**Calculate Factorial**

**Objectives :**  
     To get experience on Number Equality using JUnit Testing Library.  
**Functional Requirements:**  
Suriya and Manikandan are best friends, they try to play a game on words. If a word is given to them, they need to find as many words as possible using the characters in the given word. While playing, they taught to find how many numbers of combinations are possible. But, they find it difficult to get it. They knew they can get the combinations by a factorial. Can you write a function to determine the total number of combinations they can generate?  
**FactorialBO** class contains the following method.

|  |  |
| --- | --- |
| **Method** | **Method Description** |
| long calculateFactorial(int n) | This method accepts an integer 'n' as an argument and returns the factorial of the number 'n'. |

**Note :**  
The code for implementing the above requirement is provided as part of the code template.  
Jars for JUnit are available in the link - [**JUNIT JARS**](http://app.e-box.co.in/uploads/JUnit_Jars.zip)  
**Test Specification :**  
Create a class named **FactorialJUnit** and include the following test methods to test the **calculateFactorial** method in the above code specifications.

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Method Name** | **Method Description** |
| Test case 1 | testFactorial | This method is used to test the factorial values. |

Write a **createBoInstance** method used to create the object for FactorialBO class

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class FactorialJUnit {

FactorialBO fb;

@Before

public void createBoInstance() {

fb=new FactorialBO();

}

@Test

public void testFactorial() {

assertEquals(120,fb.calculateFactorial(5));

}

}

public class FactorialBO {

public long calculateFactorial(int n) {

long val = 1;

for(int i=1;i<=n;i++) {

val\*=i;

}

return val;

}

}

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader buff = new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter the number:");

Integer n = Integer.parseInt(buff.readLine());

System.out.println("The factorial of "+n+" is "+new FactorialBO().calculateFactorial(n));

}

}

**Email domain validation**

**Objectives :**  
     To get experience @Test and @Before annotation using JUnit Testing Library.  
  
**Functional Requirements:**  
The fair organizers have listed the accepted domains as "com", "in", "net", and "org".  
Write a program to validate the email addresses that have the above listed domain names.  
  
**Example:**  
Valid Domain mail: eboxuser@ebox.com  
Invalid Domain mail: eboxuser@ebox.edu  
  
**DomainValidationBO** class contains the following method.

|  |  |
| --- | --- |
| **Method** | **Method Description** |
| String validateMailDomain(String mail) | This method accepts an email id as argument and returns "**Valid email address**" if the mail id is valid else return "**Invalid email address**". |

**Note :**  
The code for implementing the above requirement is provided as part of the code template.  
Jars for JUnit are available in the link - [**JUNIT JARS**](http://app.e-box.co.in/uploads/JUnit_Jars.zip)  
  
**Test Specification :**  
Create a class named **DomainValidationJUnit** and include the following test methods to test the **validateMailDomain** method in the above code specifications.

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Method Name** | **Method Description** |
| Test case 1 | testValidDomain | This method is used to test valid mail id scenarios. |
| Test case 2 | testInvalidDomain | This method is used to test invalid mail id scenarios. |

Write a **createBoInstance** method used to create the object for DomainValidationBO class

public class DomainValidationBO {

public String validateMailDomain(String mail) {

String domain = (mail.substring(mail.lastIndexOf('.') + 1));

if(domain.equals("com") || domain.equals("in") || domain.equals("org") || domain.equals("net"))

return "Valid email address";

return "Invalid email address";

}

}

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter the email address");

String mail = br.readLine();

System.out.println(new DomainValidationBO().validateMailDomain(mail));

}

}

import static org.junit.Assert.assertEquals;

import org.junit.Before;

import org.junit.Test;

public class DomainValidationJUnit {

DomainValidationBO domainValidation;

@Before

public void createBoInstance() {

domainValidation = new DomainValidationBO();

}

@Test

public void testValidDomain() {

assertEquals("Valid email address", domainValidation.validateMailDomain("eboxuser@ebox.com"));

}

@Test

public void testInvalidDomain() {

assertEquals("Invalid email address", domainValidation.validateMailDomain("eboxuser@ebox.edu"));

}

}

**Arithmetic Exception**

**Objectives:**  
     To get experience in Testing Exceptions using JUnit Testing Library.  
  
**Functional Requirements:**  
An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e at runtime, it disrupts the normal flow of the program. For example, there are 10 statements in your program and there occurs an exception at statement 5, rest of the code will not be executed i.e. statement 6 to 10 will not run. If we perform exception handling, rest of the statement will be executed. That is why we use exception handling.  
  
For practice in exception handling, obtain the cost for 'n' days of an item and n as input and calculate the cost per day for the item. In case, zero is given as input for n, an arithmetic exception is thrown, handle the exception and prompt the user accordingly (Refer sample I/O).  
  
**CalculateBO** class contains the following method.

|  |  |
| --- | --- |
| **Method** | **Method Description** |
| double calculateCost(Integer cost, Integer days) | This method will get the cost and days, then calculate the cost per day return the value(cost per day = cost/days). |

**Note :**  
The code for implementing the above requirement is provided as part of the code template.  
Jars for JUnit are available in the link - [**JUNIT JARS**](http://e-box.co.in/uploads/JUnit_Jars.zip)  
**Test Specification :**  
Create a class named **CalculateJUnit** and include the following test methods to test the **calculateCost** method in the above code specifications.

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Method Name** | **Method Description** |
| Test case 1 | testCalculateCost | This method is used to test the cost per day returned by the calculateCost method. |
| Test case 2 | testCalculateCostException | This method is used to test the ArithmeticException case, whether the method is throwing ArithmeticException in days given as 0. |

Write a **createBoInstance** method used to create the object for CalculateBO class

import static org.junit.Assert.assertEquals;

import org.junit.Before;

import org.junit.Test;

public class CalculateJUnit {

@Before

public void createBoInstance() {

//fill the code

CalculateBO cbo=new CalculateBO();

}

//@SuppressWarnings("deprecation")

@Test

public void testCalculateCost() {

CalculateBO cbo=new CalculateBO();

assertEquals(20.0,cbo.calculateCost(100, 5),0.0);

//fill the code

}

@Test(expected=ArithmeticException.class)

public void testCalculateCostException() {

//fill the code

CalculateBO cbo=new CalculateBO();

cbo.calculateCost(10, 0);

//assertEquals(message,m);

}

}

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main {

public static void main(String[] args)throws IOException {

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter the cost of the item for n days");

Integer cost=Integer.parseInt(br.readLine());

System.out.println("Enter the value of n");

Integer n=Integer.parseInt(br.readLine());

double costPerDay;

try{

costPerDay = new CalculateBO().calculateCost(cost, n);

System.out.println("Cost per day of the item is "+costPerDay);

}

catch(ArithmeticException e){

System.out.println(e);

}

}

}

public class CalculateBO {

public double calculateCost(Integer cost, Integer days) throws ArithmeticException {

return (cost/days);

}

}

**Date Formats**

**Objectives:**  
     To get experience on equalTo in Basic assertThat using JUnit Testing Library.  
  
**Functional Requirements:**  
SimpleDateFormat in Java can be used to convert String to Date in Java. java.text.SimpleDateFormat is an implementation of DateFormat which defines a date pattern and can convert a particular String which follows that pattern into Date in Java. Write a program to convert the dates given by the user into different formats.  
  
Create following methods in **DateFormatBO** class,

|  |  |
| --- | --- |
| **Method** | **Method Description** |
| String convertToFormat1(String value) | This method accepts a String with "MM-dd-yyyy" format and converts it into a String of "EEE, MMM d, yy" format and returns the value. |
| String convertToFormat2(String value) | This method accepts a String with "MM-dd-yyyy" format and converts it into a String of "dd.MM.yyyy" format and returns the value. |
| String convertToFormat3(String value) | This method accepts a String with "MM-dd-yyyy" format and converts it into a String of "dd/MM/yyyy" format and returns the value. |

**Note :**  
The code for implementing the above requirement is provided as part of the code template.  
Jars for JUnit are available in the link - [**JUNIT JARS**](http://e-box.co.in/uploads/JUnit_Jars.zip)

import static org.junit.Assert.assertThat;

import org.hamcrest.CoreMatchers;

import org.junit.Test;

public class DateFormatJUnit {

@Test

public void createBoInstance() {

//fill the code

DateFormatBO dbo1=new DateFormatBO();

}

@Test

public void testConvertToFormat1() {

DateFormatBO dbo1=new DateFormatBO();

assertThat("Thu, Dec 12, 91",CoreMatchers.equalTo(dbo1.convertToFormat1("12-12-1991")));

// assertThat("12.12.1991",equalsTo(dbo1.convertToFormat1("12-12-1991")))

}

@Test

public void testConvertToFormat2() {

//fill the code

DateFormatBO dbo1=new DateFormatBO();

assertThat("12.12.1991",CoreMatchers.equalTo(dbo1.convertToFormat2("12-12-1991")));

}

@Test

public void testConvertToFormat3() {

//fill the code

DateFormatBO dbo1=new DateFormatBO();

assertThat("12/12/1991",CoreMatchers.equalTo(dbo1.convertToFormat3("12-12-1991")));

}

}

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.text.ParseException;

public class Main {

public static void main(String []args) throws IOException, ParseException {

BufferedReader buff = new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter the date to be formatted:(MM-dd-yyyy)");

String value = buff.readLine();

DateFormatBO boIns = new DateFormatBO();

System.out.println("Date Format with EEE, MMM d, yy : " + boIns.convertToFormat1(value));

System.out.println("Date Format with dd.MM.yyyy : " + boIns.convertToFormat2(value));

System.out.println("Date Format with dd dd/MM/yyyy : " + boIns.convertToFormat3(value));

}

}

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class DateFormatBO {

public String convertToFormat1(String value) {

SimpleDateFormat s1 = new SimpleDateFormat("MM-dd-yyyy");

SimpleDateFormat s2 = new SimpleDateFormat("EEE, MMM d, yy");

Date d1;

try { d1 = s1.parse(value); }

catch (ParseException e) { d1 = new Date(); }

return s2.format(d1);

}

public String convertToFormat2(String value) {

SimpleDateFormat s1 = new SimpleDateFormat("MM-dd-yyyy");

SimpleDateFormat s2 = new SimpleDateFormat("dd.MM.yyyy");

Date d1;

try { d1 = s1.parse(value); }

catch (ParseException e) { d1 = new Date(); }

return s2.format(d1);

}

public String convertToFormat3(String value) {

SimpleDateFormat s1 = new SimpleDateFormat("MM-dd-yyyy");

SimpleDateFormat s2 = new SimpleDateFormat("dd/MM/yyyy");

Date d1;

try { d1 = s1.parse(value); }

catch (ParseException e) { d1 = new Date(); }

return s2.format(d1);

}

}

**Finding Square**

**Objectives:**  
     To get experience in Custom assertThat using JUnit Testing Library.  
  
**Functional Requirements:**  
Write a  program to accept an integer as argument and print the square of that integer.  
  
**SquareBO** class contains the following method.

|  |  |
| --- | --- |
| **Method** | **Method Description** |
| Integer findSquareValue(Integer n) | This method will get an Integer as argument and return the square of the Integer. |

**Note :**  
The code for implementing the above requirement is provided as part of the code template.  
Jars for JUnit are available in the link - [**JUNIT JARS**](http://e-box.co.in/uploads/JUnit_Jars.zip)  
  
**Test Specification :**  
Create a class named **SquareJUnit** and include the following test methods to test the **findSquareValue** method in the above code specifications.

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Method Name** | **Method Description** |
| Test case 1 | testFindSquareValue | This method is used to test the square of the Integer. |

Write a **createBoInstance** method used to create the object for SquareBO class  
  
Create  a class named **SquareChecker** extending **TypeSafeMatcher**, override the appropriate methods and display the custom error message as shown below if the computed square is wrong.  
java.lang.AssertionError:  Expected: Expected was: <10> but:  was <11>

import org.junit.Before;

import static org.junit.Assert.\*;

import static org.hamcrest.CoreMatchers.\*;

import org.junit.Before;

import org.junit.Test;

public class SquareJUnit {

SquareBO sb;

@Before

public void createBoInstance() {

sb=new SquareBO();

}

@Test

public void testFindSquareValue() {

sb=new SquareBO;

assertThat(9,SquareChecker.checkSquare(9))

}

}

class SquareChecker extends TypeSafeMatcher

{

public static Matcher<Integer> checkSquare(final Integer value) {

return new TypeSafeMatcher<Integer>() {

@Override

public void describeTo(Description description) {

description.appendText(""+value+"");

}

protected boolean matchesSafely(Integer n) {

return (value==n);

}

public void describeMismatchSafely(final Integer n, final Description mismatchDescription) {

mismatchDescription.appendText("java.lang.AssertionError: Expected:"

+ "Expected was: <"+n+"> but: was <" + (n+1)+ ">");

}

};

}

}

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main {

public static void main(String args[]) throws IOException {

BufferedReader buff = new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter the number:");

Integer n = Integer.parseInt(buff.readLine());

System.out.println(new SquareBO().findSquareValue(n));

}

}

public class SquareBO {

public Integer findSquareValue(Integer n) {

return n\*n;

}

}