Task 1.

```
import org.junit.jupiter.params. ParameterizedTest ;
import org.junit.jupiter.params.provider.Arguments ;
import org.junit.jupiter.params.provider. MethodSource ;
import java.util.stream.Stream ;
import static org.junit.jupiter.api.Assertions . assertEquals ;
public class CalculatorTest {
Calculator calculator = new Calculator();
@ParameterizedTest
@MethodSource ( "provideNumbersAdd" )
void shouldAddTwoNumbers ( double first, double second, double
result) {
assertEquals (result, calculator .add(first, second));
}
@ParameterizedTest
@MethodSource ( "provideNumbersSubtraction" )
void shouldSubtractionTwoNumbers ( double first, double second,
double result) {
assertEquals (result, calculator .subtraction(first, second));
}
@ParameterizedTest
@MethodSource ( "provideNumbersMultiply" )
void shouldMultiplyTwoNumbers ( double first, double second,
double result) {
assertEquals (result, calculator .multiply(first, second));
}
@ParameterizedTest
@MethodSource ( "provideNumbersDivide" )
void shouldDivideTwoNumbers ( double first, double second, double
result) {
assertEquals (result, calculator .divide(first, second));
}
private static Stream < Arguments > provideNumbersAdd () {
return Stream . of ( Arguments . of ( 1 , 2 , 3 ),
Arguments . of ( 3 , 4 , 7 ),
Arguments . of ( 5 , 6 , 11 ));
}
private static Stream < Arguments > provideNumbersSubtraction () {
return Stream . of ( Arguments . of ( 110 , 20 , 90 ),
Arguments . of ( 15 , 5 , 10 ),
Arguments . of ( 15 , 6 , 9 ));
}
private static Stream < Arguments > provideNumbersMultiply () {
return Stream . of ( Arguments . of ( 1 , 2 , 2 ),
Arguments . of ( 3 , 4 , 12 ),
Arguments . of ( 5 , 6 , 30 ));
}
private static Stream < Arguments > provideNumbersDivide () {
return Stream . of ( Arguments . of ( 12 , 4 , 3 ),
Arguments . of ( 30 , 6 , 5 ),
Arguments . of ( 5 , 1 , 5 ));
}
}
```

Task 2

```
void shouldThrowIllegalArgumentException () {
```

```
final IllegalArgumentException exception =
assertThrows ( IllegalArgumentException . class ,
() -> calculator .divide( 100 , 0 ));
assertThat ( exception ).hasMessage( "divide can't be 0" )
.hasNoCause();
}
```

Task 3

```
import org.junit.jupiter.api. Test ;
import org.junit.jupiter.api.extension. ExtendWith ;
import org.mockito. InjectMocks ;
import org.mockito. Mock ;
import org.mockito.junit.jupiter.MockitoExtension ;
import java.util.NoSuchElementException ;
import java.util.Optional ;
import static org.assertj.core.api.Assertions . assertThat ;
import static
org.assertj.core.api.Assertions . assertThatExceptionOfType ;
import static org.mockito.Mockito .*;
@ExtendWith ( MockitoExtension . class )
public class AnimalServiceTest {
private static final long ANIMAL_ID = 1L ;
private static final String ANIMAL_TYPE = "cat" ;
private static final Animal ANIMAL = new Animal( ANIMAL_ID ,
ANIMAL_TYPE , "Filemon" );
@Mock
private AnimalRepository animalRepository ;
@InjectMocks
private AnimalService animalService ;
@Test
void shouldGetAnimalById () {
when ( animalRepository .findById( ANIMAL_ID )).thenReturn( Optional
. of ( A
NIMAL ));
final Animal animal =
animalService .getAnimalById( ANIMAL_ID );
assertThat ( animal ).isEqualTo( ANIMAL );
verify ( animalRepository ).findById( ANIMAL_ID );
}
@Test
void shouldGetAnimalByType () {
when ( animalRepository .findByType( ANIMAL_TYPE )).thenReturn(
Optional .
of ( ANIMAL ));
final Animal animal =
animalService .getAnimalByType( ANIMAL_TYPE );
assertThat ( animal ).isEqualTo( ANIMAL );
verify ( animalRepository ).findByType( ANIMAL_TYPE );
}
@Test
void shouldCreateAnimal () {
when ( animalRepository .addAnimal( ANIMAL )).thenReturn( ANIMAL );
final Animal animal = animalService .create( ANIMAL );
assertThat ( animal ).isEqualTo( ANIMAL );
}
@Test
```

```
void shouldThrowWhenAnimalDoesNotExist () {
when ( animalRepository .findById( ANIMAL_ID )).thenReturn( Optional
. empt
y ());
assertThatExceptionOfType ( NoSuchElementException . class )
.isThrownBy(() ->
animalService .getAnimalById( ANIMAL_ID ));
verify ( animalRepository ).findById( ANIMAL_ID );
verifyNoMoreInteractions ( animalRepository );
}
}
```

Task 4

```
import java.io.File ;
import java.io.FileNotFoundException ;
import java.util.Scanner ;
public class FileReader {
public int counterCharacters ( final File file) {
int counter = 0 ;
String extension = "" ;
if (file != null && file.exists()) {
String name = file.getName();
extension = name .substring( name .lastIndexOf( "." ));
if (! ".csv" .equalsIgnoreCase(extension)) {
throw new IllegalArgumentException( "wrong
extension" );
}
} else {
throw new IllegalArgumentException( "no found file" );
}
try ( Scanner scanner = new Scanner(file)) {
while ( scanner .hasNextLine()) {
counter += scanner .nextLine().length();
}
} catch ( FileNotFoundException e) {
e.printStackTrace();
}
return counter;
}
}
import org.junit.jupiter.api. Test ;
import java.io.File ;
import static org.assertj.core.api.Assertions . assertThat ;
import static org.junit.jupiter.api.Assertions . assertEquals ;
import static org.junit.jupiter.api.Assertions . assertThrows ;
public class FileReaderTest {
FileReader fileReader = new FileReader();
@Test
void shouldCounterCharactersExpected () {
assertEquals ( 10 , fileReader .counterCharacters( new
File( "book.csv" )));
}
@Test
void shouldThrowWhenFileDoesNotExist () {
final IllegalArgumentException exception =
assertThrows ( IllegalArgumentException . class ,
```

```
() -> fileReader .counterCharacters( new
File( "book2.csv" )));
assertThat ( exception ).hasMessage( "no found file" )
.hasNoCause();
}
@Test
void shouldThrowWhenFileHasWrongExtension () {
final IllegalArgumentException exception =
assertThrows ( IllegalArgumentException . class ,
() -> fileReader .counterCharacters( new
File( "book.txt" )));
assertThat ( exception ).hasMessage( "wrong extension" )
.hasNoCause();
}
}
```

Task 5

```
import lombok. AllArgsConstructor ;
import lombok. Data ;
import lombok. NoArgsConstructor ;
@Data
@NoArgsConstructor
@AllArgsConstructor
public class User {
private String name ;
private String surname ;
private String email ;
public boolean validName () {
return this . name .toLowerCase().matches( "([a - z]+)" );
}
public boolean validSurname () {
return this . surname .toLowerCase().matches( "([a - z/-]+)" );
}
public boolean validEmail () {
return
this . email .toLowerCase().matches( "([a - z0 - 9+_.-]+@(.+))" );
}
public boolean valid () {
if (! this .validName()) {
throw new IllegalArgumentException( "bad name" );
}
if (! this .validSurname()) {
throw new IllegalArgumentException( "bad surname" );
}
if (! this .validEmail()) {
throw new IllegalArgumentException( "bad email" );
}
return true ;
}
}
import org.junit.jupiter.params. ParameterizedTest ;
import org.junit.jupiter.params.provider. CsvFileSource ;
import static org.assertj.core.api.Assertions . assertThat ;
import static org.junit.jupiter.api.Assertions .*;
public class UserTest {
@ParameterizedTest
```

```
@CsvFileSource (resources = "/email.txt" , delimiterString = ";" )
void shouldValidUser ( String input) {
String [] words = input.split( "," );
User user = new User( words [ 0 ], words [ 1 ], words [ 2 ]);
try {
assertTrue ( user .valid());
} catch ( IllegalArgumentException e) {
final IllegalArgumentException exception =
assertThrows ( IllegalArgumentException . class ,
user ::valid);
assertThat ( exception ).hasNoCause();
}
}
}
```