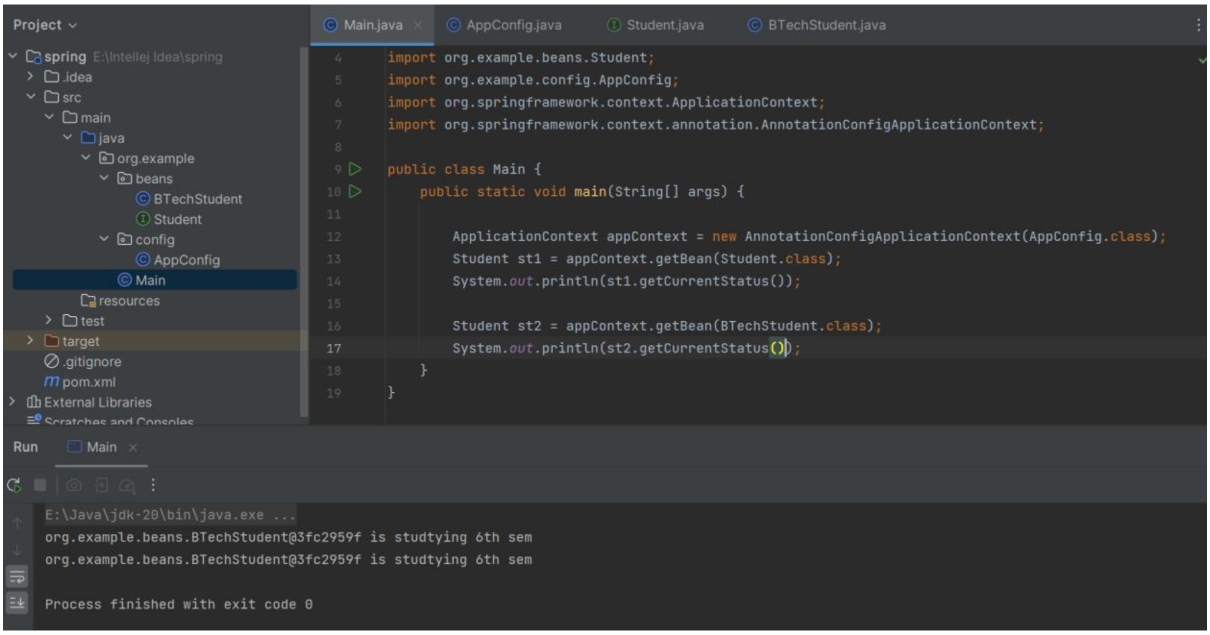
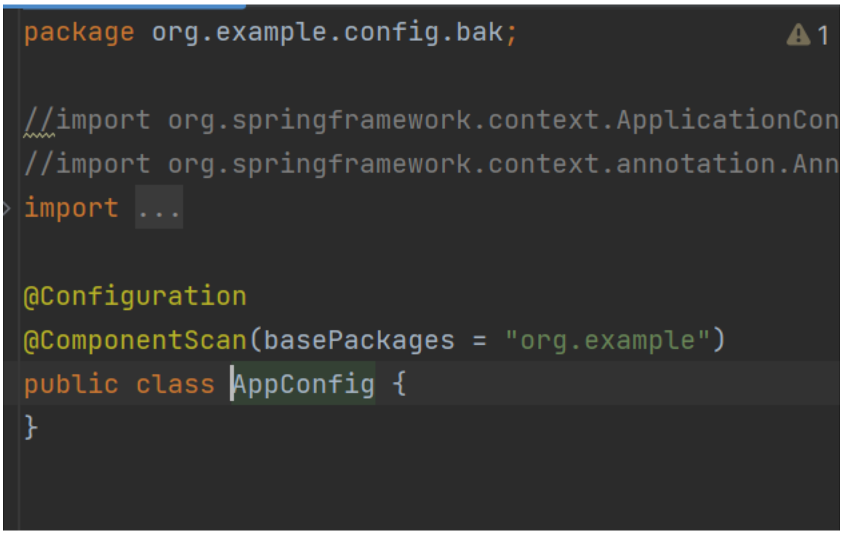
1. Application context in beans java class @Component is there



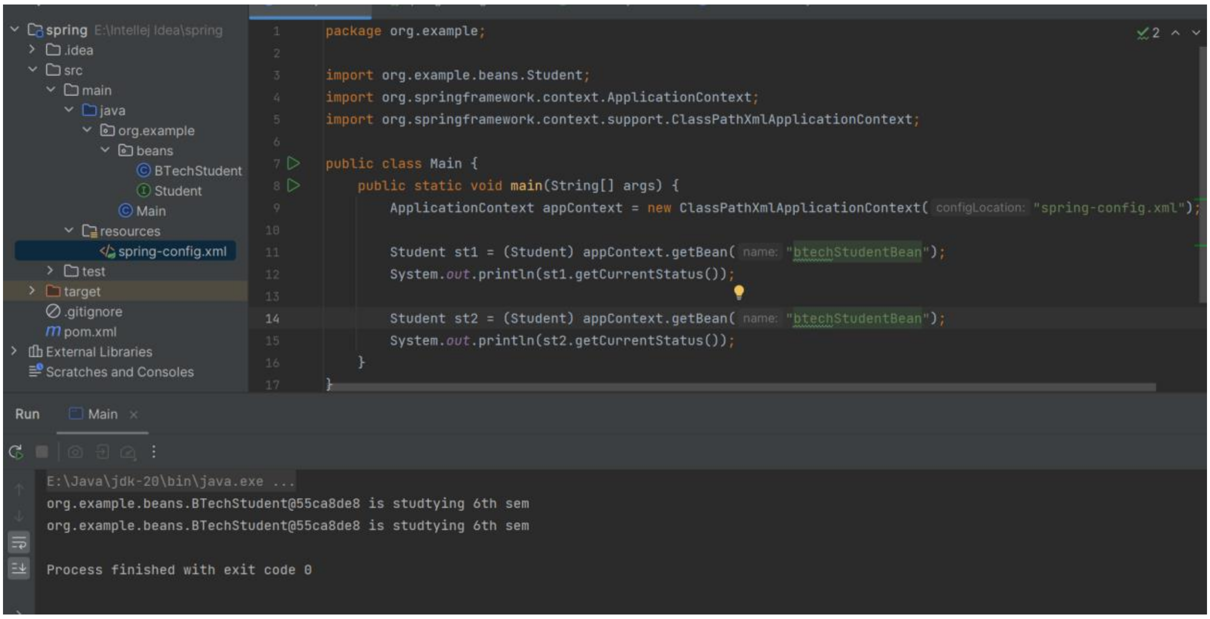
```java
import org.example.beans.Student;
import org.example.config.AppConfig;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class Main {
    public static void main(String[] args) {

        ApplicationContext appContext = new AnnotationConfigApplicationContext(AppConfig.class);
        Student st1 = appContext.getBean(Student.class);
        System.out.println(st1.getCurrentStatus());

        Student st2 = appContext.getBean(BTechStudent.class);
        System.out.println(st2.getCurrentStatus());
    }
}
```

```
E:\Java\jdk-20\bin\java.exe ...
org.example.beans.BTechStudent@3fc2959f is studtying 6th sem
org.example.beans.BTechStudent@3fc2959f is studtying 6th sem

Process finished with exit code 0
```



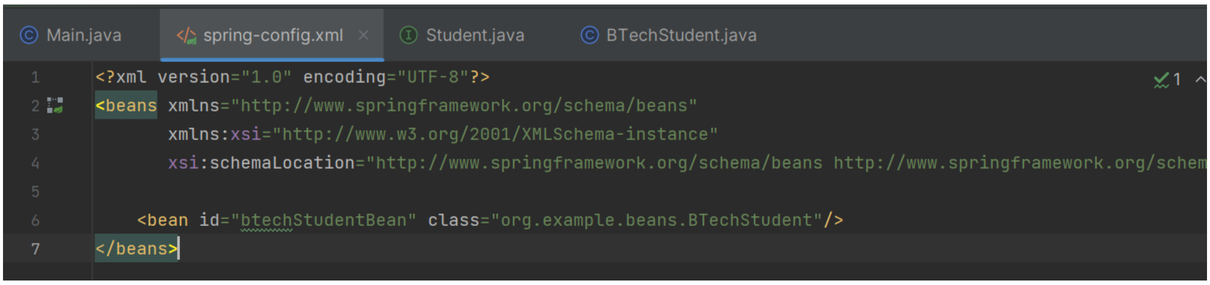
```java
package org.example.config.bak;

//import org.springframework.context.ApplicationCon
//import org.springframework.context.annotation.Ann
import ...

@Configuration
@ComponentScan(basePackages = "org.example")
public class AppConfig {
}
```

2. Traditional way xml: in bean Java class @Component is not there

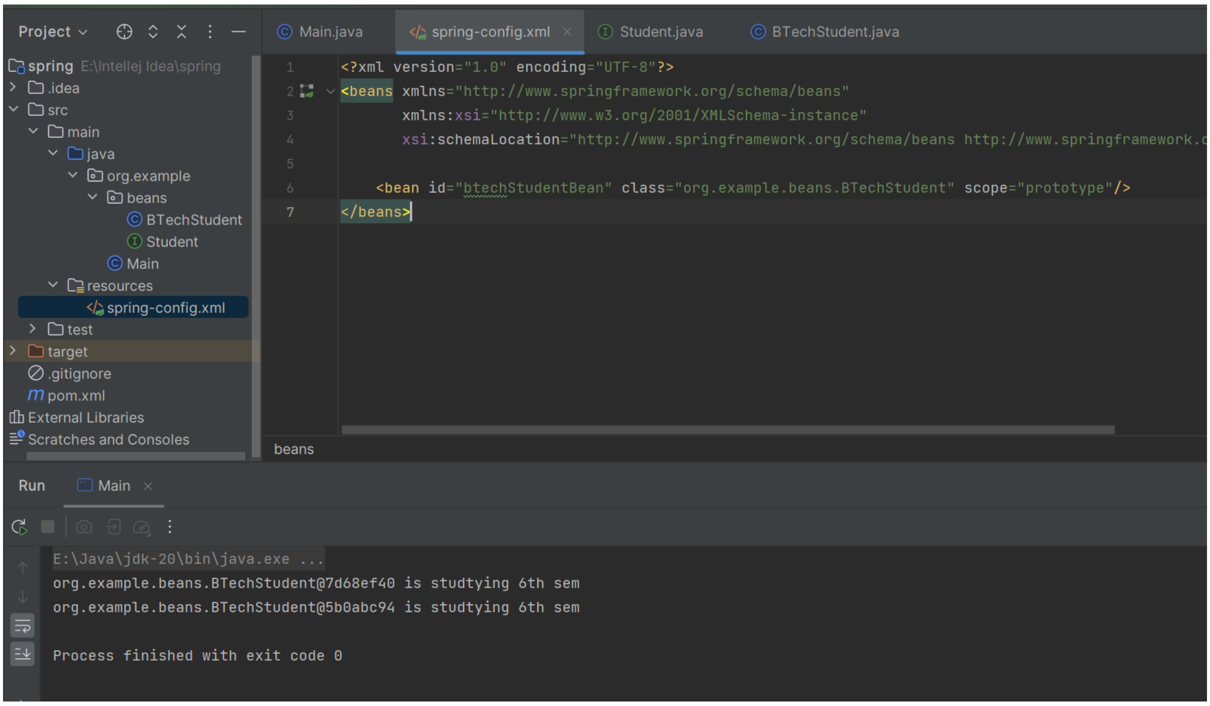


```
package org.example;

import org.example.beans.Student;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {
    public static void main(String[] args) {
        ApplicationContext appContext = new ClassPathXmlApplicationContext( configLocation: "spring-config.xml");

        Student st1 = (Student) appContext.getBean( name: "btechStudentBean");
        System.out.println(st1.getCurrentStatus());

        Student st2 = (Student) appContext.getBean( name: "btechStudentBean");
        System.out.println(st2.getCurrentStatus());
    }
}
```

```
E:\Java\jdk-20\bin\java.exe ...
org.example.beans.BTechStudent@55ca8de8 is studtying 6th sem
org.example.beans.BTechStudent@55ca8de8 is studtying 6th sem

Process finished with exit code 0
```

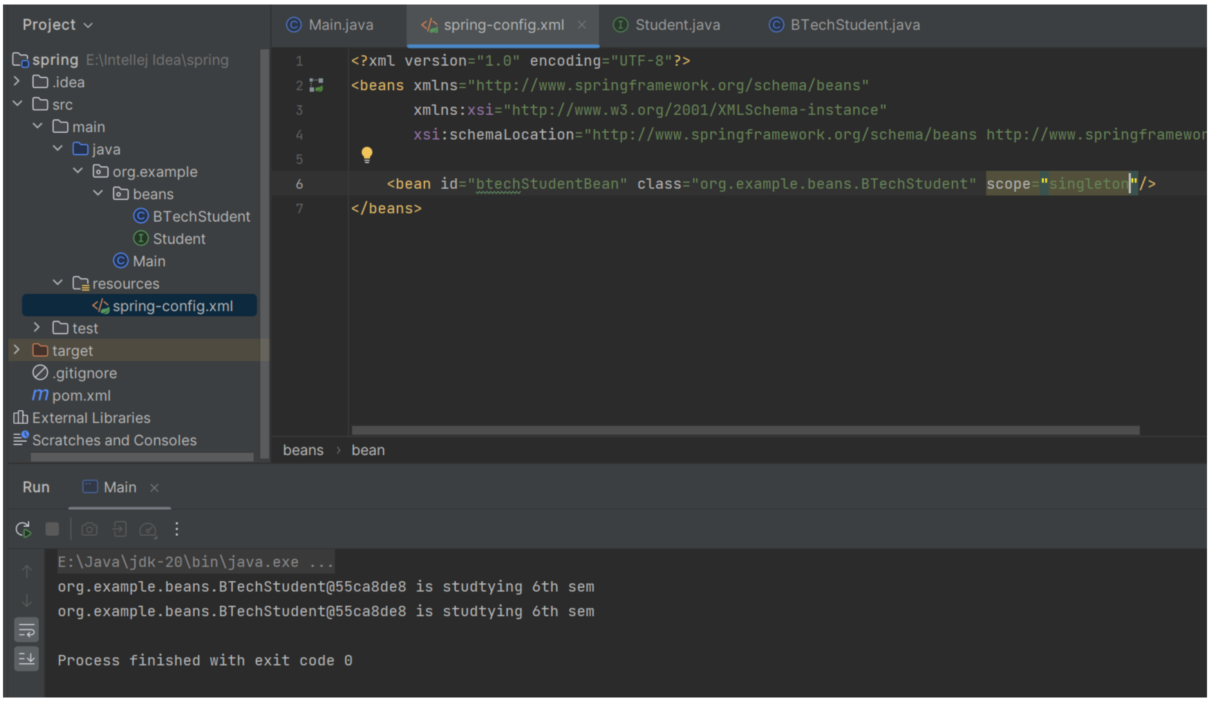


```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schem

    <bean id="btechStudentBean" class="org.example.beans.BTechStudent"/>
</beans>
```



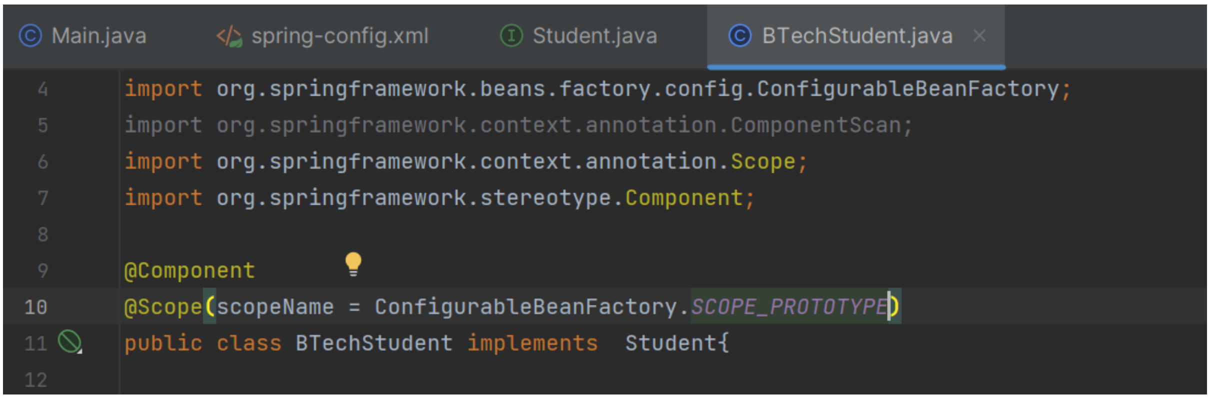
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.c

    <bean id="btechStudentBean" class="org.example.beans.BTechStudent" scope="prototype"/>
</beans>
```

```
E:\Java\jdk-20\bin\java.exe ...
org.example.beans.BTechStudent@7d68ef40 is studtying 6th sem
org.example.beans.BTechStudent@5b0abc94 is studtying 6th sem

Process finished with exit code 0
```



```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframewor

    <bean id="btechStudentBean" class="org.example.beans.BTechStudent" scope="singleton"/>
</beans>
```

```
E:\Java\jdk-20\bin\java.exe ...
org.example.beans.BTechStudent@55ca8de8 is studtying 6th sem
org.example.beans.BTechStudent@55ca8de8 is studtying 6th sem

Process finished with exit code 0
```



```java
import org.springframework.beans.factory.config.ConfigurableBeanFactory;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Component
@Scope(scopeName = ConfigurableBeanFactory.SCOPE_PROTOTYPE)
public class BTechStudent implements  Student{

```

Useful Methods: ApplicationContext and BeanFactory

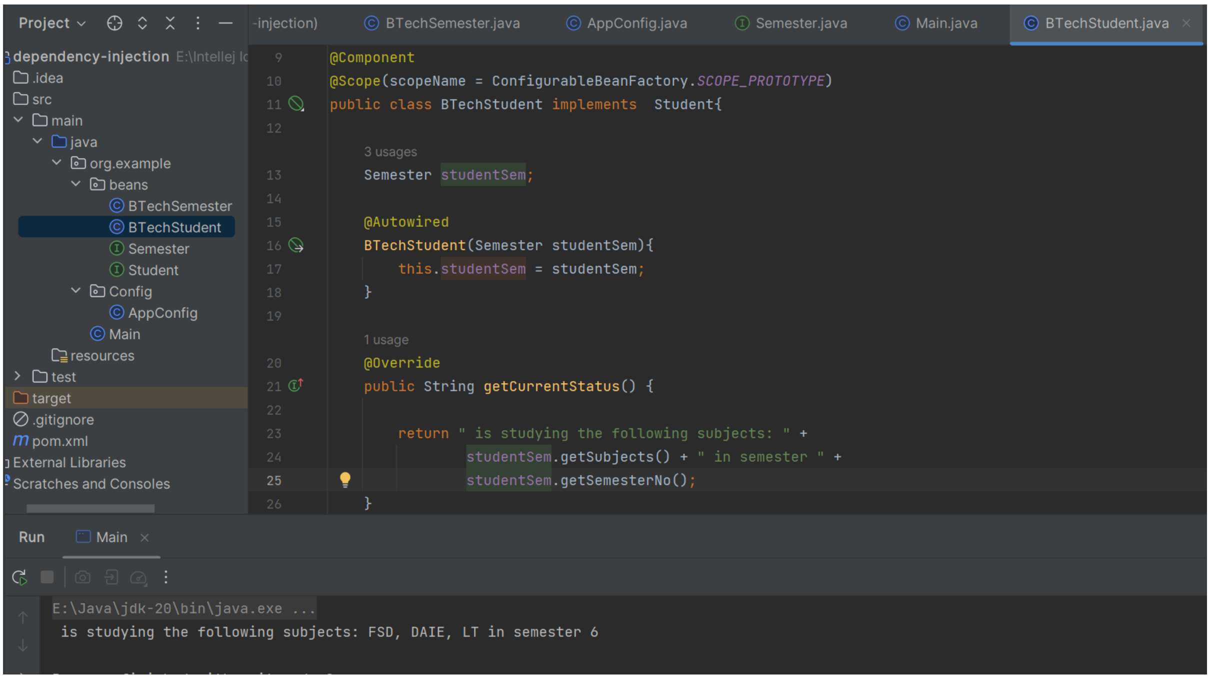
- boolean containsBean(String beanName)
  - Returns **true** if bean factory **contains** instance with the given name.
- Object getBean(String beanName)
  - We need to type **cast returned object** into desired type.
- T getBean(String beanName, Class<T> desiredType)
  - **Returns bean** instance of desired type registered with given name.
- Class<?> getType(String **beanName**)
  - **Returns** the **Class** of the **bean** having provided **name**.
- boolean isSingleton(String beanName)
  - **Returns true** if bean is a **singleton** object

Problems we face due to Direct Dependency

- Replace BTechSemester class with another improved class:
  - For example, BTechSemester -> ExpressBTechSemester class.
  - We need to modify all the classes that were using BTechSemester class.

- Modification in BTechSemester class:
  - For example, change in parameters of its constructor.
- We cannot test BTechStudent class without including BTechSemester class.
  - For example, using testing framework such as Junit.

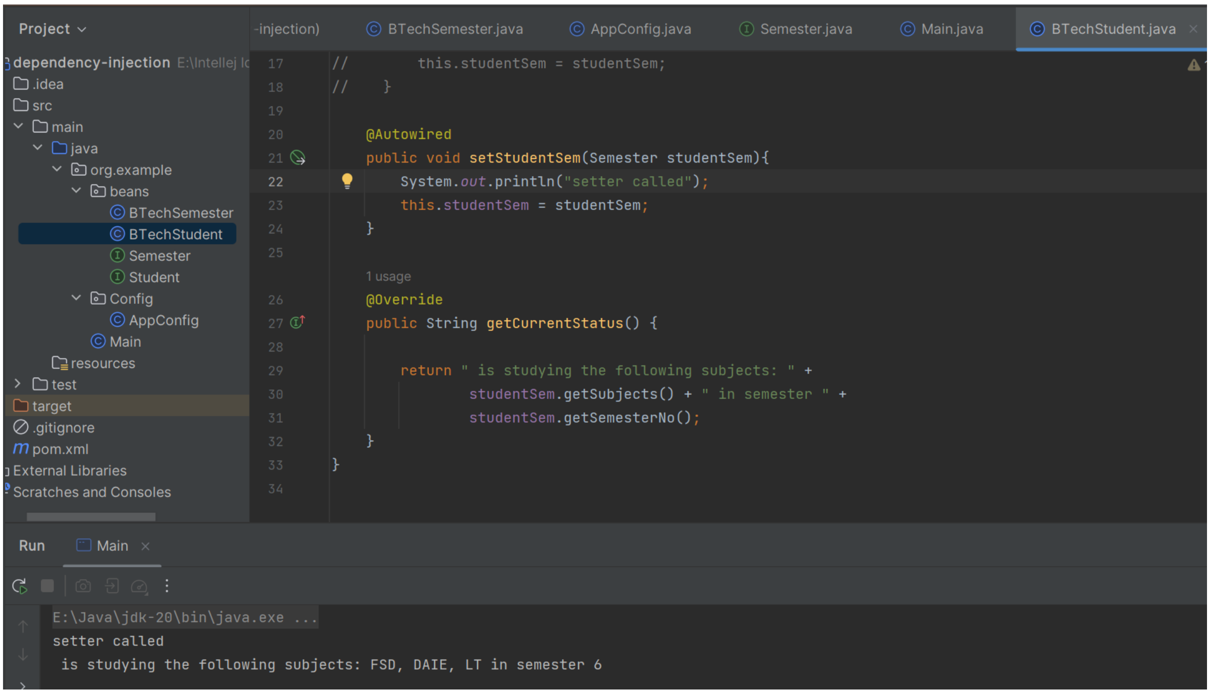
3. creating object using autowiring in constructor



```java
@Component
@Scope(scopeName = ConfigurableBeanFactory.SCOPE_PROTOTYPE)
public class BTechStudent implements  Student{

    Semester studentSem;

    @Autowired
    BTechStudent(Semester studentSem){
        this.studentSem = studentSem;
    }

    @Override
    public String getCurrentStatus() {

        return " is studying the following subjects: " +
                studentSem.getSubjects() + " in semester " +
                studentSem.getSemesterNo();
    }
```

```
E:\Java\jdk-20\bin\java.exe ...
 is studying the following subjects: FSD, DAIE, LT in semester 6
```

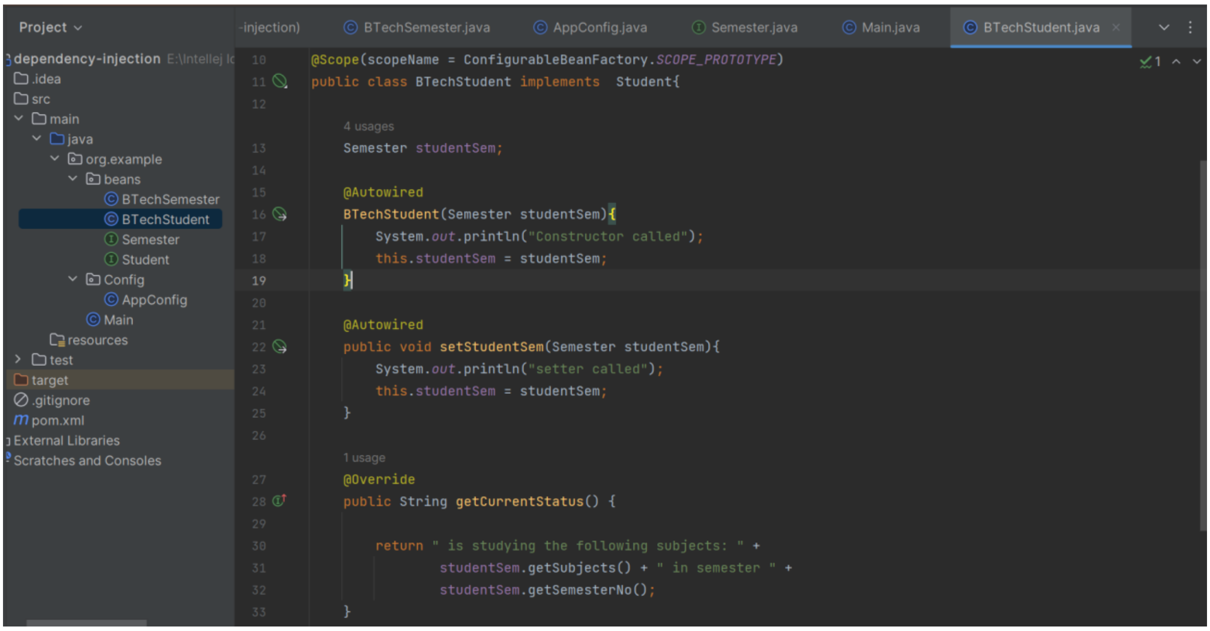
4.AutoWiring using setter



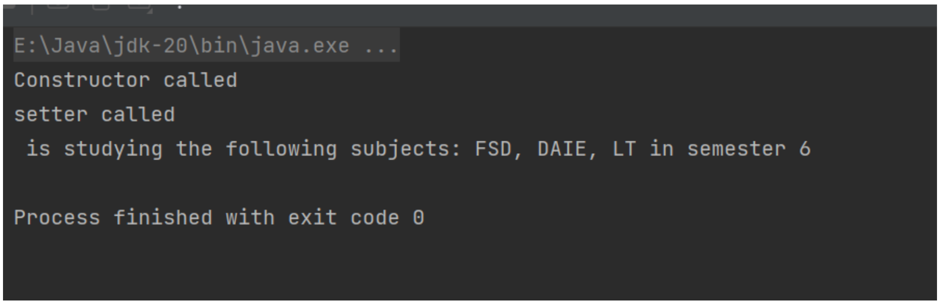
```
//        this.studentSem = studentSem;
//    }

    @Autowired
    public void setStudentSem(Semester studentSem){
        System.out.println("setter called");
        this.studentSem = studentSem;
    }

    1 usage
    @Override
    public String getCurrentStatus() {

        return " is studying the following subjects: " +
                studentSem.getSubjects() + " in semester " +
                studentSem.getSemesterNo();
    }
}
```

```
E:\Java\jdk-20\bin\java.exe ...
setter called
 is studying the following subjects: FSD, DAIE, LT in semester 6
```

5. using both:

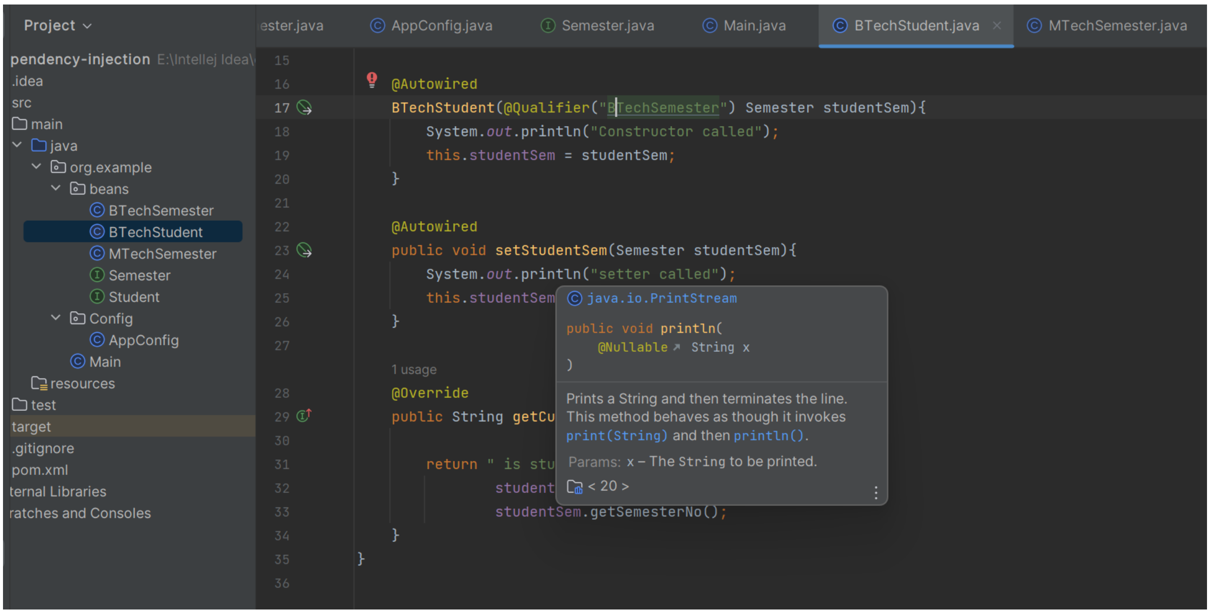


```
@Scope(scopeName = ConfigurableBeanFactory.SCOPE_PROTOTYPE)
public class BTechStudent implements  Student{

    4 usages
    Semester studentSem;

    @Autowired
    BTechStudent(Semester studentSem){
        System.out.println("Constructor called");
        this.studentSem = studentSem;
    }

    @Autowired
    public void setStudentSem(Semester studentSem){
        System.out.println("setter called");
        this.studentSem = studentSem;
    }

    1 usage
    @Override
    public String getCurrentStatus() {

        return " is studying the following subjects: " +
                studentSem.getSubjects() + " in semester " +
                studentSem.getSemesterNo();
    }
```



```
E:\Java\jdk-20\bin\java.exe ...
Constructor called
setter called
 is studying the following subjects: FSD, DAIE, LT in semester 6

Process finished with exit code 0
```

6. when I make MTechSemester implements Semester then autwired gets confused where to go BTech or MTech @Qualifier



```
    @Autowired
    BTechStudent(@Qualifier("BTechSemester") Semester studentSem){
        System.out.println("Constructor called");
        this.studentSem = studentSem;
    }

    @Autowired
    public void setStudentSem(Semester studentSem){
        System.out.println("setter called");
        this.studentSem
    }

    1 usage
    @Override
    public String getCu

        return " is stu
                student
                studentSem.getSemesterNo();
    }
}
```

java.io.PrintStream

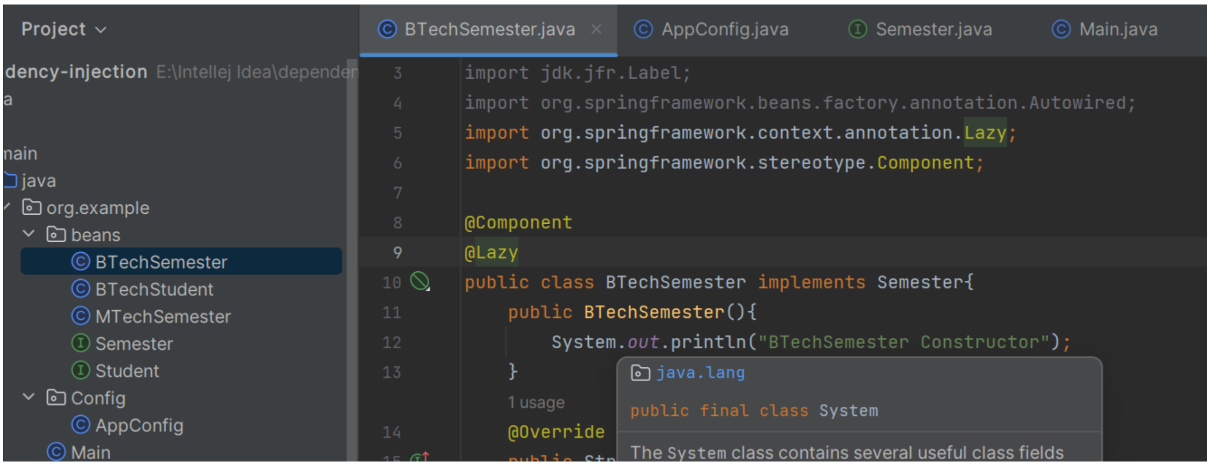
```
public void println(
    @Nullable String x
)
```

Prints a String and then terminates the line. This method behaves as though it invokes print(String) and then println().

Params: x – The String to be printed.

< 20 >

Now calling main both btech and mtech constructor called
so to remove unnecessary constructor call add lazy

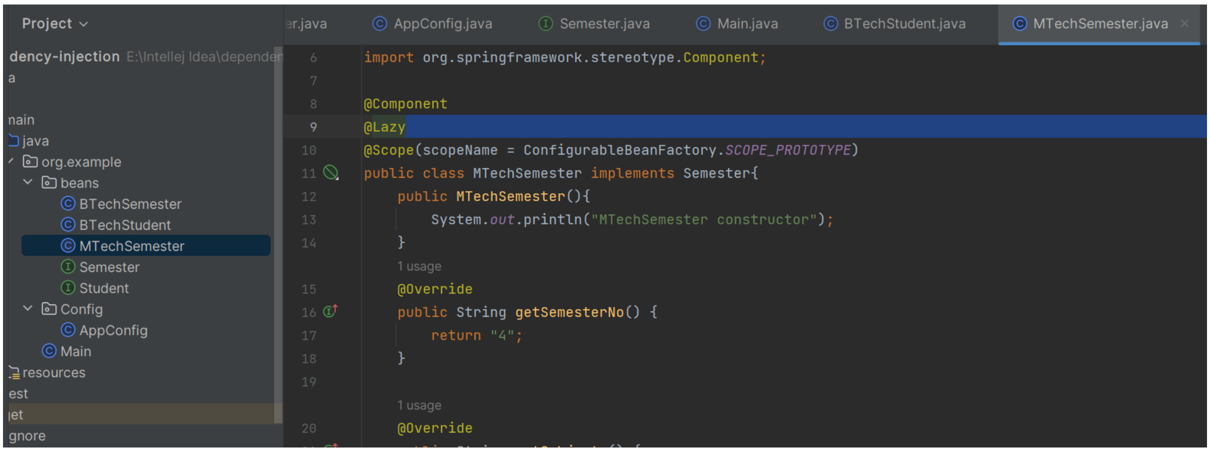


```
import jdk.jfr.Label;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Lazy;
import org.springframework.stereotype.Component;

@Component
@Lazy
public class BTechSemester implements Semester{
    public BTechSemester(){
        System.out.println("BTechSemester Constructor");
    }
    1 usage
    @Override
```

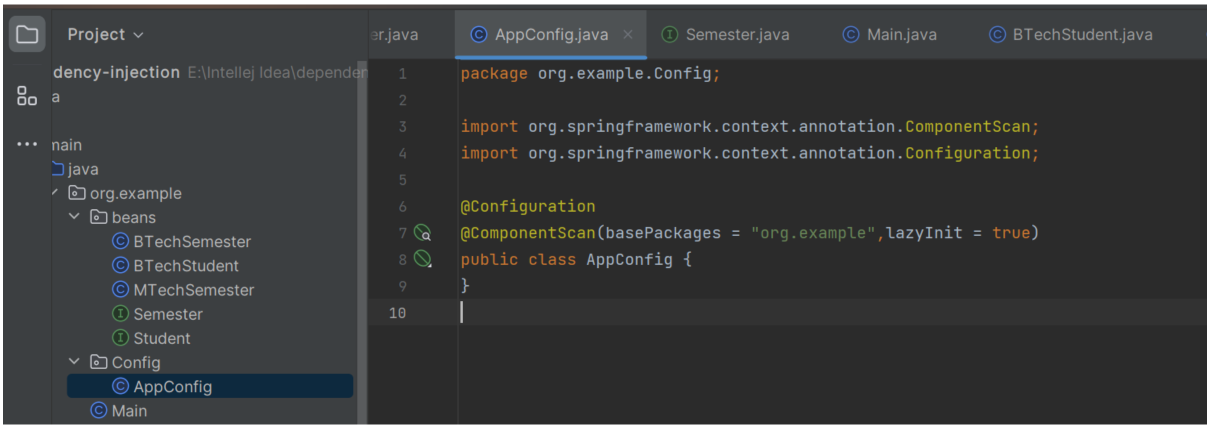
java.lang

```
public final class System
```

The System class contains several useful class fields

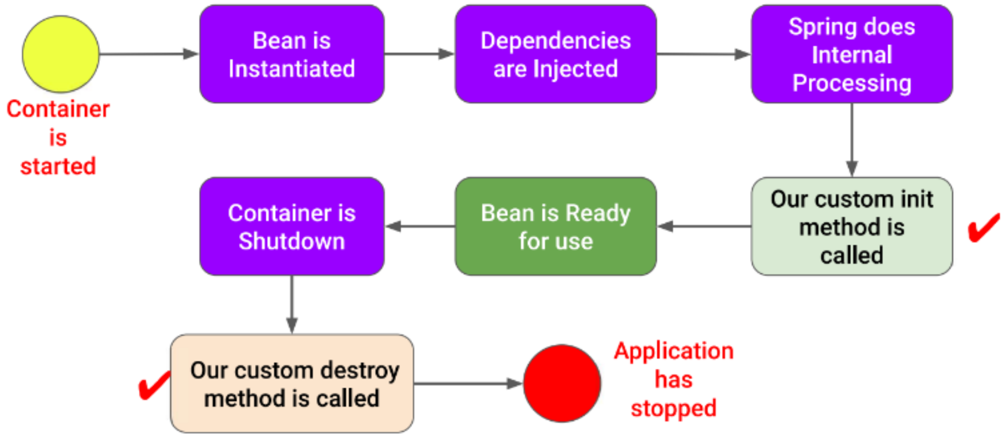


```
import org.springframework.stereotype.Component;

@Component
@Lazy
@Scope(scopeName = ConfigurableBeanFactory.SCOPE_PROTOTYPE)
public class MTechSemester implements Semester{
    public MTechSemester(){
        System.out.println("MTechSemester constructor");
    }
    1 usage
    @Override
    public String getSemesterNo() {
        return "4";
    }

    1 usage
    @Override
```



```java
package org.example.Config;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = "org.example",lazyInit = true)
public class AppConfig {
}
```

Lifecycle of Bean



Container is started

Bean is Instantiated

Dependencies are Injected

Spring does Internal Processing

Our custom init method is called ✔

Bean is Ready for use

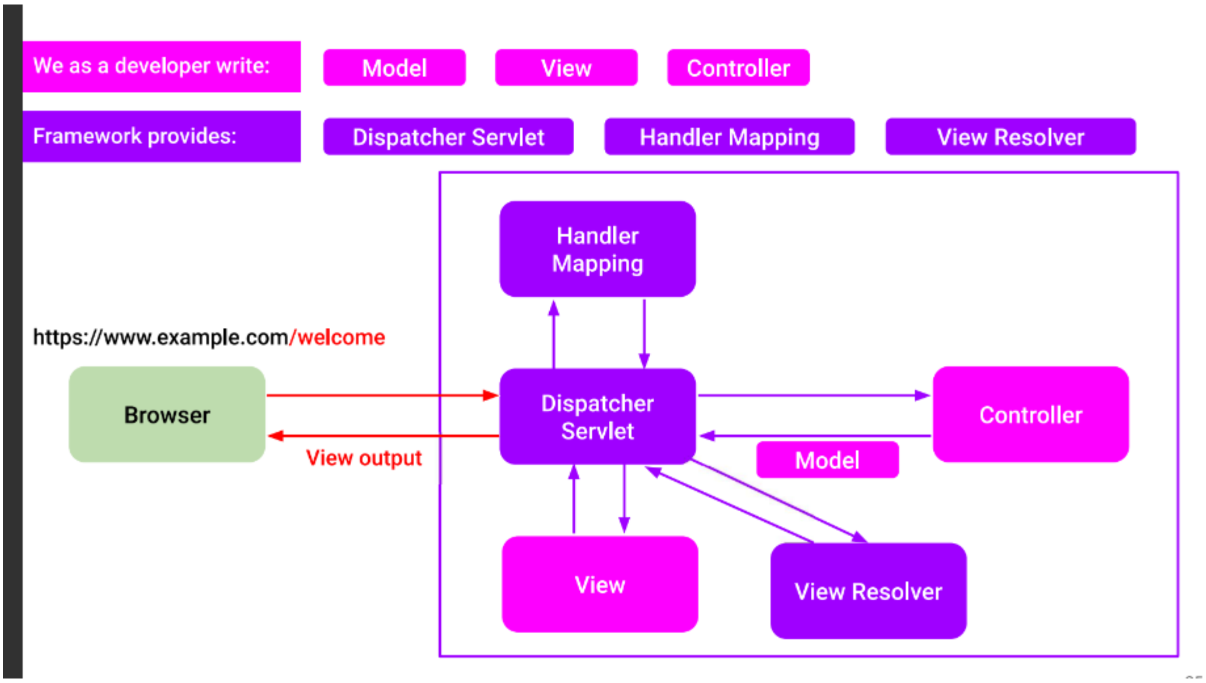
Container is Shutdown

✔ Our custom destroy method is called

Application has stopped

✔Spring allows us to write our own initialization and destruction logic in these methods

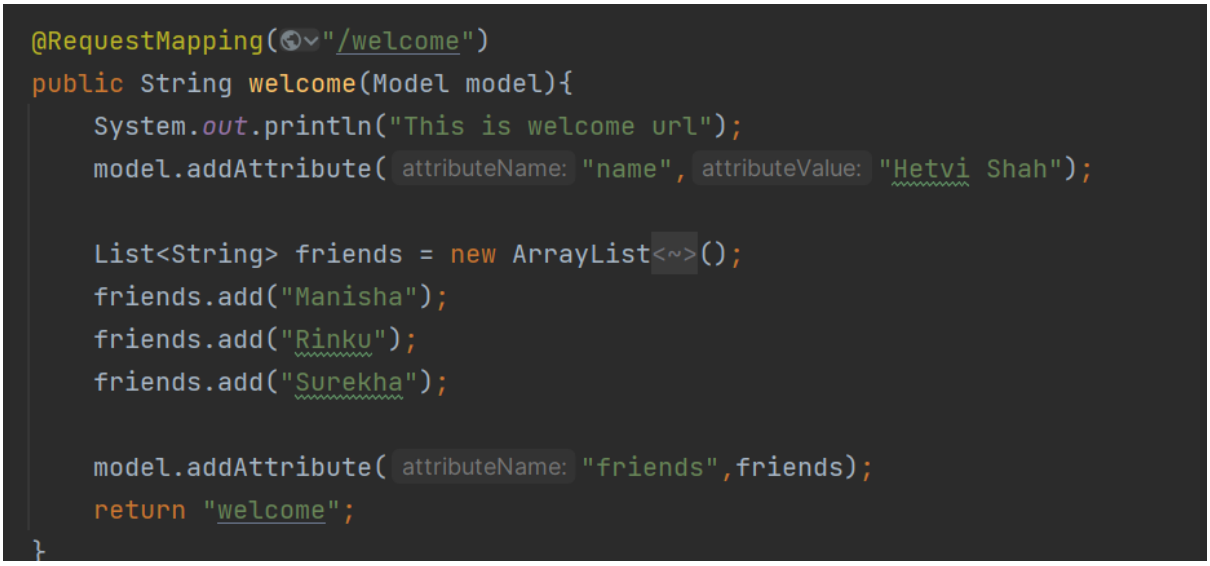
Spring mvc:



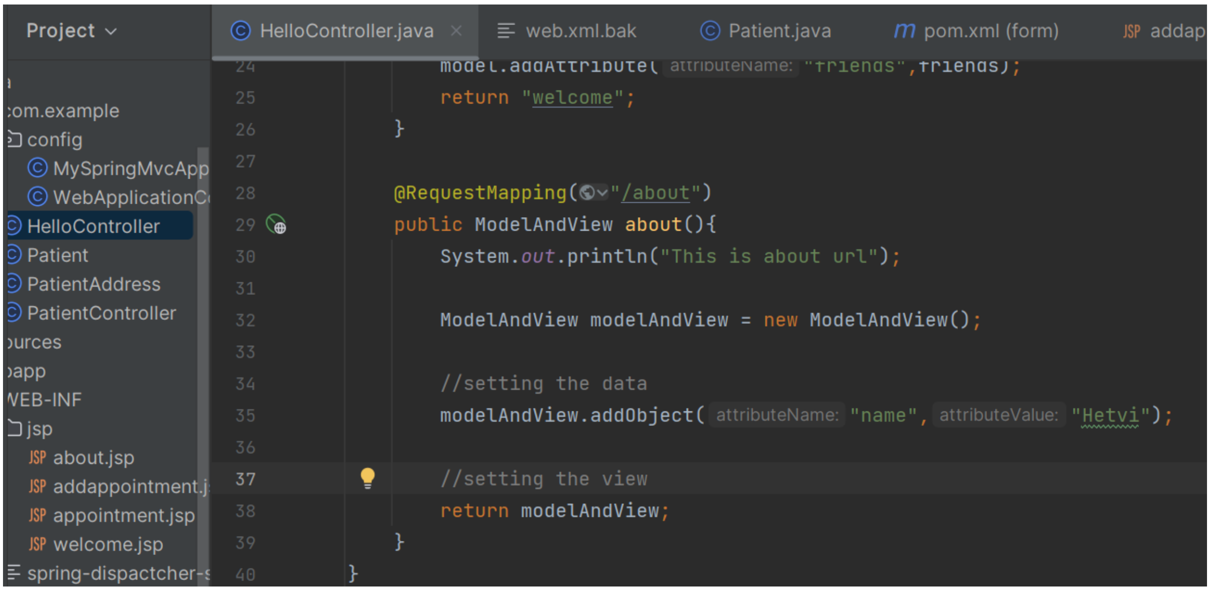
We as a developer write: Model View Controller

Framework provides: Dispatcher Servlet Handler Mapping View Resolver

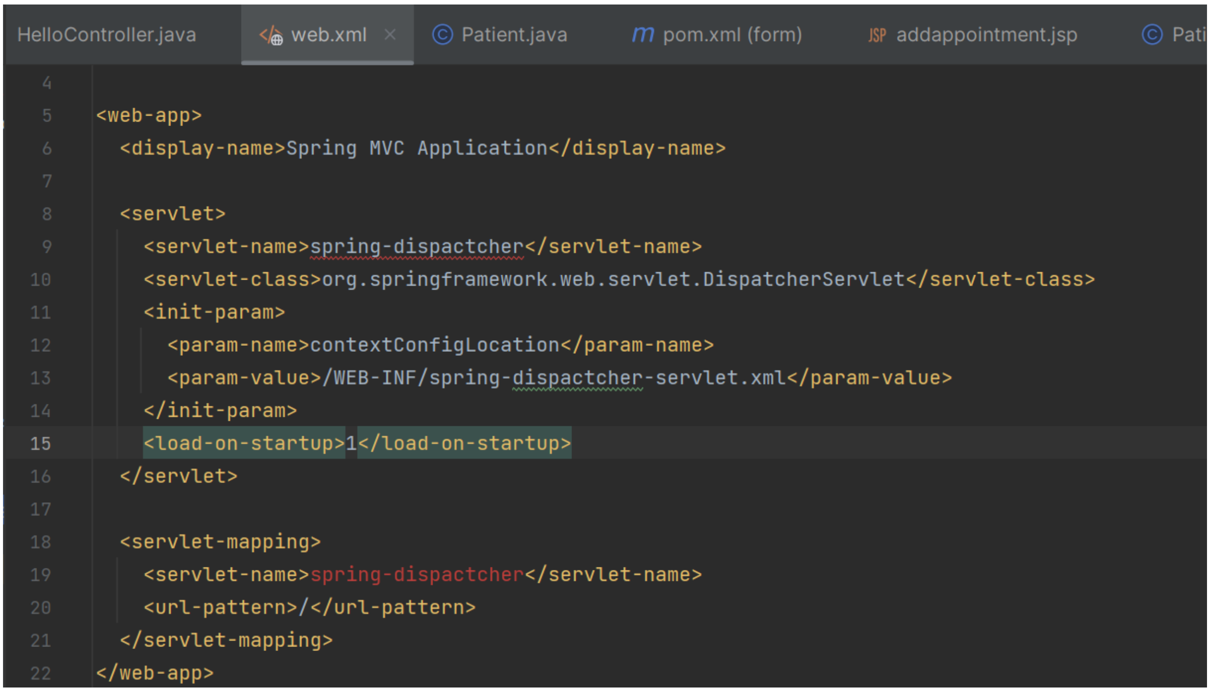
1. Controller with xml file:



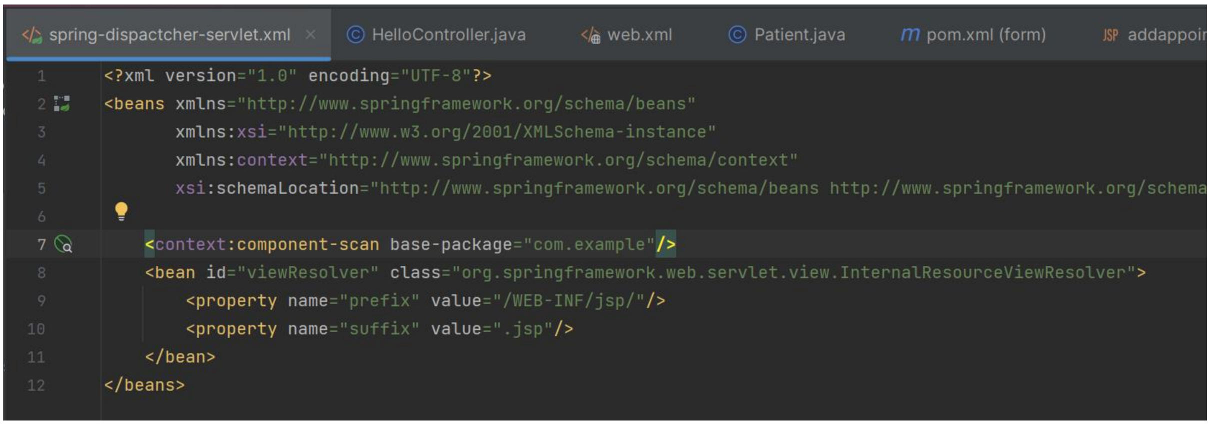
```
@RequestMapping("/welcome")
public String welcome(Model model){
    System.out.println("This is welcome url");
    model.addAttribute( attributeName: "name", attributeValue: "Hetvi Shah");

    List<String> friends = new ArrayList<~>();
    friends.add("Manisha");
    friends.add("Rinku");
    friends.add("Surekha");

    model.addAttribute( attributeName: "friends",friends);
    return "welcome";
}
```



```java
        model.addAttribute( attributeName: "friends",friends);
        return "welcome";
    }

    @RequestMapping("/about")
    public ModelAndView about(){
        System.out.println("This is about url");

        ModelAndView modelAndView = new ModelAndView();

        //setting the data
        modelAndView.addObject( attributeName: "name", attributeValue: "Hetvi");

        //setting the view
        return modelAndView;
    }
}
```

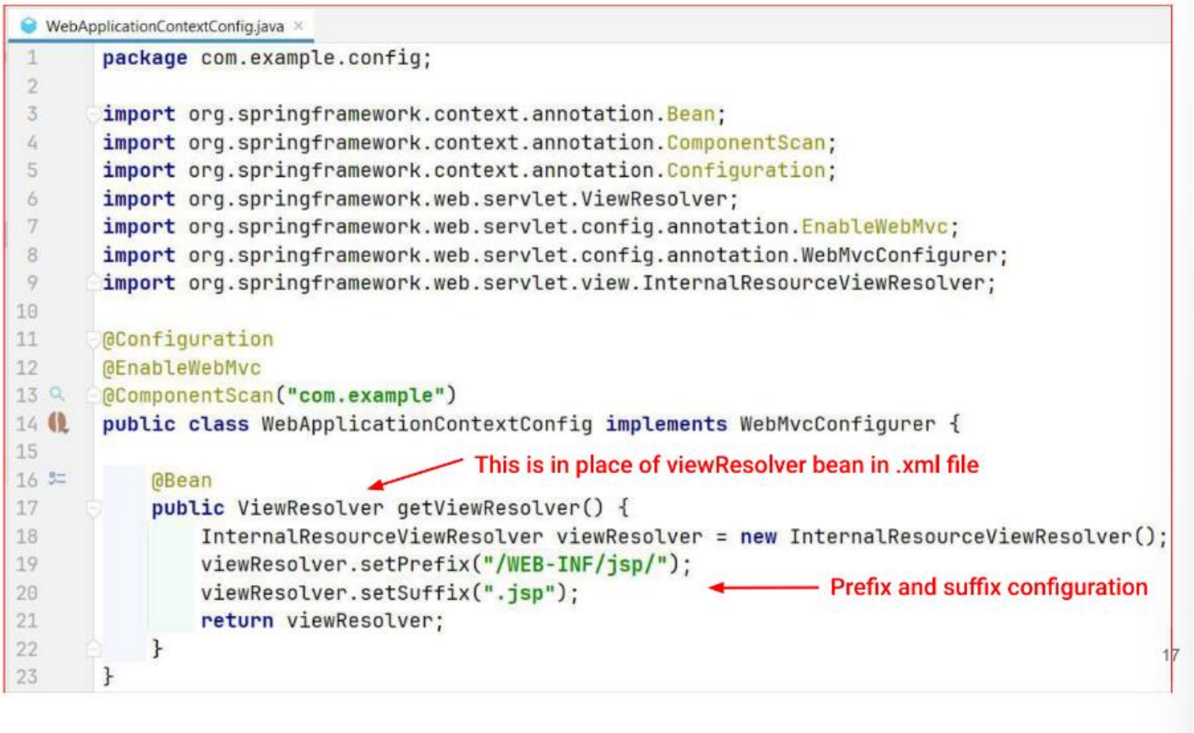


```xml

<web-app>
  <display-name>Spring MVC Application</display-name>

  <servlet>
    <servlet-name>spring-dispactcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/spring-dispactcher-servlet.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>spring-dispactcher</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```



```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema

    <context:component-scan base-package="com.example"/>
    <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
    </bean>
</beans>
```

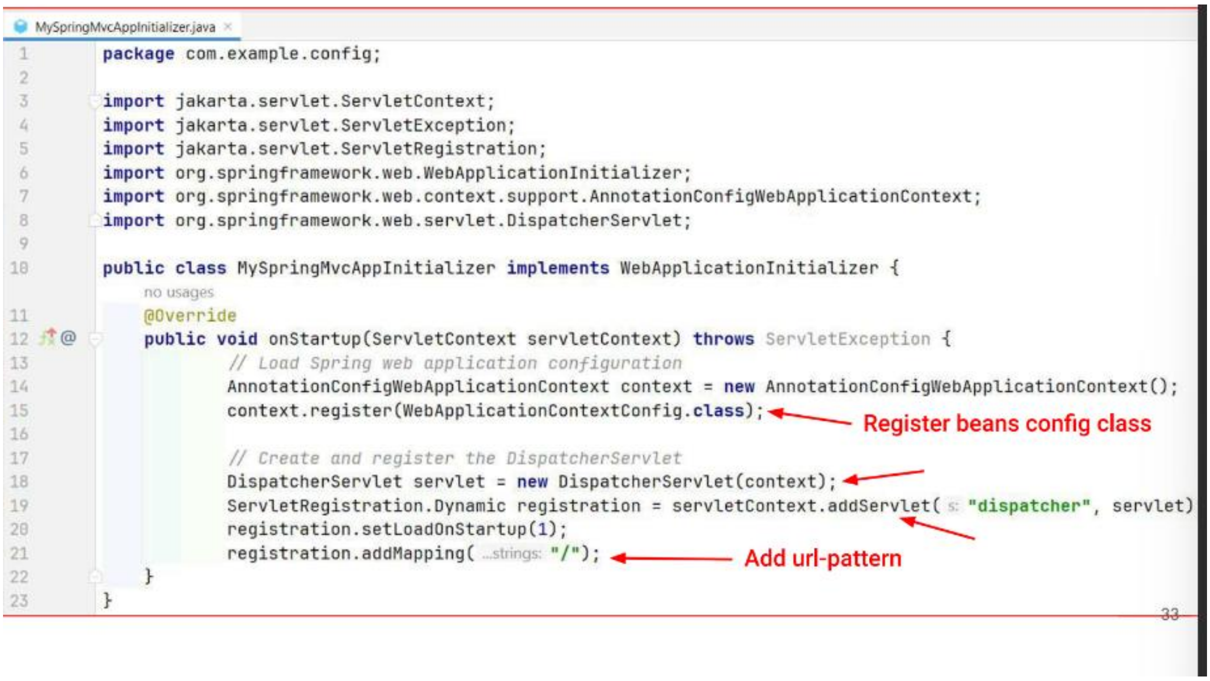
2. without xml file



```java
package com.example.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.ViewResolver;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.view.InternalResourceViewResolver;

@Configuration
@EnableWebMvc
@ComponentScan("com.example")
public class WebApplicationContextConfig implements WebMvcConfigurer {

    @Bean
    public ViewResolver getViewResolver() {
        InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();
        viewResolver.setPrefix("/WEB-INF/jsp/");
        viewResolver.setSuffix(".jsp");
        return viewResolver;
    }
}
```

This is in place of viewResolver bean in .xml file

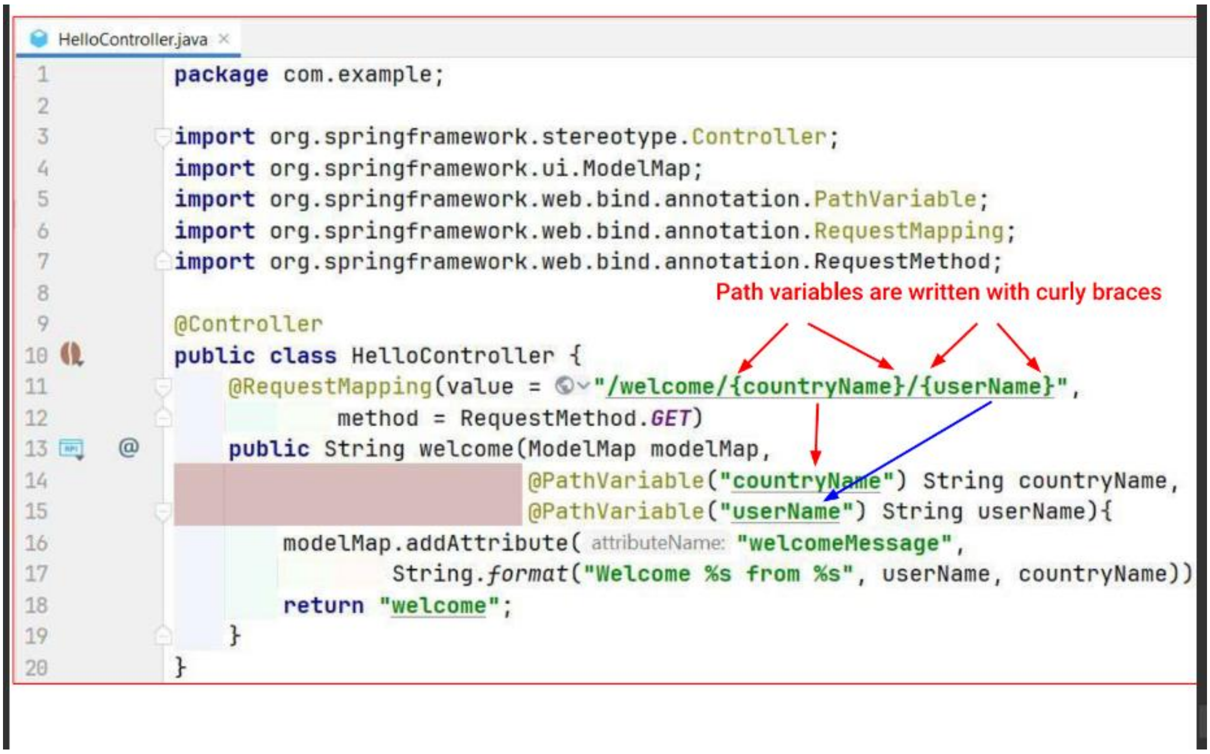
Prefix and suffix configuration



```java
package com.example.config;

import jakarta.servlet.ServletContext;
import jakarta.servlet.ServletException;
import jakarta.servlet.ServletRegistration;
import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;

public class MySpringMvcAppInitializer implements WebApplicationInitializer {
    no usages
    @Override
    public void onStartup(ServletContext servletContext) throws ServletException {
        // Load Spring web application configuration
        AnnotationConfigWebApplicationContext context = new AnnotationConfigWebApplicationContext();
        context.register(WebApplicationContextConfig.class);

        // Create and register the DispatcherServlet
        DispatcherServlet servlet = new DispatcherServlet(context);
        ServletRegistration.Dynamic registration = servletContext.addServlet( s: "dispatcher", servlet)
        registration.setLoadOnStartup(1);
        registration.addMapping( ...strings: "/");
    }
}
```

Register beans config class

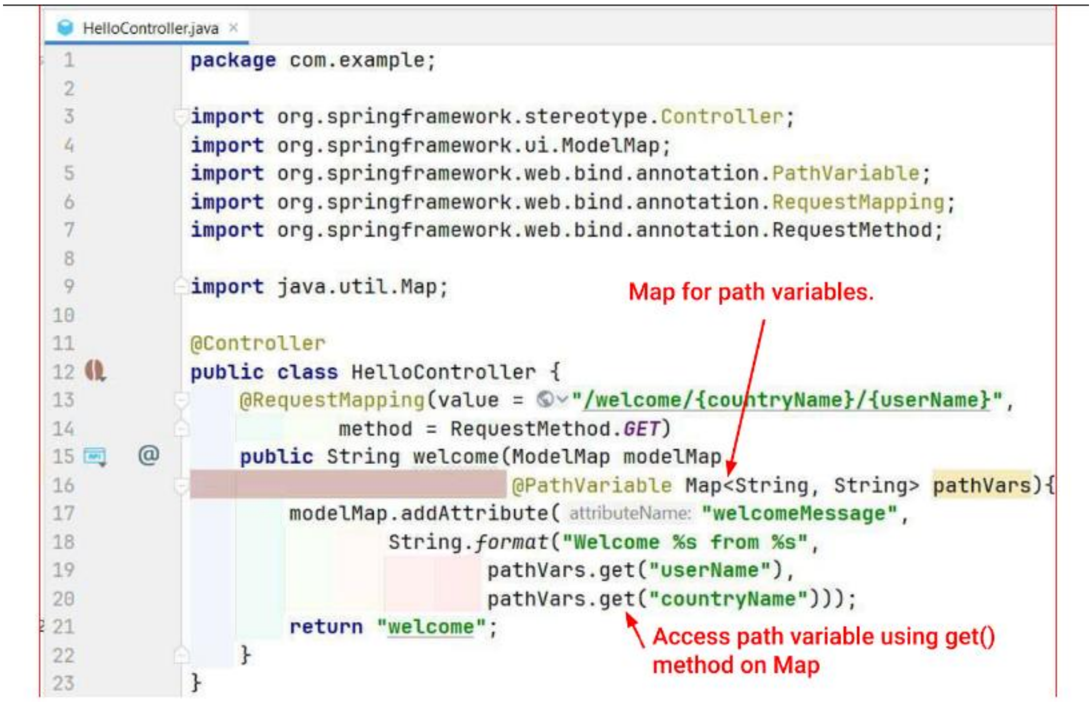
Add url-pattern



HelloController.java

```java
package com.example;

import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class HelloController {
    @RequestMapping(value = "/welcome/{countryName}/{userName}",
            method = RequestMethod.GET)
    public String welcome(ModelMap modelMap,
                          @PathVariable("countryName") String countryName,
                          @PathVariable("userName") String userName){
        modelMap.addAttribute( attributeName: "welcomeMessage",
                String.format("Welcome %s from %s", userName, countryName))
        return "welcome";
    }
}
```

Path variables are written with curly braces

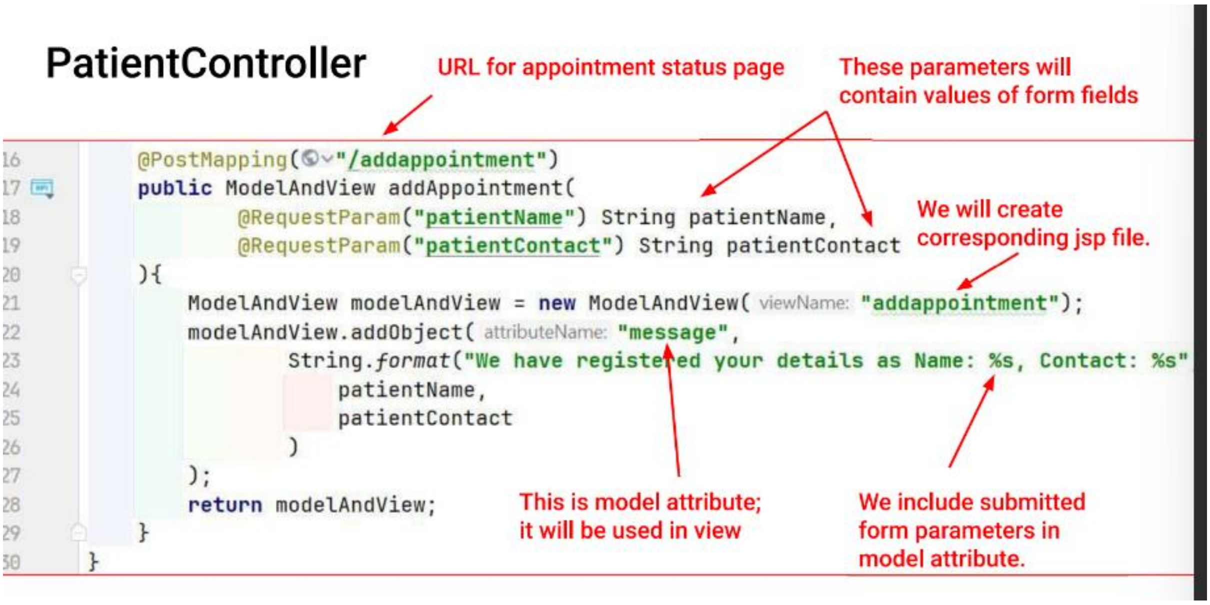


HelloController.java

```java
package com.example;

import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import java.util.Map;

@Controller
public class HelloController {
    @RequestMapping(value = "/welcome/{countryName}/{userName}",
            method = RequestMethod.GET)
    public String welcome(ModelMap modelMap,
                          @PathVariable Map<String, String> pathVars){
        modelMap.addAttribute( attributeName: "welcomeMessage",
                String.format("Welcome %s from %s",
                        pathVars.get("userName"),
                        pathVars.get("countryName")));
        return "welcome";
    }
}
```

Map for path variables.

Access path variable using get() method on Map



PatientController

URL for appointment status page

These parameters will contain values of form fields

We will create corresponding jsp file.

```
@PostMapping("/addappointment")
public ModelAndView addAppointment(
        @RequestParam("patientName") String patientName,
        @RequestParam("patientContact") String patientContact
){
    ModelAndView modelAndView = new ModelAndView( viewName: "addappointment");
    modelAndView.addObject( attributeName: "message",
            String.format("We have registered your details as Name: %s, Contact: %s"
                    patientName,
                    patientContact
            )
    );
    return modelAndView;
}
}
```

This is model attribute; it will be used in view

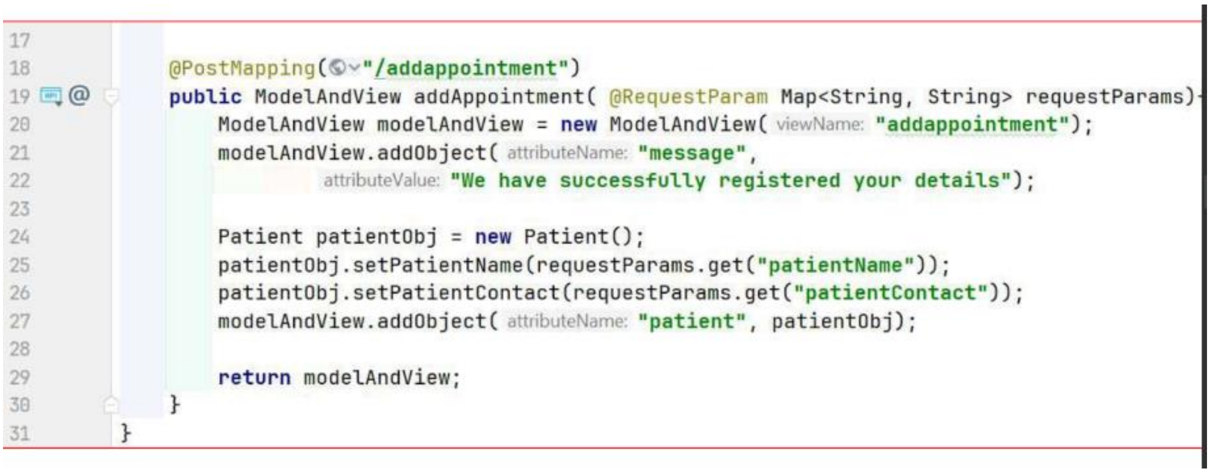
We include submitted form parameters in model attribute.

field.

To take default value for student's name, we write the header of request handler method as follows:

This default element name, is not needed for single element.

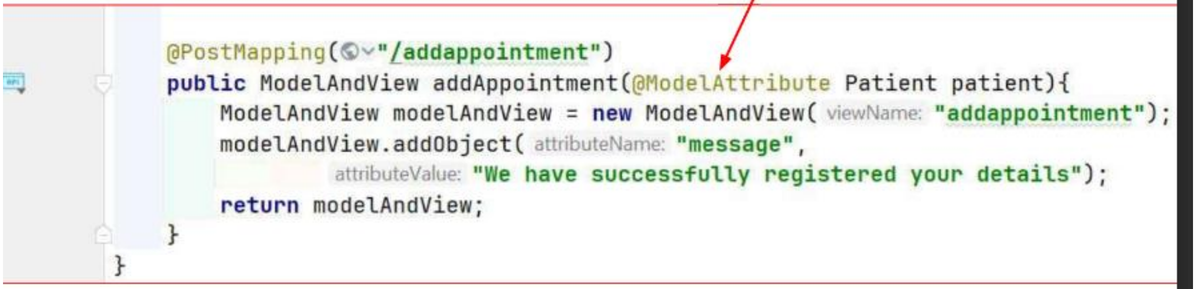
- @RequestParam(value="patientContact", defaultValue = "Not available") String patientContact



```
@PostMapping("/addappointment")
public ModelAndView addAppointment( @RequestParam Map<String, String> requestParams)
    ModelAndView modelAndView = new ModelAndView( viewName: "addappointment");
    modelAndView.addObject( attributeName: "message",
            attributeValue: "We have successfully registered your details");

    Patient patientObj = new Patient();
    patientObj.setPatientName(requestParams.get("patientName"));
    patientObj.setPatientContact(requestParams.get("patientContact"));
    modelAndView.addObject( attributeName: "patient", patientObj);

    return modelAndView;
}
}
```

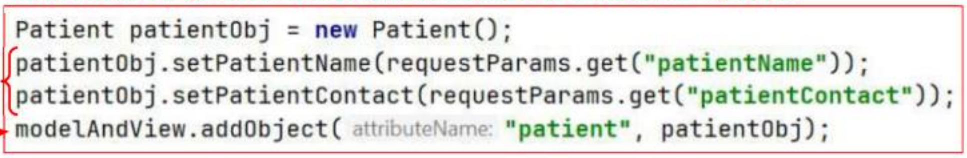
Update PatientController

1. Does data binding
2. Adds patient object as a model attribute



```
    @PostMapping(🌐∨"/addappointment")
    public ModelAndView addAppointment(@ModelAttribute Patient patient){
        ModelAndView modelAndView = new ModelAndView( viewName: "addappointment");
        modelAndView.addObject( attributeName: "message",
                attributeValue: "We have successfully registered your details");
        return modelAndView;
    }
}
```

With the use of @ModelAttribute, we do not need to write following steps:

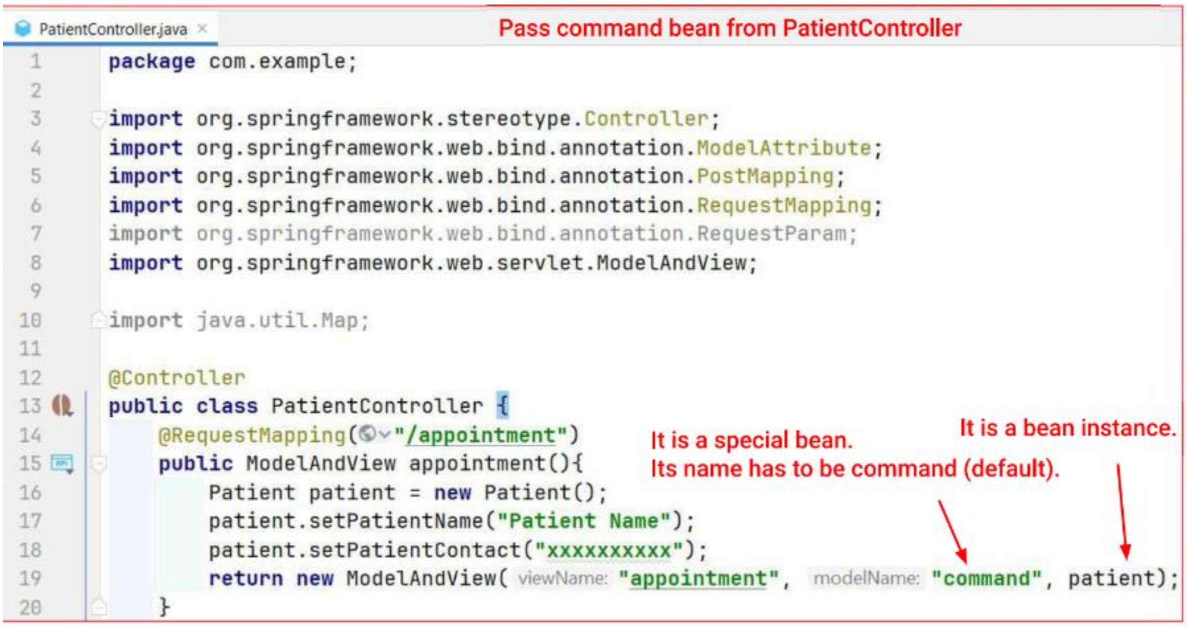


```
Patient patientObj = new Patient();
patientObj.setPatientName(requestParams.get("patientName"));
patientObj.setPatientContact(requestParams.get("patientContact"));
modelAndView.addObject( attributeName: "patient", patientObj);
```

ata binding

Add attribute

Pass command bean from PatientController

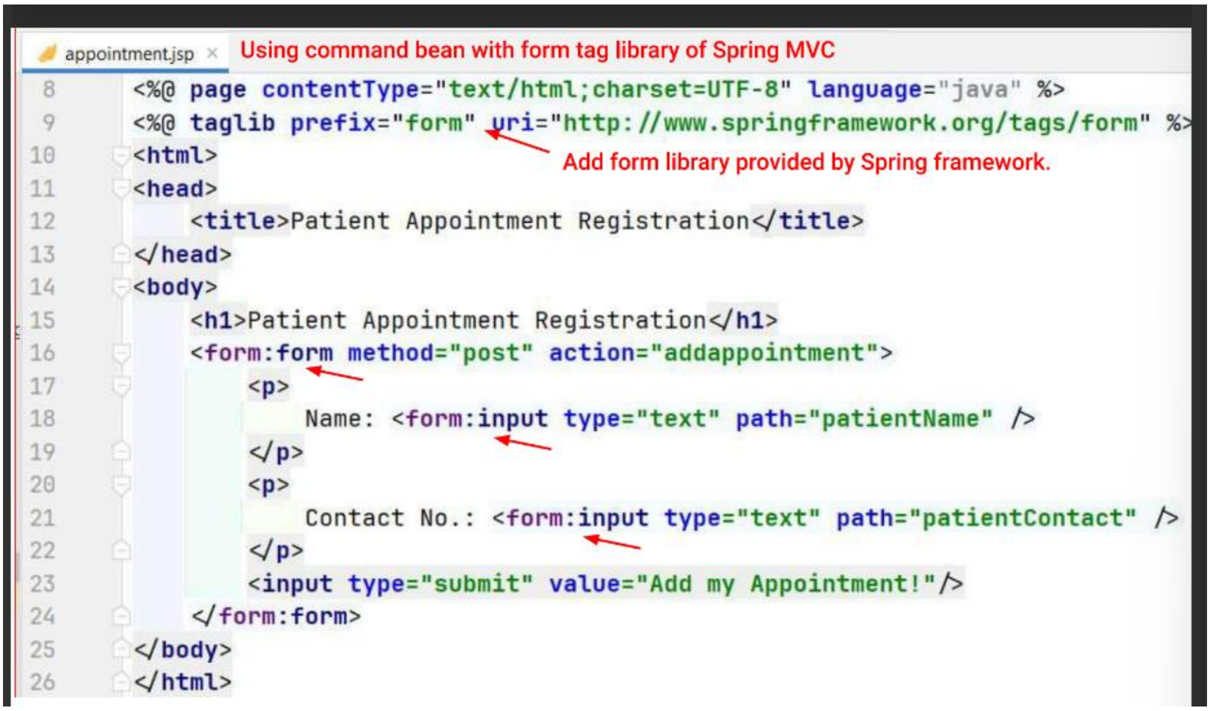


PatientController.java

```
package com.example;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;

import java.util.Map;

@Controller
public class PatientController {
    @RequestMapping(🌐∨"/appointment")
    public ModelAndView appointment(){
        Patient patient = new Patient();
        patient.setPatientName("Patient Name");
        patient.setPatientContact("xxxxxxxxxx");
        return new ModelAndView( viewName: "appointment",  modelName: "command", patient);
    }
```

It is a special bean.
Its name has to be command (default).

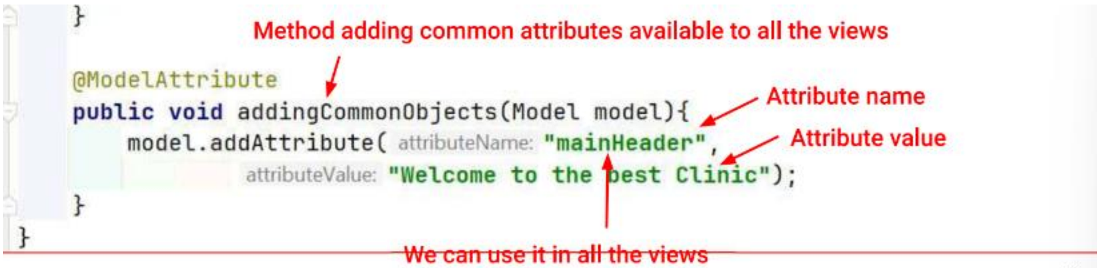
It is a bean instance.



Using command bean with form tag library of Spring MVC

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
<head>
    <title>Patient Appointment Registration</title>
</head>
<body>
    <h1>Patient Appointment Registration</h1>
    <form:form method="post" action="addappointment">
        <p>
            Name: <form:input type="text" path="patientName" />
        </p>
        <p>
            Contact No.: <form:input type="text" path="patientContact" />
        </p>
        <input type="submit" value="Add my Appointment!"/>
    </form:form>
</body>
</html>
```

Add form library provided by Spring framework.



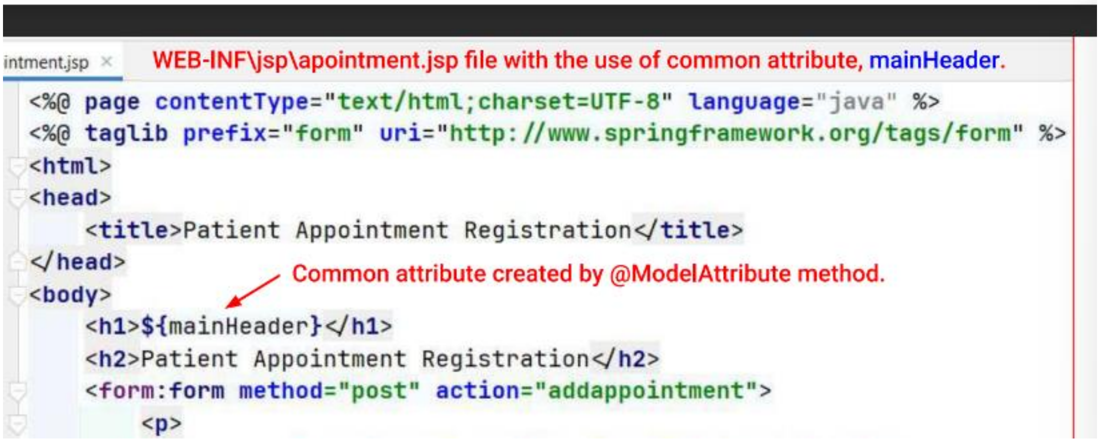
Method adding common attributes available to all the views

```
    }

    @ModelAttribute
    public void addingCommonObjects(Model model){
        model.addAttribute( attributeName: "mainHeader",
                attributeValue: "Welcome to the best Clinic");
    }
}
```

Attribute name

Attribute value

We can use it in all the views



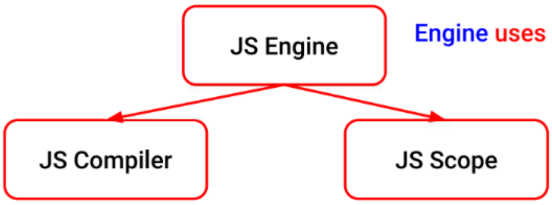
WEB-INF\jsp\apointment.jsp file with the use of common attribute, mainHeader.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
<head>
    <title>Patient Appointment Registration</title>
</head>
<body>
    <h1>${mainHeader}</h1>
    <h2>Patient Appointment Registration</h2>
    <form:form method="post" action="addappointment">
        <p>
```

Common attribute created by @ModelAttribute method.

How Scope is Handled in JavaScript

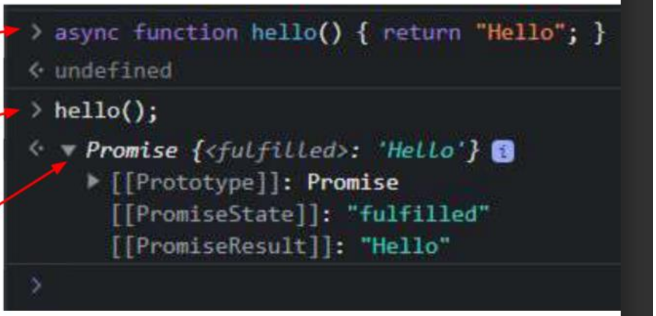
- There are three separate components while JavaScript code execution:
  - JS Engine: Responsible for start-to-finish compilation and execution of JS program.
  - JS Compiler: It helps engine. Responsible for parsing and code-generation.
  - JS Scope: It helps engine. It collects and maintains a lookup list of all the declared identifiers.



Engine uses Compiler and Scope.

How to Create a Promise Implicitly?

- Now, we write and call an async function in console of Web Browser.



```
> async function hello() { return "Hello"; }
<· undefined
> hello();
<· ▼ Promise {<fulfilled>: 'Hello'}
    ▶ [[Prototype]]: Promise
      [[PromiseState]]: "fulfilled"
      [[PromiseResult]]: "Hello"
>
```

Create an async function.

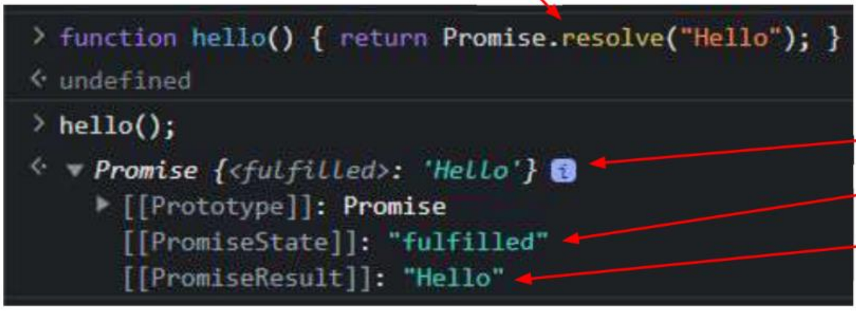
Call the async function.

Return value of an async function is a promise object.

How to Create a Promise Explicitly?

- We explicitly return a promise object.

Explicitly return a Promise object.



```
> function hello() { return Promise.resolve("Hello"); }
< undefined
> hello();
< ▼ Promise {<fulfilled>: 'Hello'} ⓘ
    ▶ [[Prototype]]: Promise
      [[PromiseState]]: "fulfilled"
      [[PromiseResult]]: "Hello"
```

Promise object

Promise state

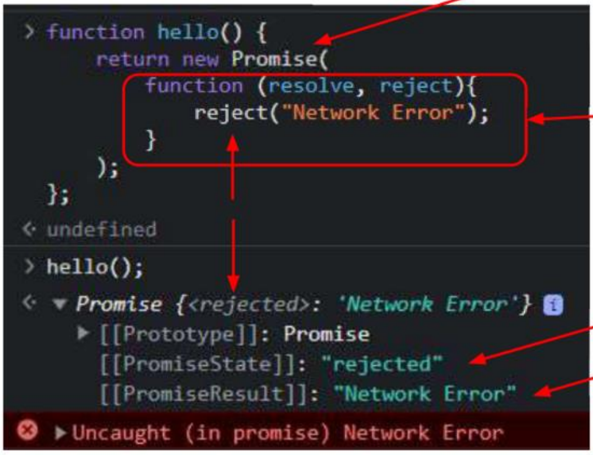
Promise error

How to Create a Promise Explicitly using constructor?

- Reject Promise:

Create a Promise object.

- We return a rejected promise using reject function that we get via executor function.



```
> function hello() {
      return new Promise(
          function (resolve, reject){
              reject("Network Error");
          }
      );
  };
< undefined
> hello();
< ▼ Promise {<rejected>: 'Network Error'} ⓘ
    ▶ [[Prototype]]: Promise
      [[PromiseState]]: "rejected"
      [[PromiseResult]]: "Network Error"
⊗ ▶Uncaught (in promise) Network Error
```

Executor function

Promise state is rejected.

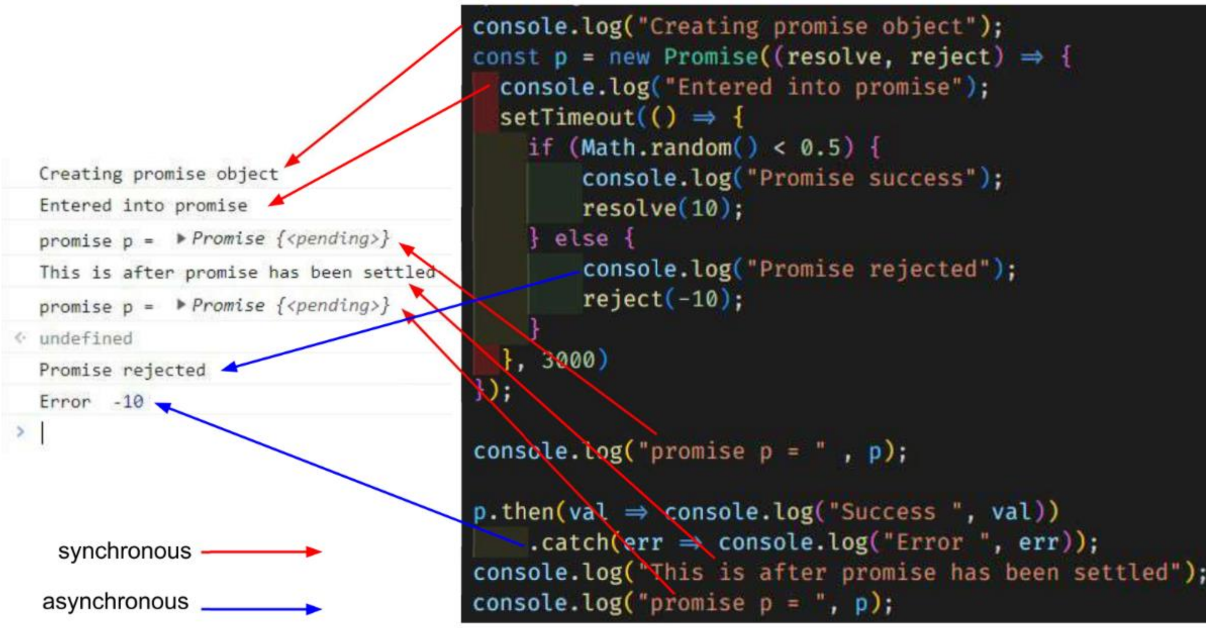
Promise result is an error.

How do we pass our callback functions?

- We pass our callback functions to promise object using:
  - then() for registration of success event handler.
  - catch() for registration of failure event handler.
  - finally() for registration of finally event handler.

How to use then(), catch(), and finally()

- then() is used to register callback function to be called for success event (when the promise resolves and produces a value).
  - (response) => { process response }
- catch() is used to register callback function for failure event.
  - (error) => { process error }
- finally() is used to register callback function for finally of promise.
  - (anything) => { process cleanup }

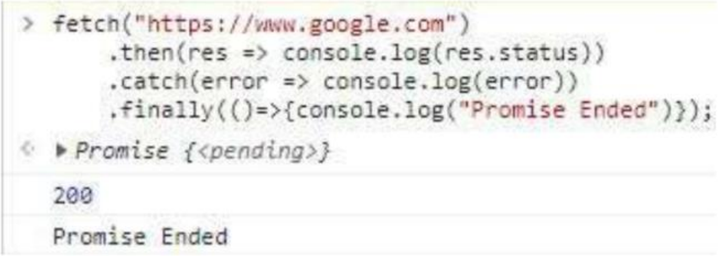


```
Creating promise object
Entered into promise
promise p =  ▸ Promise {<pending>}
This is after promise has been settled
promise p =  ▸ Promise {<pending>}
<· undefined
Promise rejected
Error  -10
> |
```

```
console.log("Creating promise object");
const p = new Promise((resolve, reject) => {
  console.log("Entered into promise");
  setTimeout(() => {
    if (Math.random() < 0.5) {
      console.log("Promise success");
      resolve(10);
    } else {
      console.log("Promise rejected");
      reject(-10);
    }
  }, 3000)
});

console.log("promise p = " , p);

p.then(val => console.log("Success ", val))
    .catch(err => console.log("Error ", err));
console.log("This is after promise has been settled");
console.log("promise p = ", p);
```

Using Promise in Console of Browser

```
fetch("https://www.google.com")
	.then(res => console.log(res.status))
	.catch(error => console.log(error))
	.finally(()=>{console.log("Promise Ended")});
```

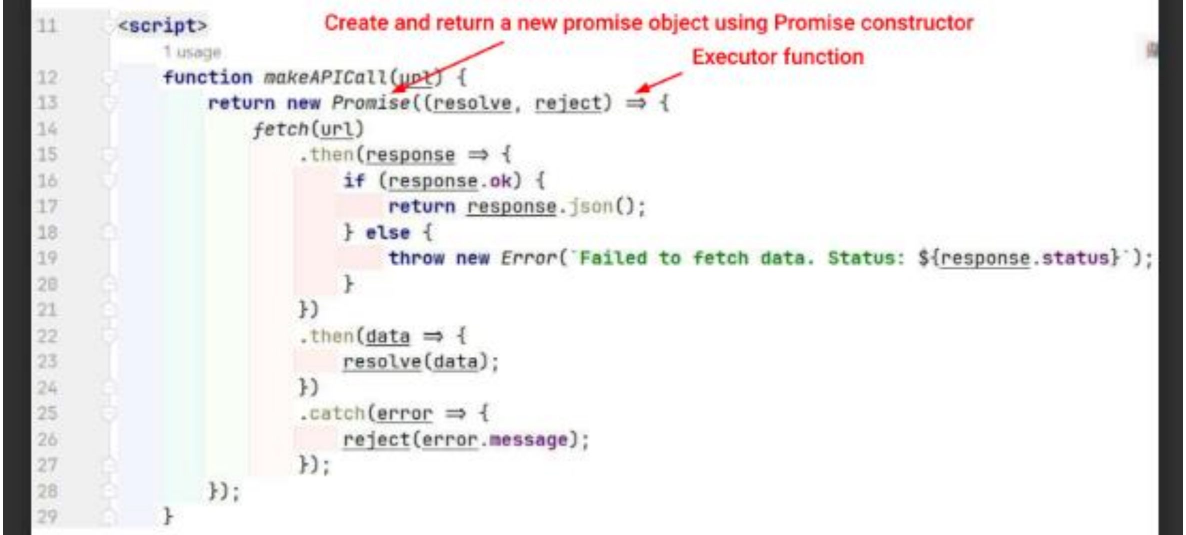


```
> fetch("https://www.google.com")
      .then(res => console.log(res.status))
      .catch(error => console.log(error))
      .finally(()=>{console.log("Promise Ended")});
< ▸ Promise {<pending>}
  200
  Promise Ended
```

Macro Tasks and Micro Tasks

- JavaScript engine is single threaded, so how does it make asynchronous calls?
  - API calls and setTimeout() or setInterval() are executed by browser and not by JavaScript.
- There are two queues that browser uses to inform to the JavaScript engine about the tasks given to them.
  - Macro tasks queue.
    - Used for setTimeout() or setInterval().
  - Micro tasks queue (Have higher priority)
    - Used for promises.

js-promise-1.html

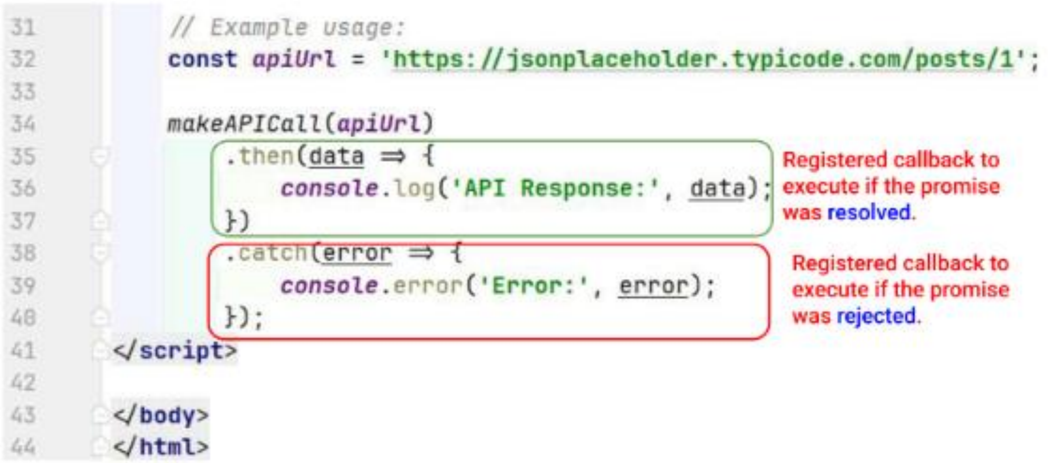


Create and return a new promise object using Promise constructor

Executor function

```
<script>
    1 usage
    function makeAPICall(url) {
        return new Promise((resolve, reject) => {
            fetch(url)
                .then(response => {
                    if (response.ok) {
                        return response.json();
                    } else {
                        throw new Error(`Failed to fetch data. Status: ${response.status}`);
                    }
                })
                .then(data => {
                    resolve(data);
                })
                .catch(error => {
                    reject(error.message);
                });
        });
    }
```

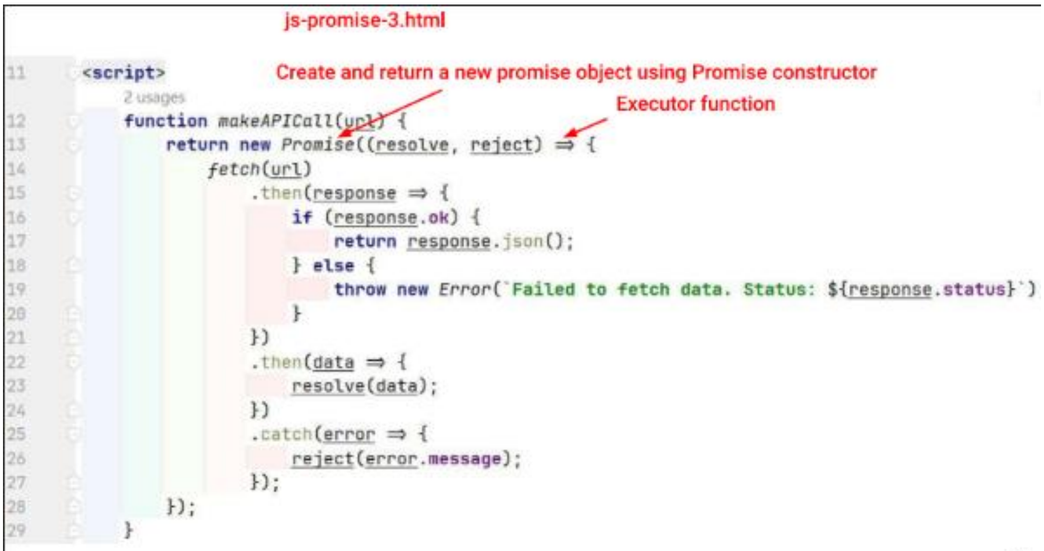
js-promise-1.html



```
    // Example usage:
    const apiUrl = 'https://jsonplaceholder.typicode.com/posts/1';

    makeAPICall(apiUrl)
        .then(data => {
            console.log('API Response:', data);
        })
        .catch(error => {
            console.error('Error:', error);
        });
</script>

</body>
</html>
```

Registered callback to execute if the promise was resolved.

Registered callback to execute if the promise was rejected.

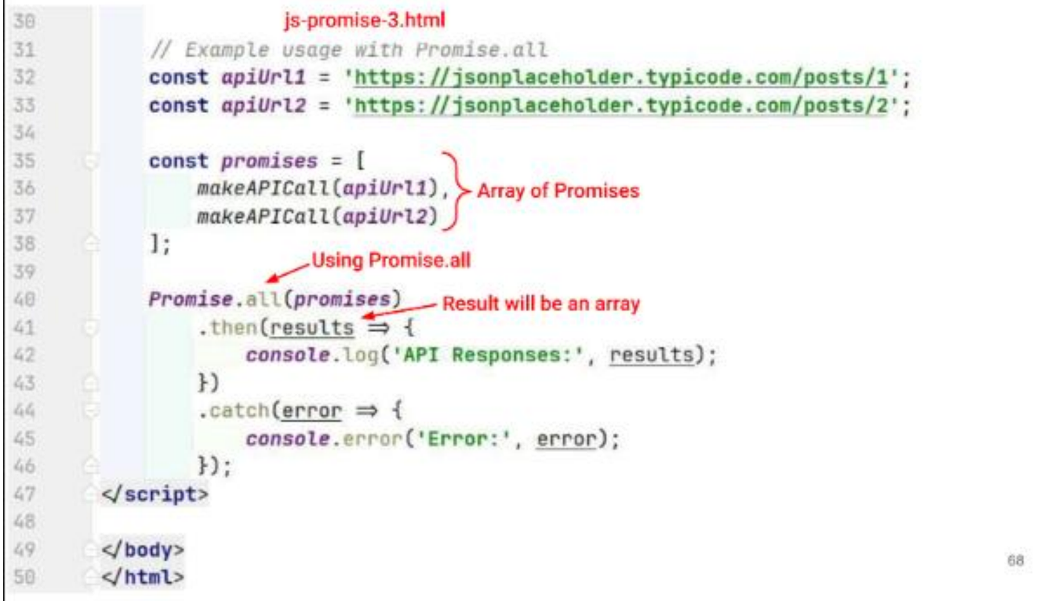


js-promise-3.html

Create and return a new promise object using Promise constructor

Executor function

```
<script>
    2 usages
    function makeAPICall(url) {
        return new Promise((resolve, reject) => {
            fetch(url)
                .then(response => {
                    if (response.ok) {
                        return response.json();
                    } else {
                        throw new Error(`Failed to fetch data. Status: ${response.status}`)
                    }
                })
                .then(data => {
                    resolve(data);
                })
                .catch(error => {
                    reject(error.message);
                });
        });
    }
```



js-promise-3.html

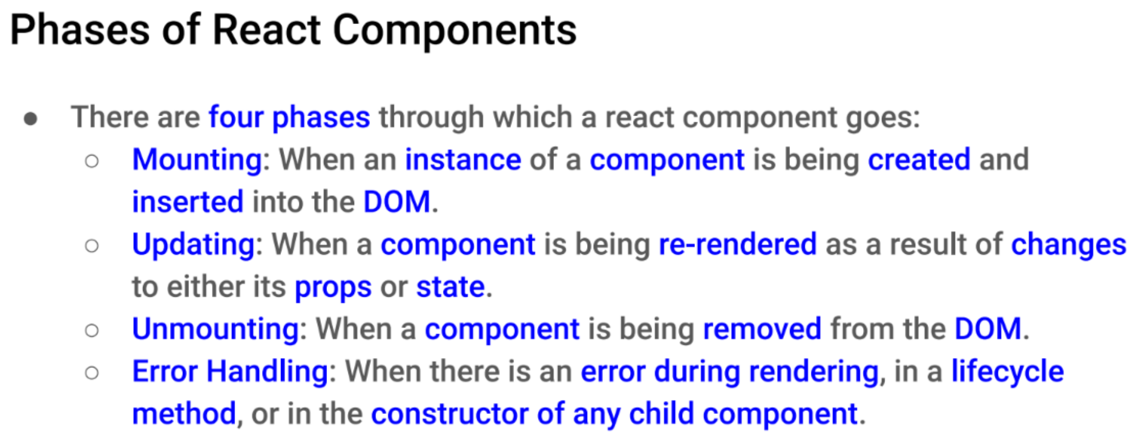
```
    // Example usage with Promise.all
    const apiUrl1 = 'https://jsonplaceholder.typicode.com/posts/1';
    const apiUrl2 = 'https://jsonplaceholder.typicode.com/posts/2';

    const promises = [
        makeAPICall(apiUrl1),
        makeAPICall(apiUrl2)
    ];

    Promise.all(promises)
        .then(results => {
            console.log('API Responses:', results);
        })
        .catch(error => {
            console.error('Error:', error);
        });
</script>

</body>
</html>
```

Array of Promises

Using Promise.all

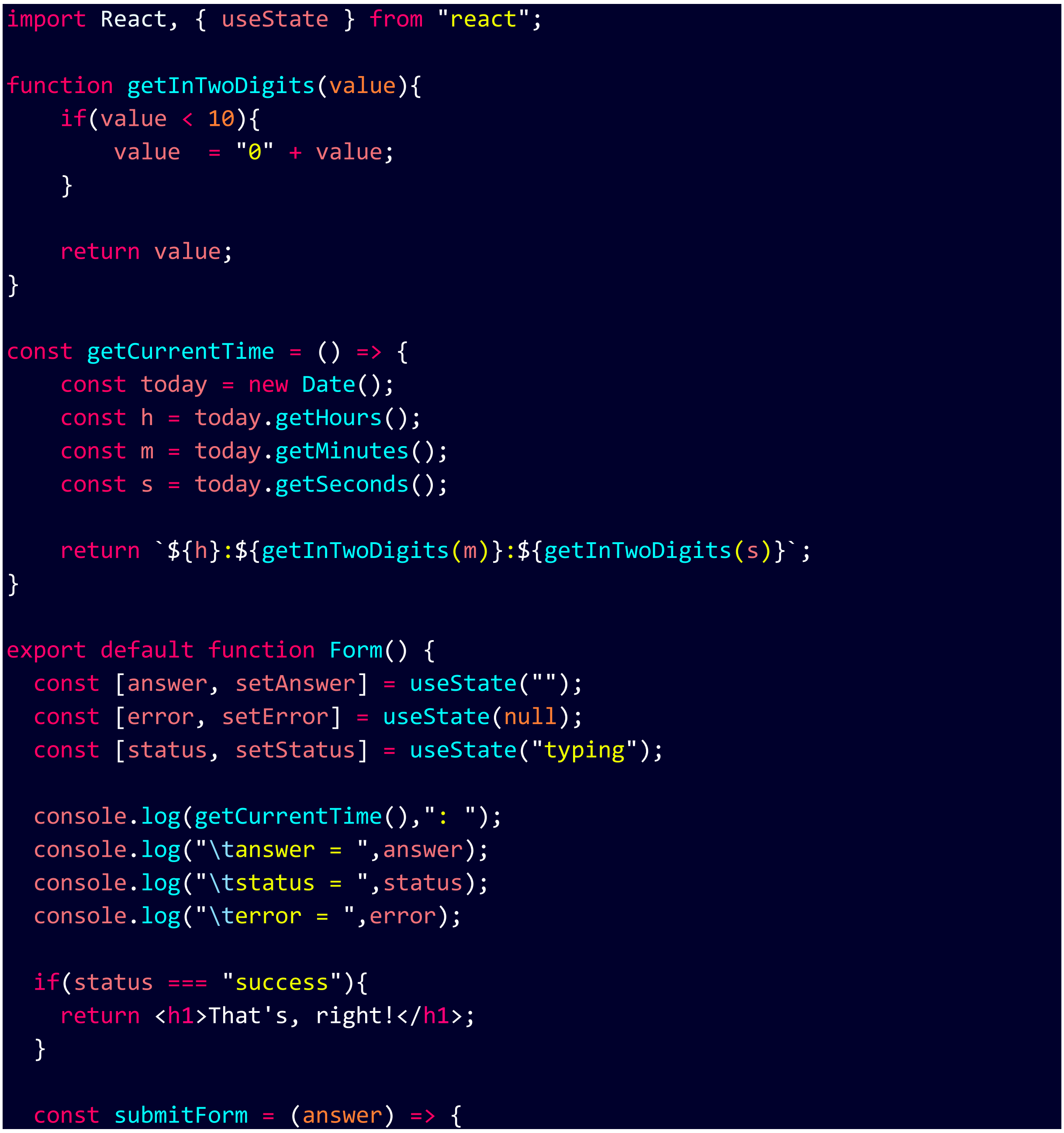
Result will be an array

React

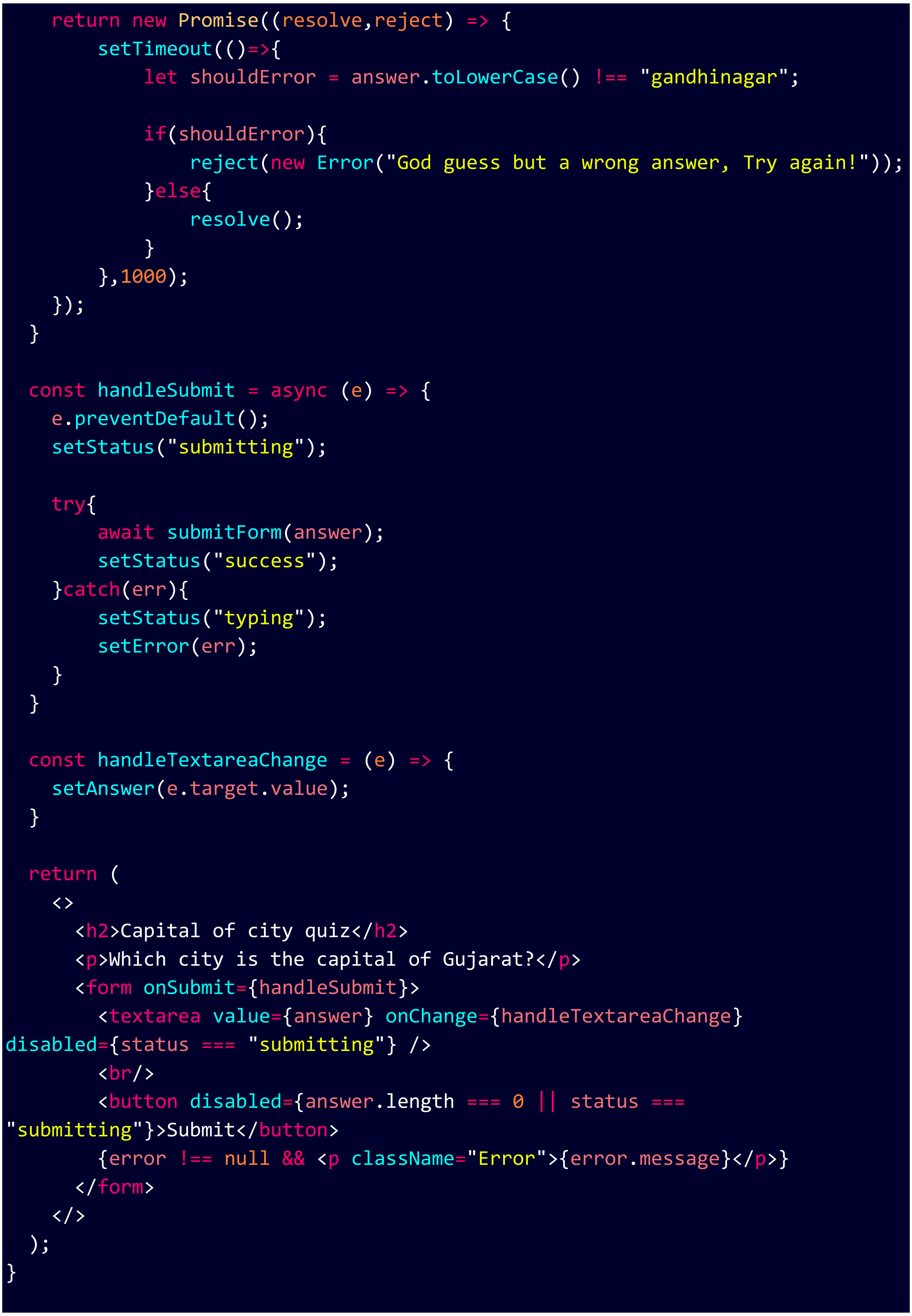


Phases of React Components

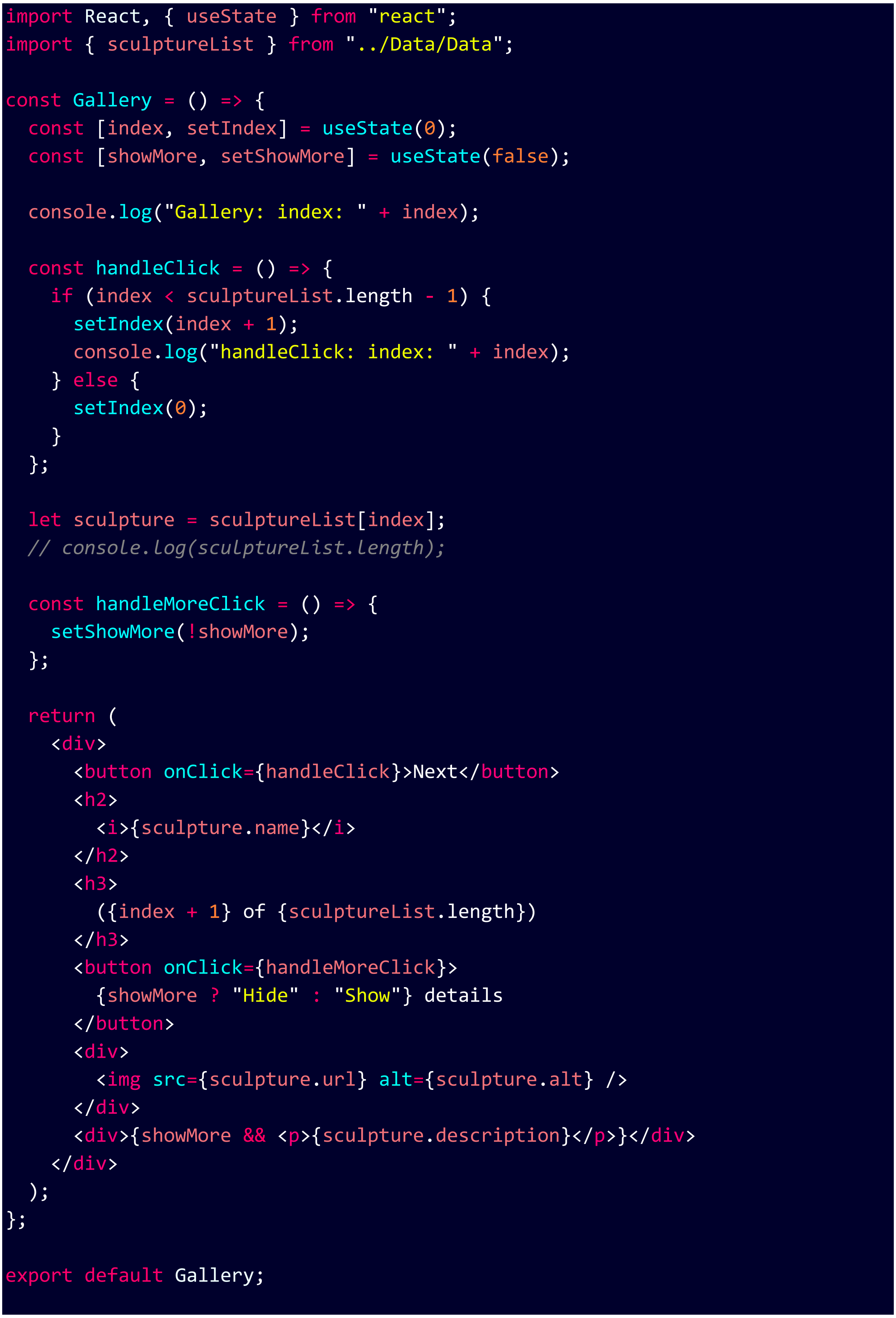
- There are four phases through which a react component goes:
  - Mounting: When an instance of a component is being created and inserted into the DOM.
  - Updating: When a component is being re-rendered as a result of changes to either its props or state.
  - Unmounting: When a component is being removed from the DOM.
  - Error Handling: When there is an error during rendering, in a lifecycle method, or in the constructor of any child component.



```
import React, { useState } from "react";

function getInTwoDigits(value){
    if(value < 10){
        value  = "0" + value;
    }

    return value;
}

const getCurrentTime = () => {
    const today = new Date();
    const h = today.getHours();
    const m = today.getMinutes();
    const s = today.getSeconds();

    return `${h}:${getInTwoDigits(m)}:${getInTwoDigits(s)}`;
}

export default function Form() {
  const [answer, setAnswer] = useState("");
  const [error, setError] = useState(null);
  const [status, setStatus] = useState("typing");

  console.log(getCurrentTime(),": ");
  console.log("\tanswer = ",answer);
  console.log("\tstatus = ",status);
  console.log("\terror = ",error);

  if(status === "success"){
    return <h1>That's, right!</h1>;
  }

  const submitForm = (answer) => {
```



```
  return new Promise((resolve,reject) => {
    setTimeout(()=>{
      let shouldError = answer.toLowerCase() !== "gandhinagar";

      if(shouldError){
        reject(new Error("God guess but a wrong answer, Try again!"));
      }else{
        resolve();
      }
    },1000);
  });
 }

 const handleSubmit = async (e) => {
  e.preventDefault();
  setStatus("submitting");

  try{
    await submitForm(answer);
    setStatus("success");
  }catch(err){
    setStatus("typing");
    setError(err);
  }
 }

 const handleTextareaChange = (e) => {
  setAnswer(e.target.value);
 }

 return (
  <>
   <h2>Capital of city quiz</h2>
   <p>Which city is the capital of Gujarat?</p>
   <form onSubmit={handleSubmit}>
    <textarea value={answer} onChange={handleTextareaChange}
disabled={status === "submitting"} />
    <br/>
    <button disabled={answer.length === 0 || status ===
"submitting"}>Submit</button>
    {error !== null && <p className="Error">{error.message}</p>}
   </form>
  </>
 );
}
```



```
import React, { useState } from "react";
import { sculptureList } from "../Data/Data";

const Gallery = () => {
  const [index, setIndex] = useState(0);
  const [showMore, setShowMore] = useState(false);

  console.log("Gallery: index: " + index);

  const handleClick = () => {
    if (index < sculptureList.length - 1) {
      setIndex(index + 1);
      console.log("handleClick: index: " + index);
    } else {
      setIndex(0);
    }
  };

  let sculpture = sculptureList[index];
  // console.log(sculptureList.length);

  const handleMoreClick = () => {
    setShowMore(!showMore);
  };

  return (
    <div>
      <button onClick={handleClick}>Next</button>
      <h2>
        <i>{sculpture.name}</i>
      </h2>
      <h3>
        ({index + 1} of {sculptureList.length})
      </h3>
      <button onClick={handleMoreClick}>
        {showMore ? "Hide" : "Show"} details
      </button>
      <div>
        <img src={sculpture.url} alt={sculpture.alt} />
      </div>
      <div>{showMore && <p>{sculpture.description}</p>}</div>
    </div>
  );
};

export default Gallery;
```