Spring Boot Auto Configuration

What is auto-configuration

- Auto-configuration is a key feature that makes it easy to get started with Spring-based applications

- It automatically configures a set of default beans and settings based on the dependencies and configuration available on the classpath

- It works by scanning the classpath for relevant Spring components and auto-configure them based on a set of predefined rules

Auto-configuration example

- If the application includes the spring-boot-starter-web dependency, it will automatically configure a set of beans and settings related to web applications, such as a Tomcat servlet container, a dispatcher servlet, and a set of default web-related configuration settings

Auto-configuration customization

- Auto-configuration is designed to work out of the box
- But it can also be customized and the default settings can be overrided
- Customization can be done by providing custom configuration files, overriding default beans, or excluding certain auto-configuration classes from the application context

Conditional Auto-configuration

- Spring also provides a set of conditions that can be used to selectively apply auto-configuration classes based on the presence or absence of certain configuration settings

- For example, an auto-configuration class might only be applied if a certain property is set in the application.properties file, or if a certain bean is not already present in the application context

Understanding @EnableXxx annotations

- @EnableXxx annotations are a set of meta-annotations
- They allow developers to easily enable certain features or behaviors
- They are typically used to enable features from third-party libraries
- For example @EnableCaching enable caching support and @EnableAsync enable asynchronous processing support

@EnableXxx annotation examples

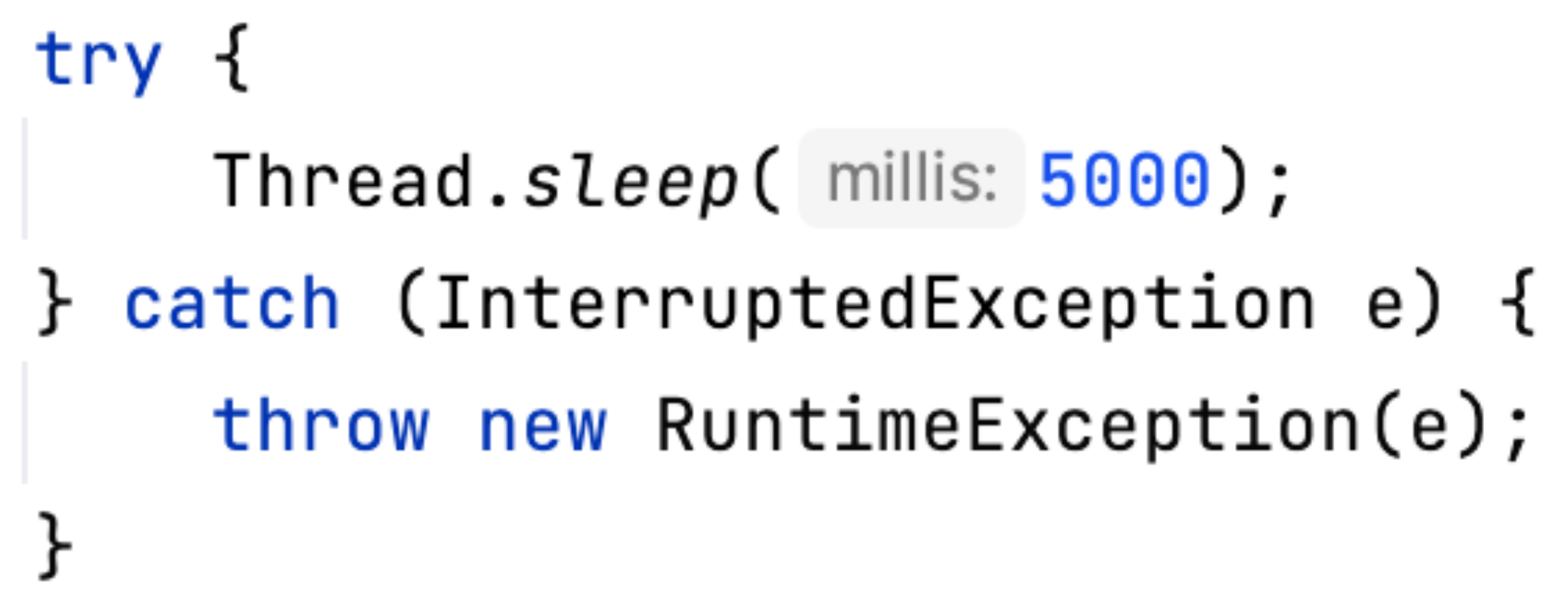
- The @EnableScheduling annotation is used to enable scheduling
- @EnableScheduling is typically used with the @Scheduled annotation for specifying the timing and frequency of scheduled tasks
- The @EnableCaching annotation is used to enable caching support
- @EnableCaching is typically used in combination with caching annotations such as @Cachable, @CachePut and @CacheEvict

Demo 1 - @EnableXxx annotations

- @EnableScheduling annotation used to schedule running a method

Exercise 1.1 - @EnableCaching

- Reuse the solution from Managing Beans And Dependency Injection
- In the sum method of the Calculator class, add code like this example before returning from the method to make the method less efficient:
- Then run the project and call a Controller method and it should take at least 5 seconds to reply



```
try {
    Thread.sleep( millis: 5000);
} catch (InterruptedException e) {
    throw new RuntimeException(e);
}
```

Exercise 1.2 - @EnableCaching

- Now speed up the functionality by enabling caching to the sum method
- Add the @EnableCaching annotation to the Calculator class
- Add the @Cacheable(”myCache”) annotation to the sum method
- Now try to run the application and you should notice that the first time you calculate two numbers it takes 5 seconds, but the second time you calculate the same numbers it will now go very fast since the return value of these two numbers is now automatically cached