@ModelAttribute (slightly longer explanation)

Product class (Model)

```
public class Product {

    private String fieldValue_One = "prewritten info";

    public String getFieldValue_One() {
        return fieldValue_One;
    }

    public void setFieldValue_One(String fieldValue_One) {
        this.fieldValue_One = fieldValue_One;
    }
}
```

Controller

```
@Controller
@RequestMapping("/first")
public class TestController {

    @GetMapping("/see")
    public String showIndex(Product product)
    {
        return "firstPage";
    }


    @PostMapping("/show")
    public String saveProduct(Product product)
    {
        return "secondPage";
    }

}
```

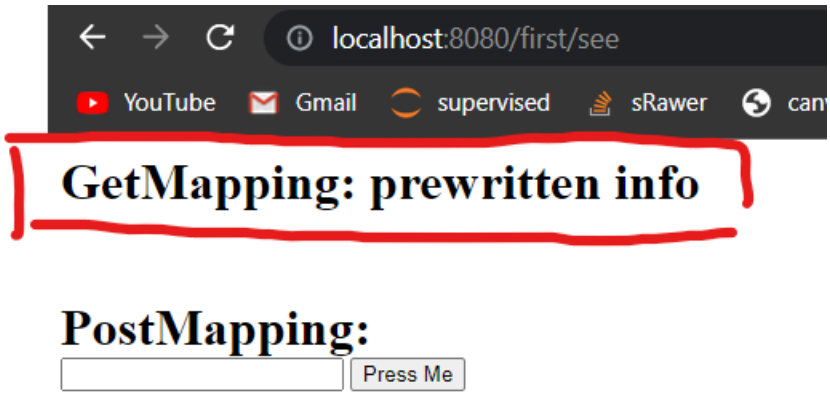
Referencing the first method with the @GetMapping, this method translates into →

1. Client sends request that maps to the end point "/see"
2. Create **Product** object and inject into the argument of showIndex
3. Create a **ModelAttribute** with the name "product", store this object in the Model Map field (a storage area for model attributes)
4. Return a string called "firstPage", use string to search for corresponding html file, send html as a response to the client

Html Page returned to browser

```
<!DOCTYPE html>
    <html>
        <head>
            <meta name="viewport" content="width=device-width, intial-scale=1"/>
            <link rel="stylesheet" href="/style.css"/>
        </head>
        <body>


    <h1>GetMapping: </h1><h1 th:text="${product.getFieldValue_One}"> </h1>

    </br></br></br></br>
    <h1>PostMapping:</h1>
    <form th:action="@{./show}" method="post" th:object="${product}" >
        <div >
        <input type="text" th:field="*{fieldValue_One}"/><span></span>
        <button type="submit">Press Me</button>
        </div>
    </form>



        </body>
    </html>
```

Browser parses html page and the result



Question is do you need to specify @ModelAttribute?

Answer is yes, because it makes it easier for other programmers to read your code even though it's just syntactic sugar.

> **@ModelAttribute**
>
> For access to an existing attribute in the model (instantiated if not present) with data binding and validation applied. See @ModelAttribute as well as Model and DataBinder.
>
> **Note that use of @ModelAttribute is optional** (for example, to set its attributes). See "Any other argument" at the end of this table.

.

https://stackoverflow.com/questions/58330323/when-we-must-use-modelattribute-and-how-it-works#:~:text=Note%20that%20use%20of%20%40ModelAttribute,%2C%20to%20set%20its%20attributes).

But actually . . . there's more to it:

```
@Controller
@RequestMapping("/first")
public class TestController {

    @GetMapping("/see")
    public String showIndex(String s, Product product)
    {
        return "firstPage";
    }


    @PostMapping("/show")
    public String saveProduct(Product product)
    {
        return "secondPage";
    }

}
```

Take a look at this code, I have introduced a string object.

Now that is an implicit way of writing this

```
@GetMapping("/see")
public String showIndex(@RequestParam(required=false)String s, Product product)
{
    return "firstPage";
}
```

So, if I type in the url this line

```
localhost:8080/first/see?s=bert was here
```

And also write an extra line of code inside the showIndex method (so I can see the value in the console)

```
@GetMapping("/see")
public String showIndex(String s, Product product)
{
    System.out.println(s);
    return "firstPage";
}
```

Console

```
[7]        : Initializing Spring DispatcherServlet 'dispatcherServlet'
2022-12-17 13:04:27.356  INFO 3568 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServl
et         : Initializing Servlet 'dispatcherServlet'
2022-12-17 13:04:27.358  INFO 3568 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServl
et         : Completed initialization in 2 ms
bert was here
```

But wait a minute, didn't we just talked about ModelAttribute annotation being optional and how Spring Boot would automatically turn those parameters without any annotation into a ModelAttribute object and store it in a Model Map field?

In the case of a String object, where it's ambiguous, Spring Boot will consider this as an implicit `@RequestParam(required=false)` situation. So if we want the string object to be a modelattribute, then we would need to specify in the parameter using @ModelAttribute annotation.

To prove this, let's write a line of code in the html page.

```
<!DOCTYPE html>
    <html>
        <head>
            <meta name="viewport" content="width=device-width, intial-scale=1"/>
            <link rel="stylesheet" href="/style.css"/>
        </head>
        <body>


    <h1>GetMapping: </h1><h1 th:text="${product.getFieldValue_One}"> </h1>

    </br></br></br></br>
    <h1>PostMapping:</h1>
    <form th:action="@{./show}" method="post" th:object="${product}" >
        <div >
        <input type="text" th:field="*{fieldValue_One}"/><span></span>
        <button type="submit">Press Me</button>
        </div>
    </form>

    </br></br></br></br>
    <h1>String object: </h1><h1 th:text="${string.hashCode}"></h1>

        </body>
    </html>
```

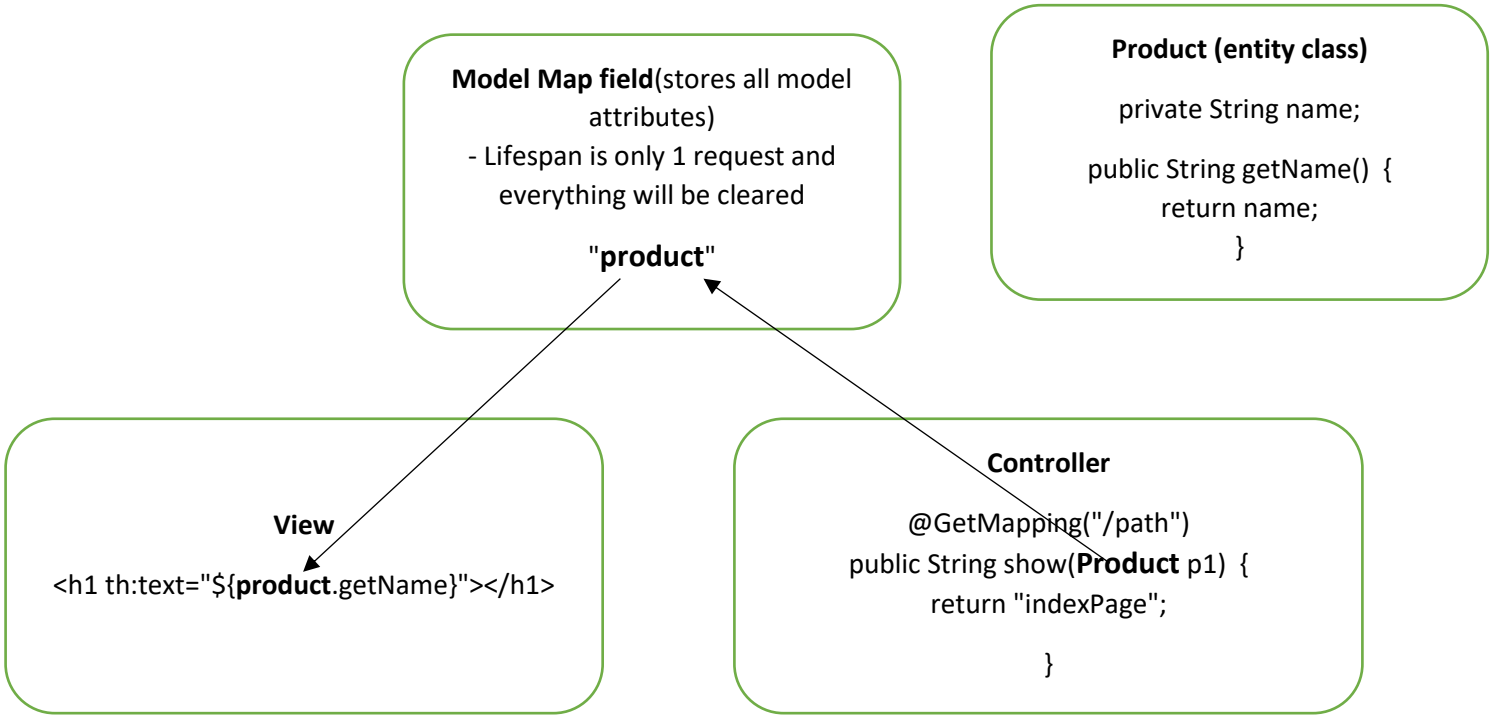
Now this line of code is **special**, it will throw an exception if this *reference* with the name "string" is null.

So from what we previously understood, even if @ModelAttribute annotation is not specified (in @GetMapping), the parameters of the method will still be turned into ModelAttribute objects.

```
@Controller
@RequestMapping("/first")
public class TestController {

    @GetMapping("/see")
    public String showIndex(Product product)
    {
        return "firstPage";
    }


    @PostMapping("/show")
    public String saveProduct(Product product)
    {
        return "secondPage";
    }

}
```

\

Visual representation



Test 1:

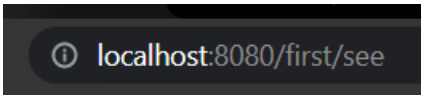
Controller

```
@GetMapping("/see")
public String showIndex(String s, Product product)
{
    return "firstPage";
}
```

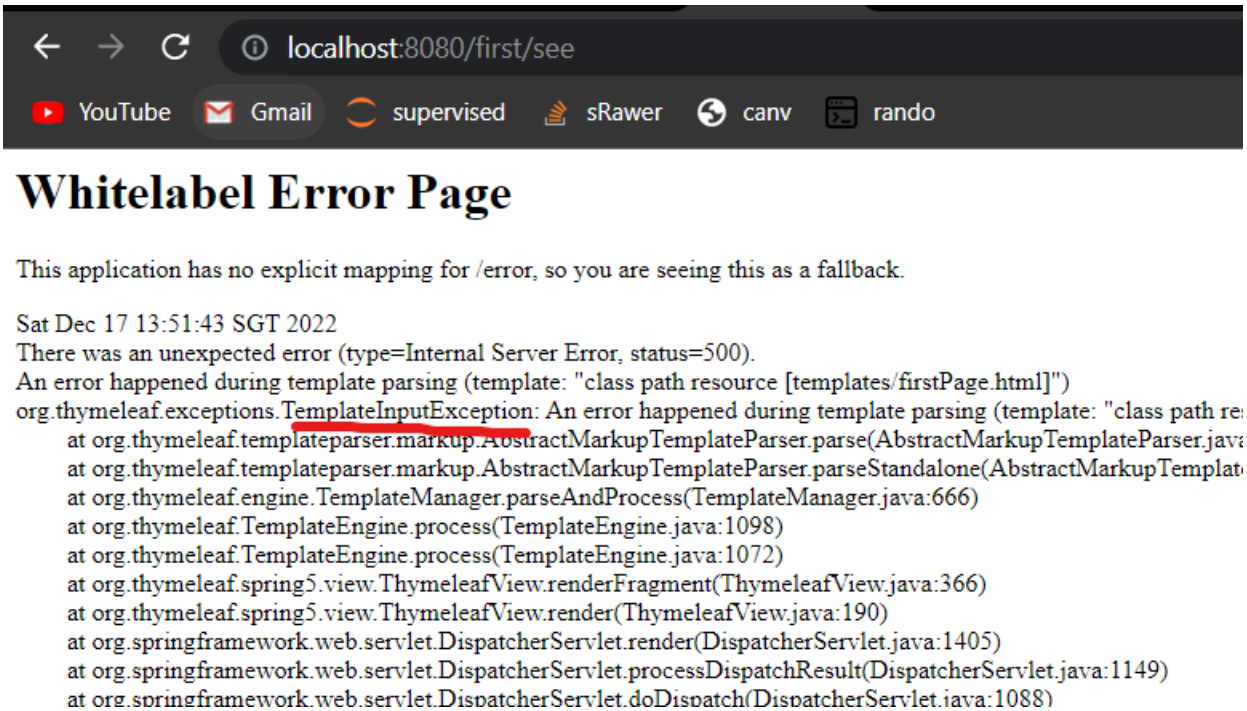
View

```
<h1>String object: </h1><h1 th:text="${string.hashCode}"></h1>
```

Browser URL input



Result



Spring Boot treats

```
/see")
 showIndex(String s, Product product)
```

as a @RequestParam parameter, so it's expecting an input through the URL, so it didn't create a ModelAttribute object, and that is the reason why we got this exception because in the HTML page we are invoking the method **hashCode**()

from the reference named "string"

```
th:text="${string.hashCode}"></h1>
```

.

And "string" is actually null, because no ModelAttribute named "string" was created.

Test 2:

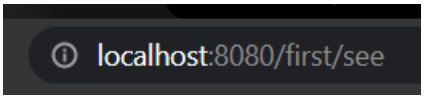
Controller

```
@GetMapping("/see")
public String showIndex(@ModelAttribute String s, Product product)
{
    return "firstPage";
}
```

View

```
<h1>String object: </h1><h1 th:text="${string.hashCode}"></h1>
```

Browser URL input



Result

So since a ModelAtrribute named "string" has been created, and we can safely use the **hashCode** function on the string object, hence there's no exception.

The 0 value is because an empty string has no value, hence the hashcode of it is 0.