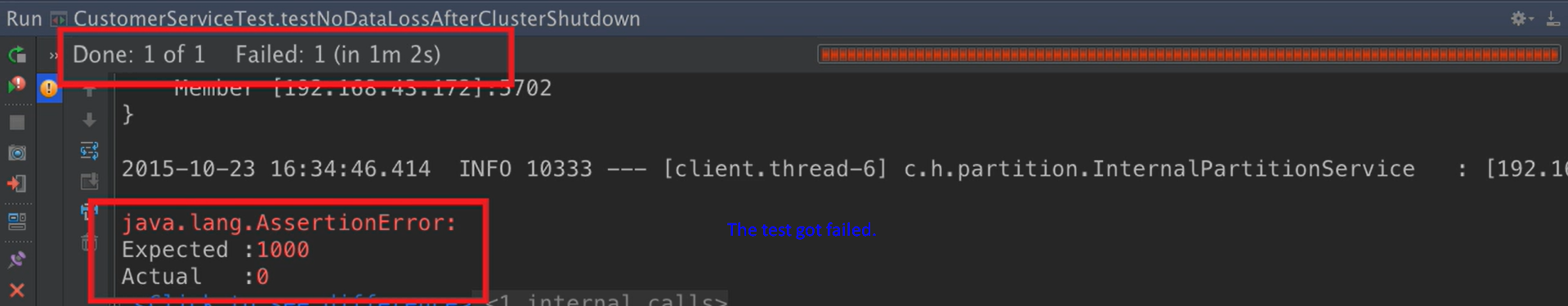
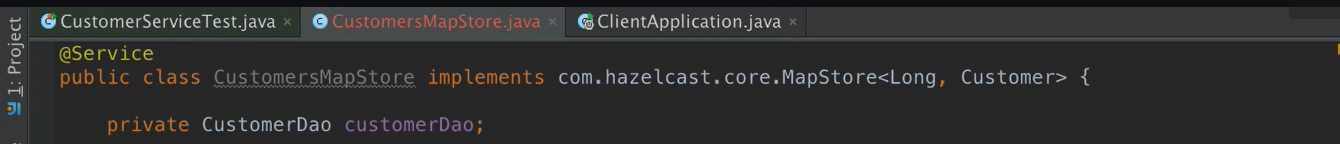
1. **Agenda**:
   1. We want to test a case where we will shut down all the 4 **Storage Nodes (Cluster Members)** and then will start 3 **Storage Nodes** to see if the data stored in cluster is available even later on or not.
2. Text

   Description automatically generated
   1. When there is no **Storage Node** (**Cluster Member**) for a **Client Node** to connect with, it will attempt a few times and then stop with an error “It couldn’t connect with the **Cluster**”.
   2. To get this test to work, I need to tell the Hazelcast client to keep trying to connect to the cluster indefinitely.
   3. To do this, we’ve created the following Bean Definition inside our client app configuration.  
      This bean creates **ClientConfig.java** instance.  
      This is the parent object for all configuration options for the clients.  
      On that client config, I get the default network configuration object & set the connection attempt limit to zero (zero means **unlimited**).  
      So, Hazelcast client instance continually try to reconnect.  
      Text

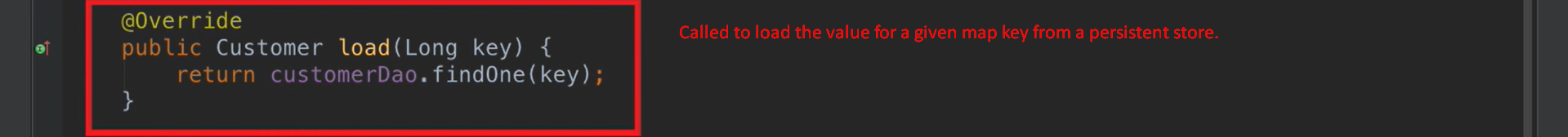
      Description automatically generated
   4. In Hazelcast, you can pretty much configure everything either in code or XML configuration file.  
      But in this course, we will be using only code version.
3. Let’s run the test case.
4. When we run the test case, it gets failed & this is what is expected.  
   As when we shut down all of the **Four Storage Nodes**, then there is nowhere for **Hazelcast** to store the data.  
   So, when we start the **three** **Storage Nodes** up, there is no data within the cluster to redistribute & therefore the map size is actually zero.  
   So, therefore out of the box, **Hazelcast** doesn’t persist data to a **persistence store**. It is completely **in-memory**.
5. However, for certain **Data Structures** (distributed data structure being one of them), we can configure Hazelcast to write the data to a persistent store when the data in that data structure is created, updated & deleted.  
   Text

   Description automatically generated with medium confidence
   1. To do this, we need two things.
      1. Create a class responsible to persist data (This class will be used by Hazelcast to perform CRUD operations and in turn this class will use DAO class to perform persistence operations on DB).
      2. Configure Hazelcast to use this class.
6. **Let’s create the class responsible to persist the data**.
   1. Create a class implementing **com.hazelcast.core.MapStore<Customer<Long, Customer>**.
   2. 

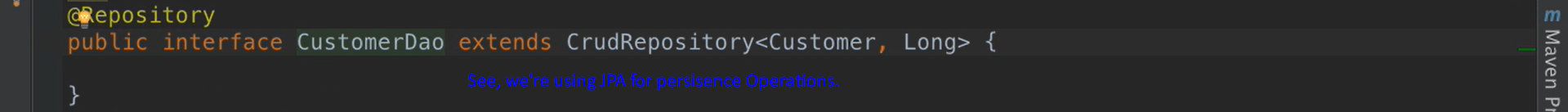
In the above slide, you can see CustomerDao which extends CrudRepository<Customer, Long>. So, we are using JPA for persistence operations.   
You can use any DB. It is up to you. Hazelcast will call the overridden methods from **com.hazelcast.core.MapStore<Long, Customer>**.

* 1. Graphical user interface

     Description automatically generated
  2. Text

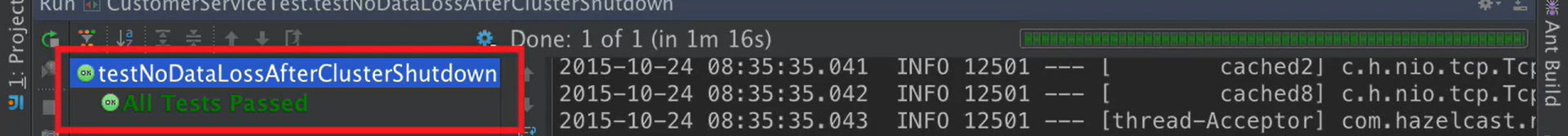
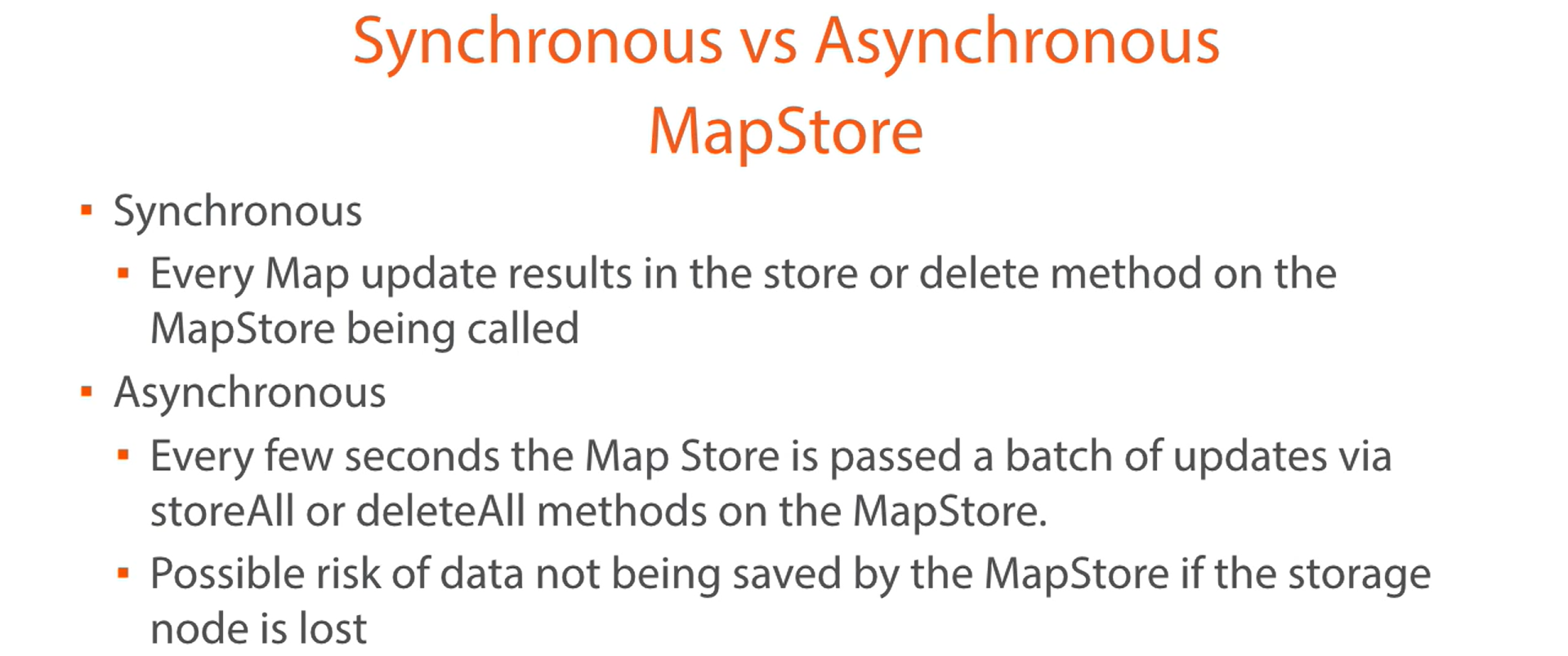
     Description automatically generated with medium confidence  
     
  3. In the below slide, there was a mistake. So, red text is the **rectification**.  
     Text

     Description automatically generated  
     Actually, the strategy in the above explanation allows for really fast loading of the initial data into the cluster from persistent store.

1. What you want to use as a persistent store is up to you.
2. For simplicity, I am using SQL DB & I am using JPA to access the data within the DB.
3. 
4. Now, we have Customer **MapStore** (**CustomerMapStore.java**) created. We need to inform **Hazelcast** about it & it is **Storage Nodes** responsible for storing data by calling the overridden methods from **com.hazelcast.core.MapStore<Long, Customer> (Concrete Implementation: CustomerMapStore.java)**
5. Configuration to inform Hazelcast to inform about the **MapStore implementation**.
   1. Similar to the Hazelcast Client configuration object, we have Config object for Storage Node (Cluster Members) Configuration.
   2. Text

      Description automatically generated
      1. In the above slide, we create Config object that represents the overall configuration for our cluster members.
      2. Then we create **MapConfig.java** object for my Customers Map.
      3. Then we create **MapStoreConfig.java** and set the implementation to be **CustomerMapStore.java** object we received as parameter.
      4. Brief:
         1. **MapStoreConfig:** To set **MapStore** implementation.
         2. **MapConfig** will contain **MapStoreConfig**.
         3. **Config** will contain **MapConfig**.
   3. Now, next step is to inform Storage Node about the MapStore (Concrete Implementation: CustomerMapStore.java)  
      Go to the StorageNodeApplication.java where we’re creating **HazelcastInstance** (which is basically **Storage Node**).  
      Text

      Description automatically generated
6. Text

   Description automatically generated
7. 
8. When setting up a MapStore, you can configure it to write to the MapStore either immediately or asynchronously in a **write behind** fashion.
9. When using **synchronous write**, each individual insert, update or delete on the map cause the operation run immediately on the store.   
   That is why your IMap and Persistent Store are in sync.
10. In the ansynchronous method, the changes are not written immediately to the MapStore.  
    Instead, they are batched together and storeAll(), deleteAll() are called a short while later.  
    
11. Text

    Description automatically generated  
    So, it is the choice you have to make based on performance or risk.