**Junit:**

**1.\*\*\*\*\*\*\*\*ArithmeticException\*\*\*\*\*\*\*\*:**

**Main.java**

Path: Junit\ArithmeticException\src\Main.java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main {

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

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

System.out.println("Enter the container price :");

int contPrice = Integer.parseInt(br.readLine());

System.out.println("Enter the number of items in the container :");

int noOfItem = Integer.parseInt(br.readLine());

System.out.println("The average price of the item is Rs." + new ShipmentBO().calculateAveragePrice(contPrice, noOfItem));

}

}

**ShipmentBO.java**

Path: Junit\ArithmeticException\src\ShipmentBO.java

public class ShipmentBO {

public double calculateAveragePrice(int contPrice,int noOfItem) {

int avgPrice = 0;

avgPrice = contPrice / noOfItem;

return avgPrice;

}

}

**ShipmentJunit.java**

Path:Junit\ArithmeticException\test\ShipmentJunit.java

import static org.junit.Assert.\*;

import org.junit.After;

import org.junit.Before;

import org.junit.Rule;

import org.junit.Test;

import org.junit.rules.ExpectedException;

public class ShipmentJunit {

ShipmentBO shipmentBO;

@Before

public void init() {

shipmentBO = new ShipmentBO();

}

@Test

public void testCalculatAverageWithoutZero() {

assertTrue(5==shipmentBO.calculateAveragePrice(10, 2));

assertTrue(20==shipmentBO.calculateAveragePrice(100, 5));

}

@Test(expected = ArithmeticException.class)

public void testCalculateAverageWithZero() {

assertTrue(5==shipmentBO.calculateAveragePrice(10,0));

}

@After

public void destroy() {

shipmentBO = null;

}

}

**2.\*\*\*\*\*\*\*\* Averageweightofitemsincontainer\*\*\*\*\*\*\***

**Main.Java**

Path: Junit\Averageweightofitemsincontainer\src\Main.java

import java.util.Scanner;

public class Main {

public static void main(String args[]) {

Scanner s = new Scanner(System.in);

System.out.println("Enter n1");

int n1 = s.nextInt();

System.out.println("Enter the weight of n1");

double wt1 = s.nextDouble();

System.out.println("Enter n2");

int n2 = s.nextInt();

System.out.println("Enter the weight of n2");

double wt2 = s.nextDouble();

ShipmentBO shipmentBO = new ShipmentBO();

double average = shipmentBO.calculateAverage(n1, wt1, n2, wt2);

System.out.printf("Average weight : %.2f", average);

}

}

**Main1.java**

Path: Junit\Averageweightofitemsincontainer\src\Main1.java

import java.util.Scanner;

import java.util.Arrays;

import java.util.ArrayList;

public class Main1 {

int ports\_num;

static int i, j, x, y, a, k;

public static void main(String args[]) {

Scanner scan = new Scanner(System.in);

System.out.println("Enter number of ports :");

int ports\_num = Integer.parseInt(scan.nextLine());

String[] data1 = new String[ports\_num];

if (ports\_num <= 0) {

System.out.println("Not entered port number accurately");

} else {

System.out.println("Enter port details :");

for (i = 0; i < ports\_num; i++) {

data1[i] = scan.nextLine();

}

}

for (k = 0; k < ports\_num; k++) {

ArrayList aList = new ArrayList(

Arrays.asList(data1[k].split("\\|")));

x = 0;

a = 0;

j = 2;

// while (j < aList.size()) {

// if (aList.get(j) == 1) {

// a = a + 1;

// }

// j = j + 1;

// }

System.out.println("One mode of transportation :");

if (a > 0 && a < 2) {

System.out.println(aList.get(0) + " " + aList.get(1));

} else {

System.out.println("No such transportation available");

}

System.out.println("More than one mode of transportation :");

if (a > 1 && a < 4) {

System.out.println(aList.get(0) + " " + aList.get(1));

} else {

System.out.println("No such transportation available");

}

}

if (scan != null)

scan.close();

}

}

**ShipmentBO.java:**

Path: Junit\Averageweightofitemsincontainer\src\ShipmentBO.java

public class ShipmentBO {

public double calculateAverage(int n1, double wt1, int n2, double wt2) {

if (n1 + n2 == 0)

return 0;

return ((n1 \* wt1) + (n2 \* wt2)) / (n1 + n2);

}

}

**ShipmentJunit.java:**

Path: Junit\Averageweightofitemsincontainer\test

import static org.junit.Assert.assertEquals;

import static org.junit.Assert.assertFalse;

import static org.junit.Assert.assertNotNull;

import static org.junit.Assert.assertNull;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

public class ShipmentJUnit {

double delta = 0.01;

ShipmentBO shipment;

@Before

public void setup() {

shipment = new ShipmentBO();

}

@Test

public void testCalculateAverage() {

assertEquals(0, shipment.calculateAverage(0, 0, 0, 0),0);

assertEquals(42.5, shipment.calculateAverage(1, 20, 3, 50),0);

}

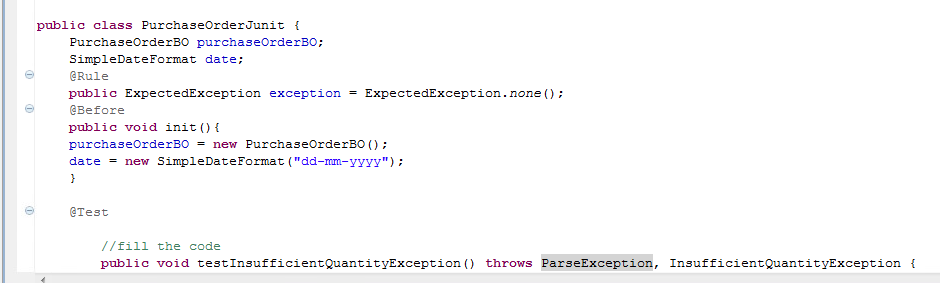
}

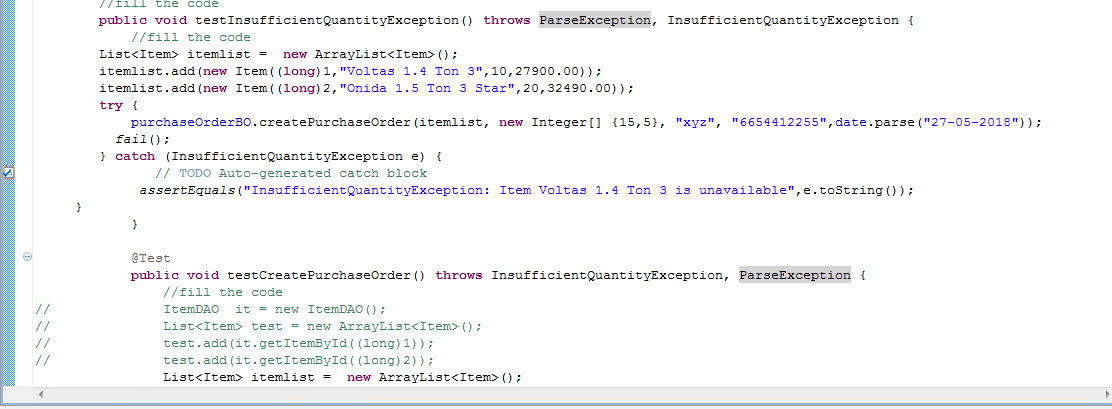
**3.\*\*\*\*\*\*** **Build a shipment report using List\*\*\*\*\*:**

**4.\*\*\*\*\*\*** **CargoObjectEquality\*\*\*\*\*:**

**5. \* Purchase order :**

import java.util.ArrayList;import java.util.List;import org.junit.Test;import org.junit.rules.ExpectedException;import static org.junit.Assert.assertEquals;import static org.junit.Assert.\*;import java.text.ParseException;import java.text.SimpleDateFormat;import static org.hamcrest.Matchers.contains;import static org.hamcrest.CoreMatchers.is;import static org.hamcrest.Matchers.hasProperty;import org.junit.Before;import org.junit.Rule;









import java.util.ArrayList;

import java.util.List;

import org.junit.Test;

import org.junit.rules.ExpectedException;

import static org.junit.Assert.assertEquals;

import static org.junit.Assert.\*;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import static org.hamcrest.Matchers.contains;

import static org.hamcrest.CoreMatchers.is;

import static org.hamcrest.Matchers.hasProperty;

import org.junit.Before;

import org.junit.Rule;

public class PurchaseOrderJunit {

PurchaseOrderBO purchaseOrderBO;

SimpleDateFormat date;

@Rule

public ExpectedException exception = ExpectedException.none();

@Before

public void init(){

purchaseOrderBO = new PurchaseOrderBO();

date = new SimpleDateFormat("dd-mm-yyyy");

}

@Test

//fill the code

public void testInsufficientQuantityException() throws ParseException, InsufficientQuantityException {

//fill the code

List<Item> itemlist =  new ArrayList<Item>();

        itemlist.add(new Item((long)1,"Voltas 1.4 Ton 3",10,27900.00));

        itemlist.add(new Item((long)2,"Onida 1.5 Ton 3 Star",20,32490.00));

        try {

        purchaseOrderBO.createPurchaseOrder(itemlist, new Integer[] {15,5}, "xyz", "6654412255",date.parse("27-05-2018"));

          fail();

        } catch (InsufficientQuantityException e) {

               // TODO Auto-generated catch block

             assertEquals("InsufficientQuantityException: Item Voltas 1.4 Ton 3 is unavailable",e.toString());

     }

}

@Test

public void testCreatePurchaseOrder() throws InsufficientQuantityException, ParseException {

//fill the code

// ItemDAO  it = new ItemDAO();

// List<Item> test = new ArrayList<Item>();

// test.add(it.getItemById((long)1));

// test.add(it.getItemById((long)2));

List<Item> itemlist =  new ArrayList<Item>();

             itemlist.add(new Item((long)1,"Voltas 1.4 Ton 3",10,27900.00));

             itemlist.add(new Item((long)2,"Onida 1.5 Ton 3 Star",20,32490.00));

try {

purchaseOrderBO.createPurchaseOrder(itemlist, new Integer[] {5,6}, "xyz", "6654412255",date.parse("27-05-2018"));

             } catch (InsufficientQuantityException e) {

                    // TODO Auto-generated catch block

                    e.printStackTrace();

             }

             List<PurchaseOrder> purchaseOrders= purchaseOrderBO.getAllPurchaseOrder();

             assertThat(purchaseOrders,contains(hasProperty("customerName",is("xyz"))));

}

           @Test

public void testRemoveItem()throws InsufficientQuantityException, ParseException {

        //fill the code

          ItemDAO  it = new ItemDAO();

          List<Item> test = new ArrayList<Item>();

          test.add(it.getItemById((long)1));

          test.add(it.getItemById((long)2));

          Integer[] t = {2,3};

          purchaseOrderBO.createPurchaseOrder(test,t,"Test", "90035380", date.parse("2016-09-09"));

          purchaseOrderBO.removeItemFromPurchaseOrder((long)1);

//           assertThat("Item is removed successfully",purchaseOrderBO.getAllPurchaseOrder().get(0).getOrderLine(), hasSize(1));

          List<PurchaseOrder> purchaseOrders= purchaseOrderBO.getAllPurchaseOrder();

          assertFalse(purchaseOrders.contains(hasProperty("customerName",is("Test"))));

          }

          }

**6. Invoice search**

**Junit: Invoice Serach**

import java.util.List;

import org.junit.\*;

import org.junit.rules.ExpectedException;

import static org.junit.Assert.\*;

import static org.hamcrest.Matchers.\*;

public class InvoiceJUnit {

InvoiceBO invoiceBO;

    @Rule

    public ExpectedException exception = ExpectedException.none();

    @Before

    public void createObjectForInvoiceBO() {

        //fill the code

        invoiceBO = new InvoiceBO();

    }

    @Test

    public void testFindInvoiceByAmount() throws InvalidAmountException {

        //fill the code

        assertThat(invoiceBO.findInvoiceByAmount(38000.0),containsInAnyOrder(new Invoice(101,"Ricky",5,38000.0,4000.0,"Pending")));

        assertTrue(invoiceBO.findInvoiceByAmount(0.0).isEmpty());

    }

    @Test

    public void testFindInvoiceByAmount\_Exception() throws InvalidAmountException {

        //fill the code

        try{

        invoiceBO.findInvoiceByAmount(-8.0);

        fail();

        } catch(InvalidAmountException e){

            assertEquals(e.getMessage(),"Amount is Invalid");

        }

    }

    @Test

    public void testFindPendingInvoice() {

        //fill the code

        assertThat(invoiceBO.findPendingInvoice(),containsInAnyOrder(new Invoice(101,"Ricky",5,38000.0,4000.0,"Pending"),new Invoice(102,"Jack",4,74000.0,21000.0,"Pending"),

                new Invoice(104,"Peter",3,47000.0,8000.0,"Pending"),new Invoice(105,"Willium",8,68000.0,17000.0,"Pending"),

                new Invoice(107,"John",6,85000.0,3000.0,"Pending"),new Invoice(108,"Parker",3,47000.0,8000.0,"Pending"),

                new Invoice(109,"Augestine",8,68000.0,17000.0,"Pending")));

    }

}

**Shipment (compare date)**

