**Maps of sObjects**

Map keys and values can be of any data type, including sObject types, such as Account.

Maps can hold sObjects both in their keys and values. A map key represents a unique value that maps to a map value. For example, a common key would be an ID that maps to an account (a specific sObject type). This example shows how to define a map whose keys are of type ID and whose values are of type Account.

|  |  |
| --- | --- |
| 1 | Map<ID, Account> m = new Map<ID, Account>(); |

As with primitive types, you can populate map key-value pairs when the map is declared by using curly brace ({}) syntax. Within the curly braces, specify the key first, then specify the value for that key using =>. This example creates a map of integers to accounts lists and adds one entry using the account list created earlier.

|  |  |  |
| --- | --- | --- |
| 1 | Account[] accs = new Account[5]; // Account[] is synonymous with List<Account> | |
| 2 | Map<Integer, List<Account>> m4 = new Map<Integer, List<Account>>{1 => accs}; |

Maps allow sObjects in their keys. You should use sObjects in the keys only when the sObject field values won’t change.

Auto-Populating Map Entries from a SOQL Query

When working with SOQL queries, maps can be populated from the results returned by the SOQL query. The map key should be declared with an ID or String data type, and the map value should be declared as an sObject data type.

This example shows how to populate a new map from a query. In the example, the SOQL query returns a list of accounts with their Id and Name fields. The new operator uses the returned list of accounts to create a map.

|  |  |  |
| --- | --- | --- |
| 1 | | // Populate map from SOQL query |
| 2 | Map<ID, Account> m = new Map<ID, Account>([SELECT Id, Name FROM Account LIMIT 10]); | | |

|  |  |  |
| --- | --- | --- |
| 3 | // After populating the map, iterate through the map entries | |
| 4 | for (ID idKey : m.keyset()) { |

|  |  |  |
| --- | --- | --- |
| 5 | Account a = m.get(idKey); | |
| 6 | System.debug(a); |

|  |  |
| --- | --- |
| 7 | } |

One common usage of this map type is for in-memory “joins” between two tables.

Using Map Methods

The Map class exposes various methods that you can use to work with map elements, such as adding, removing, or retrieving elements. This example uses Map methods to add new elements and retrieve existing elements from the map. This example also checks for the existence of a key and gets the set of all keys. The map in this example has one element with an integer key and an account value.

[view source](https://developer.salesforce.com/docs/#viewSource)

[print](https://developer.salesforce.com/docs/#printSource)[?](https://developer.salesforce.com/docs/#about)

|  |  |  |
| --- | --- | --- |
| 1 | Account myAcct = new Account();                        //Define a new account | |
| 2 | Map<Integer, Account> m = new Map<Integer, Account>(); // Define a new map |

|  |  |  |
| --- | --- | --- |
| 3 | m.put(1, myAcct);                  // Insert a new key-value pair in the map | |
| 4 | System.assert(!m.containsKey(3));  // Assert that the map contains a key |

|  |  |  |
| --- | --- | --- |
| 5 | | Account a = m.get(1);              // Retrieve a value, given a particular key |
| 6 | Set<Integer> s = m.keySet();       // Return a set that contains all of the keys in the map | |

* [**sObject Map Considerations**](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_map_sobject_considerations.htm)