory leak is a point.

The `if` body should look like this

```
elements[indexOf(value)] = elements[size - 1];
elements[--size] = null;
```

Refactor - extract indexOf

Ask the candidate

Do you see any refactorings

You can extract an `indexOf` method from both `contains` and `remove`.

Index of should use the existing for loop and return `-1` if no element is found.

`contains` should become one line and the candidate had the potential to introduce a bug in `remove`.



If the candidate does not add a guard clause checking the given value is contained in the array. There is potential for an `IndexOutOfBoundsException` exception. Ask the candidate to spot it and add a test to prevent it

If the candidate adds a guard clause to check the element is contained suggest they have added untested code and see if they can spot it. If they spot it quickly add points if it requires coaching remove points.

If the candidate suggests wrapping the the whole method in an `if` rather than using a guard clause coach them to a guard

If the candidate suggests using `-1` rather than `contains` ask them why, if they can't explain the trade of multiple loops vs readability deduct points.

The final method should use a guard that tests if the given value is not contained to return.

ignoresDuplicates

Add a test that adds `"1"` to `one` then assert the size of `one` is 1.

The candidate should suggest a guard clause using the contains method. Anything other than that loses points.

grows



Make sure you can use IntelliJ Generators for this one.

Introduce a single argument constructor and test a new set that has an initial capacity of 1.

Add 2 things to the set and see an `OutOfBounds` exception.

add a check in `add` if `full` then `grow`.

full

| How do we know if we are full

Full should check that `size == elements.length`

grow

| How should we grow the Set

Create a new bigger set that is 2x the original size.

Use `System.arraycopy` to move everything over into the new bigger set and reassign.

Ending the Interview


If the candidate did well and you feel confident in moving them forward to the next interview, give them this feedback in real time and ***ask the candidate directly when***

they are available to come in for a full final day interview.

Ideally, the candidate is able to give you a few days of when they are available. Check the Focused Labs Interview calendar (below) to see whether the team is available to interview. If there are no other candidates interviewing, you can tentatively confirm the candidate for that day. If the candidate provides multiple days, the team's preference would be to not conduct interviews on Fridays (due to retros and demos). Please put the candidate's availability in your Greenhouse scorecard as soon as possible so that the recruiting team can work on coordinating the next interview.


Google Calendar

With Google's free online calendar, it's easy to keep track of life's important events all in one place.

 <https://calendar.google.com/calendar/u/0?cid=Y19qbDNsY3FwNGllbWI0cTBjdDBoaWo4OWEwMEBncm91cC5jYWxlbmRhci5nb29nbGUuY29t>

If the candidate did not do well, you can thank them for their time and complete your Greenhouse scorecard with your feedback.

Other Notes

 [TPI Scoring Rubric](#)