GUAVA - CACHING UTILITIES

Guava provides a very powerful memory based caching mechanism by an interface LoadingCache<K,V>. Values are automatically loaded in the cache and it provides many utility methods useful for caching needs.

Interface Declaration

Following is the declaration for **com.google.common.cache.LoadingCache<K,V>** interface:

```
@Beta
@GwtCompatible
public interface LoadingCache<K,V>
   extends Cache<K,V>, Function<K,V>
```

Interface Methods

Sr.No	Method & Description
1	**V apply***Kkey* Deprecated. Provided to satisfy the Function interface; use get*K* or getUnchecked*K* instead.
2	**ConcurrentMap<K,V> asMap** Returns a view of the entries stored in this cache as a thread-safe map.
3	**V get***Kkey* Returns the value associated with key in this cache, first loading that value if necessary.
4	**ImmutableMap<K,V> getAll***Iterable < ?extendsK > keys* Returns a map of the values associated with keys, creating or retrieving those values if necessary.
5	**V getUnchecked***Kkey* Returns the value associated with key in this cache, first loading that value if necessary.
6	**void refresh***Kkey* Loads a new value for key, possibly asynchronously.

Example of LoadingCache

Create the following java program using any editor of your choice in say **C:/> Guava.**

GuavaTester.java

```
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
```

```java
import com.google.common.base.MoreObjects;
import com.google.common.cache.CacheBuilder;
import com.google.common.cache.CacheLoader;
import com.google.common.cache.LoadingCache;

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

      //create a cache for employees based on their employee id
      LoadingCache<String, Employee> employeeCache =
         CacheBuilder.newBuilder()
            .maximumSize(100) // maximum 100 records can be cached
            .expireAfterAccess(30, TimeUnit.MINUTES) // cache will expire after 30
minutes of access
            .build(new CacheLoader<String, Employee>(){ // build the cacheloader

               @Override
               public Employee load(String empId) throws Exception {
                  //make the expensive call
                  return getFromDatabase(empId);
               }
            });

      try {
         //on first invocation, cache will be populated with corresponding
         //employee record
         System.out.println("Invocation #1");
         System.out.println(employeeCache.get("100"));
         System.out.println(employeeCache.get("103"));
         System.out.println(employeeCache.get("110"));

         //second invocation, data will be returned from cache
         System.out.println("Invocation #2");
         System.out.println(employeeCache.get("100"));
         System.out.println(employeeCache.get("103"));
         System.out.println(employeeCache.get("110"));

      }catch (ExecutionException e) {
         e.printStackTrace();
      }
   }

   private static Employee getFromDatabase(String empId) {

      Employee e1 = new Employee("Mahesh", "Finance", "100");
      Employee e2 = new Employee("Rohan", "IT", "103");
      Employee e3 = new Employee("Sohan", "Admin", "110");

      Map<String, Employee> database = new HashMap<String, Employee>();

      database.put("100", e1);
      database.put("103", e2);
      database.put("110", e3);

      System.out.println("Database hit for" + empId);

      return database.get(empId);
   }
}

class Employee {
   String name;
   String dept;
   String emplD;

   public Employee(String name, String dept, String empID) {
      this.name = name;
      this.dept = dept;
      this.emplD = empID;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getDept() {
      return dept;
   }

   public void setDept(String dept) {
      this.dept = dept;
   }

   public String getEmplD() {
      return emplD;
   }

   public void setEmplD(String emplD) {
      this.emplD = emplD;
   }

   @Override
   public String toString() {
      return MoreObjects.toStringHelper(Employee.class)
      .add("Name", name)
      .add("Department", dept)
      .add("Emp Id", emplD).toString();
   }
}
```

Verify the Result

Compile the class using **javac** compiler as follows:

```
C:\Guava>javac GuavaTester.java
```

Now run the GuavaTester to see the result.

```
C:\Guava>java GuavaTester
```

See the result.

```
Invocation #1
Database hit for100
Employee{Name=Mahesh, Department=Finance, Emp Id=100}
Database hit for103
Employee{Name=Rohan, Department=IT, Emp Id=103}
Database hit for110
Employee{Name=Sohan, Department=Admin, Emp Id=110}
Invocation #2
Employee{Name=Mahesh, Department=Finance, Emp Id=100}
Employee{Name=Rohan, Department=IT, Emp Id=103}
Employee{Name=Sohan, Department=Admin, Emp Id=110}
```