public class CalculateArea {

SquareService squareService;

RectangleService rectangleService;

CircleService circleService;

CalculateArea(SquareService squareService, RectangleService rectangeService, CircleService circleService)

{

this.squareService = squareService;

this.rectangleService = rectangeService;

this.circleService = circleService;

}

public Double calculateArea(Type type, Double... r )

{

switch (type)

{

case RECTANGLE:

if(r.length >=2)

return rectangleService.area(r[0],r[1]);

else

throw new RuntimeException("Missing required params");

case SQUARE:

if(r.length >=1)

return squareService.area(r[0]);

else

throw new RuntimeException("Missing required param");

case CIRCLE:

if(r.length >=1)

return circleService.area(r[0]);

else

throw new RuntimeException("Missing required param");

default:

throw new RuntimeException("Operation not supported");

}

}

}

public class SquareService {

public Double area(double r)

{

return r \* r;

}

}

public class RectangleService {

public Double area(Double r, Double h)

{

return r \* h;

}

}

public class CircleService {

public Double area(Double r)

{

return Math.PI \* r \* r;

}

}

public enum Type {

RECTANGLE,SQUARE,CIRCLE;

}

import org.junit.Assert;

import org.junit.Before;

import org.junit.Test;

import org.mockito.Mockito;

public class CalculateAreaTest {

RectangleService rectangleService;

SquareService squareService;

CircleService circleService;

CalculateArea calculateArea;

@Before

public void init()

{

rectangleService = Mockito.mock(RectangleService.class);

squareService = Mockito.mock(SquareService.class);

circleService = Mockito.mock(CircleService.class);

calculateArea = new CalculateArea(squareService,rectangleService,circleService);

}

@Test

public void calculateRectangleAreaTest()

{

Mockito.when(rectangleService.area(5.0d,4.0d)).thenReturn(20d);

Double calculatedArea = this.calculateArea.calculateArea(Type.RECTANGLE, 5.0d, 4.0d);

Assert.assertEquals(new Double(20d),calculatedArea);

}

}

@Component

public class CalculateArea {

SquareService squareService;

RectangleService rectangleService;

CircleService circleService;

public CalculateArea(@Autowired SquareService squareService, @Autowired RectangleService rectangeService, @Autowired CircleService circleService)

{

this.squareService = squareService;

this.rectangleService = rectangeService;

this.circleService = circleService;

}

public Double calculateArea(Type type, Double... r )

{

// (same implementation as before)

}

}

@Service

public class SquareService {

public Double area(double r)

{

return r\*r;

}

}

import org.springframework.stereotype.Service;

@Service

public class CircleService {

public Double area(Double r)

{

return Math.PI \* r \* r;

}

}

import org.springframework.stereotype.Service;

@Service

public class RectangleService {

public Double area(Double r, Double h)

{

return r\*h;

}

}

@Component

public class CalculateArea {

@Autowired

SquareService squareService;

@Autowired

RectangleService rectangleService;

@Autowired

CircleService circleService;

public Double calculateArea(Type type, Double... r )

{

// (same implementation as before)

}

}

public class CalculateAreaTest {

@Mock

RectangleService rectangleService;

@Mock

SquareService squareService;

@Mock

CircleService circleService;

@InjectMocks

CalculateArea calculateArea;

@Before

public void init()

{

MockitoAnnotations.initMocks(this);

}

@Test

public void calculateRectangleAreaTest()

{

Mockito.when(rectangleService.area(5.0d,4.0d)).thenReturn(20d);

Double calculatedArea = this.calculateArea.calculateArea(Type.RECTANGLE, 5.0d, 4.0d);

Assert.assertEquals(new Double(20d),calculatedArea);

}

}

@Service

public class RectangleService {

public Double area(Double r, Double h)

{

log();

return r\*h;

}

public void log() {

System.out.println("skip this");

}

}

@RunWith(MockitoJUnitRunner.class)

public class CalculateAreaTest {

@Spy

RectangleService rectangleService;

@Mock

SquareService squareService;

@Mock

CircleService circleService;

@InjectMocks

CalculateArea calculateArea;

@Test

public void calculateRectangleAreaTest()

{

Mockito.doCallRealMethod().when(rectangleService).log();

Mockito.when(rectangleService.area(5.0d,4.0d)).thenReturn(20d);

Double calculatedArea = this.calculateArea.calculateArea(Type.RECTANGLE, 5.0d, 4.0d);

Assert.assertEquals(new Double(20d),calculatedArea);

}

}

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.ResponseBody;

@Controller

public class AreaController {

@Autowired

CalculateArea calculateArea;

@RequestMapping(value = "api/area", method = RequestMethod.GET)

@ResponseBody

public ResponseEntity calculateArea(

@RequestParam("type") String type,

@RequestParam("param1") String param1,

@RequestParam(value = "param2", required = false) String param2

) {

try {

Double area = calculateArea.calculateArea(

Type.valueOf(type),

Double.parseDouble(param1),

Double.parseDouble(param2)

);

return new ResponseEntity(area, HttpStatus.OK);

}

catch (Exception e)

{

return new ResponseEntity(e.getCause(), HttpStatus.INTERNAL\_SERVER\_ERROR);

}

}

}

import org.junit.Assert;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.Mockito;

import org.mockito.junit.MockitoJUnitRunner;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

@RunWith(MockitoJUnitRunner.class)

public class AreaControllerTest {

@Mock

CalculateArea calculateArea;

@InjectMocks

AreaController areaController;

@Test

public void calculateAreaTest()

{

Mockito

.when(calculateArea.calculateArea(Type.RECTANGLE,5.0d, 4.0d))

.thenReturn(20d);

ResponseEntity responseEntity = areaController.calculateArea("RECTANGLE", "5", "4");

Assert.assertEquals(HttpStatus.OK,responseEntity.getStatusCode());

Assert.assertEquals(20d,responseEntity.getBody());

}

}

import org.junit.Before;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.Mockito;

import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import static org.springframework.test.web.servlet.setup.MockMvcBuilders.standaloneSetup;

@RunWith(SpringJUnit4ClassRunner.class)

public class AreaControllerTest {

@Mock

CalculateArea calculateArea;

@InjectMocks

AreaController areaController;

MockMvc mockMvc;

@Before

public void init()

{

mockMvc = standaloneSetup(areaController).build();

}

@Test

public void calculateAreaTest() throws Exception {

Mockito

.when(calculateArea.calculateArea(Type.RECTANGLE,5.0d, 4.0d))

.thenReturn(20d);

mockMvc.perform(

MockMvcRequestBuilders.get("/api/area?type=RECTANGLE&param1=5&param2=4")

)

.andExpect(status().isOk())

.andExpect(content().string("20.0"));

}

}

@Configuration

public class TestConfig {

@Bean

public AreaController areaController()

{

return new AreaController();

}

@Bean

public CalculateArea calculateArea()

{

return new CalculateArea();

}

@Bean

public RectangleService rectangleService()

{

return new RectangleService();

}

@Bean

public SquareService squareService()

{

return new SquareService();

}

@Bean

public CircleService circleService()

{

return new CircleService();

}

}

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import static org.springframework.test.web.servlet.setup.MockMvcBuilders.standaloneSetup;

@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration(classes = {TestConfig.class})

public class AreaControllerIntegrationTest {

@Autowired

AreaController areaController;

MockMvc mockMvc;

@Before

public void init()

{

mockMvc = standaloneSetup(areaController).build();

}

@Test

public void calculateAreaTest() throws Exception {

mockMvc.perform(

MockMvcRequestBuilders.get("/api/area?type=RECTANGLE&param1=5&param2=4")

)

.andExpect(status().isOk())

.andExpect(content().string("20.0"));

}

}