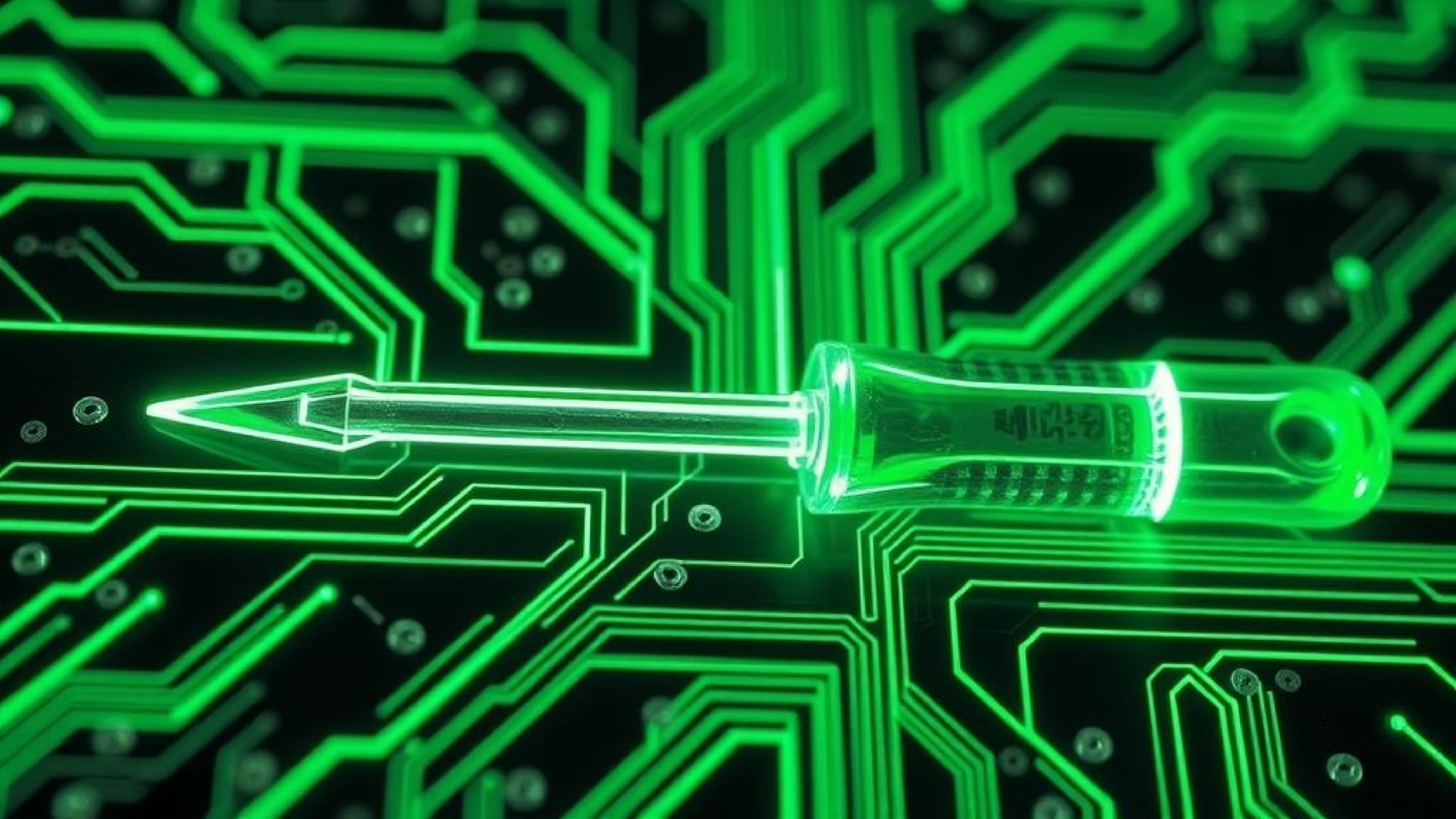
MAHARISHI INTERNATIONAL UNIVERSITY

HIGHER EDUCATION • FOR HIGHER CONSCIOUSNESS



Spring AI Tool Calling

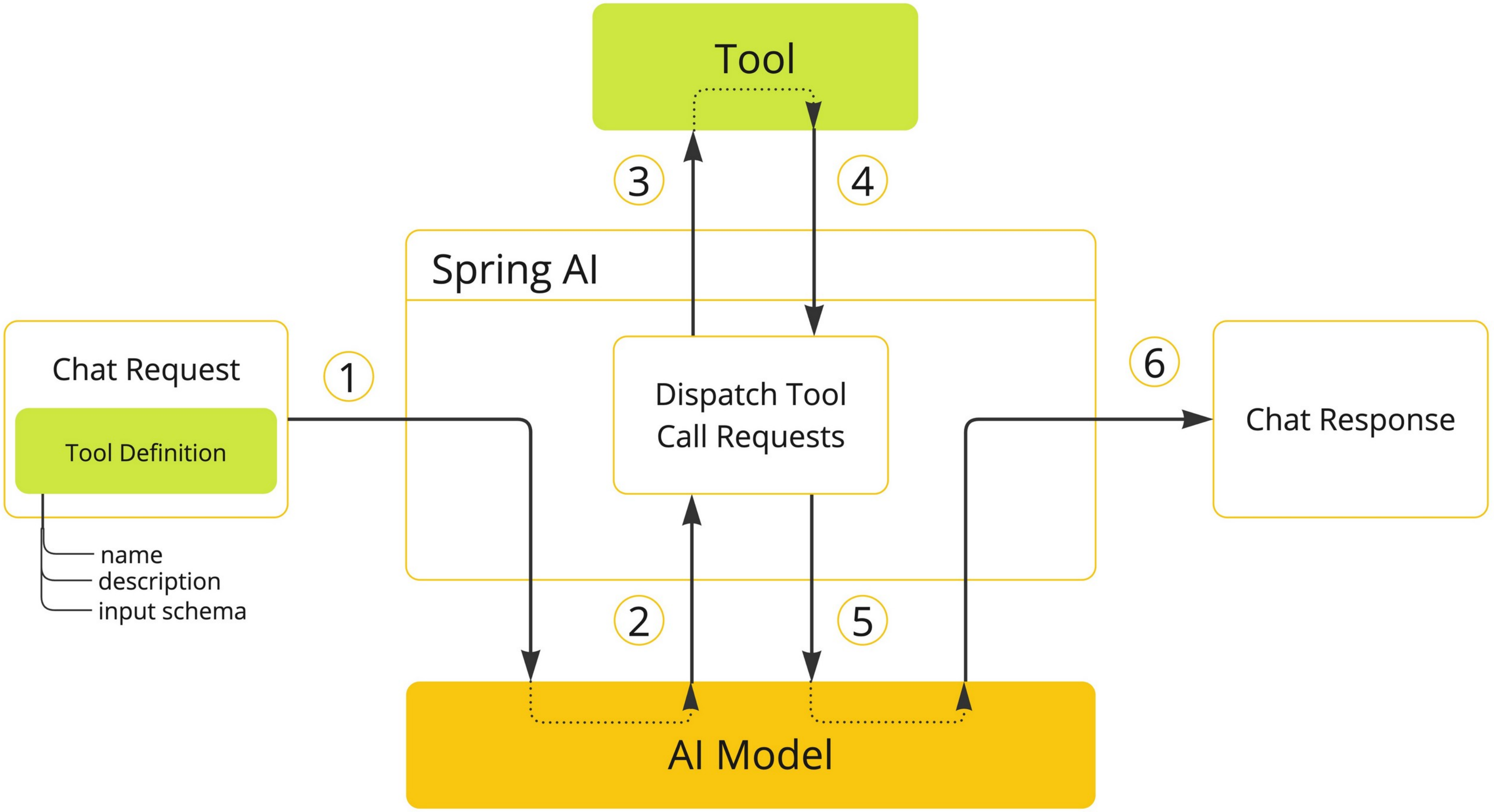
- Tool calling (also known as function calling) is a common pattern in AI applications allowing a model to interact with a set of APIs, or tools, augmenting its capabilities.

- Model Context Protocol (MCP) is a programming language agnostic protocol for tool calling
  - Requires you to make MCP servers and clients
  - We will not cover it in this presentation

Overview

- Declarative Methods
- Programmatic Functions
- Tool Inputs
- Tool Ouput

Structure



Tool
3
4
Spring AI
Chat Request
Tool Definition
1
Dispatch Tool Call Requests
6
Chat Response
name
description
input schema
2
5
AI Model

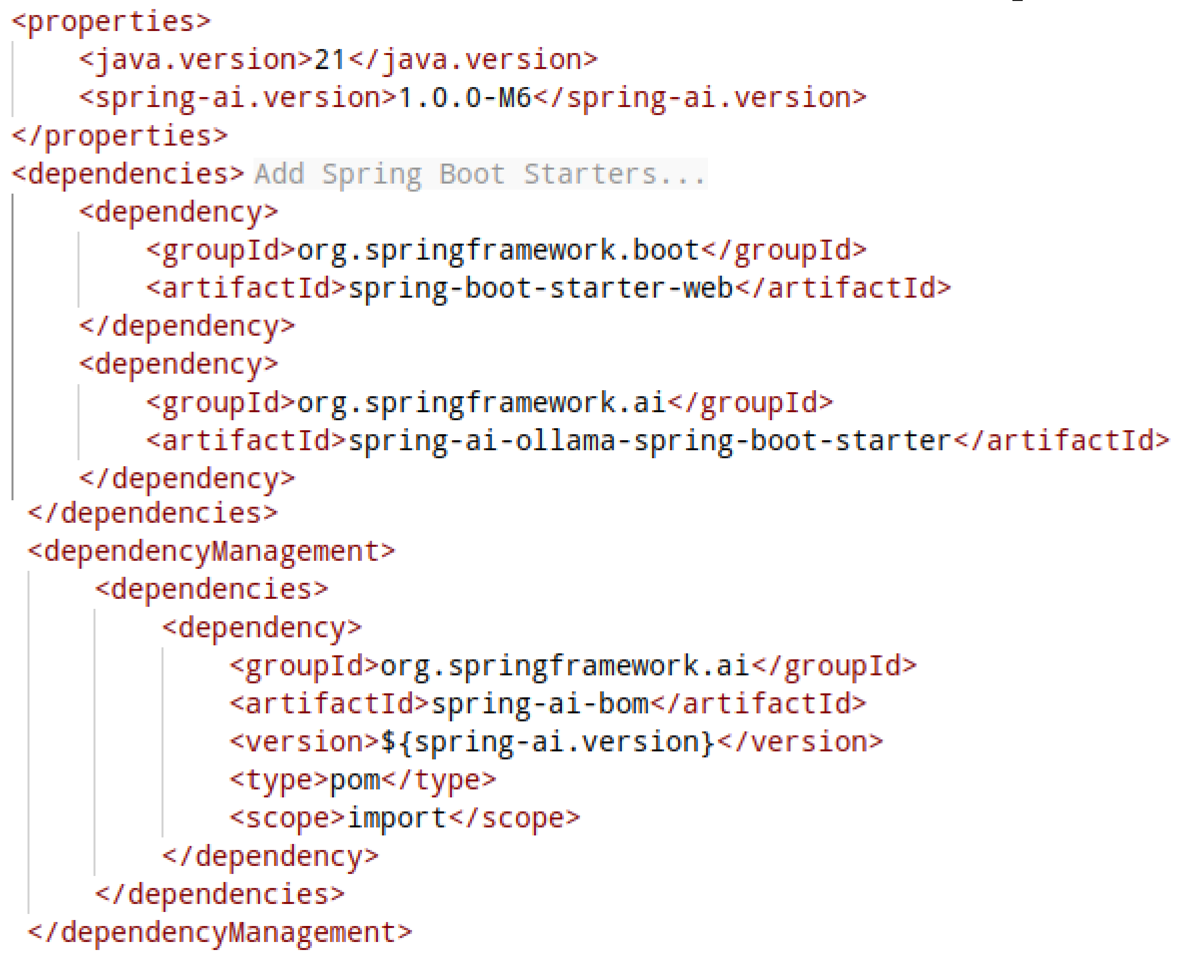
Different Ways to Create

- Methods as Tools
- Functions as Tools

- Declarative Specificaion (@Tool)
- Programmatic Specification (callback)

Declarative Methods

Maven Dependencies



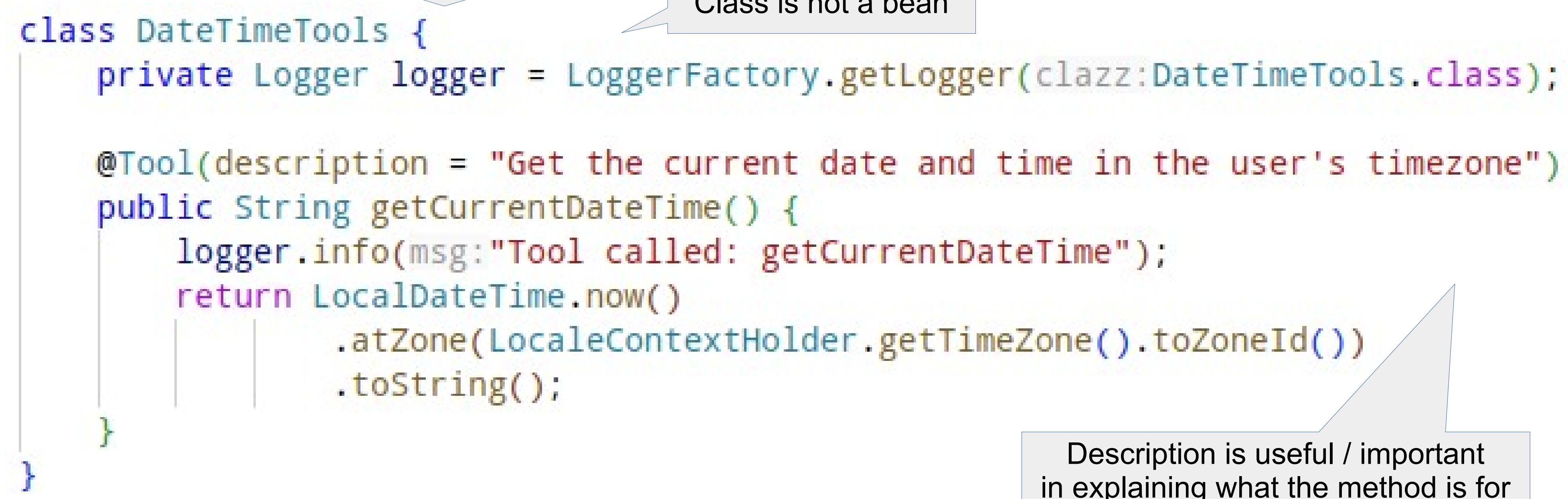
```
<properties>
    <java.version>21</java.version>
    <spring-ai.version>1.0.0-M6</spring-ai.version>
</properties>
<dependencies> Add Spring Boot Starters...
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.ai</groupId>
        <artifactId>spring-ai-ollama-spring-boot-starter</artifactId>
    </dependency>
 </dependencies>
 <dependencyManagement>
     <dependencies>
         <dependency>
             <groupId>org.springframework.ai</groupId>
             <artifactId>spring-ai-bom</artifactId>
             <version>${spring-ai.version}</version>
             <type>pom</type>
             <scope>import</scope>
         </dependency>
     </dependencies>
 </dependencyManagement>
```

From Spring Initializr

DateTimeTool

Simply add the @Tool annotation
to a regular method

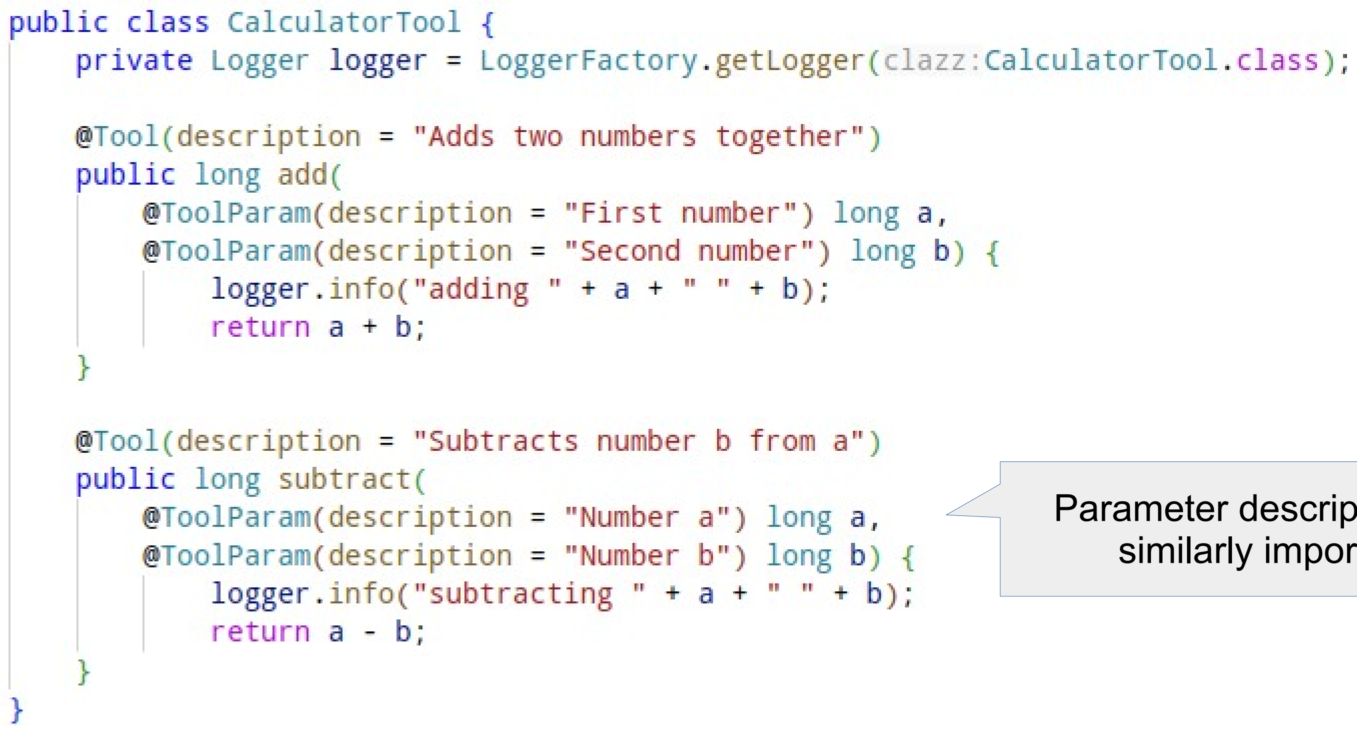
Class is not a bean



```
class DateTimeTools {
    private Logger logger = LoggerFactory.getLogger(clazz:DateTimeTools.class);

    @Tool(description = "Get the current date and time in the user's timezone")
    public String getCurrentDateTime() {
        logger.info(msg:"Tool called: getCurrentDateTime");
        return LocalDateTime.now()
                .atZone(LocaleContextHolder.getTimeZone().toZoneId())
                .toString();
    }
}
```

Description is useful / important
in explaining what the method is for
to the LLM

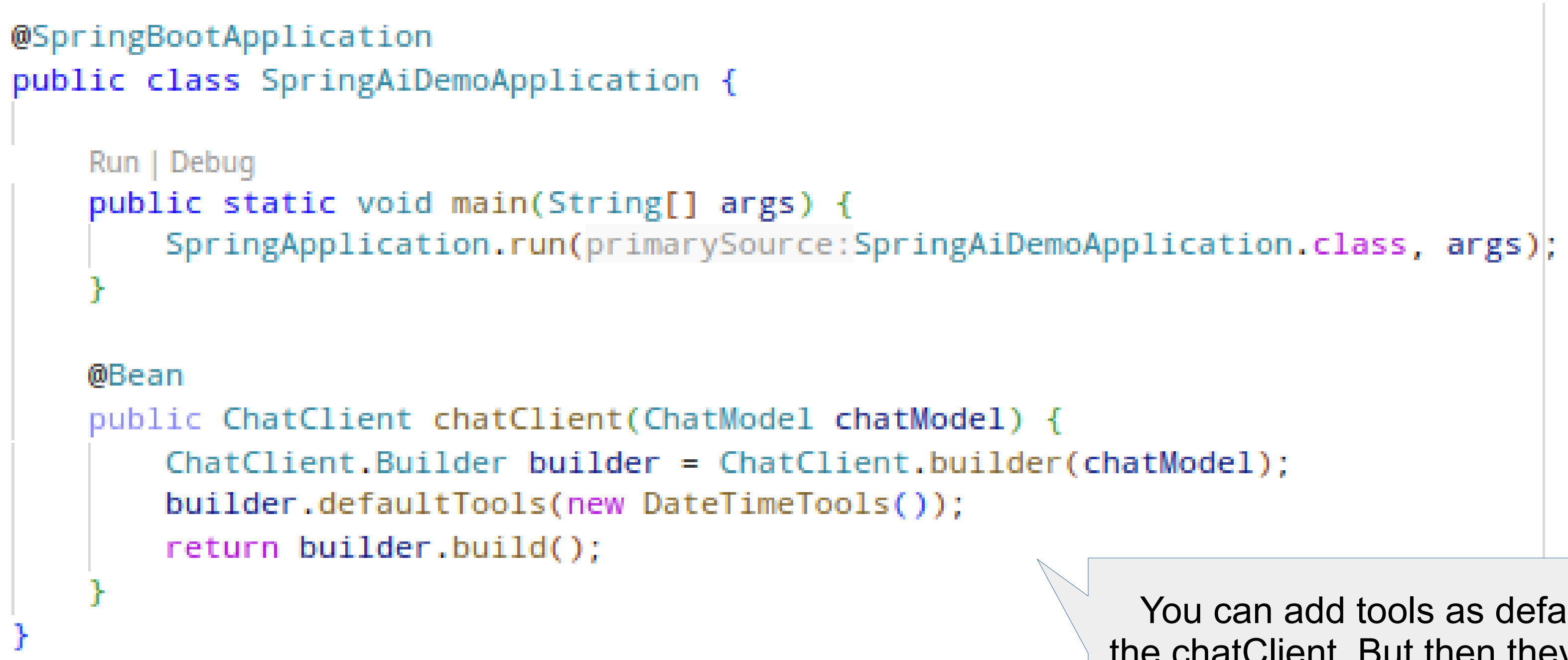
Calculator Tool



```java
public class CalculatorTool {
    private Logger logger = LoggerFactory.getLogger(clazz:CalculatorTool.class);

    @Tool(description = "Adds two numbers together")
    public long add(
        @ToolParam(description = "First number") long a,
        @ToolParam(description = "Second number") long b) {
            logger.info("adding " + a + " " + b);
            return a + b;
    }

    @Tool(description = "Subtracts number b from a")
    public long subtract(
        @ToolParam(description = "Number a") long a,
        @ToolParam(description = "Number b") long b) {
            logger.info("subtracting " + a + " " + b);
            return a - b;
    }
}
```

Parameter description are similarly important

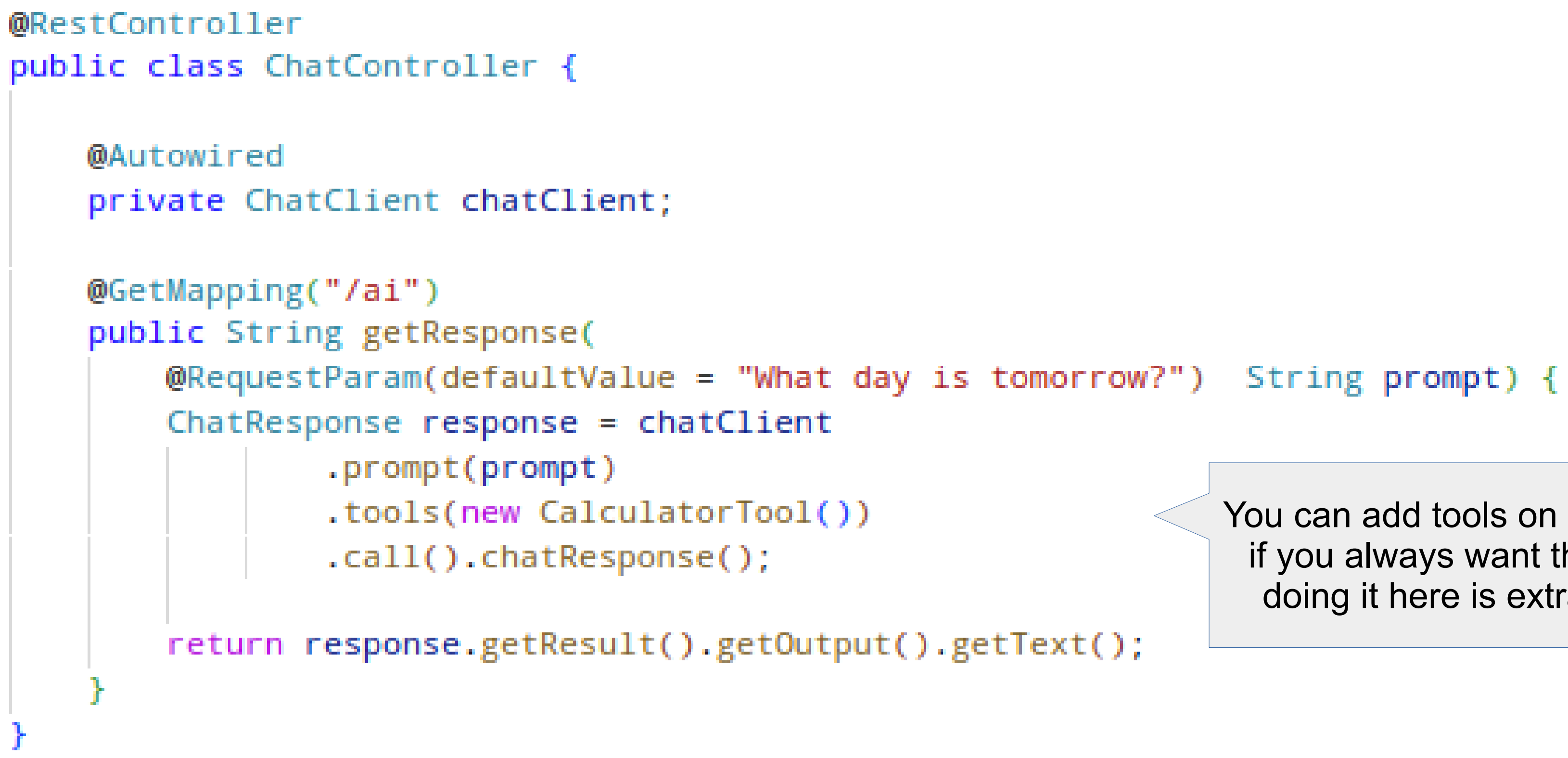
SpringBootApplication



```
@SpringBootApplication
public class SpringAiDemoApplication {

    Run | Debug
    public static void main(String[] args) {
        SpringApplication.run(primarySource:SpringAiDemoApplication.class, args);
    }

    @Bean
    public ChatClient chatClient(ChatModel chatModel) {
        ChatClient.Builder builder = ChatClient.builder(chatModel);
        builder.defaultTools(new DateTimeTools());
        return builder.build();
    }
}
```

You can add tools as default to the chatClient. But then they might also be available in situations where you didn't want them

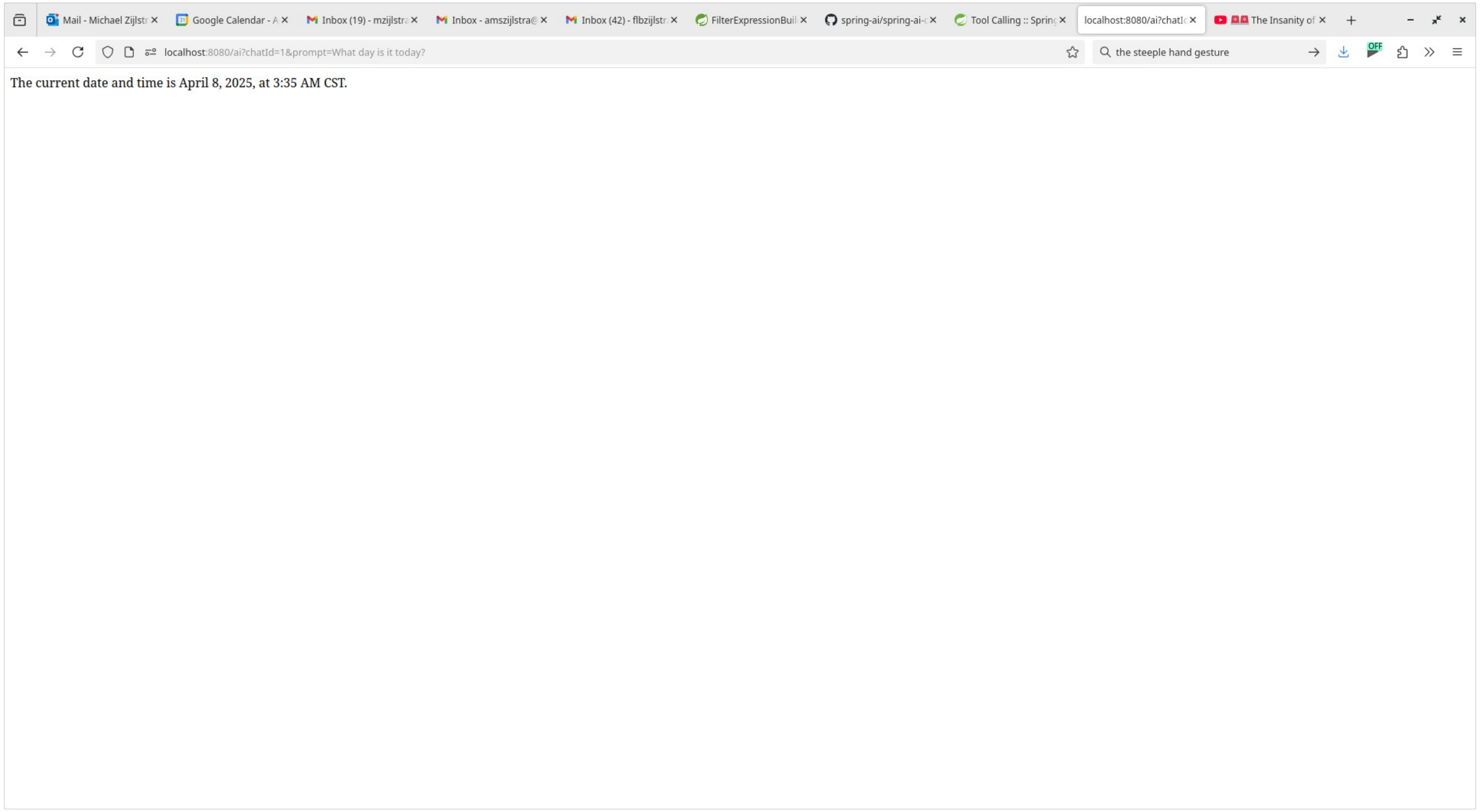
Controller



```
@RestController
public class ChatController {

    @Autowired
    private ChatClient chatClient;

    @GetMapping("/ai")
    public String getResponse(
        @RequestParam(defaultValue = "What day is tomorrow?")  String prompt) {
        ChatResponse response = chatClient
                .prompt(prompt)
                .tools(new CalculatorTool())
                .call().chatResponse();

        return response.getResult().getOutput().getText();
    }
}
```

You can add tools on the response, if you always want them present doing it here is extra overhead

Demo



The current date and time is April 8, 2025, at 3:35 AM CST.

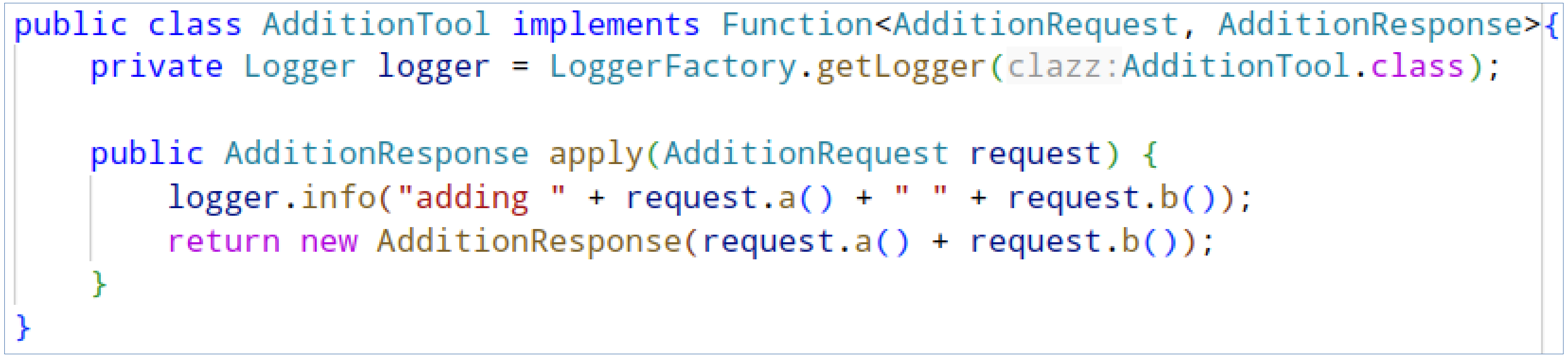
Method Tool Limitations

- The following types are not currently supported as parameters or return types for methods used as tools:
  - `Optional`
  - Asynchronous types (e.g. `CompletableFuture`, `Future`)
  - Reactive types (e.g. `Flow`, `Mono`, `Flux`)
  - Functional types (e.g. `Function`, `Supplier`, `Consumer`).

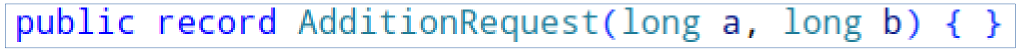
- Functional types are supported using the function-based tool specification approach.

Programmatic Functions

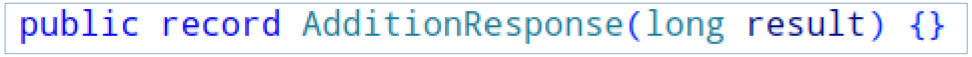
AdditionTool



```java
public class AdditionTool implements Function<AdditionRequest, AdditionResponse>{
    private Logger logger = LoggerFactory.getLogger(clazz:AdditionTool.class);

    public AdditionResponse apply(AdditionRequest request) {
        logger.info("adding " + request.a() + " " + request.b());
        return new AdditionResponse(request.a() + request.b());
    }
}
```



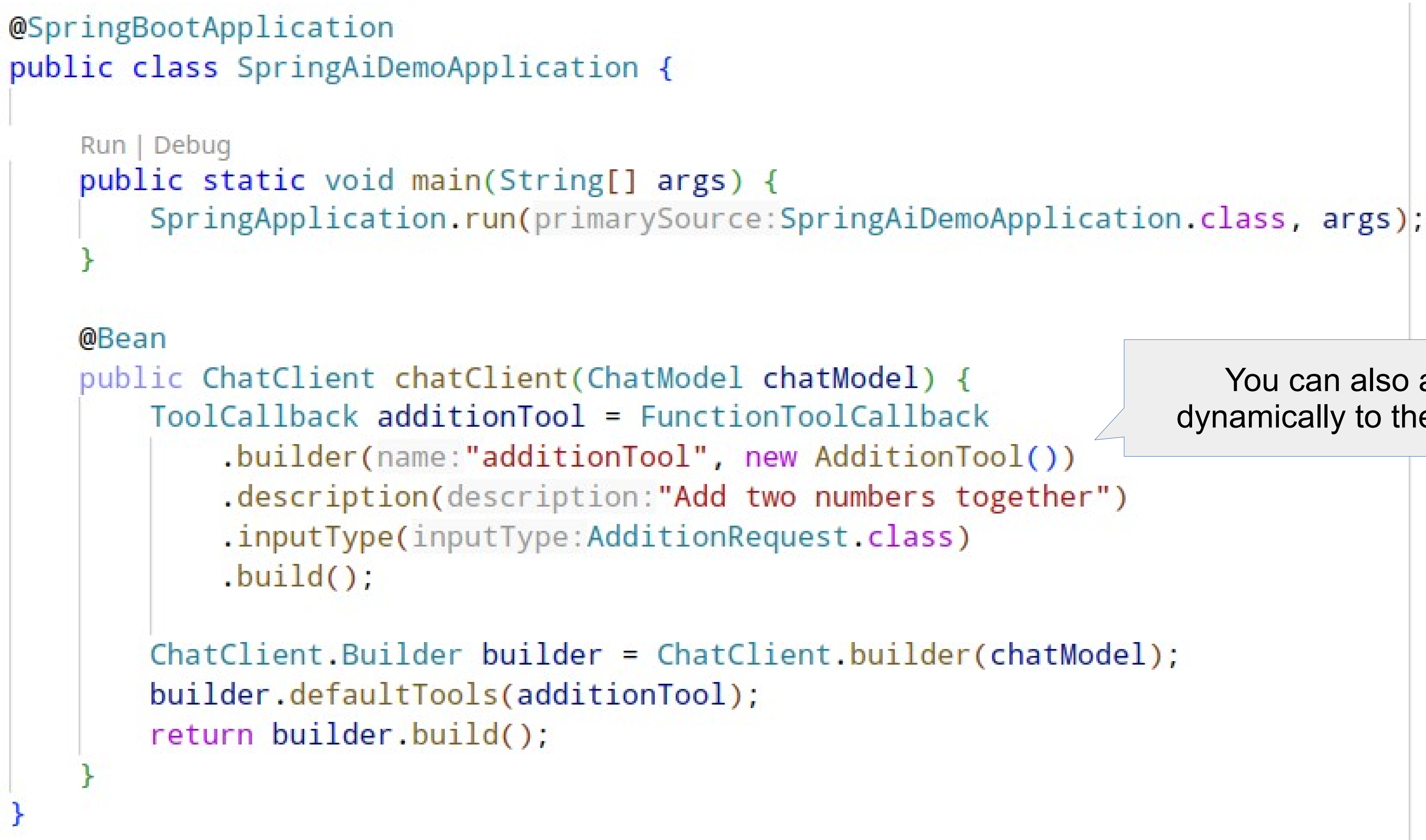
```java
public record AdditionRequest(long a, long b) { }
```



```java
public record AdditionResponse(long result) {}
```

Need 3 files for each function that you want the LLM to call

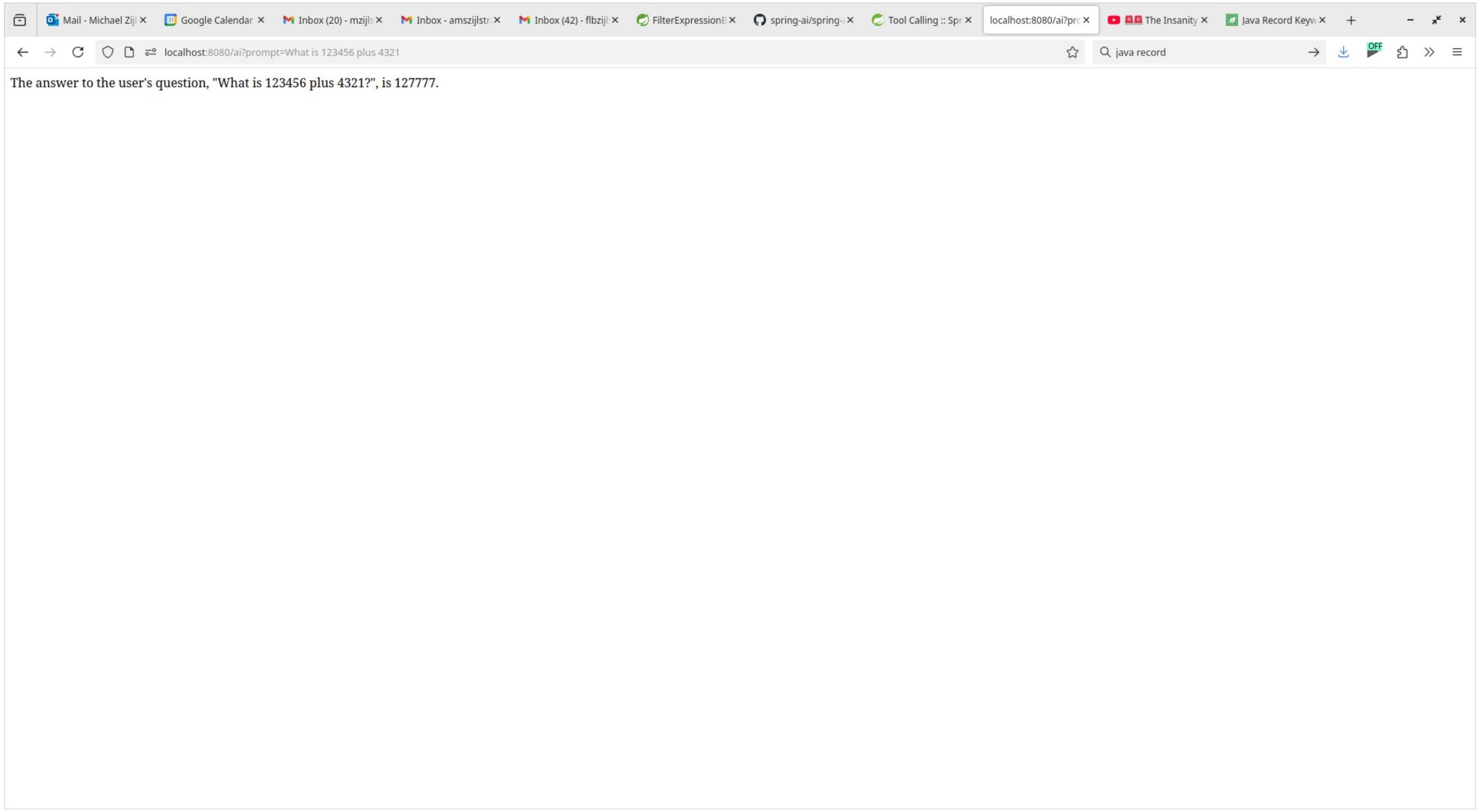
SpringBootApplication



```
@SpringBootApplication
public class SpringAiDemoApplication {

    Run | Debug
    public static void main(String[] args) {
        SpringApplication.run(primarySource:SpringAiDemoApplication.class, args);
    }

    @Bean
    public ChatClient chatClient(ChatModel chatModel) {
        ToolCallback additionTool = FunctionToolCallback
            .builder(name:"additionTool", new AdditionTool())
            .description(description:"Add two numbers together")
            .inputType(inputType:AdditionRequest.class)
            .build();

        ChatClient.Builder builder = ChatClient.builder(chatModel);
        builder.defaultTools(additionTool);
        return builder.build();
    }
}
```

You can also add it dynamically to the prompt

Demo



```
The answer to the user's question, "What is 123456 plus 4321?", is 127777.
```

Function Tool Limitations

- The following types are not currently supported as input or output types for functions used as tools:
  - Primitive types
  - `Optional`
  - Collection types (`List`, `Map`, `Array`, `Set`)
  - Asynchronous types (`CompletableFuture`, `Future`)
  - Reactive types (`Flow`, `Mono`, `Flux`)

- Primitive types and collections are supported using the method-based tool specification approach.

Tool Inputs

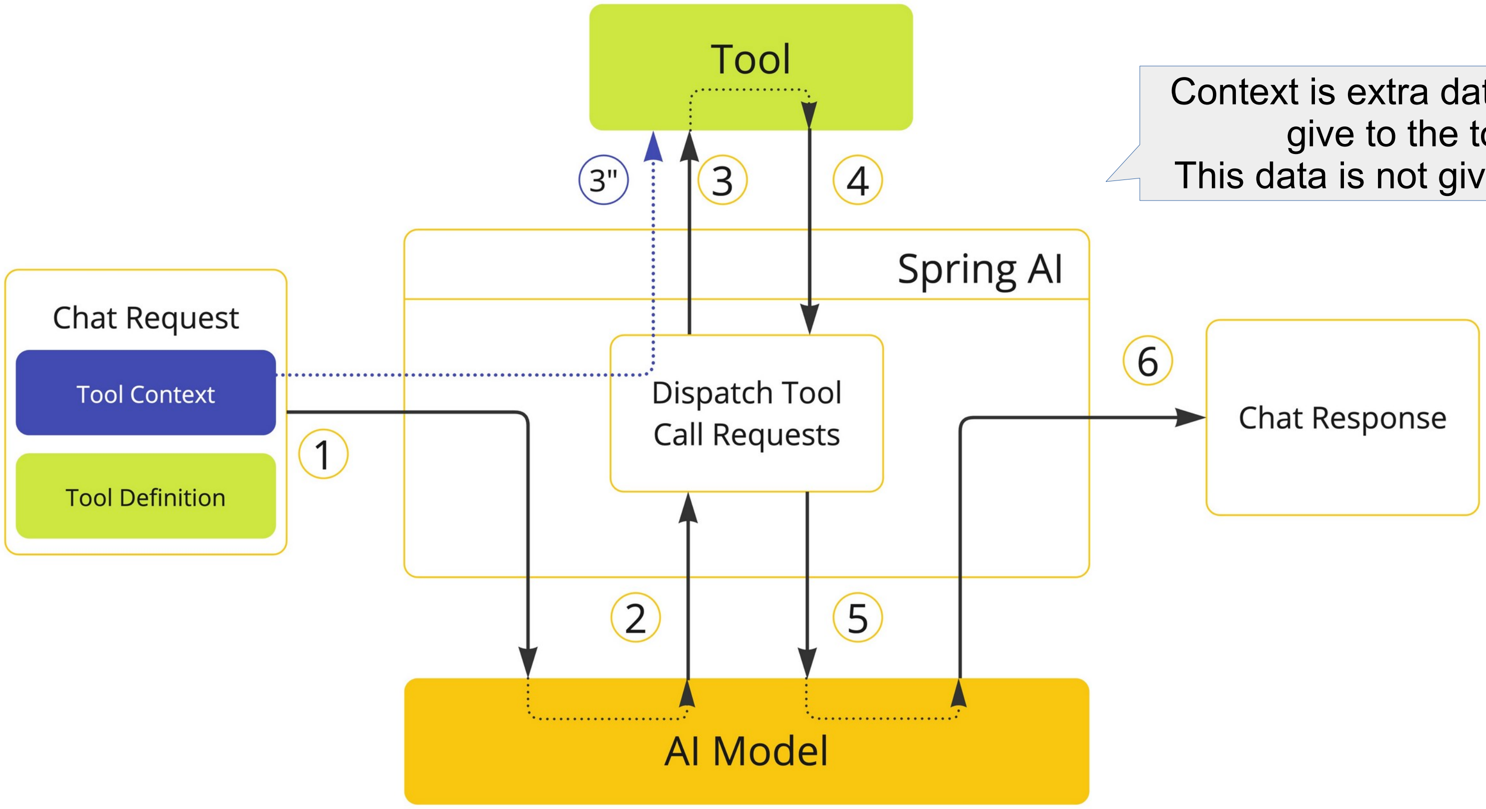
Required / Optional

- By default a parameter is considered required
- You can make a parameter optional with:
  - `@ToolParam(required=false)` from Spring AI
  - `@JsonProperty(required=false)` from Jackson
  - `@Schema(required=false)` from Swagger
  - `@Nullable` from Spring Framework

Description

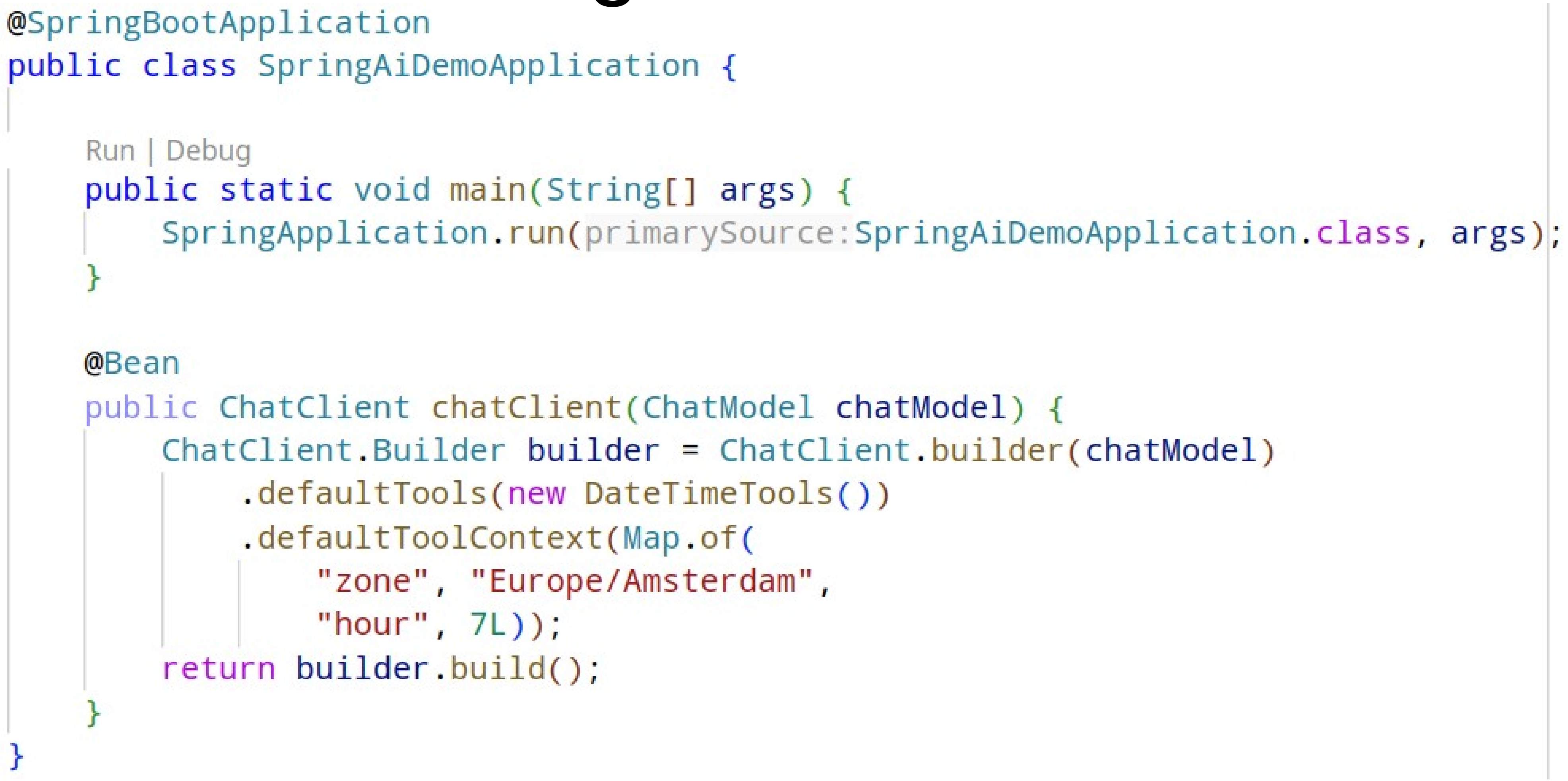
- Parameter desciption can be provided with:
  - `@ToolParam(description = "…")`
    from Spring AI
  - `@JsonClassDescription(description = "…")`
    from Jackson
  - `@JsonPropertyDescription(description = "…")`
    from Jackson
  - `@Schema(description = "…")`
    from Swagger.

Tool Context



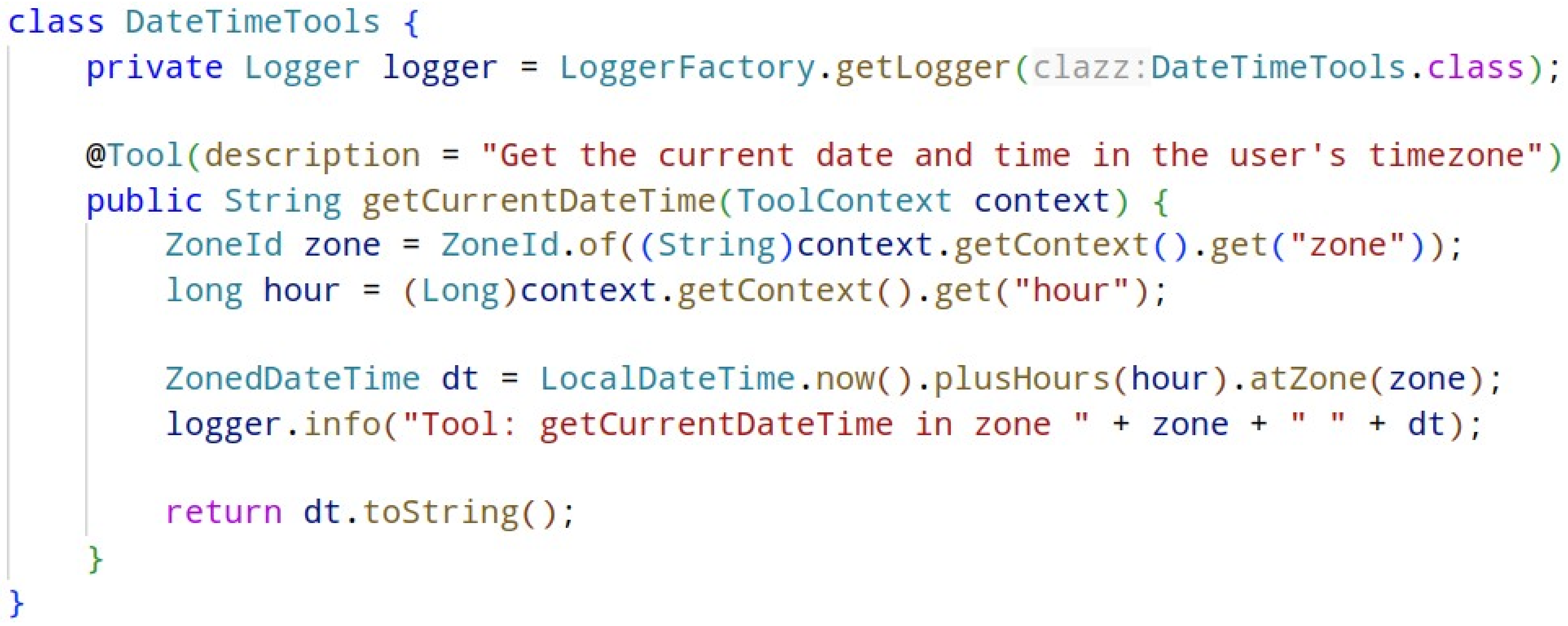
Context is extra data that you give to the tool.
This data is not given to LLM

Setting Tool Context



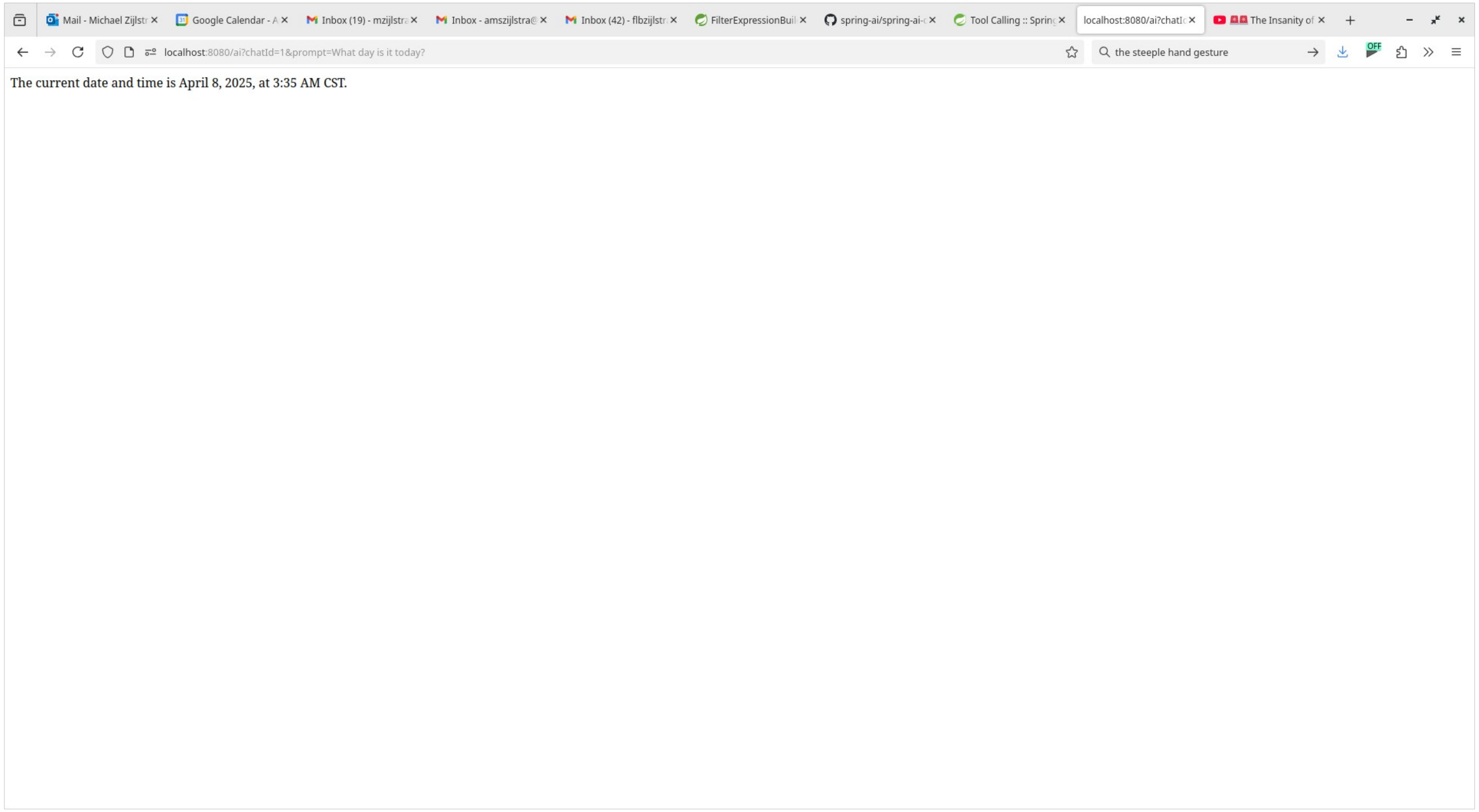
```
@SpringBootApplication
public class SpringAiDemoApplication {

    Run | Debug
    public static void main(String[] args) {
        SpringApplication.run(primarySource:SpringAiDemoApplication.class, args);
    }

    @Bean
    public ChatClient chatClient(ChatModel chatModel) {
        ChatClient.Builder builder = ChatClient.builder(chatModel)
            .defaultTools(new DateTimeTools())
            .defaultToolContext(Map.of(
                "zone", "Europe/Amsterdam",
                "hour", 7L));
        return builder.build();
    }
}
```

Using Tool Context



```java
class DateTimeTools {
    private Logger logger = LoggerFactory.getLogger(clazz:DateTimeTools.class);

    @Tool(description = "Get the current date and time in the user's timezone")
    public String getCurrentDateTime(ToolContext context) {
        ZoneId zone = ZoneId.of((String)context.getContext().get("zone"));
        long hour = (Long)context.getContext().get("hour");

        ZonedDateTime dt = LocalDateTime.now().plusHours(hour).atZone(zone);
        logger.info("Tool: getCurrentDateTime in zone " + zone + " " + dt);

        return dt.toString();
    }
}
```

Demo

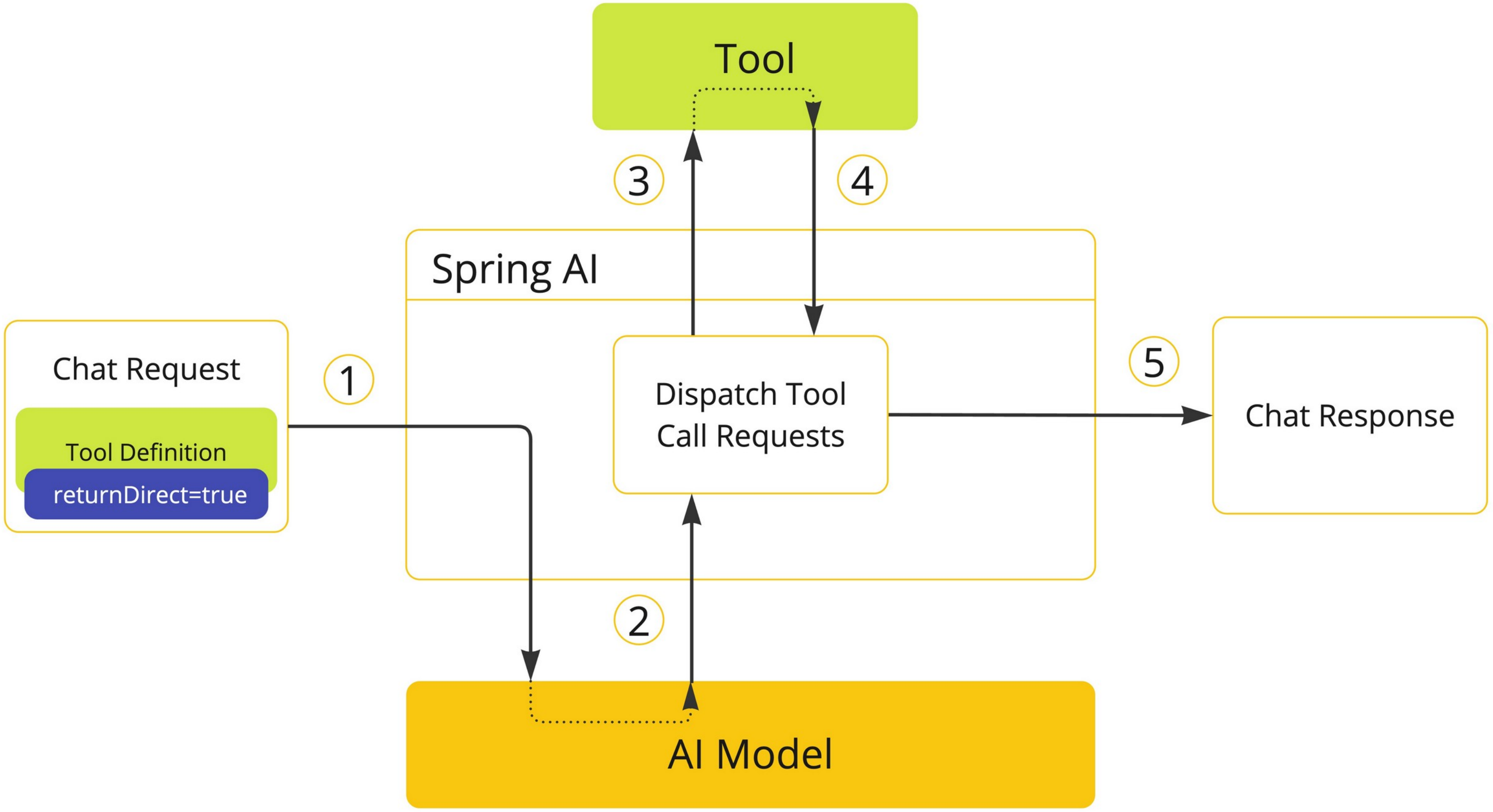


```
localhost:8080/ai?chatId=1&prompt=What day is it today?
```

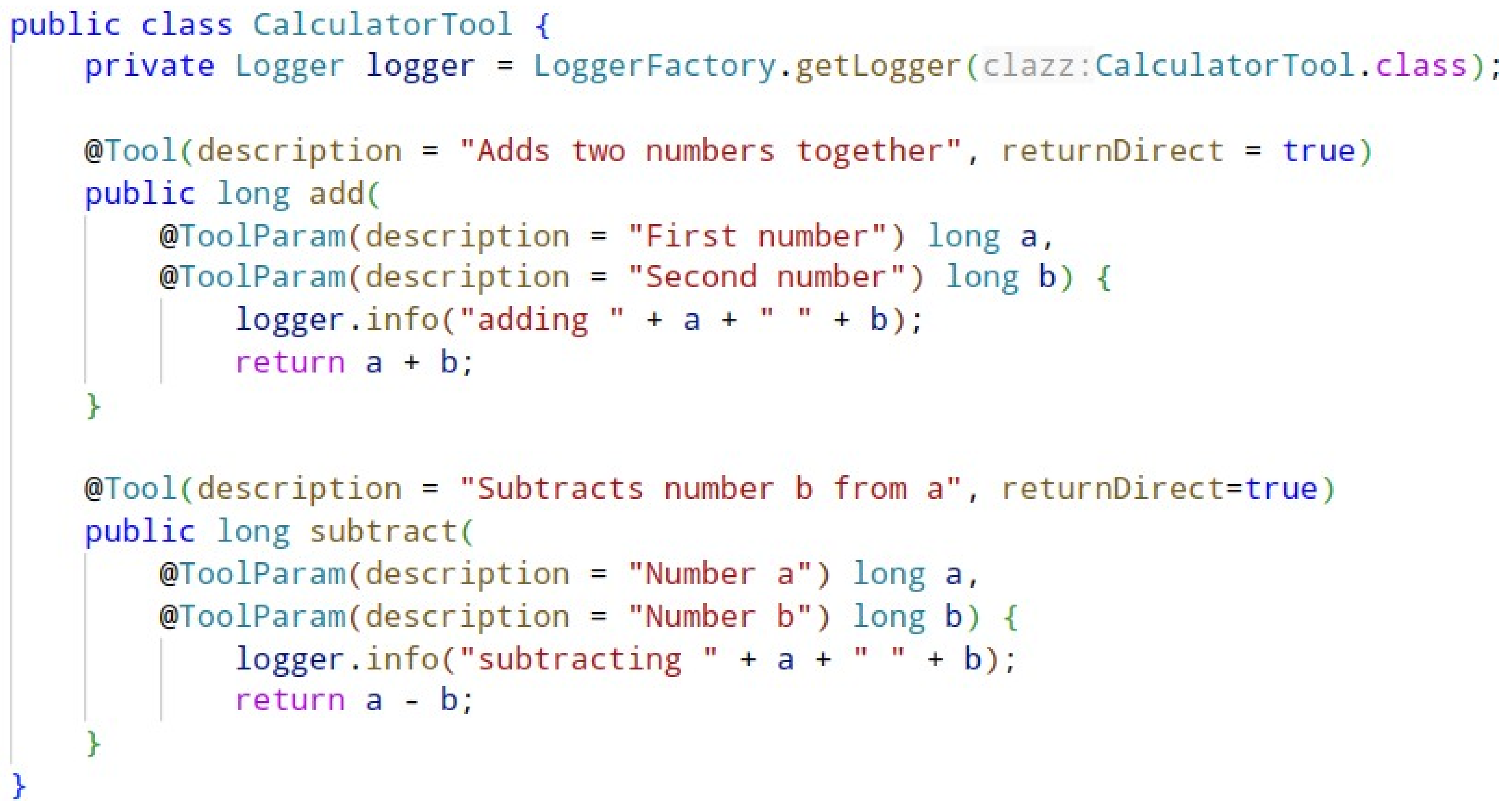
```
The current date and time is April 8, 2025, at 3:35 AM CST.
```

Tool Ouput

Return Direct

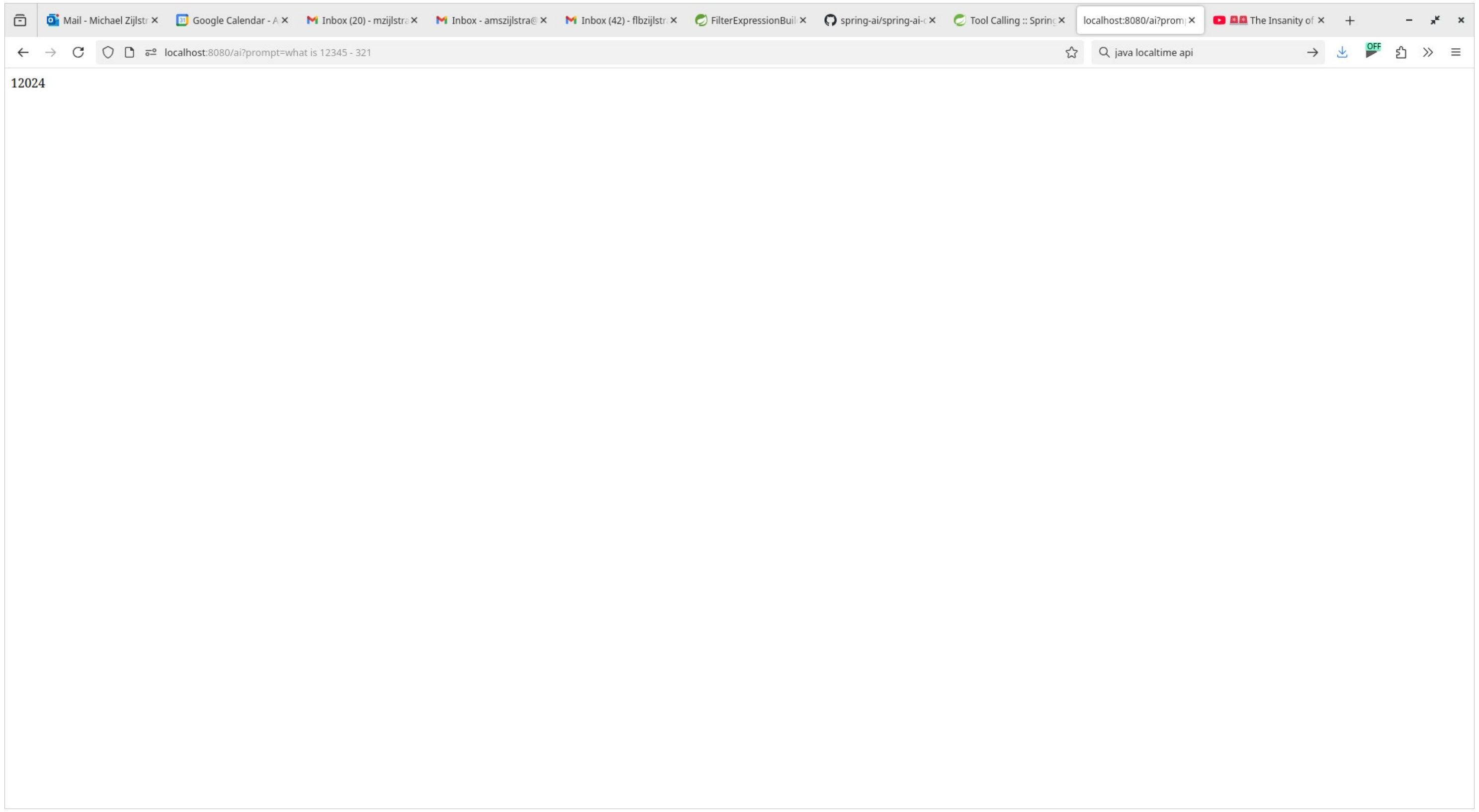


Calculator Return Direct



```java
public class CalculatorTool {
    private Logger logger = LoggerFactory.getLogger(clazz:CalculatorTool.class);

    @Tool(description = "Adds two numbers together", returnDirect = true)
    public long add(
        @ToolParam(description = "First number") long a,
        @ToolParam(description = "Second number") long b) {
            logger.info("adding " + a + " " + b);
            return a + b;
    }

    @Tool(description = "Subtracts number b from a", returnDirect=true)
    public long subtract(
        @ToolParam(description = "Number a") long a,
        @ToolParam(description = "Number b") long b) {
            logger.info("subtracting " + a + " " + b);
            return a - b;
    }
}
```

Demo



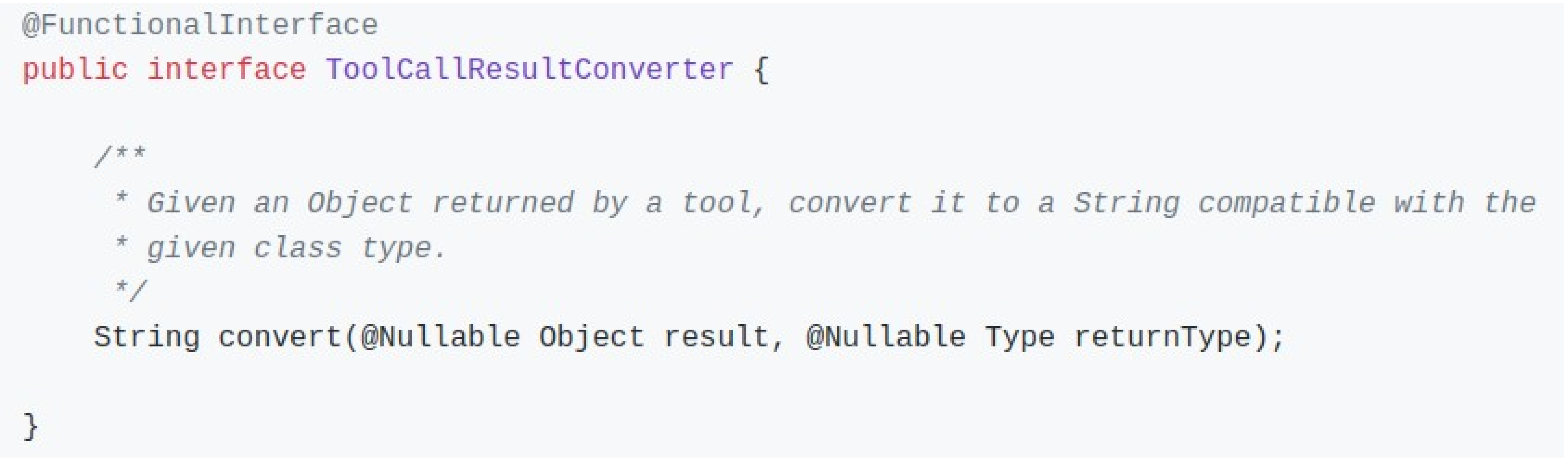
localhost:8080/ai?prompt=what is 12345 - 321

```
12024
```

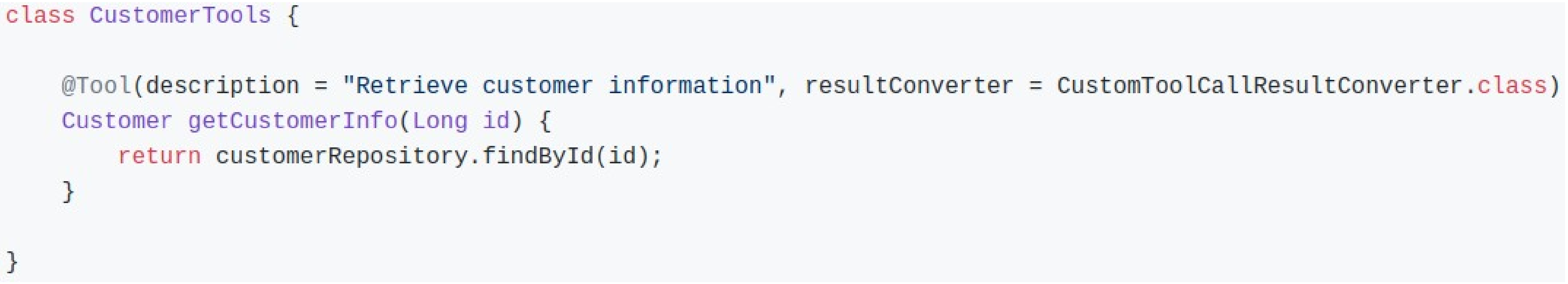
Result Conversion

- By default, the result is serialized to JSON using Jackson (DefaultToolCallResultConverter), but you can customize the serialization process by providing your own ToolCallResultConverter implementation.

From Spring AI Docs



```java
@FunctionalInterface
public interface ToolCallResultConverter {

    /**
     * Given an Object returned by a tool, convert it to a String compatible with the
     * given class type.
     */
    String convert(@Nullable Object result, @Nullable Type returnType);

}
```



```java
class CustomerTools {

    @Tool(description = "Retrieve customer information", resultConverter = CustomToolCallResultConverter.class)
    Customer getCustomerInfo(Long id) {
        return customerRepository.findById(id);
    }

}
```

Summary

- Declarative Methods
- Programmatic Functions
- Tool Inputs
- Tool Ouput

Closing Thoughts

- Tools can add pratical capablities to LLMs
- Putting this together really made me experience the limits of self hosting on old hardware
  - Llama3.2 wouldn’t call multiple tools for a prompt
  - Many other small models don’t support tool calling
  - I’m interested in testing this with OpenAI

MAHARISHI INTERNATIONAL UNIVERSITY

HIGHER EDUCATION • FOR HIGHER CONSCIOUSNESS