EXERCISE 1

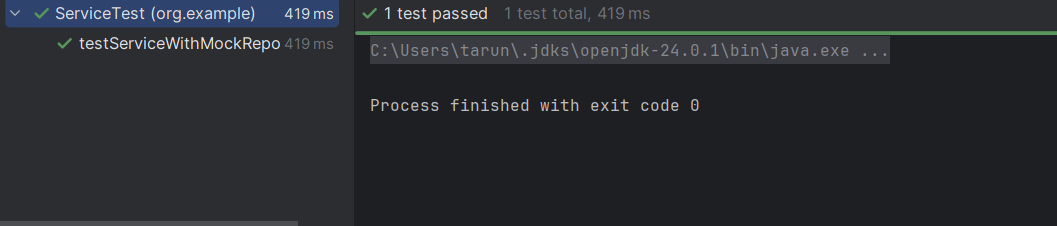
package org.example;  
  
public interface Repository {  
 String getdata();  
  
  
}

package org.example;  
  
public class Service {  
 private Repository repository;  
 public Service(Repository repository){  
 this.repository=repository;  
 }  
 public String processdata(){  
 String data=repository.getdata();  
 return "Processed "+data;  
 }  
  
  
}

TEST

package org.example;  
  
import org.junit.Test;  
  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.mockito.Mockito.\*;  
  
public class ServiceTest {  
 @Test  
 public void testServiceWithMockRepository(){  
 Repository mockRepository = *mock*(Repository.class);  
 *when*(mockRepository.getdata()).thenReturn("Mock Data");  
 Service service = new Service(mockRepository);  
 String result = service.processdata();  
 *assertEquals*("Processed Mock Data", result);  
 }  
}

OUTPUT



EXERCISE 2

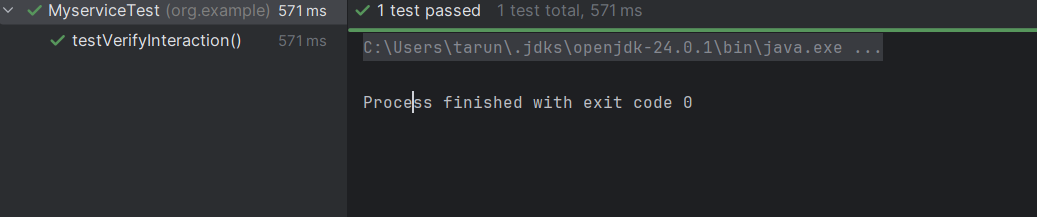
package org.example;  
  
public interface ExternalApi {  
 String getdata();  
}

package org.example;  
  
public class Myservice {  
 private ExternalApi externalApi;  
  
public Myservice(ExternalApi externalApi){  
 this.externalApi=externalApi;  
}  
public String fetchdata(){  
  
 return externalApi.getdata();  
}  
}

TEST

package org.example;  
  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
  
public class MyserviceTest {  
  
 @Test  
 public void testVerifyInteraction() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
  
 Myservice service = new Myservice(mockApi);  
 service.fetchdata();  
  
 *verify*(mockApi).getdata();  
 }  
}

OUTPUT



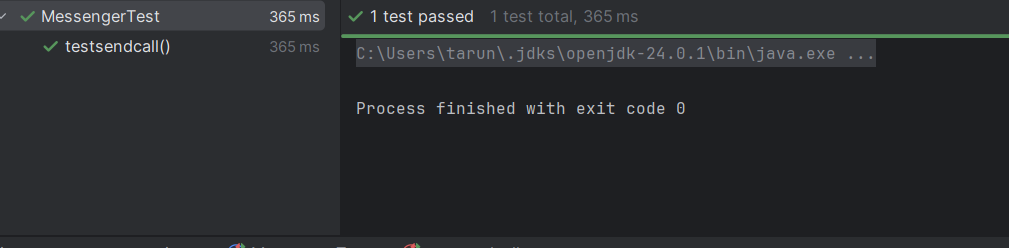
EXERCISE 3

package org.example;  
  
public interface NotificationService {  
 void send(String recipient,String message);  
}

package org.example;  
  
public class Messenger {  
 private NotificationService notificationService;  
 public Messenger(NotificationService notificationService){  
 this.notificationService=notificationService;  
 }  
 public void notifyUser(String user){  
 notificationService.send(user,"WELCOME"+user);  
 }  
}

import org.example.Messenger;  
import org.example.NotificationService;  
import org.junit.jupiter.api.Test;  
import org.mockito.ArgumentMatchers;  
  
import static org.mockito.ArgumentMatchers.*contains*;  
import static org.mockito.Mockito.*mock*;  
import static org.mockito.Mockito.*verify*;  
  
public class MessengerTest {  
 @Test  
 public void testsendcall(){  
 NotificationService service=*mock*(NotificationService.class);  
 Messenger messenger= new Messenger(service);  
 messenger.notifyUser("Tarun");  
 *verify*(service).send(ArgumentMatchers.*eq*("Tarun"),*contains*("WELCOME"));  
 }  
}

OUTPUT

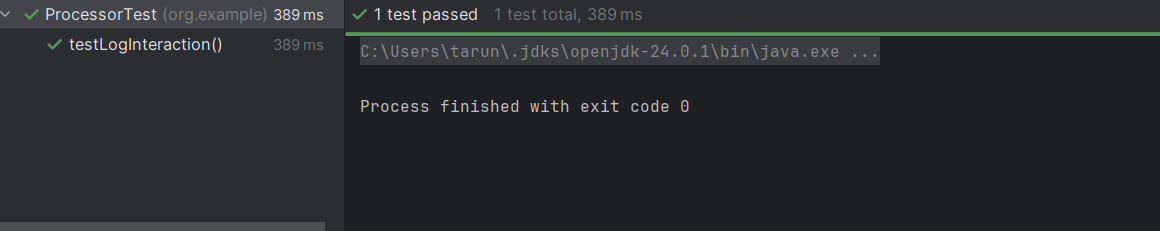


EXERCISE 4

package org.example;  
  
public interface Logger {  
 void log(String message);  
}

package org.example;  
public class Processor {  
 private final Logger logger;  
 public Processor(Logger logger) {  
 this.logger = logger;  
 }  
 public void process() {  
 logger.log("Processing complete");  
 }  
}

package org.example;  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
public class ProcessorTest {  
 @Test  
 public void testLogInteraction() {  
 Logger mockLogger = *mock*(Logger.class);  
 *doNothing*().when(mockLogger).log(*anyString*());  
 Processor processor = new Processor(mockLogger);  
 processor.process();  
 *verify*(mockLogger).log("Processing complete");  
 }  
}

OUTPUT

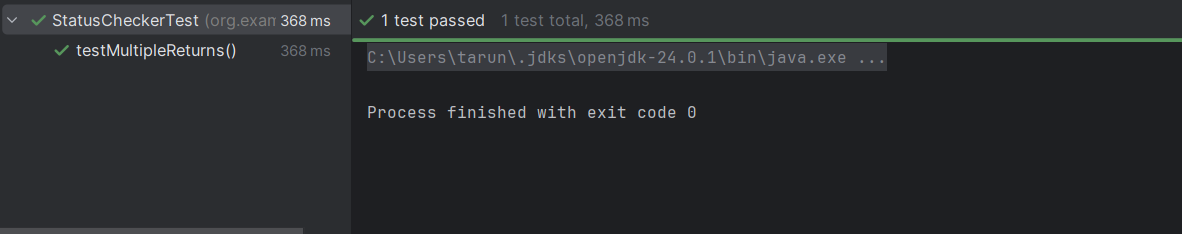
EXERCISE 5

package org.example;  
  
public interface ExternalApi {  
 String getStatus();  
}

package org.example;  
public class StatusChecker {  
 private final ExternalApi api;  
 public StatusChecker(ExternalApi api) {  
 this.api = api;  
 }  
 public String checkStatusTwice() {  
 return api.getStatus() + " & " + api.getStatus();  
 }  
}

package org.example;  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
import static org.mockito.Mockito.\*;  
public class StatusCheckerTest {  
 @Test  
 public void testMultipleReturns() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
 *when*(mockApi.getStatus())  
 .thenReturn("Online")  
 .thenReturn("Offline");  
 StatusChecker checker = new StatusChecker(mockApi);  
 String result = checker.checkStatusTwice();  
 *assertEquals*("Online & Offline", result);  
 }  
}

OUTPUT



EXERCISE 6

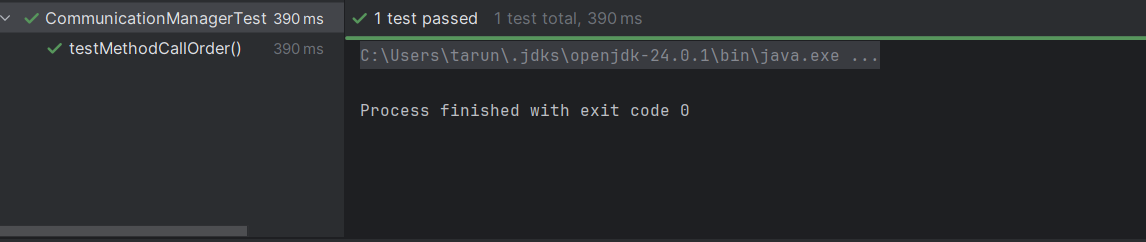
package org.example;  
  
public interface MessageService {  
 void connect();  
 void sendMessage(String message);  
 void disconnect();  
  
}

package org.example;  
public class CommunicationManager {  
 private final MessageService service;  
 public CommunicationManager(MessageService service) {  
 this.service = service;  
 }  
 public void performCommunication() {  
 service.connect();  
 service.sendMessage("Hello");  
 service.disconnect();  
 }  
}

package org.example;  
import org.junit.jupiter.api.Test;  
import org.mockito.InOrder;  
import static org.mockito.Mockito.\*;  
public class CommunicationManagerTest {  
 @Test  
 public void testMethodCallOrder() {  
 MessageService mockService = *mock*(MessageService.class);  
 CommunicationManager manager = new CommunicationManager(mockService);  
 manager.performCommunication();  
 InOrder inOrder = *inOrder*(mockService);  
 inOrder.verify(mockService).connect();  
 inOrder.verify(mockService).sendMessage("Hello");  
 inOrder.verify(mockService).disconnect();  
 }

}

OUTPUT



EXERCISE 7

package org.example;  
  
public interface ErrorLogger {  
 void log(String message);  
}

package org.example;  
public class ErrorReporter {  
 private final ErrorLogger logger;  
 public ErrorReporter(ErrorLogger logger) {  
 this.logger = logger;  
 }  
 public void report(String msg) {  
 logger.log(msg);  
 }  
}

package org.example;  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.*assertThrows*;  
import static org.mockito.Mockito.\*;  
public class ErrorReporterTest {  
 @Test  
 public void testVoidMethodThrowsException() {  
 ErrorLogger mockLogger = *mock*(ErrorLogger.class);  
 *doThrow*(new RuntimeException("Logging failed"))  
 .when(mockLogger).log("ERROR");  
 ErrorReporter reporter = new ErrorReporter(mockLogger);  
 *assertThrows*(RuntimeException.class, () -> {  
 reporter.report("ERROR");  
 });  
 *verify*(mockLogger).log("ERROR");  
 }  
}

OUTPUT

