Spring uses several design patterns

1. Dependency injection pattern

Spring manages the dependencies between objects, allowing for loose coupling

We have constructor injection as the most preferred way

1. Factory pattern (creational)

The getBean method in spring will return an object from the context when we provide it with a classname and / or a bean name

1. Template method (behavioural)

RestTemplate uses this design pattern. There are a series of fixed steps in an algorithm that the flow has to go thru, and the user can make small customizations on his end such as providing the url, the return type expected etc.

1. Singleton pattern (creational)

When we create a bean using the @Component annotation, spring will create only one bean of this and place it in the context by default, and this single bean wil be provided whenever a bean of its class is asked for.

1. Façade pattern (structural)

Spring framework quite heavily uses the façade design pattern. The façade design pattern is responsible for providing an API that is easy to use, that abstracts away the complicated steps and details that are performed under the hood. RestTemplate is an example of this.

1. Proxy pattern (structural)

Spring AOP uses a proxy pattern that is essentially using a placeholder (proxy) object to intercept method calls and apply cross cutting concerns such as logging