Java Object Cache 1.0 Component Specification

1. Design

An existing standalone Java batch job has a flaw in that it that it performs an expensive operation in order to build an object and then it discards that object, not knowing if it will need that same object on the next iteration.
This component design is for an object caching mechanism which will cache objects during the execution of the job.

1.1 Design Patterns

1.1.1 Strategy

CacheManagerImpl uses CachedObjectDAO, CacheStatisticsPrinter and MemoryCache implementations which are pluggable.

In addition, this component defines the interface CacheManager which can be used by external code as a strategy.

1.1.2 DAO

CachedObjectDAO is the DAO for the DTO CachedObject.

1.1.3 DTO

CachedObject and CacheStatistics are both DTO.

1.2 Industry Standards

JDBC
SQL

1.3 Required Algorithms

Please refer to TCUML method doc for details not listed below.

1.3.1 Logging

Logging will happen in all public methods of CacheManagerImpl, but not in setters/getters.

- Method entrance and exit will be logged with DEBUG level.
  Entrance format: [Entering method {className.methodName}]
  Exit format: [Exiting method {className.methodName}]. Only do this if there are no exceptio ns.
  Also note the time of the method call duration. Which would be the time from entrance to exit.
- Method request and response parameters will be logged with DEBUG level
  Format for request parameters: [Input parameters[{request_parameter_name_1}: { request_parameter_value_1}, {request_parameter_name_2}: { request_parameter_value_2}, etc.)]]

  Format for the response: [Output parameter {response_value}]. Only do this if there are no e xceptions and the return value is not void.
- All exceptions will be logged at ERROR level, and automatically log inner exceptions as well.
  Format: Simply log the text of exception: [Error in method {className.methodName}: Details {error details}]

In general, the order of the logging in a method should be as follows:
1. Method entry
2. Log method entry
3. If error occurs, log it and skip to step5
4. Log method exit
5. Method exit

1.4 Component Class Overview

com.ibm.support.electronic.cache:

CacheManager(interface):

This interface defines the contract for managing an object cache. It provides method to get objects from cache, put objects into cache, and clear the cache. In addition, it provides a way to get and print out cache statistics. Before using the cache, the clear() method should first be called to remove any old data in the persistence storage.

Thread Safety:
The implementation of this interface must be thread-safe

MemoryCache(interface):

This interface defines the contract for managing the in-memory cache of the component. It defines the method to put, get, and remove CachedObject to/from an in-memory storage it maintains. It provides the method to clear the memory cache. In addition and most importantly, it provides the getReplaceableCachedObject() method that acts as the replacement strategy that picks the element to remove from the memory cache during memory shortage. Note that this interface doesn't throw any exception. This is intended because the memory cache should be as simple as possible and the strategy for determining the next replaceable object should be based on some kind of calculation, which should not throw exception. Implementations must have a constructor that takes a Properties object as the sole parameter.

Thread Safety:
The implementation of this interface does not have to be thread-safe

CacheStatisticsPrinter(interface):

This interface defines the contract for printing cache statistics to a print writer.
Implementations should have a 0-argument constructor.
Thread Safety:
The implementation of this interface does not have to be thread-safe

CachedObjectDAO(interface):

This interface defines the methods to get, save (including insert/update depending on if the row already exists), delete a CachedObject from persistence storage, as well as to delete all CachedObject instances of one particular cache set from persistence storage.
Implementations must have a constructor that takes a Properties object as the sole parameter.

Thread Safety:
The implementation of this interface does not have to be thread-safe

com.ibm.support.electronic.cache.cachemanager:

CacheManagerImpl:

This is the default implementation of CacheManager. The object to put into the cache must implement Serializable, and must be responsible for its own serialization (writing its own readObject() and writeObject()) if the serialization of its own involves custom behaviors. It uses MemoryCache to manage the in-memory cache and CachedObjectDAO to manage the database table. MemoryCache also provides the replacement strategy for picking the element to remove from memory cache and put to database when the memory cache is out of configured memory.

Thread Safety:
This class is thread-safe because it's properly synchronized. The synchronization guarantees that the mutable state of the class is always modified by one single thread at a time.

com.ibm.support.electronic.cache.dao:

CachedObjectDAOImpl:

This class is a JDBC implementation of CachedObjectDAO. It simply executes proper SQL statements to perform the operation. Auto-commit mode is turned off.

Thread Safety:
This class is thread-safe because it's immutable assuming the passed in parameters are used thread-safely by the caller.

com.ibm.support.electronic.cache.memorycache:

MemoryCacheImpl:

This is the MemoryCache implementation that uses Least Recently Used strategy to get the element to remove from memory cache. It maintains a LinkedHashMap to store the cached objects so that it can get the least recently used element from the map, which will be the element to remove. The LinkedHashMap can achieve this function because both its put and get methods result in a structural modification (see http://download.oracle.com/javase/6/docs/api/java/util/LinkedHashMap.html for details)

Thread Safety:
This class is not thread-safe because it has mutable state. The caller should use it with proper synchronization.

com.ibm.support.electronic.cache.model:

CachedObject:

This is a simple POJO that stores a serialized object in a cache set. The id and cacheSetName together identify an instance of this class. This component stores all cached objects in serialized form, and each cached object is serialized and wrapped in one instance of this class.

Thread Safety:
This class is not thread-safe because it's mutable

CacheStatistics:

This is a simple class that stores statistics about the cache. Its setter/getter doesn't perform validation.
Thread Safety:
This class is not thread-safe because it's mutable

com.ibm.support.electronic.cache.printer:

CacheStatisticsPrinterImpl:

This is the default implementation of CacheStatisticsPrinter. It simply prints out the properties of the passed in CacheStatistics one by one using the passed in print writer.
Thread Safety:
This class is not thread-safe because the passed in parameters are not thread-safe.

1.5 Component Exception Definitions

1.5.1 Custom exceptions

PersistenceException:

This is the thrown if any error occurs with the persistence storage. It's thrown by CacheManager(and its implementations), CachedObjectDAO(and its implementations).

Thread Safety:
This class is not thread-safe because the base class is not thread-safe.

ObjectCacheConfigurationException:

This is thrown if any configuration error occurs in this component. It's thrown by CachedObjectDAOImpl, CacheManagerImpl.

Thread Safety:
This class is not thread-safe because the base class is not thread-safe.

ObjectCacheException:

This is the base exception of all checked exception encountered by this component. It's thrown by CacheManager(and its implementations).

Thread Safety:
This class is not thread-safe because the base class is not thread-safe.

StatisticsPrintingException:

This is the thrown if any error occurs during printing cache statistics. It's thrown by CacheManager(and its implementations), CacheStatisticsPrinter(and its implementations).

Thread Safety:
This class is not thread-safe because the base class is not thread-safe.

1.5.2 System exceptions

IllegalArgumentException:

This will be used to signal that an input parameter to the component is illegal.

1.6 Thread Safety

This component is thread-safe because CacheManager requires implementations to be thread-safe and CacheManagerImpl is thread-safe as stated in 1.4.

2. Environment Requirements

2.1 Environment

Development language: Java1.5

Compile target: Java1.5

DB2 v9+

2.2 TopCoder Software Components

None

2.3 Third Party Components

Apache Log4j 1.2.15 (http://logging.apache.org/log4j/1.2/index.html)

3. Installation and Configuration

3.1 Configuration Parameters

Configuration for CacheManagerImpl:

Name	Description	Value
cachedObjectDAOClass	The full class name of the CachedObjectDAO instance used by CacheManagerImpl. Default to com.ibm.support.electronic.cache.dao.CachedObjectDAOImpl	String. Not empty. Optional.
maxMemorySize	The max memory size in kilobytes. Default to 0.	Integer string. It must be non-negative. Optional.
memoryCacheClass	The full class name of the MemoryCache instance used	String. Not empty. Optional.

Name	Description	Value
	by CacheManagerImpl. Default to com.ibm.support.electronic.cache.memorycache.MemoryCacheImpl	
cacheSetName	The name of the cache set this cache manager operates on. Default to “default”.	String. Not empty. Optional.
cacheStatisticsPrinterClass	The full class name of the CacheStatisticsPrinter instance used by CacheManagerImpl. Default to com.ibm.support.electronic.cache.printer.CacheStatisticsPrinterImpl	String. Not empty. Optional.

Configuration for CachedObjectDAOImpl:

Name	Description	Value
user	The username used for connecting to the database. Default to an empty string. If it’s empty, authentication is not performed.	String. Not empty. Optional.
password	The password used to connect the database. Default to an empty string.	String. Not empty. Optional.
url	The connection string used when establishing a database connection.	String. Not empty. Required.
driver	The full JDBC driver class name. Default to “com.ibm.db2.jcc.DB2Driver”.	String. Not empty. Optional.

Configuration for MemoryCacheImpl:

Name	Description	Value
initialCapacity	The initial capacity of the LinkedHashMap used to store objects. Default to 1000.	Integer String. Not empty. Optional.
loadFactor	The load factor of the LinkedHashMap used to store objects. Default to 0.75.	Float String. Not empty. Optional.

3.2 Dependencies Configuration

Log4j must be configured properly.
The table given in the attached “DDL.txt” should be set up in the database.

3.3 Package Structure

com.ibm.support.electronic.cache
com.ibm.support.electronic.cache.cachemanager
com.ibm.support.electronic.cache.dao
com.ibm.support.electronic.cache.memorycache
com.ibm.support.electronic.cache.model

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute ‘ant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

Please see the demo.

4.3 Demo

4.3.1 Demo Input

Assuming we have the following class to cache:

```
public class Foo implements Serializable {
    private String name;
    private int age;
    // This is for making this POJO larger for demo purpose
    private byte[] bytes = new byte[1000];

    public String getName() {
        return name;
    }

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

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public byte[] getBytes() {
        return bytes;
    }

    public void setBytes(byte[] bytes) {
        this.bytes = bytes;
    }
}
```

The sample configuration file this demo uses is the attached “config.properties”.

4.3.2 API Usage

```
// Create a CacheManagerImpl with default configuration file
CacheManager manager = new CacheManagerImpl();
// Create a CacheManagerImpl with a given configuration file
CacheManager manager2 = new CacheManagerImpl("config.properties");
// Assuming this Properties object contains configuration parameters
Properties prop = new Properties();
// Create a CacheManagerImpl with a given Properties object
CacheManager manager3 = new CacheManagerImpl(prop);
// Create a CacheManagerImpl with a given Properties object and a cache set name
CacheManager manager4 = new CacheManagerImpl(prop, "Set 1");

// First clear the cache to delete any possible old data in persistence storage
manager.clear();

// Get an object that doesn't exist in the cache
Object object = manager.get("fooA");
```

```
// It should be null
assertNull(object);

// Create a Foo
Foo foo1 = new Foo();
foo1.setAge(10);
foo1.setName("fooA");

// Put it into the cache (this one will go to memory cache)
manager.put("fooA", foo1);
// Get it from cache
Foo fooFromCache = (Foo) manager.get("fooA");
// Its values should be the same as foo1
assertEquals(foo1.getAge(), fooFromCache.getAge());
assertEquals(foo1.getName(), fooFromCache.getName());

// Create another Foo
Foo foo2 = new Foo();
foo2.setAge(20);
foo2.setName("fooB");

// Put it into the cache (this one will go to memory cache)
manager.put("fooB", foo2);

// Get it from cache
fooFromCache = (Foo)manager.get("fooB");
// Its values should be the same as foo2
assertEquals(foo2.getAge(), fooFromCache.getAge());
assertEquals(foo2.getName(), fooFromCache.getName());

// Create a PrintWriter
PrintWriter pw = new PrintWriter(System.out);
// Print the current statistics
manager.printStatistics(pw);
```

The following content will be printed:

```
Number of items in the cache: 2
Keys in the cache:
Key fooB is accessed 1 times.
Key fooA is accessed 1 times.
Number of items in memory: 2
Number of items in persistent storage: 0
Number of misses: 1
```

The one miss is caused by “manager.get("fooA")” at the beginning of this demo.

```
// Create another Foo
Foo foo3 = new Foo();
foo3.setAge(30);
foo3.setName("fooC");

// This one will go directly to database because the 3 cached Foo require 1138 * 3 bytes, which is
larger than the configured 3K in-memory cache
manager.put("fooC", foo3);

// At this point, the table CACHED_OBJECT will have a row with id=fooC and
cache_set_name=default

// Now accessing fooC will force the least recently used in-memory object fooA to be removed from
memory cache (because manager uses MemoryCacheImpl, which uses Least Recently Used
replacement strategy) and put to database, and table CACHED_OBJECT will have a new row with
id=fooA and cache_set_name=default. Note that because fooC is moved from database to memory
```

cache, its corresponding row is deleted. So at this time, CACHED_OBJECT only has one row (for fooA)
```
fooFromCache = (Foo)manager.get("fooC");
```

// Print the current statistics
```
manager.printStatistics(pw);
```
// Remember to close the print writer when it's no longer in use
```
pw.close();
```

The following content will be printed:
```
Number of items in the cache: 3
Keys in the cache:
Key fooB is accessed 1 times.
Key fooC is accessed 1 times.
Key fooA is accessed 1 times.
Number of items in memory: 2
Number of items in persistent storage: 1
Number of misses: 1
```

// We can also get a CacheStatistics instead of printing it out
```
CacheStatistics statistics = manager.getCacheStatistics();
```
// Accessing its getter to get details
```
System.out.println(statistics.getInMemoryItemCount());
```

// We can remove one particular cached object by setting it to null (the associated table row will be deleted)
```
manager.put("fooA", null);
assertNull(manager.get("fooA"));
```

// We can replace a cached object with another. Note that replacing it with the same key will result in the statistics for that key be reset. It’s like it’s a brand-new key.
```
manager.put("fooB", foo1);
fooFromCache = (Foo)manager.get("fooB");
```
// Now the retrieved Foo's values are the same as those of foo1
```
assertEquals(foo1.getAge(), fooFromCache.getAge());
assertEquals(foo1.getName(), fooFromCache.getName());
```

// Clear the cache set (the associated table rows will be deleted)
```
manager.clear();
```
// After clearing the cache, there's nothing in the cache
```
assertNull(manager.get("fooB"));
assertNull(manager.get("fooC"));
```

5. Future Enhancements

None