



This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and extend across the width of the page. There are no margins, text, or other markings on the paper.

SPRING AI

GENERATIVE ARTIFICIAL INTELLIGENCE CON JAVA

Simone Scannapieco

Corso avanzato per Venis S.p.A, Venezia, Italia

Novembre 2025

Note

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

-  Simone Scannapieco

Note

[illegible]

 Simone Scannapieco
 Spring AI - Corso avanzato
 Venis S.p.A, Venezia, IT
 4 / 18

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

-
- The diagram illustrates the flow of a tool call request through the Spring AI framework. It features three main components: a green 'Tool' box at the top, a yellow 'Spring AI' box in the center, and an orange 'AI Model' box at the bottom. A 'Chat Request' box on the left contains a green 'Tool Definition' box, which lists 'name', 'description', and 'input schema'. A 'Chat Response' box is on the right. The flow is numbered 1 through 6: 1. A solid arrow from 'Tool Definition' to 'Dispatch Tool Call Requests' in Spring AI. 2. A solid arrow from 'AI Model' to 'Dispatch Tool Call Requests'. 3. A solid arrow from 'Dispatch Tool Call Requests' to 'Tool'. 4. A solid arrow from 'Tool' to 'Dispatch Tool Call Requests'. 5. A dashed arrow from 'Dispatch Tool Call Requests' to 'AI Model'. 6. A solid arrow from 'Dispatch Tool Call Requests' to 'Chat Response'.

 Simone Scannapieco

Spring AI - Corso avanzato

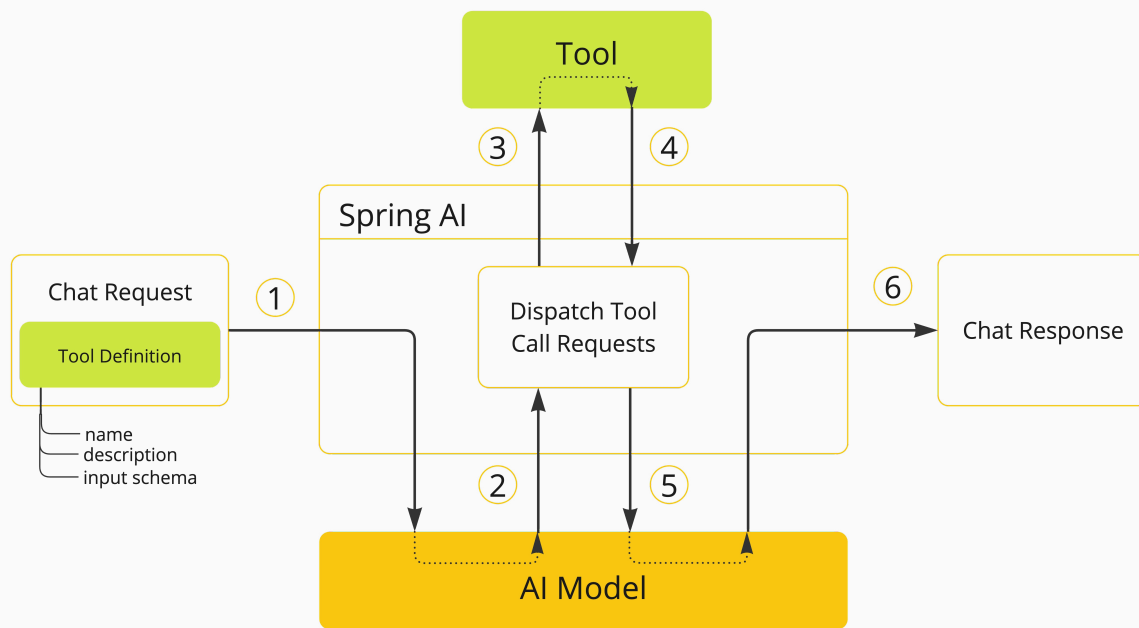
 Venis S.p.A, Venezia, IT

5 / 18

Note

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

- 2** Se il LLM decide di chiamare un *tool*, invia una *response* con nome del *tool* e parametri istanziati, seguendo lo schema definito per il *tool*



©Spring AI

Note

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

-
- The diagram illustrates the Spring AI Tool Dispatching Flow, showing the interaction between a Chat Request, Spring AI, an AI Model, and a Tool.
- Components:**
- Chat Request:** Contains a **Tool Definition** (name, description, input schema).
 - Spring AI:** Contains the **Dispatch Tool Call Requests** component.
 - AI Model:** The model that processes the request.
 - Tool:** The external tool that performs the action.
- Flow Steps:**
- 1:** The **Chat Request** (containing the **Tool Definition**) is sent to the **Dispatch Tool Call Requests** component within **Spring AI**.
 - 2:** The **Dispatch Tool Call Requests** component sends the request to the **AI Model**.
 - 3:** The **AI Model** sends the result back to the **Dispatch Tool Call Requests** component.
 - 4:** The **Dispatch Tool Call Requests** component sends the request to the **Tool**.
 - 5:** The **Tool** sends the result back to the **Dispatch Tool Call Requests** component.
 - 6:** The **Dispatch Tool Call Requests** component sends the final **Chat Response**.

 Simone Scannapieco

Note

[illegible]

-
- The diagram illustrates the flow of a tool call in Spring AI, involving three main components: Chat Request, Spring AI, and AI Model.
- Chat Request** (Left): Contains a **Tool Definition** (green box) with attributes: `name`, `description`, and `input schema`.
 - Spring AI** (Center): A large container with a **Dispatch Tool Call Requests** box inside.
 - AI Model** (Bottom): The model that processes requests and returns responses.
 - Tool** (Top): The external tool being called.
- The flow is numbered 1 through 6:
- 1**: Chat Request sends the Tool Definition to Spring AI.
 - 2**: Spring AI sends the Tool Definition to the AI Model.
 - 3**: AI Model sends a request to the Tool.
 - 4**: Tool returns a response to Spring AI.
 - 5**: Spring AI sends the response back to the AI Model.
 - 6**: AI Model sends the final Chat Response to the Chat Request.
- ```
graph TD; subgraph Chat_Request [Chat Request]; TD[Tool Definition]; end; subgraph Spring_AI [Spring AI]; DTCR[Dispatch Tool Call Requests]; end; subgraph AI_Model [AI Model]; end; Tool[Tool]; Chat_Request -- 1 --> Spring_AI; Spring_AI -- 2 --> AI_Model; AI_Model -. 3 .-> Tool; Tool -- 4 --> Spring_AI; Spring_AI -- 5 --> AI_Model; AI_Model -- 6 --> Chat_Response[Chat Response];
```

 Simone Scannapieco

### Note

[illegible]

- 
- The diagram illustrates the Spring AI Tool Call Flow, showing the interaction between a Chat Request, Spring AI, a Tool, and an AI Model.
- Components:**
- Chat Request:** Contains a **Tool Definition** (name, description, input schema).
  - Spring AI:** Contains the **Dispatch Tool Call Requests** component.
  - Tool:** A component that can be called by the AI Model.
  - AI Model:** The model that generates the response based on the chat request and tool calls.
- Flow Steps:**
- 1:** The **Chat Request** (containing the **Tool Definition**) is sent to **Spring AI**.
  - 2:** **Spring AI** sends the **Dispatch Tool Call Requests** to the **AI Model**.
  - 3:** The **AI Model** calls the **Tool** (indicated by a dashed arrow).
  - 4:** The **Tool** returns the result to **Spring AI**.
  - 5:** **Spring AI** sends the tool call results back to the **AI Model**.
  - 6:** The **AI Model** generates the final **Chat Response**.

 Simone Scannapieco

Spring AI - Corso avanzato

 Venis S.p.A, Venezia, IT

5 / 18

### Note

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

- 
- The diagram illustrates the Spring AI Tool Dispatching Flow, showing the interaction between a Chat Request, Spring AI, an AI Model, and a Tool.
- Components:**
- Chat Request:** Contains a **Tool Definition** (green box) with attributes: name, description, and input schema.
  - Spring AI:** Contains the **Dispatch Tool Call Requests** component (white box).
  - AI Model:** (Yellow box) Receives requests and returns results.
  - Tool:** (Green box) Provides the actual tool implementation.
- Flow Steps:**
- 1:** Chat Request (Tool Definition) is sent to Spring AI.
  - 2:** Spring AI sends the request to the AI Model.
  - 3:** AI Model returns the result to Spring AI.
  - 4:** Spring AI sends the request to the Tool.
  - 5:** Tool returns the result to Spring AI.
  - 6:** Spring AI sends the final Chat Response.
- ```
graph LR; CR[Chat Request] -- 1 --> SA[Spring AI]; subgraph SA [Spring AI]; DTCR[Dispatch Tool Call Requests]; end; SA -- 2 --> AM[AI Model]; AM -- 3 --> SA; SA -- 4 --> T[Tool]; T -- 5 --> SA; SA -- 6 --> CR2[Chat Response];
```

 Simone Scannapieco

Spring AI - Corso avanzato

 Venis S.p.A, Venezia, IT

5 / 18

Note

[illegible]

-
- The diagram illustrates the flow of a tool call request in Spring AI. It shows the interaction between a Chat Request, Spring AI, an AI Model, and a Tool.
- Chat Request:** Contains a **Tool Definition** (green box) and a **returnDirect=true** flag (blue box).
 - Spring AI:** Contains a **Dispatch Tool Call Requests** component.
 - AI Model:** A yellow box at the bottom.
 - Tool:** A green box at the top.
- The flow is numbered 1 through 5:
- 1:** The Chat Request is sent to the Spring AI Dispatch Tool Call Requests component.
 - 2:** The Spring AI Dispatch Tool Call Requests component sends the request to the AI Model.
 - 3:** The AI Model sends the response back to the Spring AI Dispatch Tool Call Requests component.
 - 4:** The Spring AI Dispatch Tool Call Requests component sends the response to the Tool.
 - 5:** The Tool sends the final Chat Response back to the Spring AI Dispatch Tool Call Requests component.

 Simone Scannapieco

Spring AI - Corso avanzato

 Venis S.p.A, Venezia, IT

6 / 18

Note

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

- 7/18

Note

This image shows a single sheet of white paper with horizontal blue or grey ruling lines. The lines are evenly spaced and run across the width of the page. There are approximately 20 lines visible. The paper has a slight shadow on the right side, suggesting it's resting on a surface. There is no handwriting or other markings on the paper.

TOOL CALLING

METHOD AS TOOL

Note

[illegible]

```
@Component
public class TimeTools {

    @Tool(name="getCurrentLocalTime",
        description="Ottieni l'ora corrente nel fuso orario dell'utente.")
    String getCurrentLocalTime() {
        ...
    }

    @Tool(name="getCurrentTime",
        description="Ottieni l'ora corrente nel fuso orario specificato.",
        returnDirect=true)
    public String getCurrentTime(@ToolParam(description = "Valore che rappresenta il fuso orario.") String timeZone) {
        ...
    }
}
```

- Simone Scannapieco
 Spring AI - Corso avanzato
 Venis S.p.A, Venezia, IT
 8 / 18

[illegible]

```
@Component
public class TimeTools {

    @Tool(name="getCurrentLocalTime",
        description="Ottieni l'ora corrente nel fuso orario dell'utente.")
    String getCurrentLocalTime() {
        ...
    }

    @Tool(name="getCurrentTime",
        description="Ottieni l'ora corrente nel fuso orario specificato.",
        returnDirect=true)
    public String getCurrentTime(@ToolParam(description = "Valore che rappresenta il fuso orario.") String timeZone) {
        ...
    }
}
```

- >_ (thinking) – Scansiono i tool a disposizione... –
- >_ (thinking) – Vedo che ho dei tool che gestiscono il recupero dell'ora... –
- >_ (thinking) – Il primo tool è relativo alla posizione dell'utente... –
- >_ (thinking) – Il secondo tool è più astratto e determina la posizione attraverso parametrizzazione del fuso orario... –
- >_ (thinking) – ...Il fuso orario di New York è America/New York... –
- >_ "R: Utilizza il tool `getCurrentTime('America/New York')`."

 Simone Scannapieco

Note

[illegible]

Tool-aware ChatClient - approccio statico

```
@Bean
public ChatClient ollamaTimeToolsChatClient(OllamaChatModel ollamaChatModel, TimeTools timeTools) {

    ChatClient.Builder chatClientBuilder = ChatClient.builder(ollamaChatModel);

    return chatClientBuilder
        .defaultTools(timeTools)
        ...
        .build();

}
```

Tool-aware ChatClient - approccio dinamico

```
@Override
public Answer getOllamaTimeToolsLocalTimeAnswer(QuestionRequest request) {

    return new Answer(this.ollamaTimeToolsChatClient
        .prompt()
        .tools(timeTools)
        ...
        .call()
        .content());
}
```

Note

[illegible]

```
public class TimeTools {

    String getCurrentLocalTime() {
        ...
    }

    public String getCurrentTime(
        @ToolParam(description = "Valore che rappresenta il fuso orario.") String timeZone) {
        ...
    }

}
```



Note

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

```
import java.lang.reflect.Method;

Method method = MethodToolCallbackReflectionUtils.findMethod(TimeTools.class, "getCurrentLocalTime");
ToolCallback toolCallback = MethodToolCallback.builder()
    .toolDefinition(ToolDefinitions.builder(method)
        .name("getCurrentLocalTime")
        .description("Ottieni l'ora corrente nel fuso orario dell'utente.")
        .build())
    .toolMethod(method)
    .toolObject(new TimeTools())
    .toolMetadata(ToolMetadata.builder()
        .returnDirect(true)
        .build())
    .build();
```

-  Simone Scannapieco

Note

This image shows a single sheet of white paper with horizontal blue or grey ruling lines. The lines are evenly spaced and run across the width of the page, providing a template for handwriting practice. There are no margins, text, or other markings on the paper.

```
@Bean
public ChatClient ollamaTimeToolsChatClient(OllamaChatModel ollamaChatModel) {

    ChatClient.Builder chatClientBuilder = ChatClient.builder(ollamaChatModel);

    return chatClientBuilder
        .defaultToolCallbacks(toolCallback)
        ...
        .build();
}
```

```
@Override
public Answer getOllamaTimeToolsLocalTimeAnswer(QuestionRequest request) {

    return new Answer(this.ollamaTimeToolsChatClient
        .prompt()
        .toolCallbacks(toolCallback)
        ...
        .call()
        .content());
}
```

Note

[illegible]

-  Simone Scannapieco

This image shows a single sheet of white paper with horizontal blue ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

TOOL CALLING

FUNCTION AS TOOL

Note

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

```
@Configuration(proxyBeanMethods = false)
class WeatherTools {

    public static final String GET_TEMP_IN_LOCATION_FUNCTION_NAME = "getTemperatureInLocation";

    @Bean(GET_TEMP_IN_LOCATION_FUNCTION_NAME)
    @Description("Ottieni la temperatura corrente nella località specificata.")
    Function<WeatherRequest, TemperatureResponse> currentTemperature() {
        return new TemperatureService();
    }
}

public enum Unit { C, F }

public record WeatherRequest(
    @ToolParam(description = "Il nome di una città o di una nazione.") String location, Unit unit) {}

public record TemperatureResponse(double temp, Unit unit) {}
```

- ⚠️ **Tipi funzionali supportati:** `Function<I, O>`, `Supplier<O>`, `Consumer<I>`, `BiFunction<I1, I2, O>`
- ➡️ Ciascun funzionale disponibile come *tool* attraverso definizione `@Bean`
 - ⚠️ Definire nome come **costante esportabile** per mancanza di garanzia *type safety*
- ➡️ `@Description` come *description* di MAT
- ➡️ `@ToolParam` con medesima logica di MAT

Note

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

```
@Bean
public ChatClient ollamaWeatherToolsChatClient(OllamaChatModel ollamaChatModel) {

    ChatClient.Builder chatClientBuilder = ChatClient.builder(ollamaChatModel);

    return chatClientBuilder
        .defaultToolNames(WeatherTools.GET_TEMP_IN_LOCATION_FUNCTION_NAME)
        ...
        .build();
}
```

```
@Override
public TemperatureResponse getOllamaTemperatureToolAnswer(@RequestBody WeatherRequest request) {

    return this.ollamaWeatherToolsChatClient
        .toolNames(WeatherTools.GET_TEMP_IN_LOCATION_FUNCTION_NAME)
        ...
        .call()
        .content();
}
```

This image shows a single sheet of white paper with horizontal blue or grey ruling lines. The lines are evenly spaced and run across the width of the page. There are approximately 20 lines visible. The paper has a slight shadow on the right side, suggesting it's resting on a surface. There is no handwriting or other markings on the paper.

```
public class TemperatureService implements Function<WeatherRequest, TemperatureResponse> {

    public WeatherResponse apply(WeatherRequest request) {
        ...
    }

}

public enum Unit { C, F }

public record WeatherRequest(
    @ToolParam(description = "Il nome di una città o di una nazione.") String location, Unit unit) {}

public record TemperatureResponse(double temp, Unit unit) {}
```

- ➔ Richiesta sola la logica pertinente al funzionale scelto (es. `apply` per `Function`)

Note

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Utilizzo approccio *function-tool* callback

```
public static final String GET_TEMP_IN_LOCATION_FUNCTION_NAME = "getTemperatureInLocation";

ToolCallback toolCallback = FunctionToolCallback
    .builder(GET_TEMP_IN_LOCATION_FUNCTION_NAME, new TemperatureService())
    .description("Ottieni la temperatura corrente nella località specificata.")
    .inputType(WeatherRequest.class)
    .toolMetadata(ToolMetadata.builder()
        .returnDirect(true)
        .build())
    .build();
```

- ➡ `FunctionToolCallback.Builder` linka un identificativo testuale ad una istanza di una nuova *function tool*
- ➡ Costruito un `ToolCallback` che definisce la logica di utilizzo del *tool*
 - ➡ `ToolDefinitions.Builder` permette di definire nome, descrizione e schema del *tool*
 - ➡ `ToolMetadata.Builder` responsabile della definizione strategia *default* vs *direct*
- ⚠ Definizione `ToolCallback` *under-the-hood* anche con approccio dichiarativo

Note

[illegible]

```
@Bean
public ChatClient ollamaWeatherToolsChatClient(OllamaChatModel ollamaChatModel) {

    ChatClient.Builder chatClientBuilder = ChatClient.builder(ollamaChatModel);

    return chatClientBuilder
        .defaultToolCallbacks(toolCallBack)
        ...
        .build();
}
```

```
@Override
public TemperatureResponse getOllamaTemperatureToolAnswer(@RequestBody WeatherRequest request) {

    return this.ollamaWeatherToolsChatClient
        .toolCallBacks(toolCallback)
        ...
        .call()
        .content();
}
```

Note

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.