Couchbase support for Java source code

- Supported Client Libraries
- Supported Operations (2.x)
- Supported Operations (3.x)
- Objects (2.x)
- Objects (3.x)
- Links (2.x)
- Links (3.x)
- What results can you expect?
    - Cluster and Bucket Object
    - Insert operations
    - Select operations
    - Update Operations
    - Delete Operations
    - Batch Operations
    - N1QL Queries
- Evolutions
- Limitations
- Future

CAST supports **Couchbase** via its NoSQL for Java extension. Details about the support provided for **Java SDK source code** is discussed below.

Supported Client Libraries

Java-Client-2.7.7
java-client 3.1.6

Supported Operations (2.x)

Operation	Methods Supported
Insert	• insert • upsert
Select	• getAndTouch • getAndLock • exists • get
Delete	• remove
Update	• replace • prepend • append

Supported Operations (3.x)

Operation	Methods Supported
Insert	• insert • upsert

Select	• getAndTouch • getAndLock • exists • get
Delete	• remove
Update	• replace

Objects (2.x)

Icon	Description
	Java Couchbase Cluster
	Java Couchbase Bucket
	Java Couchbase Unknown Cluster
	Java Couchbase Unknown Bucket

Objects (3.x)

Icon	Description
	Java Couchbase Cluster
	Java Couchbase Bucket
	Java Couchbase Collection
	Java Couchbase Unknown Cluster
	Java Couchbase Unknown Bucket
	Java Couchbase Unknown Collection

Links (2.x)

Links are created for transaction and function point needs:

Link type	Source and destination of link	Methods supported
belongsTo	From Java Couchbase Collection object to Java Couchbase Bucket object and Java Couchbase Bucket object to Java Couchbase Cluster object	

useLink	Between the caller Couchbase Java Method objects and Java Couchbase Buckets/Collection objects	• Upsert • N1QL. parameterized • N1QL.simple
useSelectLink		• get() • async().get() • getAndTouch() • async(). getAndTouch() • getAndLock() • async(). getAndLock() • exists() • async().exists()
useUpdateLink		• replace() • async(). replace() • append() • async(). append() • prepend() • async.prepend()
useDeleteLink		• remove() • async(). remove()
useInsertLink		• insert • async().insert()

Links (3.x)

Link type	Source and destination of link	Methods supported
parentLink	• Between Couchbase Connection object and Bucket object • Between Couchbase bucket object and Collection object	
useLink	Between the caller Couchbase Java Method objects and Couchbase Buckets objects	• Upsert
useSelectLink		• get() • async().get() • getAndTouch() • async().getAndTouch() • getAndLock() • async().getAndLock() • exists() • async().exists()
useUpdateLink		• replace() • async().replace()
useDeleteLink		• remove() • async().remove()

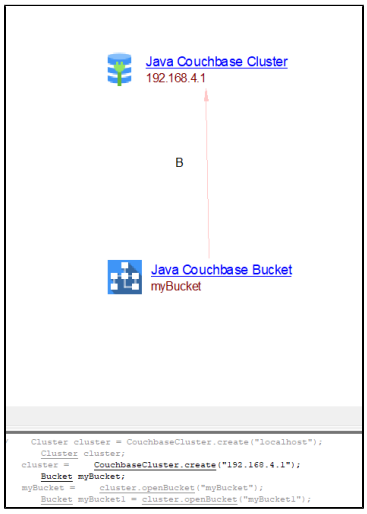
useInsertLink		• insert • async().insert()

What results can you expect?

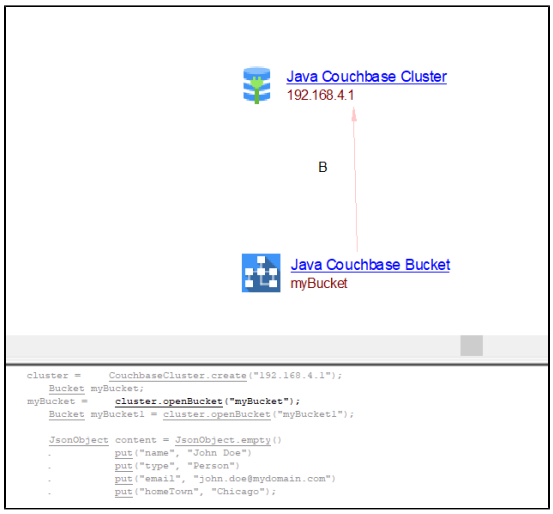
Once the analysis/snapshot generation is completed, you can view the results in the normal manner (for example via CAST Enlighten). Some examples are shown below.

Cluster and Bucket Object

```
public class App
{
    public static void main( String[] args )
    {

        System.out.println("Create connection");
                  final Cluster cluster = CouchbaseCluster.create("127.0.0.1");
            cluster.authenticate("Administrator", "Password");
        }
}
```

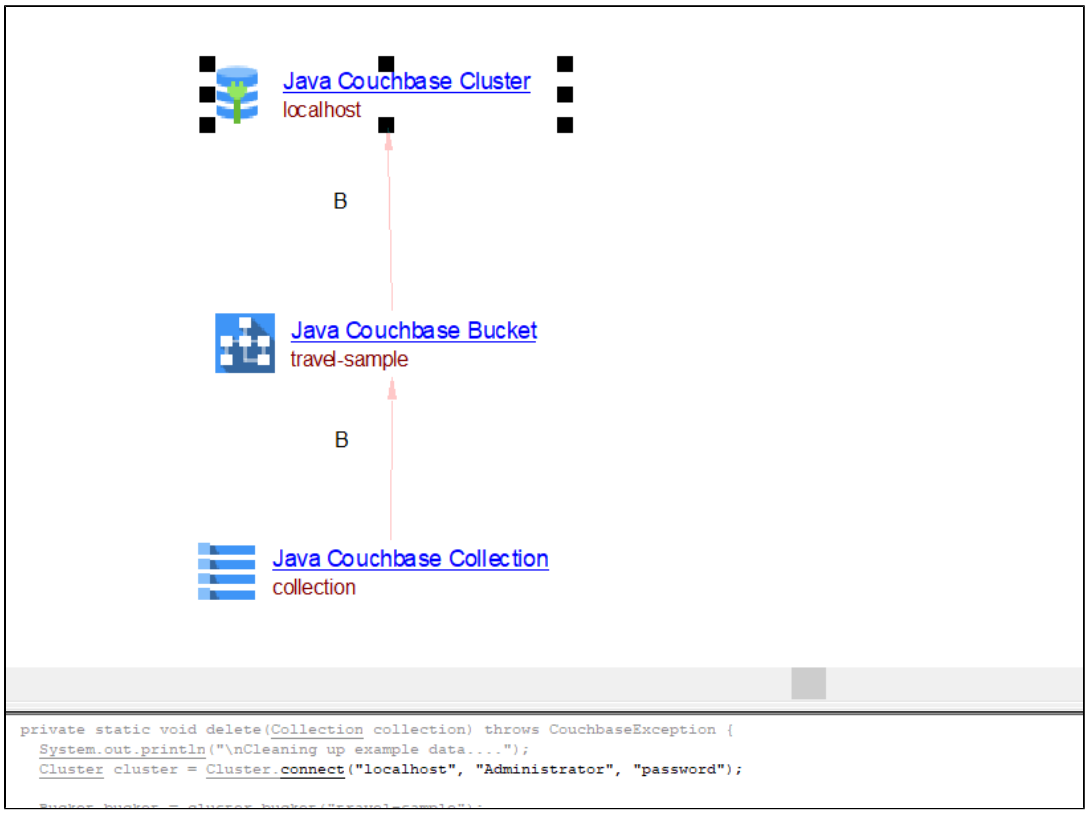



```
public class App
{
    public static void main( String[] args )
    {

        System.out.println("Create connection");
            final Cluster cluster = CouchbaseCluster.create("127.0.0.1");
            cluster.authenticate("Administrator", "Password");
            Bucket myBucket1 = cluster.openBucket("example1");
    }
}
```



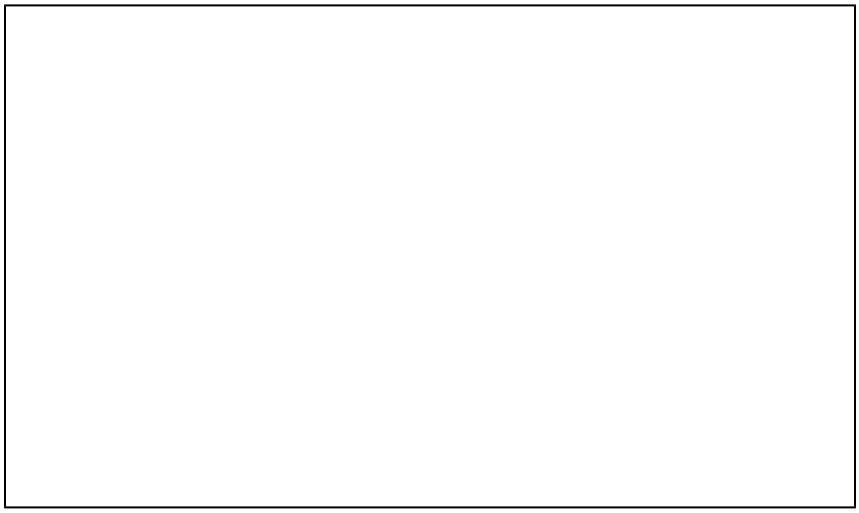
Connection, Bucket and Cluster object

```
public static void main(String... args) {

  Cluster cluster = Cluster.connect("localhost", "Administrator", "password");

  Bucket bucket = cluster.bucket("travel-sample");
  Scope scope = bucket.scope("_default");
  Collection collection = scope.collection("_default");
```

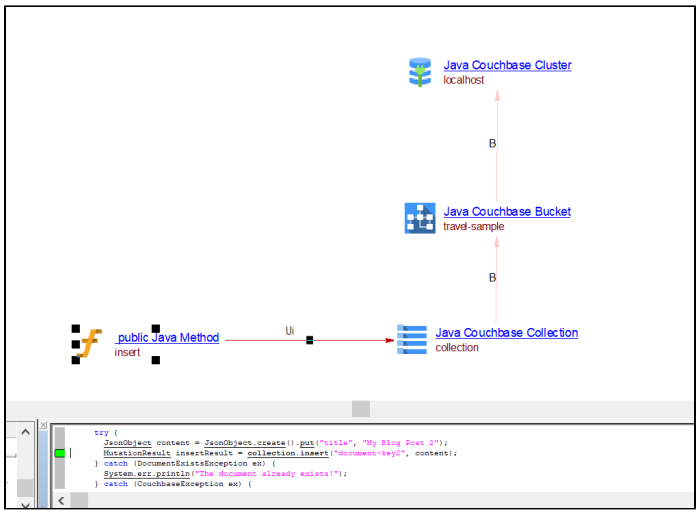


Insert operations

```
public class App
{
    public static void main( String[] args )
    {

        System.out.println("Create connection");
            final Cluster cluster = CouchbaseCluster.create("127.0.0.1");
            cluster.authenticate("Administrator", "Password");
            Bucket myBucket = cluster.openBucket("example");
            JsonObject content = JsonObject.empty()
                                .put("name", "John Doe")
                                .put("type", "Person")
                                .put("email", "john.doe@mydomain.com")
                                .put("homeTown", "Chicago");
                String id = UUID.randomUUID().toString();
            JsonDocument document = JsonDocument.create(id, content);
                JsonDocument inserted = myBucket.insert(document);
        }
}
```



```
public void insert()
{
      System.out.println("\nExample: [insert]");
      // tag::insert[]
      try {
        JsonObject content = JsonObject.create().put("title", "My Blog Post 2");
        MutationResult insertResult = collection.insert("document-key2", content);
      } catch (DocumentExistsException ex) {
        System.err.println("The document already exists!");
      } catch (CouchbaseException ex) {
        System.err.println("Something else happened: " + ex);
      }
      // end::insert[]
    }
```



Select operations

- **get()**

```
public class App
{
    public static void main( String[] args )
    {

        System.out.println("Create connection");
            final Cluster cluster = CouchbaseCluster.create("127.0.0.1");
            cluster.authenticate("Administrator", "Password");
            Bucket myBucket = cluster.openBucket("example");
            JsonObject content = JsonObject.empty()
                                .put("name", "John Doe")
                                .put("type", "Person")
                                .put("email", "john.doe@mydomain.com")
                                .put("homeTown", "Chicago");
                String id = UUID.randomUUID().toString();
            JsonDocument document = JsonDocument.create(id, content);
                JsonDocument inserted = myBucket.insert(document);
                JsonDocument document1 = JsonDocument.create(id, content1);


                JsonDocument gets = myBucket.get(document);
                System.out.println(gets);

        }
}
```

- **getAndTouch()**

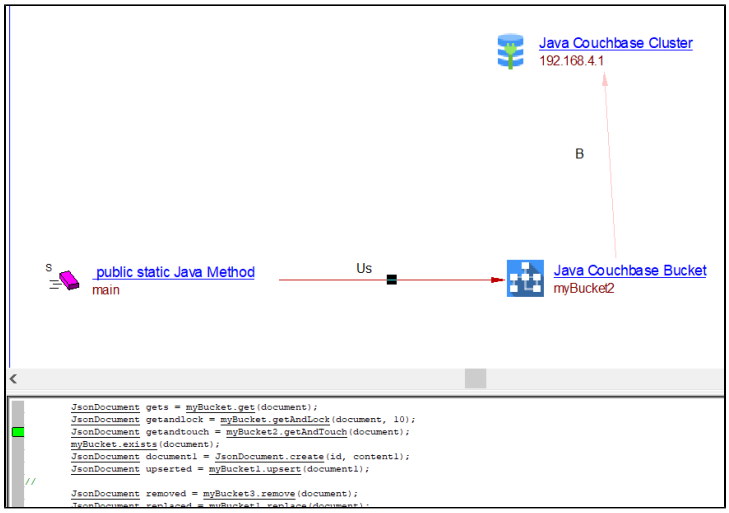
```
public class App
{
    public static void main( String[] args )
    {

        System.out.println("Create connection");
            final Cluster cluster = CouchbaseCluster.create("127.0.0.1");
            cluster.authenticate("Administrator", "password");
            Bucket myBucket = cluster.openBucket("example");
            JsonObject content = JsonObject.empty()
                               .put("name", "John Doe")
                               .put("type", "Person")
                               .put("email", "john.doe@mydomain.com")
                               .put("homeTown", "Chicago");
                String id = UUID.randomUUID().toString();
            JsonDocument document = JsonDocument.create(id, content);
                JsonDocument inserted = myBucket.insert(document);
                JsonDocument document1 = JsonDocument.create(id, content1);

                JsonDocument getandtouch = myBucket.getAndTouch(document);
                System.out.println(getandtouch);
        }
}
```

- **getAndLock()**

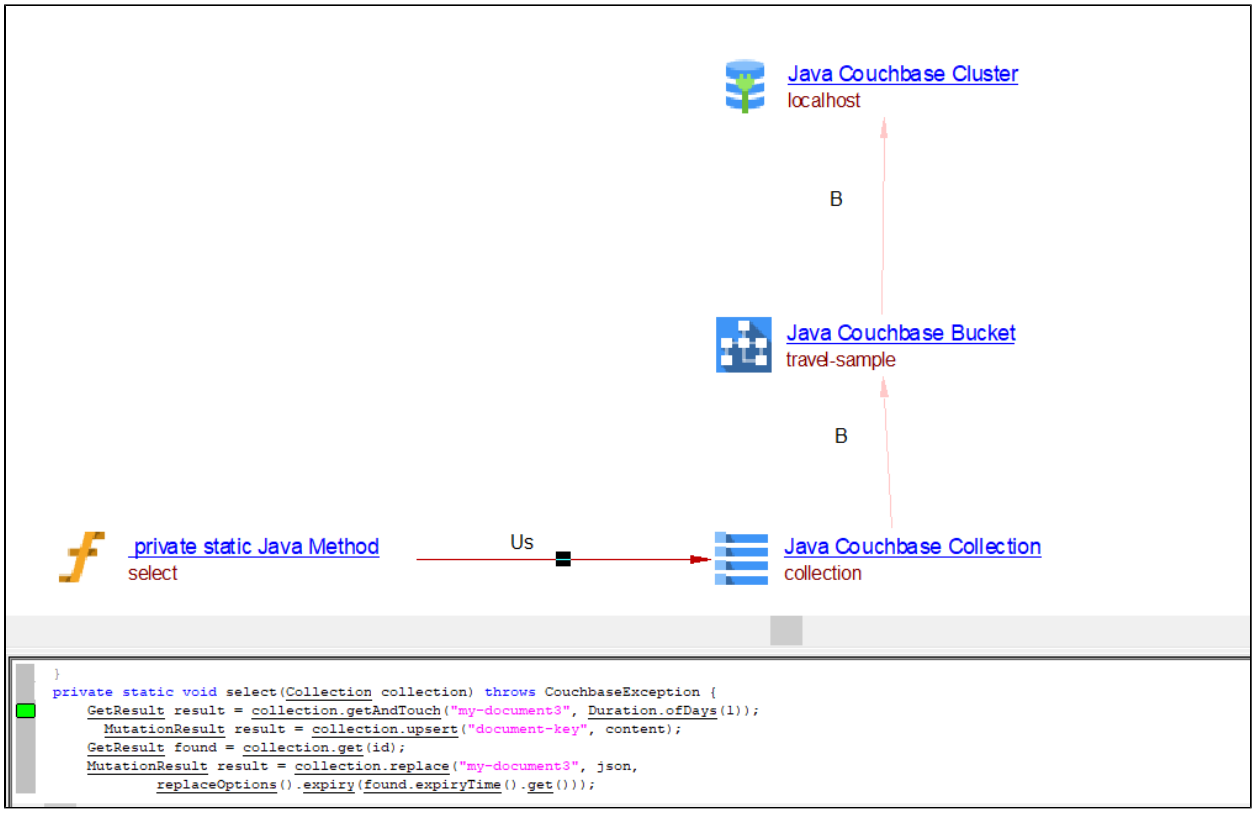
```
public class App
{
    public static void main( String[] args )
    {

        System.out.println("Create connection");
            final Cluster cluster = CouchbaseCluster.create("127.0.0.1");
            cluster.authenticate("Administrator", "Password");
            Bucket myBucket = cluster.openBucket("example");
            JsonObject content = JsonObject.empty()
                               .put("name", "John Doe")
                               .put("type", "Person")
                               .put("email", "john.doe@mydomain.com")
                               .put("homeTown", "Chicago");
                String id = UUID.randomUUID().toString();
            JsonDocument document = JsonDocument.create(id, content);
                JsonDocument inserted = myBucket.insert(document);
                JsonDocument document1 = JsonDocument.create(id, content1);

                JsonDocument getandlock = myBucket.getAndLock(document, 10);
                System.out.println(getandlock);
        }
}
```

- **exists()**

```
public class App
{
    public static void main( String[] args )
    {

        System.out.println("Create connection");
            final Cluster cluster = CouchbaseCluster.create("127.0.0.1");
            cluster.authenticate("Administrator", "Password");
            Bucket myBucket = cluster.openBucket("example");
            JsonObject content = JsonObject.empty()
                                .put("name", "John Doe")
                                .put("type", "Person")
                                .put("email", "john.doe@mydomain.com")
                                .put("homeTown", "Chicago");
                String id = UUID.randomUUID().toString();
            JsonDocument document = JsonDocument.create(id, content);
                JsonDocument inserted = myBucket.insert(document);
                JsonDocument document1 = JsonDocument.create(id, content1);

                Boolean exists = myBucket.exists(document);
                System.out.println(exists);
        }
}
```



```
public void get()
{
      System.out.println("\nExample: [get-simple]");
      // tag::get-simple[]
      try {
        GetResult getResult = collection.get("document-key");
        String title = getResult.contentAsObject().getString("title");
        System.out.println(title); // title == "My Blog Post"
      } catch (DocumentNotFoundException ex) {
        System.out.println("Document not found!");
      }
      // end::get-simple[]
    }
public void getAndTouch()
    {
      System.out.println("\nExample: [expiry-touch]");
      // tag::expiry-touch[]
      GetResult result = collection.getAndTouch("my-document3", Duration.ofDays(1));
      // end::expiry-touch[]

      System.out.println("cas value: " + result.cas());
    }
```



Update Operations

- **replace()**

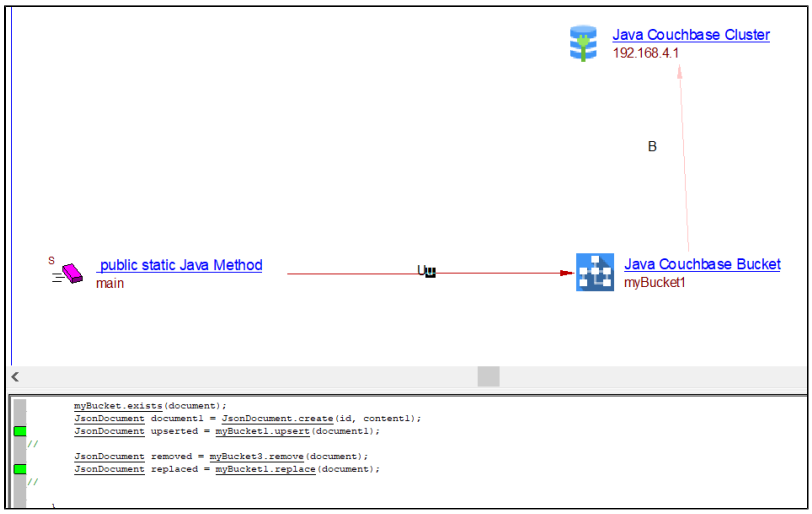
```
public class App
{
    public static void main( String[] args )
    {

        System.out.println("Create connection");
            final Cluster cluster = CouchbaseCluster.create("127.0.0.1");
            cluster.authenticate("Administrator", "Password");
            Bucket myBucket = cluster.openBucket("example");
            JsonObject content = JsonObject.empty()
                              .put("name", "John Doe")
                              .put("type", "Person")
                              .put("email", "john.doe@mydomain.com")
                              .put("homeTown", "Chicago");

                JsonObject content3 = JsonObject.empty()
                            .put("Emp_Trigram", "ABC")
                            .put("Emp_id", "001");
                JsonDocument document1 = JsonDocument.create(id, content1);

                String id = UUID.randomUUID().toString();
            JsonDocument document = JsonDocument.create(id, content);
                JsonDocument inserted = myBucket.insert(document);
                JsonDocument document1 = JsonDocument.create(id, content1);

                JsonDocument removed = myBucket.replace(document1);
                System.out.println(myBucket);
        }
}
```

- **prepend()**

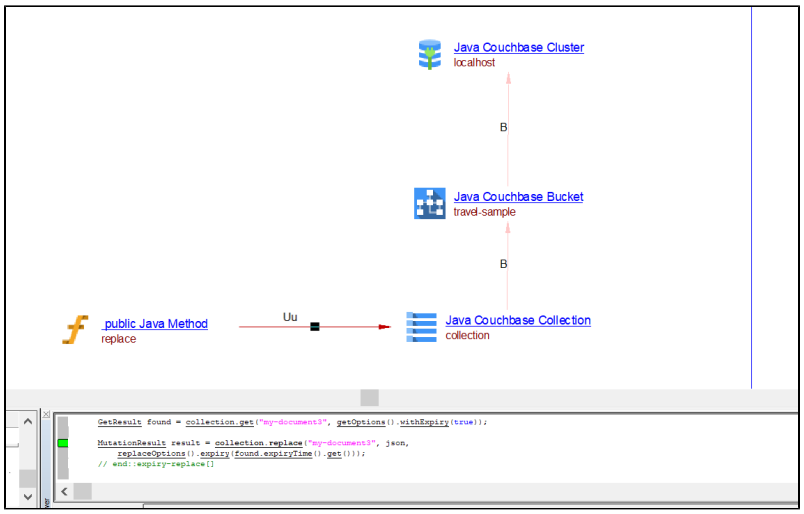
```
public class App
{
    public static void main( String[] args )
    {

        System.out.println("Create connection");
            final Cluster cluster = CouchbaseCluster.create("127.0.0.1");
            cluster.authenticate("Administrator", "Password");
            Bucket myBucket = cluster.openBucket("example");
            JsonObject content = JsonObject.empty()
                              .put("name", "John Doe")
                              .put("type", "Person")
                              .put("email", "john.doe@mydomain.com")
                              .put("homeTown", "Chicago");
                String id = UUID.randomUUID().toString();
            JsonDocument document = JsonDocument.create(id, content);
                JsonDocument inserted = myBucket.insert(document);
                JsonObject content3 = JsonObject.empty()
                            .put("Emp_Trigram", "ABC")
                            .put("Emp_id", "001");
                JsonDocument document1 = JsonDocument.create(id, content1);

                JsonDocument prepended = myBucket.prepend(document1);

        }
}
```

- **append()**

```
public class App
{
    public static void main( String[] args )
    {

        System.out.println("Create connection");
            final Cluster cluster = CouchbaseCluster.create("127.0.0.1");
            cluster.authenticate("Administrator", "Password");
            Bucket myBucket = cluster.openBucket("example");
            JsonObject content = JsonObject.empty()
                               .put("name", "John Doe")
                               .put("type", "Person")
                               .put("email", "john.doe@mydomain.com")
                               .put("homeTown", "Chicago");
                String id = UUID.randomUUID().toString();
            JsonDocument document = JsonDocument.create(id, content);
                JsonDocument inserted = myBucket.insert(document);
                JsonObject content3 = JsonObject.empty()
                            .put("Emp_Trigram", "ABC")
                            .put("Emp_id", "001");
                JsonDocument document1 = JsonDocument.create(id, content1);

                JsonDocument prepended = myBucket.append(document1);

        }
}
```



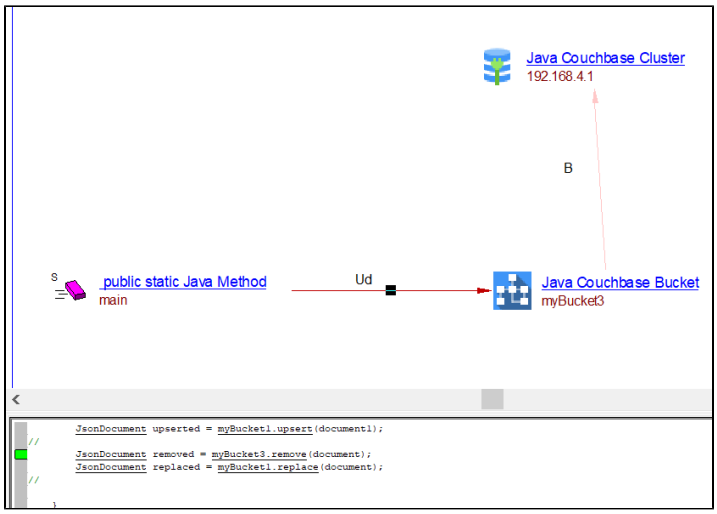

```
public void get()
{
      System.out.println("\nExample: [expiry-replace]");
      // tag::expiry-replace[]
      GetResult found = collection.get("my-document3", getOptions().withExpiry(true));

      MutationResult result = collection.replace("my-document3", json,
          replaceOptions().expiry(found.expiryTime().get()));
      // end::expiry-replace[]

      System.out.println("cas value: " + result.cas());
    }
```



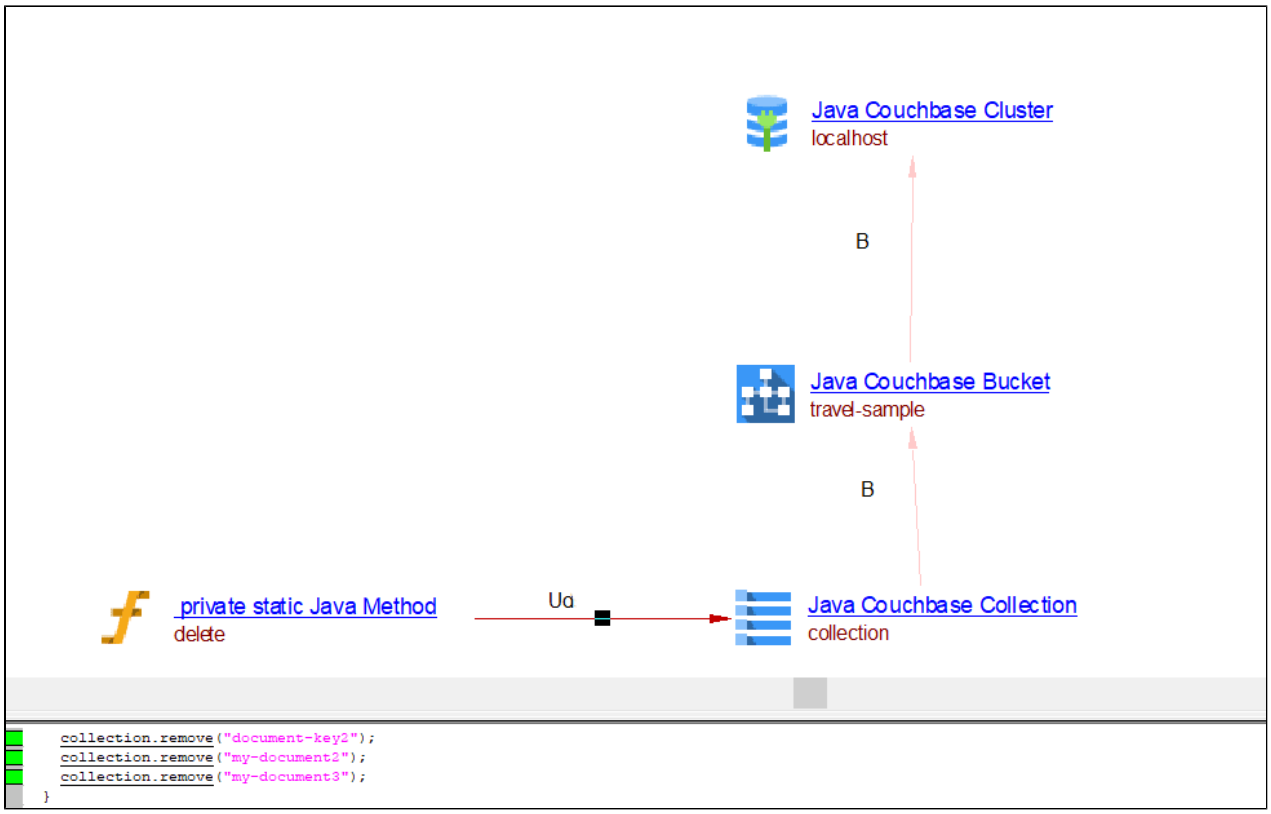
Delete Operations

- **remove()**

```
public class App
{
    public static void main( String[] args )
    {

        System.out.println("Create connection");
            final Cluster cluster = CouchbaseCluster.create("127.0.0.1");
            cluster.authenticate("Administrator", "Password");
            Bucket myBucket = cluster.openBucket("example");
            JsonObject content = JsonObject.empty()
                                .put("name", "John Doe")
                                .put("type", "Person")
                                .put("email", "john.doe@mydomain.com")
                                .put("homeTown", "Chicago");
                String id = UUID.randomUUID().toString();
            JsonDocument document = JsonDocument.create(id, content);
                JsonDocument inserted = myBucket.insert(document);
                JsonDocument document1 = JsonDocument.create(id, content1);

                JsonDocument removed = myBucket.remove(document);
                System.out.println(myBucket);
        }
}
```




```
public void remove()

{
      System.out.println("\nExample: [remove]");
      // tag::remove[]
      try {
        collection.remove("my-document");
      } catch (DocumentNotFoundException ex) {
        System.out.println("Document did not exist when trying to remove");
      }
      // end::remove[]
    }
```

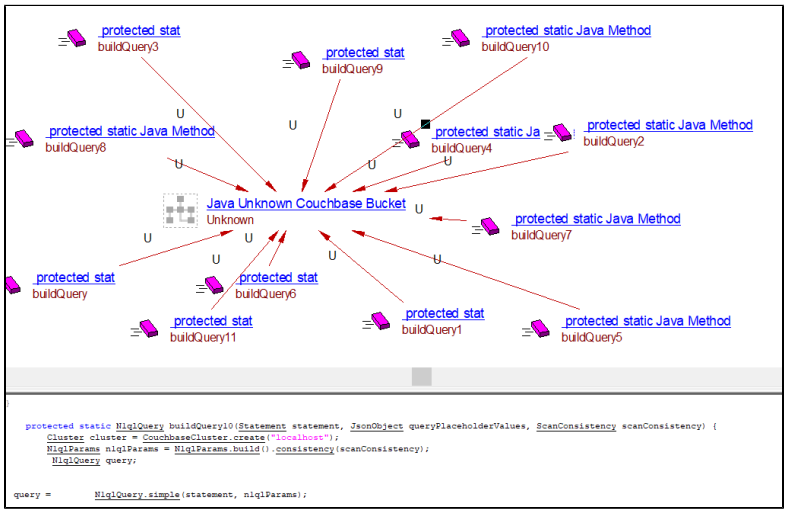


Batch Operations

```
int docsToCreate = 100;
List<JsonDocument> documents = new ArrayList<JsonDocument>();
for (int i = 0; i < docsToCreate; i++) {
    JsonObject content = JsonObject.create()
        .put("counter", i)
        .put("name", "Foo Bar");
    documents.add(JsonDocument.create("doc-"+i, content));
}

// Insert them in one batch, waiting until the last one is done.
Observable
    .from(documents)
    .flatMap(new Func1<JsonDocument, Observable<JsonDocument>>() {
        @Override
        public Observable<JsonDocument> call(final JsonDocument docToInsert) {
            return bucket.async().insert(docToInsert);
        }
    })
    .last()
    .toBlocking()
    .single();
```

N1QL Queries

```
protected static N1qlQuery buildQuery(Statement statement, JsonValue queryPlaceholderValues, ScanConsistency
scanConsistency) {
                Cluster cluster = CouchbaseCluster.create("localhost");
                N1qlParams n1qlParams = N1qlParams.build().consistency(scanConsistency);
                 N1qlQuery query;
                 if (queryPlaceholderValues instanceof JsonObject && !((JsonObject) queryPlaceholderValues).
isEmpty()) {
                  query = N1qlQuery.parameterized(statement, (JsonObject) queryPlaceholderValues,
n1qlParams);
                 } else if (queryPlaceholderValues instanceof JsonArray && !((JsonArray)
queryPlaceholderValues).isEmpty()) {
                  query = N1qlQuery.parameterized(statement, (JsonArray) queryPlaceholderValues, n1qlParams);
                 } else {
                  query = N1qlQuery.simple(statement, n1qlParams);
                 }
                 return query;
                }
```



Evolutions

- Collections are supported for SDK 3.x
- Better resolution for client and bucket

Limitations

- Only CRUD operations are supported for Java 3.x

Future

- N1QL query support in Java SDK 3.x