Sequential agents

The `SequentialAgent`

The `SequentialAgent` is a workflow agent that executes its sub-agents in the order they are specified in the list.

Use the `SequentialAgent` when you want the execution to occur in a fixed, strict order.

Example

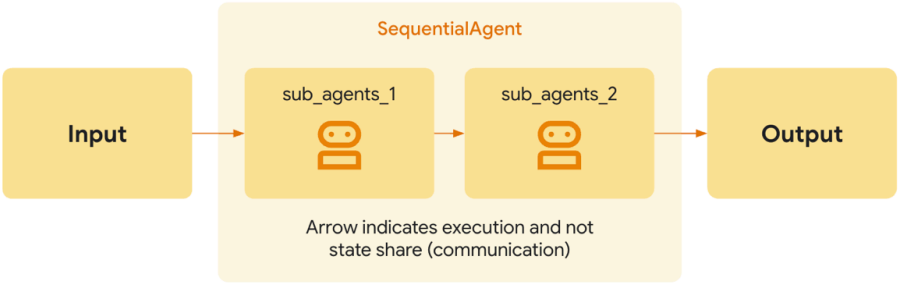
- You want to build an agent that can summarize any webpage, using two tools: `Get Page Contents` and `Summarize Page`. Because the agent must always call `Get Page Contents` before calling `Summarize Page` (you can't summarize from nothing!), you should build your agent using a `SequentialAgent`.

As with other workflow agents, the `SequentialAgent` is not powered by an LLM, and is thus deterministic in how it executes. That being said, workflow agents are concerned only with their execution (i.e. in sequence), and not their internal logic; the tools or sub-agents of a workflow agent may or may not utilize LLMs.

How it works

When the `SequentialAgent`'s `Run Async` method is called, it performs the following actions:

1. **Iteration:** It iterates through the sub agents list in the order they were provided.
2. **Sub-Agent Execution:** For each sub-agent in the list, it calls the sub-agent's `Run Async` method.



Full Example: Code Development Pipeline

Consider a simplified code development pipeline:

- **Code Writer Agent:** An LLM Agent that generates initial code based on a specification.
- **Code Reviewer Agent:** An LLM Agent that reviews the generated code for errors, style issues, and adherence to best practices. It receives the output of the Code Writer Agent.
- **Code Refactorer Agent:** An LLM Agent that takes the reviewed code (and the reviewer's comments) and refactors it to improve quality and address issues.

A `SequentialAgent` is perfect for this:

```
SequentialAgent(sub_agents=[CodeWriterAgent,
CodeReviewerAgent, CodeRefactorerAgent])
```

This ensures the code is written, *then* reviewed, and *finally* refactored, in a strict, dependable order. **The output from each sub-agent is passed to the next by storing them in state via Output Key**.

Code

Python

```
# Part of agent.py --> Follow
https://google.github.io/adk-docs/get-started/quickstart/
to learn the setup

# --- 1. Define Sub-Agents for Each Pipeline Stage ---
```

```
# Code Writer Agent
# Takes the initial specification (from user query) and
writes code.
code_writer_agent = LlmAgent(
    name="CodeWriterAgent",
    model=GEMINI_MODEL,
    # Change 3: Improved instruction
    instruction="""You are a Python Code Generator.
Based *only* on the user's request, write Python code that
fulfills the requirement.
Output *only* the complete Python code block, enclosed in
triple backticks (```python ... ```).
Do not add any other text before or after the code block.
""",
    description="Writes initial Python code based on a
specification.",
    output_key="generated_code" # Stores output in
state['generated_code']
)

# Code Reviewer Agent
# Takes the code generated by the previous agent (read
from state) and provides feedback.
code_reviewer_agent = LlmAgent(
    name="CodeReviewerAgent",
    model=GEMINI_MODEL,
    # Change 3: Improved instruction, correctly using
state key injection
    instruction="""You are an expert Python Code Reviewer.
    Your task is to provide constructive feedback on the
provided code.

    **Code to Review:**
    ```python
    {generated_code}
    ```

Review Criteria:
1. **Correctness:** Does the code work as intended? Are
there logic errors?
2. **Readability:** Is the code clear and easy to
understand? Follows PEP 8 style guidelines?
3. **Efficiency:** Is the code reasonably efficient? Any
obvious performance bottlenecks?
4. **Edge Cases:** Does the code handle potential edge
cases or invalid inputs gracefully?
5. **Best Practices:** Does the code follow common Python
best practices?

Output:
Provide your feedback as a concise, bulleted list. Focus
on the most important points for improvement.
If the code is excellent and requires no changes, simply
```

```
state: "No major issues found."
Output *only* the review comments or the "No major issues"
statement.
""",
    description="Reviews code and provides feedback.",
    output_key="review_comments", # Stores output in
state['review_comments']
)


# Code Refactorer Agent
# Takes the original code and the review comments (read
from state) and refactors the code.
code_refactorer_agent = LlmAgent(
    name="CodeRefactorerAgent",
    model=GEMINI_MODEL,
    # Change 3: Improved instruction, correctly using
state key injection
    instruction="""You are a Python Code Refactoring AI.
Your goal is to improve the given Python code based on the
provided review comments.

  **Original Code:**
  ```python
  {generated_code}
  ```

  Review Comments:
  {review_comments}

Task:
Carefully apply the suggestions from the review comments
to refactor the original code.
If the review comments state "No major issues found,"
return the original code unchanged.
Ensure the final code is complete, functional, and
includes necessary imports and docstrings.

Output:
Output *only* the final, refactored Python code block,
enclosed in triple backticks (```python ... ```).
Do not add any other text before or after the code block.
""",
    description="Refactors code based on review
comments.",
    output_key="refactored_code", # Stores output in
state['refactored_code']
)

--- 2. Create the SequentialAgent ---
This agent orchestrates the pipeline by running the
sub_agents in order.
code_pipeline_agent = SequentialAgent(
```

```
    name="CodePipelineAgent",
    sub_agents=[code_writer_agent, code_reviewer_agent,
code_refactorer_agent],
    description="Executes a sequence of code writing,
reviewing, and refactoring.",
    # The agents will run in the order provided: Writer ->
Reviewer -> Refactorer
)

# For ADK tools compatibility, the root agent must be
named `root_agent`
root_agent = code_pipeline_agent
```

Java

```
import com.google.adk.agents.LlmAgent;
import com.google.adk.agents.SequentialAgent;
import com.google.adk.events.Event;
import com.google.adk.runner.InMemoryRunner;
import com.google.adk.sessions.Session;
import com.google.genai.types.Content;
import com.google.genai.types.Part;
import io.reactivex.rxjava3.core.Flowable;

public class SequentialAgentExample {

  private static final String APP_NAME =
"CodePipelineAgent";
  private static final String USER_ID = "test_user_456";
  private static final String MODEL_NAME = "gemini-2.0-
flash";

  public static void main(String[] args) {
    SequentialAgentExample sequentialAgentExample = new
SequentialAgentExample();
    sequentialAgentExample.runAgent(
        "Write a Java function to calculate the factorial
of a number.");
  }

  public void runAgent(String prompt) {

    LlmAgent codeWriterAgent =
        LlmAgent.builder()
            .model(MODEL_NAME)
            .name("CodeWriterAgent")
            .description("Writes initial Java code based
on a specification.")
            .instruction(
                """
                You are a Java Code Generator.
                Based *only* on the user's request, write
Java code that fulfills the requirement.
```

```
                Output *only* the complete Java code
block, enclosed in triple backticks (```java ... ```).
                Do not add any other text before or after
the code block.
                """)
            .outputKey("generated_code")
            .build();

    LlmAgent codeReviewerAgent =
        LlmAgent.builder()
            .model(MODEL_NAME)
            .name("CodeReviewerAgent")
            .description("Reviews code and provides
feedback.")
            .instruction(
                """
                    You are an expert Java Code Reviewer.
                    Your task is to provide constructive
feedback on the provided code.

                    **Code to Review:**
                    ```java
                    {generated_code}
                    ```

                    Review Criteria:
                    1. **Correctness:** Does the code
work as intended? Are there logic errors?
                    2. **Readability:** Is the code clear
and easy to understand? Follows Java style guidelines?
                    3. **Efficiency:** Is the code
reasonably efficient? Any obvious performance bottlenecks?
                    4. **Edge Cases:** Does the code
handle potential edge cases or invalid inputs gracefully?
                    5. **Best Practices:** Does the code
follow common Java best practices?

                    Output:
                    Provide your feedback as a concise,
bulleted list. Focus on the most important points for
improvement.
                    If the code is excellent and requires
no changes, simply state: "No major issues found."
                    Output *only* the review comments or
the "No major issues" statement.
                """)
            .outputKey("review_comments")
            .build();

    LlmAgent codeRefactorerAgent =
        LlmAgent.builder()
            .model(MODEL_NAME)
            .name("CodeRefactorerAgent")
            .description("Refactors code based on review
```

```
comments.")
            .instruction(
                """
                You are a Java Code Refactoring AI.
                Your goal is to improve the given Java
code based on the provided review comments.

                  **Original Code:**
                  ```java
                  {generated_code}
                  ```

                  Review Comments:
                  {review_comments}

                Task:
                Carefully apply the suggestions from the
review comments to refactor the original code.
                If the review comments state "No major
issues found," return the original code unchanged.
                Ensure the final code is complete,
functional, and includes necessary imports and docstrings.

                Output:
                Output *only* the final, refactored Java
code block, enclosed in triple backticks (```java ...
```).
                Do not add any other text before or after
the code block.
                """)
            .outputKey("refactored_code")
            .build();

    SequentialAgent codePipelineAgent =
        SequentialAgent.builder()
            .name(APP_NAME)
            .description("Executes a sequence of code
writing, reviewing, and refactoring.")
            // The agents will run in the order provided:
Writer -> Reviewer -> Refactorer
            .subAgents(codeWriterAgent, codeReviewerAgent,
codeRefactorerAgent)
            .build();

    // Create an InMemoryRunner
    InMemoryRunner runner = new
InMemoryRunner(codePipelineAgent, APP_NAME);
    // InMemoryRunner automatically creates a session
service. Create a session using the service
    Session session =
runner.sessionService().createSession(APP_NAME,
USER_ID).blockingGet();
    Content userMessage =
Content.fromParts(Part.fromText(prompt));
```
```

```
    // Run the agent
    Flowable<Event> eventStream = runner.runAsync(USER_ID,
session.id(), userMessage);

    // Stream event response
    eventStream.blockingForEach(
        event -> {
          if (event.finalResponse()) {
            System.out.println(event.stringifyContent());
          }
        });
  }
}
```