Class,

In terms past, I've had students to reach out asking for clarification and I would share the approach used to complete Week 3 Milestone.  I will provide hints along the way so that you understand how the assignment was completed.

//Author Name:

//Date:

//Course ID:

//Description: This is the **contact class**. It creates and stores contact information.  
//See the Constructor for more info.

package module3;

import java.util.concurrent.atomic.AtomicLong;

public class Contact {  
private final String contactID;  
private String firstName;  
private String lastName;  
private String Number;  
private String Address;  
private static AtomicLong idGenerator = new AtomicLong();

//CONSTRUCTOR  
/\*  
\* The constructor takes first name, last name, phone number, and address as parameters.  
\* The first thing it does is generates a new ID for the contactID field.  
\*  
\* First name and last name are checked for null condition or blank fields. If either of  
\* those conditions exist, fill in the field with the phrase "NULL" so that something exists  
\* to protect data integrity while making it clear it is a placeholder.  
\* In both cases, if the first or last name is greater than 10 characters, truncate it  
\* so that only the first 10 characters are used.  
\*  
\* For the number field, if the phone number is not exactly 10 characters then fill it with  
\* the placeholder '5555555555'.  
\*  
\* Address is like first and last names. If it is blank or null, set it to "NULL".  
\* If it is more than 30 characters, truncate to the first 30 characters.  
\*/  
public Contact(String firstName, String lastName, String number, String address)  {

//CONTACTID  
//Contact ID is generated when the constructor is called. It is set as a final variable and has  
//no other getter or setter so there should be no way to change it.  
//The idGenerator is static to prevent duplicates across all contacts.  
this.contactID = String.valueOf(idGenerator.getAndIncrement());  
  
//FIRSTNAME

if (firstName == null || firstName.isBlank()) {

this.firstName = "NULL";

//If first name is longer than 10 characters, just grab the first 10 characters  
} else if(firstName.length() > 10) {  
this.firstName = firstName.substring(0, 10);  
} else {  
 this.firstName = firstName;  
}

//LASTNAME  
if (lastName == null || lastName.isBlank()) {  
this.lastName = "NULL";  
} else if(lastName.length() > 10) {  
this.lastName = lastName.substring(0,10);  
} else {  
this.lastName = lastName;  
}  
//NUMBER  
if (number == null || number.isBlank() || number.length() != 10) {  
this.Number = "5555555555";  
} else {  
this.Number = number;  
}  
//ADDRESS  
if (address == null || address.isBlank()) {  
this.Address = "NULL";  
} else if(address.length() > 30) {  
this.Address = address.substring(0,30);  
} else {  
this.Address = address;  
}  
}

//GETTERS  
public String getContactID() {  
return contactID;  
}

// Now it's your turn to finish creating the code for the 'Getters'

 .

.

.

.

Reply to Thread

Filter by:

All Posts

|

[Clear filters](javascript:void(0);)

Top of Form

Show:



Bottom of Form

Top of Form

* View profile card for Angel Cross

**Angel Cross**

July 9 at 5:48 PM

* + //Author Name:

//Date:

//Course ID:

//Description: This is the **contact service**. It maintains a list of contacts and has capabilities  
//for adding and deleting contacts, as well as updating first name, last name, phone number, and address.

package module3;

import java.util.ArrayList;

public class ContactService {  
//Start with an ArrayList of contacts to hold the list of contacts  
ArrayList<Contact> contactList = new ArrayList<Contact>();

//Display the full list of contacts to the console for error checking.  
public void displayContactList() {  
for(int counter = 0; counter < contactList.size(); counter++) {  
System.out.println("\t Contact ID: " + contactList.get(counter).getContactID());  
System.out.println("\t First Name: " + contactList.get(counter).getFirstName());  
System.out.println("\t Last Name: " + contactList.get(counter).getLastName());  
System.out.println("\t Phone Number: " + contactList.get(counter).getNumber());  
System.out.println("\t Address: " + contactList.get(counter).getAddress() + "\n");  
}  
}  
  
//Adds a new contact using the Contact constructor, then assign the new contact to the list.  
public void addContact(String firstName, String lastName, String number, String address) {  
// Create the new contact  
Contact contact = new Contact(firstName, lastName, number, address);  
contactList.add(contact);  
}

// Now it's your turn to finish creating the code

 .

.

.

.

Reply

* View profile card for Angel Cross

**Angel Cross**

July 9 at 5:48 PM

//Author Name:

//Date:

//Course ID:

//Description: This is the **unit tests** for the contact service (ContactServiceTest)

package module3;

import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.TestMethodOrder;

import static org.junit.jupiter.api.Assertions.\*;

import java.util.ArrayList;  
import org.junit.jupiter.api.DisplayName;  
import org.junit.jupiter.api.MethodOrderer.OrderAnnotation;  
import org.junit.jupiter.api.Order;

@TestMethodOrder(OrderAnnotation.class)  
public class ContactServiceTest {

/\*  
\* The following tests exercise the **ContactService** class.  
\* In each test, a new service is created with default values for each field.  
\* Then the service is requested to make some change to the list of contacts and the result  
\* is tested to ensure the actual field matches what is expected.  
\*  
\* Because each contact that gets created has a new automatically assigned contactID,  
\* and the tests are run based on that contactID, the order in which the tests are run depend  
\* on the value of each contactID. Therefore, the @Order annotation is used to keep the tests  
\* in a specific order.  
\*  
\* For evidence that the contactID is correct for each test, uncomment the service.displayContactList();  
\* prior to each assertion so that the records are displayed on the console for error checking.  
\*/  
@Test  
@DisplayName("Test to Update First Name.")  
@Order(1)  
void testUpdateFirstName() {  
ContactService service = new ContactService();  
service.addContact("Dr.", "Cross", "5555551111", "123 Lollypop Lane");  
service.updateFirstName("Sven", "0");  
//service.displayContactList();  
assertEquals("Sven",service.getContact("0").getFirstName(), "First name was not updated.");  
}

// Now it's your turn to finish creating the code

 .

.

.

.

Reply

* View profile card for Angel Cross

**Angel Cross**

July 9 at 5:49 PM

//Author Name:

//Date:

//Course ID:

//Description: This is the unit tests for the contact class (**ContactTest**).

package module3;

import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.DisplayName;  
import static org.junit.jupiter.api.Assertions.\*;

public class ContactTest {  
/\*  
\* The following tests exercise the Contact class.  
\* The first 5 tests to make sure the field does not become longer than the constraint  
\* (10 characters for first and last name, exactly 10 characters for phone number,  
\* and 30 characters for the address).  
\* The last 4 tests ensure that each field is not null.  
\* ContactID is not tested for being not null because there isn't a way to create  
\* a contact with a null contactID. Likewise it is not tested for being non-updateable  
\* because there is no way to update it.  
\*/  
@Test  
@DisplayName("Contact ID cannot have more than 10 characters")  
void testContactIDWithMoreThanTenCharacters() {  
Contact contact = new Contact("FirstName","LastName","PhoneNumbr","Address");  
if(contact.getContactID().length() > 10) {  
fail("Contact ID has more than 10 characters.");  
}  
}  
  
@Test  
@DisplayName("Contact First Name cannot have more than 10 characters")  
void testContactFirstNameWithMoreThanTenCharacters() {  
Contact contact = new Contact("OllyOllyOxenFree","LastName","PhoneNumbr","Address");  
if(contact.getFirstName().length() > 10) {  
fail("First Name has more than 10 characters.");  
}  
}

/ Now it's your turn to finish creating the code

 .

.

.

.

Bottom of Form