Alexa, what's the high temperature?

Branch name: remotedebuggingandlogging-classroom

RDE workflows:

- `remotedebuggingandlogging-classroom-phase0`
- `remotedebuggingandlogging-classroom-phase1`
- `remotedebuggingandlogging-classroom-phase2`
- `remotedebuggingandlogging-classroom-extension1`
- `remotedebuggingandlogging-classroom-extension2`

Introduction

Today, we will continue work on the Alexa Weather app. Previously, we worked on an API that let you get the average annual precipitation level (in inches) for a city. Another feature of the App is you can ask what the high temperature was on any given day, and Alexa will tell you what the historical high was on that day. So if I ask Alexa, "Alexa, what was the high temperature in Atlanta on July 4, 2020?" She would respond that the high temperature was 81 degrees Fahrenheit.

This is handled by one of the APIs of our `ATACurriculumAlexaWeatherService`. The name of this API is `GetDailyHighTemp` and it's input is a `GetDailyHighTempRequest` object and it returns a `GetDailyHighTempResponse` object.

When our clients (our customers) use our API, they provide a city name and a date in the format "yyyy-MM-dd". Those two pieces of information get set in the `GetDailyHighTempRequest` object and passed to our API. With a bit of help from behind the scenes the `GetDailyHighTempRequest` object gets passed to the `getDailyHighTemp()` method in the `GetDailyHighTempActivity` class. Common terminology used here would be that the `GetDailyHighTempActivity` class "handles" the requests to the `GetDailyHighTemp` API.

To return the high temperature, the `GetDailyHighTempActivity` first calls the `TemperatureDao` with the city and date to get a `List<Integer>` containing the temperature for each hour in that city on that date. It then takes that list of temperatures and finds the highest temperature in the list.

This high temperature is then set in a `GetDailyHighTempResponse` object. Once again, with a bit of help from behind the scenes this `GetDailyHighTempResponse` object is returned to our client.

Phase 0: Preliminaries

Before we dive into the activity, let's ensure we're set-up for success.

Let's try calling the `GetDailyHighTemp` API for Atlanta on July 4, 2020.

A request to this API with the inputs "Atlanta" and "2020-07-04" is set-up to run when we use the `remotedebuggingandlogging-classroom-phase0` workflow. This should return 81 degrees Fahrenheit. An API response returning 81 on the command line looks like:

```
{"highTemp":81}
```

GOAL: Make sure the `ATACurriculumAlexaWeatherService` is set-up and running correctly on your laptop.

Phase 0 is complete when:

- You understand the inputs and outputs of the `GetDailyHighTemp` API
- `remotedebuggingandlogging-classroom-phase0` RDE workflow returns 81 as the `highTemp`

Phase 1: Add a log statement

Currently our API doesn't write anything to the logs. If we took a look at the logs after we ran the phase 0 workflow we would see something like:

```
START RequestId: 37558cbf-f7d4-1686-0a3c-c2c516c36aa7 Version: $LATEST
30 Jul 2020 03:07:53,520 [INFO ]  (main)
com.amazon.ata.remotedebuggingandlogging.classroom.resources.LambdaMain:24
: Entered request path
30 Jul 2020 03:07:55,843 [INFO ]  (main)
com.amazon.coral.service.DirectActivityInvokerFactory:106: Service
ATACurriculumAlexaWeatherService registered with 1 operations
[GetDailyHighTemp]
30 Jul 2020 03:07:55,944 [INFO ]  (main)
com.amazon.ata.remotedebuggingandlogging.classroom.resources.LambdaMain:29
: Endpoint constructed
30 Jul 2020 03:07:55,949 [INFO ]  (main)
com.amazon.coral.service.lambda.LambdaEndpoint:196: AWS internal request
ID: 37558cbf-f7d4-1686-0a3c-c2c516c36aa7
END RequestId: 37558cbf-f7d4-1686-0a3c-c2c516c36aa7
REPORT RequestId: 37558cbf-f7d4-1686-0a3c-c2c516c36aa7   Init Duration:
8693.32 ms       Duration: 2993.17 ms    Billed Duration: 3000 ms
Memory Size: 512 MB
     Max Memory Used: 105 MB
2020-07-30 03:07:56 240.10.1.4 - - [30/Jul/2020 03:07:56] "GET
/temperature/high/Atlanta/2020-07-04 HTTP/1.1" 200 -
REPORT RequestId: c003c41e-eb5e-1d41-5026-b3f3056bd835   Init Duration:
9075.88 ms       Duration: 3408.22 ms    Billed Duration: 3500 ms
Memory Size: 512 MB
```

You may notice we have this weird request ID appearing throughout the logs, "37558cbf-f7d4-1686-0a3c-c2c516c36aa7." This style of ID with dashes and alphanumeric characters is called a UUID. Each time we make a request to the service the new request gets an ID. We can see the log files for that particular request start with the line that has "START" and the request ID and then end with the line that has "END" and the request ID. In between those two you may notice that there isn't any mention of `GetDailyHighTemp` or the `TemperatureDao`.

Let's change that so that we can see each time a request is made specifically to the `GetDailyHighTemp` API vs other APIs in this service, like our average precipitation API. But let's also include some context!

Add an info log line as the first line of the `getDailyHighTemp` method in the `GetDailyHighTempActivity` class. This info log should mention which API has been called and what the inputs to that API were.

Once you've added the code, let's take a look at the log file. The logs are in our local RDE container. You will need to connect to the local RDE container. The app name in this case is `ATAClassroomSnippets_U2`.

Hints

- How do I connect to an RDE container again?

GOAL: Add an info log to the `GetDailyHighTempActivity` class

Phase 1 is complete when:

- You've added the log to the `GetDailyHighTempActivity` class
- You've run the `remotedebuggingandlogging-classroom-phase1` workflow to call the API
- You've seen the new logs you added in your local docker container

Phase 2: Errors from Fairbanks

A customer from Fairbanks has reported an error. They requested the high temperature in Fairbanks on January 15, 2020. They're saying that Alexa didn't tell them the correct high temperature.

Luckily we had set up the `remotedebuggingandlogging-classroom-phase1` workflow to call the API for Fairbanks on January 15, 2020. What a strange coincidence!

Take note of the high temperature that was returned. Was there helpful information in the logs that made you think there could be an issue?

As a group discuss: Is there anything there that can help you find out if the wrong high temperature was returned? Can you find any information related to the inputs or the outputs of the request? We're lucky that right now the log file is pretty small and we can look at the entire thing. The presenter of your group should be prepared to share your group's findings.

To get some more information let's attach a remote debugger! Set a break point on the first line of the `getDailyHighTemp` method in the `GetDailyHighTempActivity` class. Set-up your configurations for your remote debugger. The remote debug port for this application is "5012".

Use the `remotedebuggingandlogging-classroom-phase2` workflow to trigger a call to the API. We need to call the API, so something can get caught by our break point. The phase 2 workflow, has the same inputs as the phase 1 workflow, but runs on the debug app, so you can attach a remote debugger.

Once you see the below on the command line, you will need to click on the "Run" menu and then select the name of the debug configuration you created.

```
+ curl localhost:1112/temperature/high/Fairbanks/2020-01-15
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
```

```
  Current
                                 Dload  Upload   Total   Spent    Left
  Speed
    0     0    0     0    0     0      0      0 --:--:--  0:00:09 --:--:--
  0
```

As a group step the debugger through the class, inspecting the values of the different variables throughout. As you walk through the code think about what you think the correct high temp should be. Before you step to the next line in the code predict what you think the variable values should be. As a group, try to identify the bug, but don't fix it!

Hints

- How do I set up the remote debug configuration again?

GOAL: Identify the bug that is causing the wrong high temperature to be returned for Fairbanks on January 15, 2020. But do **NOT** fix the bug.

Phase 2 is complete when:

- You have set a break point in the `GetDailyHighTempActivity` class and using a remote debugger walked through the code
- Your group has **identified** the bug

Phase 3: What logs should we add?

Take a moment to think about what logs you think we should add to the `GetDailyHighTempActivity` class to expose this issue in the log files. What would have been helpful to diagnose the bug so you didn't have to attach a remote debugger? That may not always be an option. You should only add logs to this class. For each log you think we should add, decide:

- Which line number to add the log statement after
- What the log level should be
- What the message should contain, including any helpful context

We will be learning more about handling exceptions later in this unit, so it's okay if the catch blocks look unfamiliar. Just know that the context for the Exception is available within the catch block. When we are logging exceptions, we want to pass the full exception object into the log. Here the exception object is represented by the variable `e`. Unlike including the values of other variables in our log messages we do not need to use "{}" substitution. We can just pass it `e` and the full stack trace (exception messaging) will be printed in the log.

After a couple of minutes, chat with your group about what logs you should add. Once you've compiled a comprehensive list, have the presenter of your group post the logs you want to add to the "Alexa Weather Skill" discussion group in canvas. Once you've posted, take a look at and comment on what other groups have submitted to the discussion.

GOAL: Identify which logs you want to add to the `GetDailyHighTempActivity` class

Phase 3 is complete when:

- Your group has posted the logs you want to add to the `GetDailyHighTempActivity` class in the discussion group in canvas
- You have reviewed what other groups have posted in the discussion group.

Extension 1

Now let's add those logs!

Once you've added the log lines to the `GetDailyHighTempActivity` class, run the `remotedebuggingandlogging-classroom-extension1` workflow to call the API one last time for Fairbanks on January 15, 2020. Once again connect to your local RDE container. First try using `grep` with some of the new log text you've added to find the new log lines. Does this make you realize anything you would want to change about your log messages to make them more "grep-able."

Then use less to read through the entire log files.

Hints

- Remind me how to create a Logger.

GOAL: Add the logs to the `GetDailyHighTempActivity` class and view the newly created logs in the RDE container

Extension 1 is complete when:

- You've added the logs to the `GetDailyHighTempActivity` class
- You've run the `remotedebuggingandlogging-classroom-extension1` workflow to call the API
- You've seen the new logs you added in your local docker container

Extension 2

This is not only saying the high temp was 81, but the 200 status code tells us that the API successfully executed. If the API failed for some reason we would see a status code in the 400s or 500s and instead of a body with the high temp, an error messages would be returned. Status codes will be covered in more detail in a later unit. This strange formatting you're seeing is called JSON. JSON will be covered more in the project's first mastery task.

We've set up the `remotedebuggingandlogging-classroom-extension2` workflow to call the API twice with different inputs. Try running this workflow and then taking a look at your logs locally in the RDE container.

Answer the extension quiz in canvas with what city and date each of the requests were made with and what value or extension was returned.