APEX LANGUAGE FEATURES

SOLUTIONS:

sObject Type

Convert 15-digit ID to 18-digit ID

Handle Exception

Throw An Exception

Safe Navigation Operator

Dynamic Field Values

Serialize Objects

Deserialize Objects

#100 - sObject Type

Implement the method `isTypeAccount()`, which accepts a sObject as input and returns a `true` if type of input is Account object else it should return as `false`.
Given the following test code:
```
Account acc = new Account(name='Apple')
Boolean result = isTypeAccount(acc);
```
`result` should be equal to `true`

`Solution:`

```
public Boolean isTypeAccount(sObject record)
{
    // code here

    if(record == null)
        return false;
    else return record.getSObjectType() == Account.SObjectType;


}
```

#93 - Convert 15-digit ID to 18-digit ID

Implement the method `convert15to18DigitId()` , which accepts a String of length 15 digit and returns a new String with 18 digit salesforce Id. If the input string length is not equal to 15 digits, then return `'-1'`.
Given the following test code:
```
String fifteenDigit = '0SO90000000PBDu';
String eighteenDigit = convert15to18DigitId(fifteenDigit);
```
`eighteenDigit` should be equal to `'0SO90000000PBDuGAO'`
Note:

- Use case 1: You have exported a Salesforce report with Ids. These Ids are 15 characters. You want to ensure that these Ids are not altered by Excel, you need to manage them with 18 characters.
- Use case 2: You need to compare Ids but your comparison mechanism is not case sensitive. You will have to extend them to 18 characters

Solution:

```
public String convert15to18DigitId(String fifteenDigit)
{
 //Code here
 if(fifteenDigit == null)
 return null;
 else if(fifteenDigit.length()!=15)
 return '-1';
 Id eighteenDigit = fifteenDigit;
 return eighteenDigit;
```

```
}
```

#97 - Handle Exception

Implement the method `divide` which takes two integers a and b as input, divides a by b using the / operator, and returns the result. If any exception occurs, the method should return the exception message.
Given the following test code:
`String result = divide(10,0);`
`result` should be `'Divide by 0';`
Given the following test code:
`String result = divide(100, 2);`
`result` should be `'5';`

Solution:

```
public String divide(Integer a, Integer b){
    //code here
    try
    {
        String divide = String.valueOf(a/b);
        return divide;
    }
    catch(Exception e)
    {
        return e.getMessage();
    }
}
```

#102 - Throw An Exception

Implement the method `checkAccounts`, which accepts a list of accounts as an input and returns a list of accounts. The method should behave as follows:

- If all accounts in the list have `BillingCity` present, the method should return the original list.
- If the passed list is `null` the method should throw the built-in `IllegalArgumentException` with message `'accounts should not be null'`
- If any of the accounts in the list do not have a `BillingCity` present, the method should throw the custom `AccountException` exception. Do not create new exception class - use the `AccountException` class that has already been created for you.

Given the following test code:
`List<Account> accounts = new List<Account>();`
`accounts.add(new Account(name ='Salesforce', BillingCity ='Boston'));`
`accounts.add(new Account(name ='Microsoft'));`
The method call`checkAccounts(accounts);` should fail, throwing an `AccountException`.

Solution:

```
public List<Account> checkAccounts(List<Account> accounts)
{
    // code here
    if(accounts == null)

    {

        throw new IllegalArgumentException ('accounts should not be null');

    }

    for(Account acc : accounts)

    {

        if(acc?.BillingCity == null)

        {

            throw new AccountException('Invalid BillingCity');

        }

    }

     return accounts;
}

//do not remove the following custom-defined exception
public class AccountException extends Exception {}
```

#94 - Safe Navigation Operator

Implement the method `getAccountBillingCityWithSafeNavigation`, which accepts a list of accounts as an input and returns the `BillingCity` in upper case of the **first** account in the list. Use the Safe Navigation (?.) to ensure `null` is returned in case the `BillingCity` is null.
Given the following test code:

```
List<Account> acts = new ListList<Account>();
acts.add(new Account(Name = 'Acme', BillingCity = 'Chicago'));
acts.add(new Account(Name = 'Dove', BillingCity = 'Boston'));
String result = getAccountBillingCityWithSafeNavigation(acts);
```

`result` should be `'CHICAGO'`

Solution:

```
public String getAccountBillingCityWithSafeNavigation(List<Account> accounts){
   // Code here
   for(Account acc : accounts){
```

```
  if(acc?.BillingCity == null)
      return null;
  else if(acc?.BillingCity!=null)
      return acc.BillingCity.toUpperCase();
   }
   return null;

}
```

#103 - Dynamic Field Values

Implement the method `getFieldsValue`, which accepts the following inputs:

- An account `acc`
- A list of strings `fields`, with each string in the list representing a valid field on the account object.

The method should return a list of values from the account record for fields listed in the list `fields` in the correct order.
Given the following test code:

```
Account acc = new Account(
    Name = 'Salesforce',
    BillingCity = 'Boston',
    AnnualRevenue=10000, Rating='Hot');
List fields = new List{'Industry','Name','Rating'};
List result = getFieldsValue(acc, fields);
```

`result` should be `[null, 'Salesforce', 'Hot']`

Solution:

```
public List<String> getFieldsValue(Account acc, List<String> fields)
{
    // code here
 List<String> result = new List<String>();
 for(String f : fields)
 {
     result.add(String.valueof(acc.get(f)));
 }
 return result;
}
```

#95 - Serialize sObjects

Implement the method `getAccountsInJSONFormat()`, a list of accounts and returns a list of accounts in string JSON format.
Given the following test code:

```
List<Account> accounts = new ListList<Account>();
accounts.add(new Account(Name = 'Acme', BillingCity = 'Chicago'));
```

```
accounts.add(new Account(Name = 'Dove', BillingCity = 'Boston'));
String result = getAccountsInJSONFormat(accounts);
result should be equals to
'[{"attributes":{"type":"Account"},"Name":"Acme","BillingCity":"Chicago"},{"attrib
utes":{"type":"Account"},"Name":"Dove", "BillingCity":"Boston"}]';
```

Solution:

```
public String getAccountsInJSONFormat(List<Account> accounts){
  // code here
  String JSONString = JSON.serialize(accounts);
  return JSONString;
}
```

#96 - Deserialize sObjects

Implement the method `getAccountsFromJSONString`, which takes a JSON string of a list of accounts as an input and returns a list of accounts. If the input string is empty or null, return null.
Given the following test code:

```
String inputJSON =
'[{"attributes":{"type":"Account","url":"/services/data/v55.0/sobjects/Account/001
58000002zBhUAAU"},"Id":"00158000002zBhUAAU","Name":"Customer1"},{"attributes":{"ty
pe":"Account","url":"/services/data/v55.0/sobjects/Account/00158000002zBhWAAU"},"I
d":"00158000002zBhWAAU","Name":"Customer2"}]';
List<Account> result = getAccountsFromJSONString(inputJSON);
accounts.add(new Account(Name = 'Dove', BillingCity = 'Boston'));
result should be list of accounts (Account:{Id=00158000002zBhUAAU, Name=Customer1},
Account:{Id=00158000002zBhWAAU, Name=Customer2})
```

Solution:

```
public List<Account> getAccountsFromJSONString(String inputJSON){
    // code here
    if(inputJSON==Null) return null;
    List<Account> deserializedJSON =
          (List<Account>)JSON.deserialize(inputJSON, List<Account>.class);
          return deserializedJSON;

}
```