Lesson 2 Gosu for Integration

This exercise requires that you use **TrainingApp**, **Guidewire Studio**. Start **Guidewire Studio** for **TrainingApp**. Start the server as **Debug** ‘Server’.

Exercise 1: Create a custom class

Exercise

Succeed Insurance has several integration points that need to make use of ABContact summaries. In general, it is highly desirable for integration points to transmit and work with only the data that is required.

2.1.1 Requirements

Spec 1 Create a package that uses **si** as the customer code, **ta** as the product code, **classes** as the mechanism, and **entity** as the functional area.

Spec 2 Create a class called ABContactSummary that represents summary information of an ABContact. Whenever this class is instantiated, the ExternalID property should be set.

Spec 3 Create the following class properties:

- ExternalID (int)
- If the summary’s ExternalID is 0, set the ExternalID to a unique integer no less than 1000, otherwise do nothing.
- ContactID (String)
- Name (String)
- NumCheckingAccounts (int)

Spec 4 Create a function called loadSummaryData.

- Takes an input parameter of the type entity.ABContact.
- Set the summary's ContactID to the ABContact's public ID.
- Set the summary's Name to the ABContact's DisplayName field.
- Set the summary's NumCheckingAccounts to the number of ABContact.BankAccounts where the type is checking. Use the appropriate array function to determine the number of checking accounts.

Spec 5 Create a function called buildConcatenatedSummary.

- Returns a String that contains the value of all properties as a comma-delimited list.

2.1.2 Tasks

1. **Create a new package.**
2. **Create a new class.**
3. **Create new properties and functions.**
   a) Properly comment and annotate code.
4. **Deploy code changes.**
5. **Perform verification steps.**

Hints

For the ExternalID property:

- Use a getter and setter rather than shorthand notation.
- Use a constructor to initialize the property.
- Use the sequence utility.
- Do not use Integer because it will initialize as null.

2.1.3 Verification steps

1. **Generate console output for ABContactSummary using Gosu Scratchpad.**
   a) In Studio, open Gosu Scratchpad by clicking **Tools → Gosu Scratchpad**.
   b) Write code that will test the solution:
   c) Using **trainingapp.base.QueryUtil.findContact** method, create a variable that references contact **William Andy** whose **publicID** is **ab:5**.
      - Create a new **ABContactSummary** object.
      - Initialize the **ExternalID** sequence counter.
      - Execute **loadSummaryData()** using the given contact.
      - Retrieve and print the output of the **buildConcatenatedSummary ()** function.
   d) Verify the output
2. **Verify that the output in the Debug Console has four delimited values with correct values:**
   a) Sequence number
   b) Public ID
   c) Contact name
   d) Number of checking account

Solution 1: Create a custom class

1. **Create a new package.**

   a) Right-click on **gsrc** folder and select **New ➔ Package**.

   b) Enter si.ta.classes.entity as the new package name.

2. **Create a new class.**

   c) Right click on the entity package and select **New ➔ Gosu Class**.

   d) Enter **ABContactSummary** as the new Gosu class name.

3. **Create new properties and functions.**

```
package si.ta.classes.entity

uses gw.api.system.database.SequenceUtil

/**
 * Training lab
 */
class ABContactSummary {

  construct(i: int) {
    ExternalID = i
  }

  // Declare ExternalID property using getter and setter
  var _externalID: int  // Don't use Integer because it will initialize as null.

  property get ExternalID(): int {
    return _externalID
  }

  property set ExternalID(externalId: int) {
    if(externalId == 0) {
      _externalID = SequenceUtil.next(1000, "externalID") as int
    } else
       _externalID = externalId
  }

  // Declare properties using shorthand syntax
  var _contactID: String as ContactID
  var _name: String as Name
  var _numCheckingAccounts: int as NumCheckingAccounts

  // Create functions

  /**
   * Function that initializes ABContact properties.
   */
  @Param("contact", "Input parameter of type entity.ABContact")
  function loadSummaryData(contact: ABContact): void {
    this._contactID = contact.PublicID
    this._name = contact.DisplayName
    this._numCheckingAccounts = contact.BankAccounts.countWhere(\account ->
        account.AccountType == typekey.BankAccountType.TC_CHECKING)
  }

  /**
   * Function that builds a comma-delimited list.
   */
  @Returns("A comma-delimited list of property values")
  function buildConcatenatedSummary(): String {
    return String.format("%s,%s,%s,%s", {_externalID, _contactID, _name, _numCheckingAccounts})
  }
}
```

4. **Deploy code changes.**

   - From the **Studio** menu, **Restart** the server.

5. **Perform verification steps.**

```
uses trainingapp.base.QueryUtil
uses si.ta.classes.entity.ABContactSummary

var testContact = QueryUtil.findContact("ab:5")
var newSummary = new ABContactSummary(0)
newSummary.loadSummaryData(testContact)
print(newSummary.buildConcatenatedSummary())
```