
JavaOne™

JavaOne™

Java Caching: State of the Union

Brian Oliver | Oracle Corporation
Greg Luck | Terracotta

MAKE THE FUTURE JAVA

ORACLE

ORACLE

Program Agenda

- Java Caching (JCache), JSR-107 and Caching
- JCache: More than your average Cache!
- JCache: By Example
- Available Implementations?

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Java Caching (JCache)

TERRACOTTA JavaOne ORACLE

Java Caching (JCache)

- **What?**
  - Java Caching (JCache) is an effort to standardize Caching for the Java Platform*
  - A common mechanism to create, access, update and remove information from Caches
- **How?**
  - JSR-107: Java Caching Specification (JCache)
  - Java Community Process (JCP) 2.9

Java Caching (JCache)

- **Why?**
  - Standardize! Standardize! Standardize!
    - Core Caching Concepts
    - Core Caching API
  - Provide application portability between Caching solutions
    - Big & Small, Open & Commercial
  - Caching is ubiquitous!

Java Caching (JCache)

- **Who?**
  - Joint Specification (LEADS)
    - Greg Luck
    - Brian Oliver (Oracle Corporation)
  - Expert Group (EG)
    - 10+ companies
    - 8+ individuals

Java Caching (JCache)

- **When?** (A Proposed Timetable)

Deliverable	Start	Finish
Public Review Ballot ✔	27th August 2013	9th September 2013
Proposed Final Draft		30th September 2013
Completion of Reference Implementation (RI) & Technology Compatibility Kit (TCK)		31st October 2013
Appeal Ballot (7 days)	31st October 2013	7th November 2013
Updated Deliverables	7th November 2013	14th November 2013
Final Approval Ballot	14th November 2013	28th November 2013
Final Release	**28th November 2013**	**12th December 2013**

Java Caching (JCache)

- **Which Platform?**

JCache Deliverable	Target Platform
Specification (SPEC)	Java 6+ (SE or EE)
Reference Implementation (RI)	Java 7+ (SE or EE)
Technology Compatibility Kit (TCK)	Java 7+ (SE or EE)
Demos and Samples	Java 7+ (SE or EE)

Java Caching (JCache)

Project Hosting

- JCP Project:
  - http://jcp.org/en/jsr/detail?id=107
- Source Code:
  - https://github.com/jsr107
- Forum:
  - https://groups.google.com/forum/?fromgroups#!forum/jsr107

Java Caching (JCache)

Java Community Process

How to get it.

Apache Maven: (via Maven Central Repository)

```
<dependency>
  <groupId>javax.cache</groupId>
  <artifactId>cache-api</artifactId>
  <version>0.10</version>
</dependency>
```


TERRACOTTA JavaOne ORACLE

Caches and Caching

TERRACOTTA JavaOne ORACLE

Caches and Caching

- **Cache:** A high-performance, low-latency data-structure* in which an application places a temporary copy of information that is likely to be used more than once

- **When To Use Caches?**
  - When applications use the same data more than once
  - When cost (time / resources) of making an initial copy is less than fetching or producing the data again or when faster to request from a Cache

TERRACOTTA JavaOne ORACLE

Caches and Caching

- **Implications?**
  - Caching is not a cure all!
  - Developers must know the costs (time and resources) to determine Cache effectiveness

Caches and Caching: Maps v's Cache APIs

Maps

- Key-Value Based API
- Supports Atomic Updates
- **Entries Don't Expire**
- **Entries Aren't Evicted**
- Entries Stored On-Heap

Caches

- Key-Value Based API
- Supports Atomic Updates
- **Entries May Expire**
- **Entries May Be Evicted**
- Entries Stored Anywhere (i.e.: topologies)
- Support Integration (through Loaders / Writers)
- Support Listeners (observer pattern)
- Entry Processors
- Statistics

Caches are not Maps!

TERRACOTTA JavaOne ORACLE

JCache: More than your average Cache!

TERRACOTTA JavaOne ORACLE

JCache: Features

- `java.util.ConcurrentMap` like API
- Atomic Operations
- Lock-Free
- Read-Through / Write-Through Integration Support
- Cache Event Listeners
- Fully Generic API = type-safety
- Statistics
- Annotations (for frameworks and containers)
- Store-By-Value semantics (optional store-by-reference)

JCache: Features

- Topology Agnostic
  - Topologies not defined or restricted by the specification
- Efficiently supports:
  - “local” in-memory Caching and
  - “distributed” server-based Caching

JCache: G’day World

```
// acquire a previously configured cache
Cache<Integer, String> cache =
    Caching.getCache(“my-cache”, Integer.class, String.class);

// put something in the cache
cache.put(123, “G’day World”);

// get something from the cache
String message = cache.get(123);
```

API In Depth

TERRACOTTA JavaOne ORACLE

JCache: Cache API

(does not extend Map!)

```
public interface Cache<K, V>
    extends Iterable<Cache.Entry<K, V>> {

    V get(K key);
    Map<K, V> getAll(Set<? extends K> keys);
    boolean containsKey(K key);

    void loadAll(Set<? extends K> keys,
                 CompletionListener l);

    ...
```

JCache: Cache API

```
void put(K key, V value);
V getAndPut(K key, V value);
void putAll(Map<? extends K, ? extends V> map);
boolean putIfAbsent(K key, V value);

boolean remove(K key);
boolean remove(K key, V oldValue);
V getAndRemove(K key);
```

JCache: Cache API

```
boolean replace(K key, V oldValue, V newValue);
boolean replace(K key, V value);
V getAndReplace(K key, V value);

void removeAll(Set<? extends K> keys);
void removeAll();

void clear();

Configuration<K, V> getConfiguration();
```

JCache: Cache API

```
void registerListener(
    CacheEntryListenerConfiguration<K, V> config);

void unregisterListener(
    CacheEntryListenerConfiguration<K, V> config);
```

JCache: Cache API

```
<T> T invoke(K key,
             EntryProcessor<K, V, T> processor,
             Object... arguments);

<T> Map<K, T> T invokeAll(Set<? Extends K> keys,
                          EntryProcessor<K, V, T> processor,
                          Object... arguments);
```

JCache: Cache API

```
    String getName();

    CacheManager getCacheManager();
}
```

JCache: Cache Managers

`javax.cache.CacheManager`

- Establishes, configures, manages and owns named Caches
  - Caches may be pre-define or dynamically created at runtime
- Provides Cache infrastructure and resources
- Provides Cache “scoping” (say in a Cluster)
- Provides Cache ClassLoaders (important for store-by-value)
- Provides Cache lifecycle management

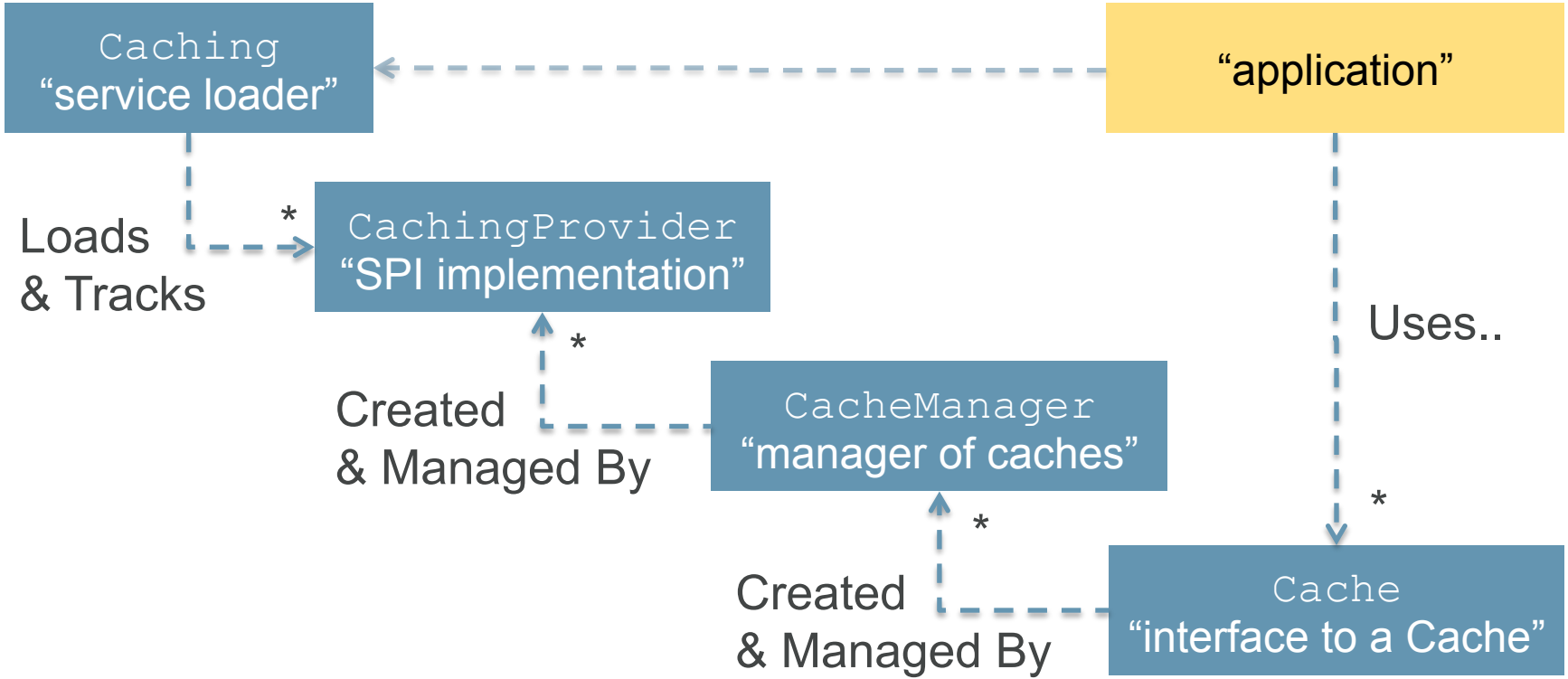
JCache: G'day World

(via a Cache Manager)

```
// acquire the default CacheManager
CacheManager manager = Caching.getCacheManager();

// acquire a previously configured cache (via CacheManager)
Cache<Integer, String> cache =
    manager.getCache("my-cache", Integer.class, String.class);

// put something in the cache
cache.put(123, "G'day World");

// get something from the cache
String message = cache.get(123);
```


TERRACOTTA JavaOne ORACLE

JCache: Runtime Structure



TERRACOTTA JavaOne ORACLE

JCache: Gudday World

(using programmatic configuration – fluent style)

```
MutableConfiguration<Integer, String> config =

    new MutableConfiguration<Integer, String>()

        .setStoreByReference(true)

        .setCacheEntryExpiryPolicy(
            new AccessedExpiryPolicy(5, TimeUnit.SECONDS));
```

JCache: Gudday World

(using programmatic configuration – fluent style)

```
// acquire the default CacheManager
CacheManager manager = Caching.getCacheManager();

// create cache with a custom configuration
Cache<Integer, String> cache =
    manager.createCache("my-cache", config);

// and perhaps later just…
Cache<Integer, String> cache =
    manager.getCache("my-cache", Integer.class, String.class);
```

Entry Processors

TERRACOTTA JavaOne ORACLE

JCache: Entry Processors

(custom atomic operations for everyone!)

```
// acquire a cache
Cache<String, Integer> cache =
    manager.getCache("my-cache", String.class, Integer.class);

// increment a cached value by 42, returning the old value
int value = cache.invoke("key", new IncrementProcessor<>(), 42);
```


TERRACOTTA JavaOne ORACLE

JCache: Entry Processors

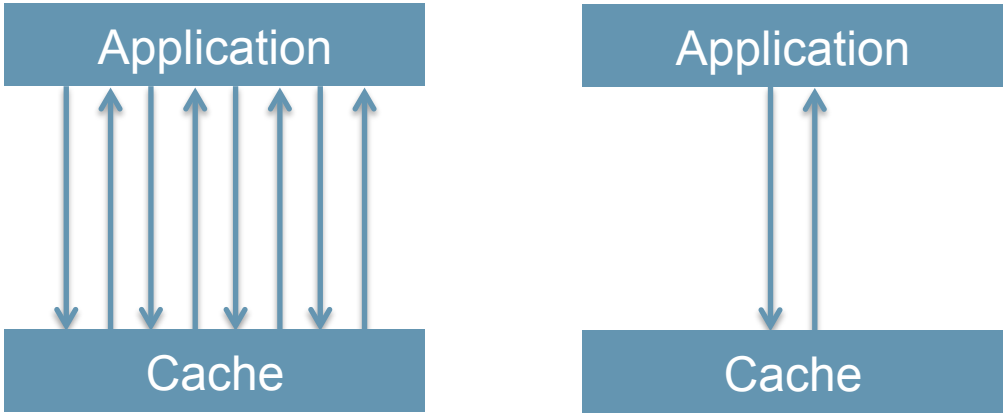
(custom atomic operations for everyone)

```
public class IncrementProcessor<K>
    implements EntryProcessor<K, Integer, Integer>, Serializable {

    @Override
    public Integer process(MutableEntry<K, Integer> entry, Object... arguments) {
        if (entry.exists()) {
            int amount = arguments.length == 0 ? 1 : (Integer)arguments[0];
            int current = entry.getValue();
            entry.setValue(count + amount);
            return current;
        } else {
            throw new IllegalStateException("no entry exists");
        }
}
```

JCache: Entry Processors

(custom atomic operations!)

- Eliminate Round-Trips! (in distributed systems)



- Enable development of a Lock-Free API! (simplifies applications)
- *May need to be Serializable (in distributed systems)

TERRACOTTA JavaOne ORACLE

JCache: Entry Processors

Which is better?


```
// using an entry processor?
int value = cache.invoke("key", new IncrementProcessor<>(), 42);


// using a lock based API?
cache.lock("key");
int current = cache.get("key");
cache.put("key", current + 42);
cache.unlock("key");
```


TERRACOTTA JavaOne ORACLE

Annotations

- JSR107 introduces a standardized set of caching annotations, which do method level caching interception on annotated classes running in dependency injection containers.
- Caching annotations are becoming increasingly popular:
  - Ehcache Annotations for Spring
  - Spring 3's caching annotations.
- JSR107 Annotations will be added to:
  - Java EE 8
  - Spring 4 (2014)

Annotation Operations

- The JSR107 annotations cover the most common cache operations:

- `@CacheResult`
- `@CachePut`
- `@CacheRemove`
- `@CacheRemoveAll`

Fully Annotated Class Example

```
@CacheDefaults(cacheName = "blogManager")
public class BlogManager {
    @CacheResult
    public Blog getBlogEntry(String title) {...}

    @CacheRemove
    public void removeBlogEntry(String title) {...}

    @CacheRemoveAll
    public void removeAllBlogs() {...}

    @CachePut
    public void createEntry(@CacheKey String title, @CacheValue Blog blog) {...}

    @CacheResult
    public Blog getEntryCached(String randomArg, @CacheKey String title){...}
}
```

Specific Overrides

```
public class DomainDao {

  @CachePut(cacheName="domainCache")
  public void updateDomain(String domainId,
      @CacheKey int index,
      @CacheValue Domain domain) {
      ...
    }
 }
```

Announcements

TERRACOTTA JavaOne ORACLE

Fully Compliant JCache early 2014.

Oracle Coherence

Fully Compliant JCACHE in 2014

TERRACOTTA JavaOne ORACLE

MAKE THE FUTURE JAVA

JavaOne™

ORACLE®

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.


JavaOne™