

# Using the new javax.cache Caching Standard

Greg Luck, CTO Terracotta  
Cameron Purdy, Founder Coherence and VP, Oracle

JavaOne 2012

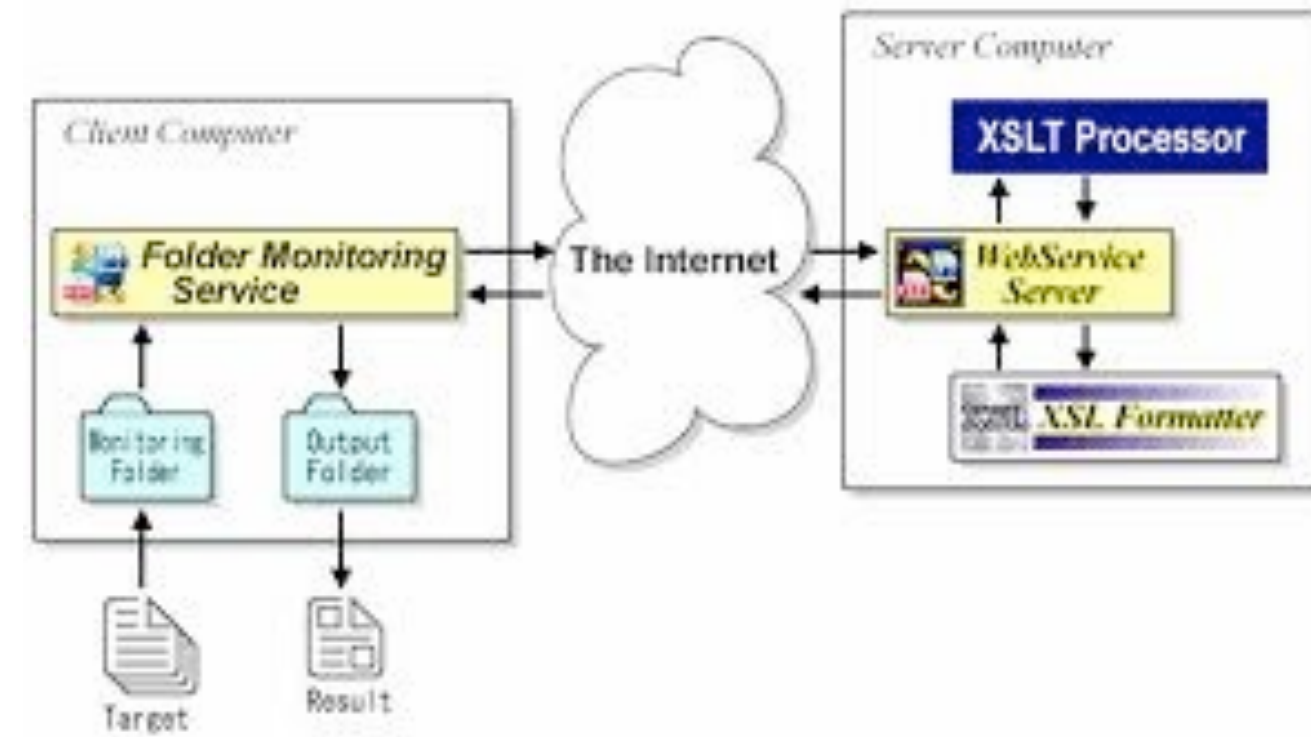
Session CON11546

# Typical Bottlenecks

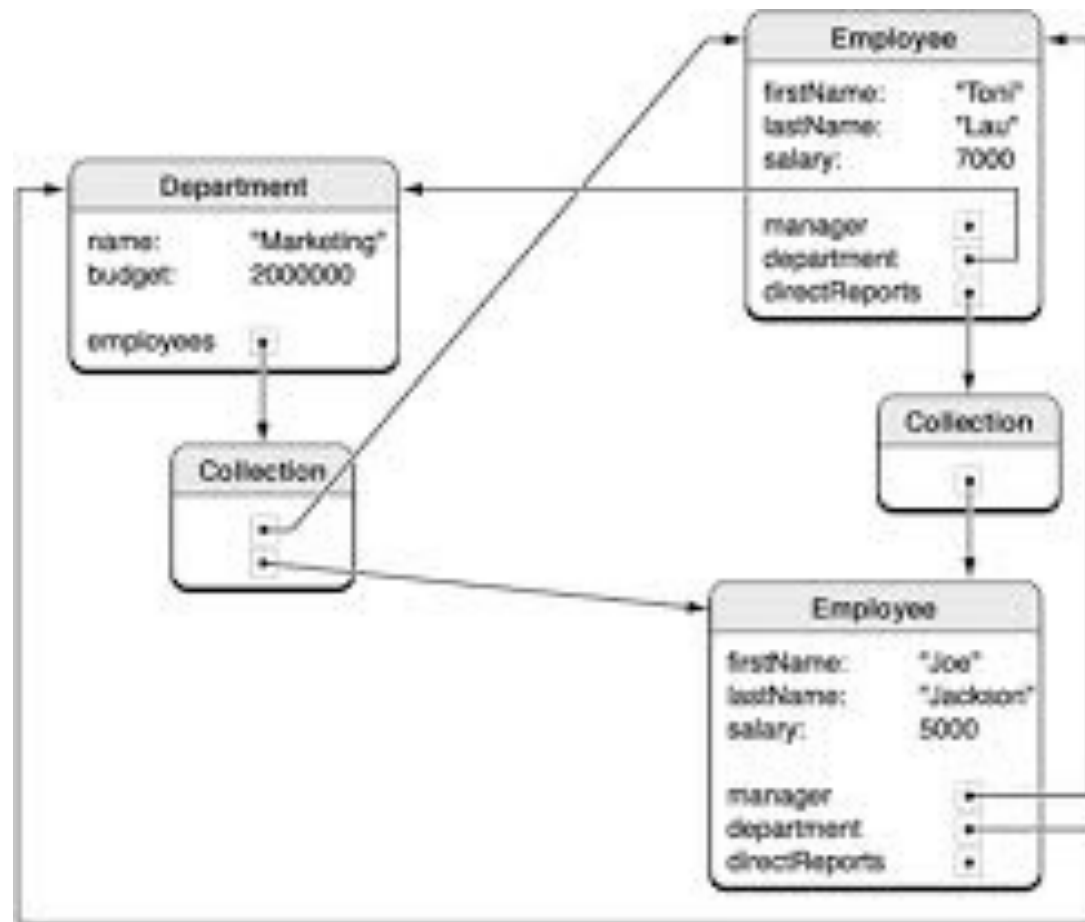
1



3



2

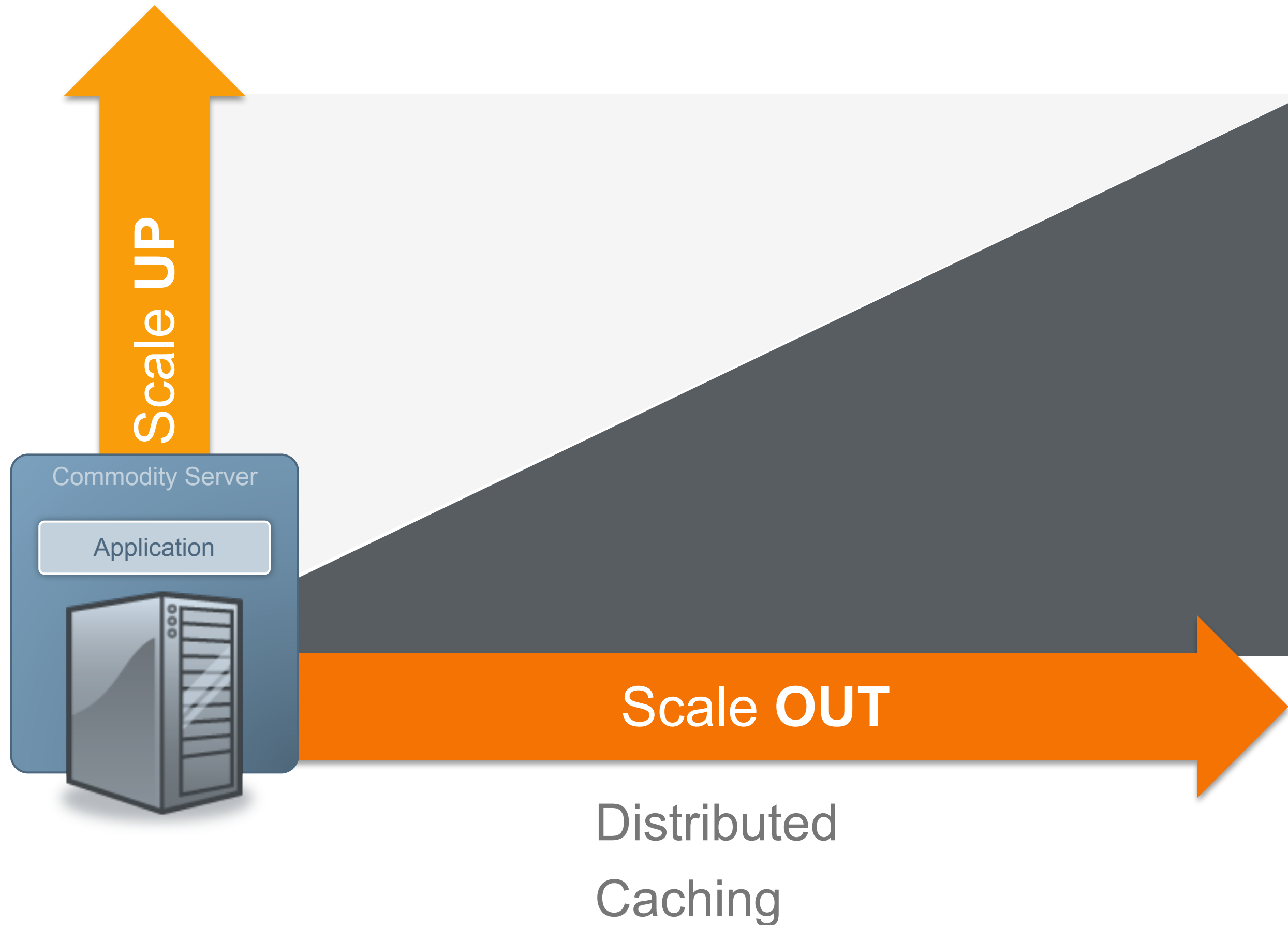


4



# Types of Scaling    Types of Caching

Standalone  
Caching  
(in-process)



# Caching Characteristics

- Fastest To Implement
- Offload
- Performance
- Scale up
- Scale out (Distributed Caches Only)
- Buffer against load variability

# JSR107: Java Caching Standard

- `javax.cache.Cache`
- Being developed by JSR107
- Java 6 and above is required
- Included in JSR 342: Java EE 7 due end of 2012
- Immediately usable by Java EE 6 and Spring
- Immediately usable by any Java based app

# Open, Transparent Standards Approach

- Terracotta and Oracle have tasked an FTE (Greg and Yannis) with developing the spec
- Developed in the open
- 15 expert group members
- Lots of healthy debate. See the mailing list: [jsr107@googlegroups.com](mailto:jsr107@googlegroups.com)
- Specification is standard spec license - free to use and implement
- Reference Implementation is Apache 2
- Tests which is the major part of the TCK is Apache 2

# Expected Implementations

- Terracotta - Ehcache
- Oracle - Coherence
- JBoss - Infinispan
- IBM - ExtemeScale
- SpringSource - Gemfire
- GridGain
- TMax
- Google App Engine Java memcache client
- Spymemcache memcache client

# Getting Started

## API in Maven Central

```
<dependency>
```

```
  <groupId>javax.cache</groupId>
```

```
  <artifactId>cache-api</artifactId>
```

```
  <version>0.3-SNAPSHOT</version>
```

```
</dependency>
```

## Everything to get started

<https://github.com/jsr107/jsr107spec>



# Key Concepts

- CacheManager => Caches
- Cache => Entries
- Entry => Key, Value
- The basic API can be thought of map--like with the following additional features:
- atomic operations, similar to `java.util.ConcurrentMap`
- read-through caching
- write-through caching
- cache event listeners
- statistics

# API Features

Map-like with the following additional features:

- atomic operations, similar to `java.util.ConcurrentMap`
- read-through caching
- write-through caching
- cache event listeners
- statistics
- transactions including all isolation levels
- caching annotations
- generics

# How to Please Everyone - No Dependencies

- Java SE - no dependencies.
- EE/Spring - provided dependencies - they are already there.

# How to Please Everyone - Optional Features

Optional Features are:

- `storeByReference`
- XA and Local Transactions
- Caching Interceptor Annotations e.g.

Options interrogation at runtime via Capabilities API:

- `ServiceProvider.isSupported(OptionalFeature feature)`
- `CacheManager.isSupported(OptionalFeature feature)`

Works for implementers and Users

# Aimed at Standalone and Distributed Caching

## Standalone Features

- `storeByReference` - allows speeds similar to CHM
- `CacheEventListener` callbacks - useful for triggering events

## Distributed Features

- `storeByValue`
- `NotificationScope` in `CacheEventListener`
- modifications/differences to `Map` and `ConcurrentHashMap` to reduce network cost. e.g.
  - `No values()` and many others.
  - Calls may not return a value e.g. `remove(Object key)` returns `boolean` rather than the old value

# Not a Data Grid Specification

- Infinispan, Coherence and Extreme Scale are Data Grids
- Ehcache and Memcache are distributed client-server caches
- NoSQL key value stores are distributed client-server key stores which could be used for caching

So:

- JSR107 does not mandate a topology
- JSR347 does - it is for data grids and builds on JSR107

# Classloading

- Caches contain data *shared* by multiple threads/JVMs which may be using Java SE, EE, OSGi or custom class loading.
- This makes class loading tricky
- A classloader can be specified when the CacheManager is created or a default is used. Either way all classes will be loaded by the CacheManager's classloader, not the environment's classloader.
- `public static CacheManager getCacheManager(ClassLoader classLoader)`
- `public static CacheManager getCacheManager(ClassLoader classLoader, String name)`

# Creating a CacheManager

## ServiceLoader Creation

We support the Java 6 `java.util.ServiceLoader` creational approach. It will automatically detect a cache implementation in your classpath. You then create a `CacheManager` with:

```
CacheManager cacheManager = CacheManagerFactory.getCacheManager();
```

or more fully:

```
CacheManager cacheManager = CacheManagerFactory.getCacheManager("app1",  
Thread.currentThread().getContextClassLoader());
```

## “new” Creation

```
CacheManager cacheManager = new RICacheManager("app1",  
Thread.currentThread().getContextClassLoader());
```



# Creating a Cache

To programmatically configure a cache named “testCache” which is set for read-through

```
CacheManager cacheManager = getCacheManager();  
Cache testCache = cacheManager.createCacheBuilder("testCache")  
.setReadThrough(true).setSize(Size.UNLIMITED).  
.setExpiry(Duration.ETERNAL).build();
```

# Using a Cache

You get caches from the `CacheManager`. To get a cache called “testCache”:

```
Cache<Integer, Date> cache = cacheManager.getCache("testCache");
```

# Putting a value in a Cache

```
Cache<Integer, Date> cache = cacheManager.getCache(cacheName);  
Date value1 = new Date();  
Integer key = 1;  
cache.put(key, value1);
```

# Getting a Value

```
Cache<Integer, Date> cache = cacheManager.getCache(cacheName);  
Date value2 = cache.get(key);
```

# Removing a mapping

```
Cache<Integer, Date> cache = cacheManager.getCache(cacheName);  
Integer key = 1;  
cache.remove(1);
```

# Exposing the underlying Cache's API

## Unwrap Method on Cache

```
<T> T unwrap(java.lang.Class<T> cls);
```

## Ehcache Example

```
net.sf.ehcache.Cache cache =  
javax.cache.cache.unwrap(net.sf.ehcache.Cache.class);
```

# IDE API Review

# Annotations

JSR107 introduces a standardised 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.



# Annotation Operations

The JSR107 annotations cover the most common cache operations including:

- `@CacheResult`
- `@CachePut`
- `@CacheRemoveEntry`
- `@CacheRemoveAll`

# Specific Overrides

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

# Fully Annotated Class Example

```
public class BlogManager {  
    @CacheResult(cacheName="blogManager")  
    public Blog getBlogEntry(String title) {...}  
  
    @CacheRemoveEntry(cacheName="blogManager")  
    public void removeBlogEntry(String title) {...}  
  
    @CacheRemoveAll(cacheName="blogManager")  
    public void removeAllBlogs() {...}  
  
    @CachePut(cacheName="blogManager")  
    public void createEntry(@CacheKeyParam String title, @CacheValue Blog blog) {...}  
  
    @CacheResult(cacheName="blogManager")  
    public Blog getEntryCached(String randomArg, @CacheKeyParam String title){...}  
}
```

# Wiring Up Spring

```
<beans ...>
```

```
  <context:annotation-config/>
```

```
  <jcache-spring:annotation-driven proxy-target-class="true"/>
```

```
  <bean id="cacheManager" class="javax.cache.Caching"  
    factory-method="getCacheManager" />
```

```
  <bean class="manager.CacheNameOnEachMethodBlogManagerImpl"/>
```

```
  <bean class="manager.ClassLevelCacheConfigBlogManagerImpl"/>
```

```
  <bean class="manager.UsingDefaultCacheNameBlogManagerImpl"/>
```

```
</beans>
```

# Wiring Up CDI

1. Create an implementation of `javax.cache.annotation.BeanProvider`
2. Declare a resource named `javax.cache.annotation.BeanProvider` in the classpath at `/META-INF/services/`.

For an example using the Weld implementation of CDI, see the `CdiBeanProvider` in our CDI test harness.

# More Information

## Jumping Off Point to Everything Else

<https://github.com/jsr107/jsr107spec>

## Maven Snippet

```
<dependency>  
  <groupId>javax.cache</groupId>  
  <artifactId>cache-api</artifactId>  
  <version>0.x</version>  
</dependency>
```