



DZone (/) > Java Zone (/java-jdk-development-tutorials-tools-news) > [How to Synchronize Blocks by the Value of the Object in Java](#)

# How to Synchronize Blocks by the Value of the Object in Java


Ever had trouble synchronizing blocks of code?

 (/users/3174562/antkorwin.html) by [Anatoliy Korovin \(/users/3174562/antkorwin.html\)](#) · Jan. 08, 19 · [Java Zone \(/java-jdk-development-tutorials-tools-news\)](#) · Tutorial

 Like (25)  Comment (7)  Save  Tweet  35.66K Views

Join the DZone community and get the full member experience.  
[\(/STATIC/REGISTRATION.HTML\)](#)



**[Report] The Total Cost of Ownership for Cloud Databases**  
What are all the costs that you need to account for when evaluating a commitment to a cloud database? Learn how to eval cost of your database in this free guide. [Download for Free ►](#)  


## the problem

sometimes, we need to synchronize blocks of code by the value of a variable.

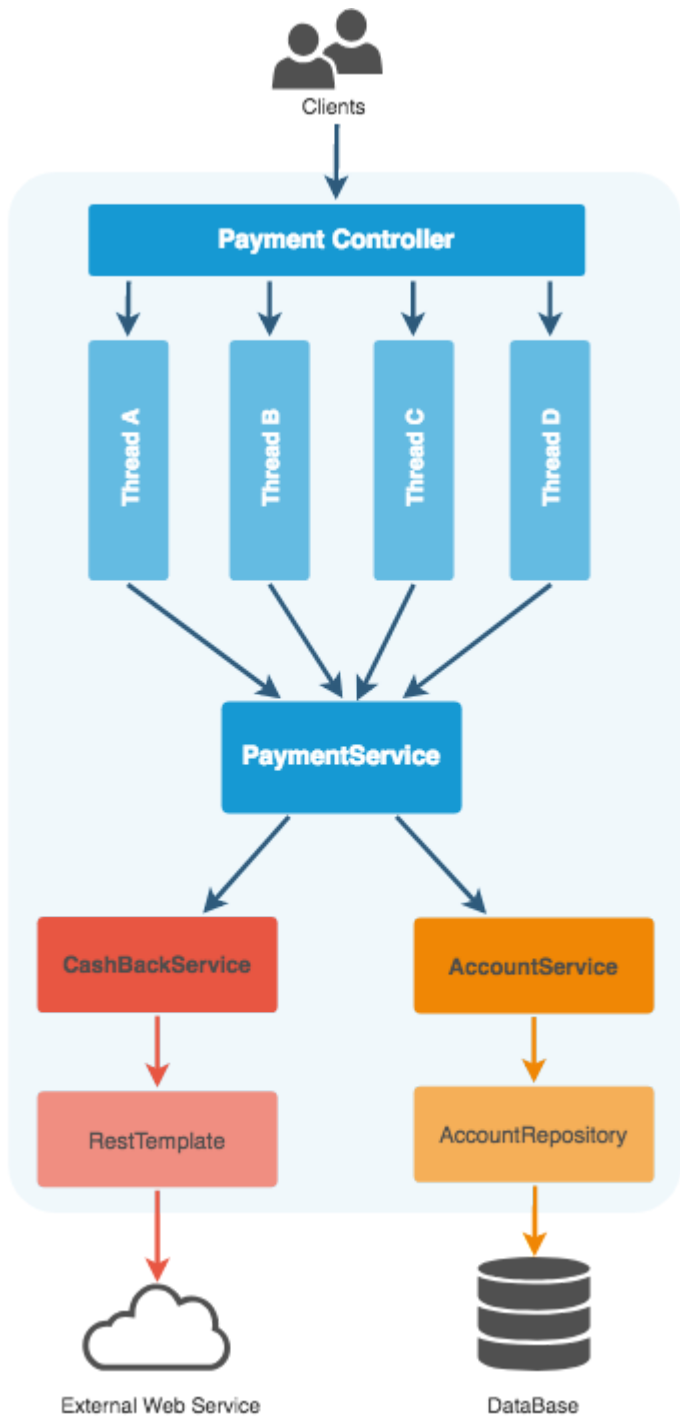
in order to understand this problem, we will consider a simple banking application that makes the following operations on each transfer of money by the client:

1. evaluates the amount of the cash back by this transfer from external web service ( `cashbackservice` )
2. performs a money transfer in the database ( `accountservice` )
3. updates the data in the cash-back evaluation system ( `cashbackservice` )

the money transfer operation looks like this:

```
1 public void withdrawmoney(uuid userid, int amountofmoney) {
2     synchronized (userid) {
3         result result = externalcashbackservice.evaluatecashback(userid, amountofmoney);
4         accountservice.transfer(userid, amountofmoney + result.getcashbackamount());
5         externalcashbackservice.cashbackcomplete(userid, result.getcashbackamount());
6     }
7 }
```

the base components of the application are shown in the following diagram:



i tried to make an example as clear as possible. the transfer of money in the payment service depends on the other two services:

- the first one is a `cashbackservice` that interacts with another (external) web application under the rest protocol. and, in order to calculate the actual cash-back, we need to synchronize transactions with this application. this is because the next amount of the cash-back may depend on the total amount of user payments.
- the second is an `accountservice` that communicates with an internal database and stores data related to accounts of its users. in this service, we can use a jpa transaction to make some actions as atomic operations in the database.



operations in the different classes by the same key, this solution does not help you at all.

## string intern

in order to ensure that the instance of the class, which contains a user id, will be the same in all synchronized blocks, we can serialize it into a string and use the `string.intern()` to obtain the same link for equals strings.

`string.intern` uses a global pool to store strings that are interned. and when you request intern on the string, you get a reference from this pool if such string exists there or else this string puts in the pool.

you can find more details about `string.intern` in [the java language specification - 3.10.5 string literals](https://docs.oracle.com/javase/specs/jls/se8/html/jls-3.html#jls-3.10.5) (<https://docs.oracle.com/javase/specs/jls/se8/html/jls-3.html#jls-3.10.5>) or in the oracle java documentation about the [string.intern](https://docs.oracle.com/javase/9/docs/api/java/lang/string.html#intern--) (<https://docs.oracle.com/javase/9/docs/api/java/lang/string.html#intern-->)

```
1 public void withdrawmoney(uuid userid, int amountofmoney) {
2     synchronized (userid.toString().intern()) {
3         ..
4     }
5 }
```

using the intern is not a good practice because the pool of strings is difficult to clean with the gc. and, your application can consume too many resources with the active use of the `string.intern`.

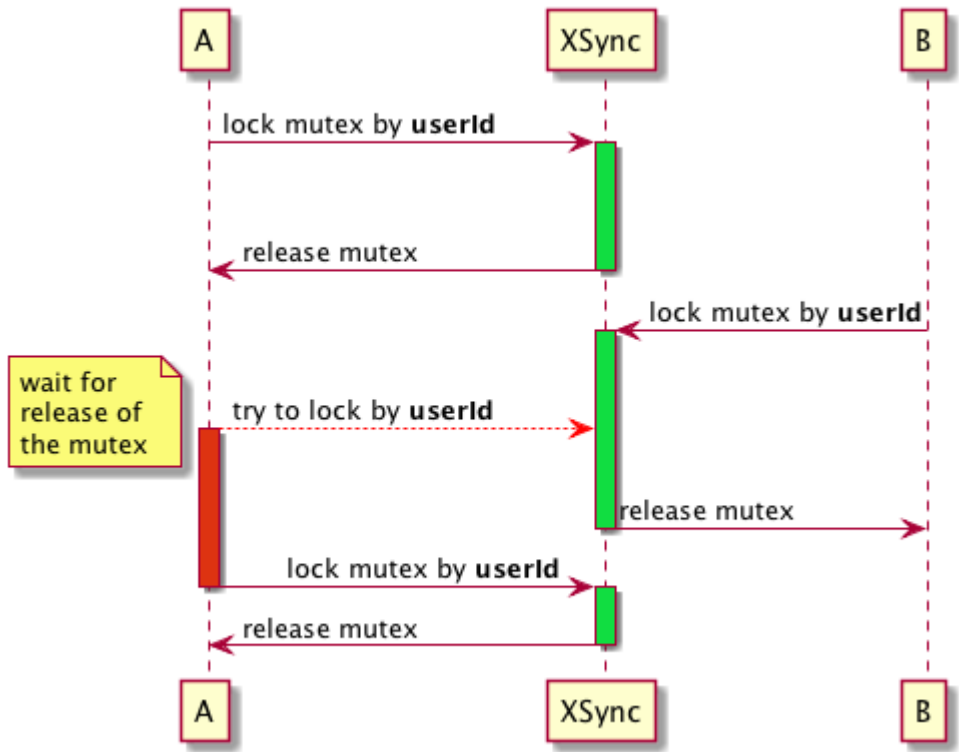
also, there is a chance that a foreign code is synchronized on the same instance of the string as your application. this can lead to deadlocks.

in general, the use of intern is better left to the internal libraries of the jdk; there are good articles by [aleksey shipilev](https://shipilev.net/) (<https://shipilev.net/>) about this concept.

## how can we solve this problem correctly?

### create your own synchronization primitive

we need to implement a behavior that describes the next diagram:



at first, we need to make a new synchronization primitive — the custom mutex. that will work by the value of the variable, and not by the reference to the object.

it will be something like a "**named mutex** ," but a little wider, with the ability to use the value of any objects for identification, not just the value of a string. you can find examples of synchronization primitives to locking by the name in other languages (c++, c#). now, we will solve this issue in java.

the solution will look something like this:

```
1 public void withdrawmoney(uuid userid, int amountofmoney) {
2     synchronized (xmutex.of(userid)) {
3         ..
4     }
5 }
```

in order to ensure that the same mutexes are obtained for equal values of variables, we will make the mutex factory.

```
1 public void withdrawmoney(uuid userid, int amountofmoney) {
2     synchronized (xmutexfactory.get(userid)) {
3         ..
4     }
5 }
6
7 public void purchase(uuid userid, int amountofmoney, vendordescription vendor) {
8     synchronized (xmutexfactory.get(userid)) {
9         ..
10    }
11 }
```

in order to return the same instance of mutex on the each of requests with equal keys, we will need to store the created mutexes. if we will store these mutexes in the simple `hashmap`, then the size of the map will increase as new keys appear. and we don't have a tool to evaluate a time when a mutex not used anywhere.

in this case, we can use the `weakreference` to save a reference to the mutex in the map, just when it uses. in order to implement this behavior, we can use the `weakhashmap` data structure. i wrote an article about this type of references a couple of months ago; you can consider it in more details here: [soft, weak, phantom references in java](http://antkorwin.com/concurrency/weakreference.html) (<http://antkorwin.com/concurrency/weakreference.html>)

our mutex factory will be based on the `weakhashmap`. the mutex factory creates a new mutex if the mutex for this `value(key)` is not found in the `hashmap`. then, the created mutex is added to the `hashmap`. using of the `weakhashmap` allows us to store a mutex in the `hashmap` while existing any references to it. and, the mutex will be removed from a `hashmap` automatically when all references to it are

we need to use a `synchronized` version of `weakhashmap` . let's see what's described in the [documentation](#) (<https://docs.oracle.com/javase/7/docs/api/java/util/WeakHashMap.html>) about it.

```
1 this class is not synchronized. a synchronized weakhashmap may be constructed
2 using the collections.synchronizedmap method.
```

it's very sad, and a little later, we'll take a closer look at the reason. but for now, let's consider an example of implementation, which is proposed by the official documentation (i mean the use of `collections.synchronizedmap` ):

```
1 public final map<xmutex<key>, weakreference<xmutex<key>>> weakhashmap =
2     collections.synchronizedmap(new weakhashmap<xmutex<key>,
3                                     weakreference<xmutex<key>>>());
4
5 public xmutex<key> getmutex(keyt key) {
6     validatekey(key);
7     return getexist(key)
8         .orElseget(() -> savenewreference(key));
9 }
10
11 private optional<xmutex<key>> getexist(keyt key) {
12     return optional.ofnullable(weakhashmap.get(xmutex.of(key)))
13         .map(weakreference::get);
14 }
15
16 private xmutex<key> savenewreference(keyt key) {
17
18     xmutex<key> mutex = xmutex.of(key);
19
20     weakreference<xmutex<key>> res = weakhashmap.put(mutex, new weakreference<>(mutex));
21     if (res != null && res.get() != null) {
22         return res.get();
23     }
24     return mutex;
25 }
```

### what about performance?

if we look at the code of the `collections.synchronizedmap` , then we find a lot of synchronizations on the global mutex, which is created in pair with a `synchronizedmap` instance.

```
1 synchronizedmap(map<k,v> m) {
2     this.m = objects.requirenonnull(m);
3     mutex = this;
4 }
```

and all other methods of the `synchronizedmap` are synchronized on mutex:

```
1 public int size() {
2     synchronized (mutex) {return m.size();}
3 }
4 public boolean containskey(object key) {
5     synchronized (mutex) {return m.containskey(key);}
6 }
7 public v get(object key) {
8     synchronized (mutex) {return m.get(key);}
9 }
10 public v put(k key, v value) {
11     synchronized (mutex) {return m.put(key, value);}
12 }
13 public v remove(object key) {
14     synchronized (mutex) {return m.remove(key);}
15 }
16 ...
17 ...
```

this solution does not have the best performance. all of these synchronizations lead us to permanent locks on each operation with a factory of mutexes.

### concurrenthashmap with a weakreference as a key

we need to look at the using of the `concurrenthashmap` . it has a better performance than `collections.synchronizedmap`. but we have one problem — the `concurrenthashmap` doesn't allow the use of weak references. this means that the garbage collector cannot delete unused mutexes.

i found two ways to solve this problem:

- the first is to create my own `concurrentmap` implementation. this is the right decision, but it will take a very long time.
- the second one is the use of the `concurrentreferencehashmap` implementation from the spring framework. this is a good implementation, but it has a couple of nuances. we will consider them below.

let's change the `xmutexfactory` implementation to use a `concurrentreferencehashmap` :

```
1 public class xmutexfactory<key> {
2
3     /**
4      * create mutex factory with default settings
5      */
6     public xmutexfactory() {
7         this.map = new concurrentreferencehashmap<>(default_initial_capacity,
8                                                     default_load_factor,
9                                                     default_concurrency_level,
10                                                     default_reference_type);
11     }
12
13     /**
14      * creates and returns a mutex by the key.
15      * if the mutex for this key already exists in the weak-map,
```



```
16 * then returns the same reference of the mutex.
17 */
18 public <mutex><key> getmutex(keyt key) {
19     return map.compute(key, (k, v) -> null).get(key);
20 }
21
22 }
```



(/users/login.html)



(/search)

RESEARCH (/research) WEBINARS (/webinars) ZONES

that’s cool!

less code, but more performance than before. let’s try to check the performance of this solution.

### create a simple benchmark

i made a small benchmark in order to select an implementation.

there are three implementations of the map involved in the test:

- collections.synchronizedmap based on the weakhashmap
- concurrenthashmap
- concurrentreferencehashmap

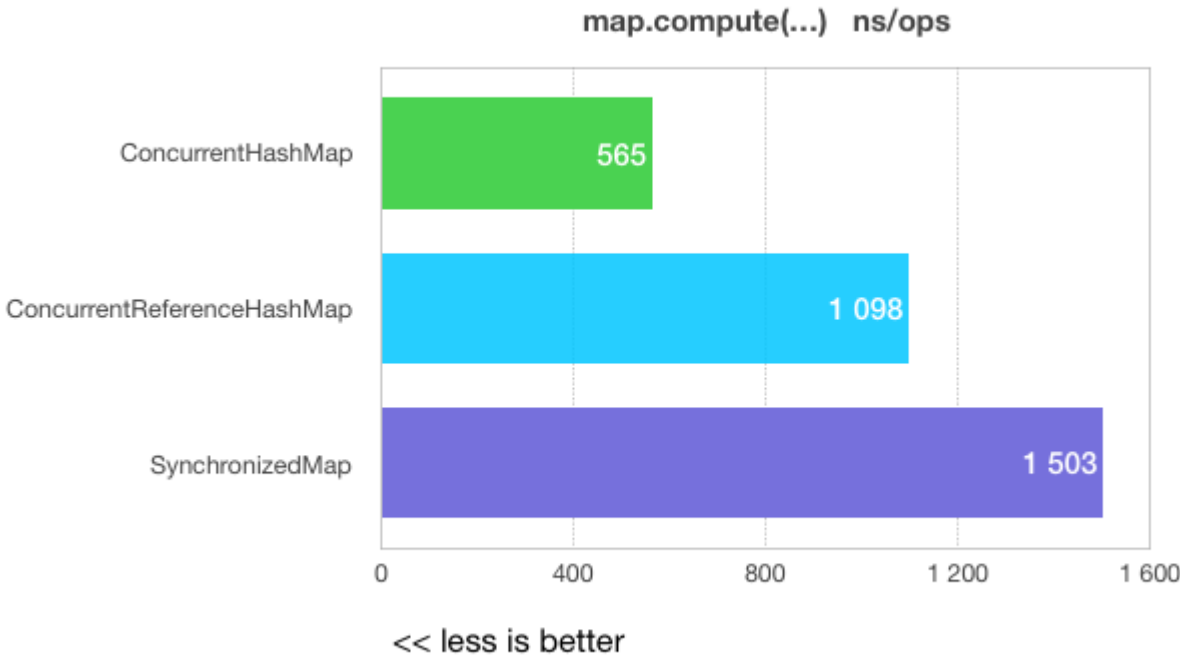
i use the concurrenthashmap in benchmark just for comparing in measurements. this implementation is not suitable for use in the factory of mutexes because it does not support the use of weak or soft references.

all benchmarks are written with using the jmh library.

1	# run complete. total time: 00:04:39
2	
3	benchmark mode cnt score error units
4	concurrentmap.concurrenthashmap thrpt 5 0,015 ? 0,004 ops/ns
5	concurrentmap.concurrentreferencehashmap thrpt 5 0,008 ? 0,001 ops/ns
6	concurrentmap.synchronizedmap thrpt 5 0,005 ? 0,001 ops/ns
7	concurrentmap.concurrenthashmap avgt 5 565,515 ? 23,638 ns/op
8	concurrentmap.concurrentreferencehashmap avgt 5 1098,939 ? 28,828 ns/op
9	concurrentmap.synchronizedmap avgt 5 1503,593 ? 150,552 ns/op
10	concurrentmap.concurrenthashmap sample 301796 663,330 ? 11,708 ns/op
11	concurrentmap.concurrentreferencehashmap sample 180062 1110,882 ? 6,928 ns/op
12	concurrentmap.synchronizedmap sample 136290 1465,543 ? 5,150 ns/op
13	concurrentmap.concurrenthashmap ss 5 336419,150 ? 617549,053 ns/op
14	concurrentmap.concurrentreferencehashmap ss 5 922844,750 ? 468380,489 ns/op
15	concurrentmap.synchronizedmap ss 5 1199159,700 ? 4339391,394 ns/op

in this micro-benchmark, i create a situation when several threads compute values in the map. you can consider the source code of this benchmark in more details here at [concurrent map benchmark](https://github.com/antkorwin/concurrent-map-benchmark/blob/master/src/test/java/com/antkorwin/concurrenttests/concurrentmapmicrobenchmark.java) (https://github.com/antkorwin/concurrent-map-benchmark/blob/master/src/test/java/com/antkorwin/concurrenttests/concurrentmapmicrobenchmark.java)

put it on the graph:



so, the concurrentreferencehashmap justifies its use in this case.

### getting started with the xsync library

i packed this code into the [xsync](https://github.com/antkorwin/xsync) (https://github.com/antkorwin/xsync) library, and you can use it as a ready solution for the synchronization on the value of variables.

in order to do it, you need to add the next dependency:

```
1 <dependency>
2   <groupid>com.antkorwin</groupid>
3   <artifactid>xsync</artifactid>
4   <version>1.1</version>
5 </dependency>
```

then, you are able to create instances of the xsync class for synchronization on types that you need. for the spring framework, you can make them as beans:

```
1 @bean
2 public xsync<uuid> xsync(){
3     return new xsync<>();
4 }
```

and now, you can use it:

```
1 @autowired
2 private xsync<String> xsync;
```

```
2 private xsync<uuid> xsync;
3
4 public void withdrawmoney(uuid userid, int amountofmoney) {
5     xsync.execute(userid, () -> {
6         Result result = externalPolicySystem.ValueBasedTransfer(userid, amountofmoney, withdraw);
7         accountservice.transfer(userid, amountofmoney, withdraw);
8     });
9 }
10
11 public void purchase(uuid userid, int amountofmoney, vendordescription vendor) {
12     xsync.execute(userid, () -> {
13         ..
14     });
15 }
```



(/users/login.html)



(/search)

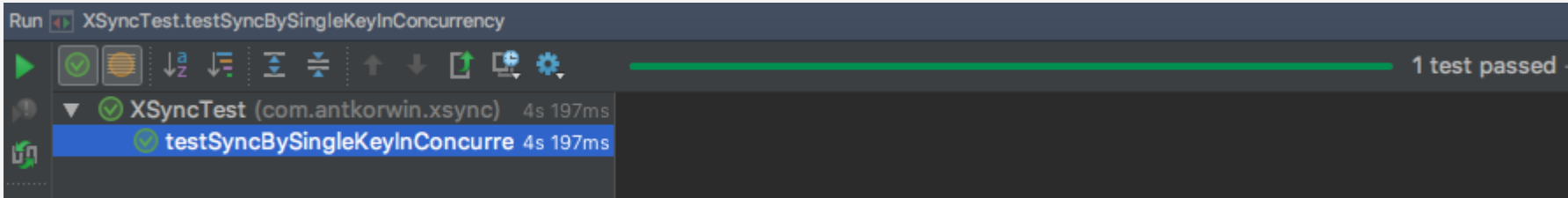
## concurrent tests

in order to be sure that this code works well, i wrote several concurrent tests.

there is an example of one of these tests:

```
1 public void testsyncbysinglekeyinconcurrency() {
2     // arrange
3     xsync<uuid> xsync = new xsync<>();
4     string id = uuid.randomUUID().toString();
5     nonatomicint var = new nonatomicint(0);
6
7     // there is a magic here:
8     // we created a parallel stream and try to increment
9     // the same nonatomic integer variable in each stream
10    intstream.range(0, thread_cnt)
11        .boxed()
12        .parallel()
13        .foreach(j -> xsync.execute(uuid.fromString(id), var::increment));
14
15    // asserts
16    await().atmost(5, TimeUnit.SECONDS)
17        .until(var::getValue, equalTo(thread_cnt));
18
19    assertions.assertThat(var.getValue()).isEqualTo(thread_cnt);
20 }
21
22 /**
23  * implementation of the does not thread safe integer variable:
24  */
25 @getter
26 @allargsconstructor
27 private class nonatomicint {
28     private int value;
29
30     public int increment() {
31         return value++;
32     }
33 }
```

let’s see the result of this test:




## references

xsync library on github: <https://github.com/antkorwin/xsync> (https://github.com/antkorwin/xsync)

examples using the xsync library: <https://github.com/antkorwin/xsync-example> (https://github.com/antkorwin/xsync-example)

the original article can be found [here](http://antkorwin.com/concurrency/synchronization_by_value.html) (http://antkorwin.com/concurrency/synchronization\_by\_value.html) .

Topics: JAVA, CONCURRENCY, SYNCHRONIZED, CONCURRENTHASHMAP, WEAKREFERENCE, MUTEX

Published at DZone with permission of Anatoliy Korovin. [See the original article here.](http://antkorwin.com/concurrency/synchronization_by_value.html)   
(http://antkorwin.com/concurrency/synchronization\_by\_value.html)  
Opinions expressed by DZone contributors are their own.

## Popular on DZone

- [Mobile App Architecture Design and Development](/articles/mobile-app-architecture-design-and-development?fromrel=true) (/articles/mobile-app-architecture-design-and-development?fromrel=true)
- [Kubernetes Multi-Tenancy Best Practices With Amazon EKS](/articles/kubernetes-multi-tenancy-best-practices-with-amazon-eks?fromrel=true) (/articles/kubernetes-multi-tenancy-best-practices-with-amazon-eks?fromrel=true)
- [Agility Office: The Backbone of Any Transformation](/articles/agility-office-the-backbone-of-any-transformation?fromrel=true) (/articles/agility-office-the-backbone-of-any-transformation?fromrel=true)
- [Four Scrum Master Success Principles](/articles/four-scrum-master-success-principles?fromrel=true) (/articles/four-scrum-master-success-principles?fromrel=true)

## Java Partner Resources



### E-Commerce Development Essentials

From SDKs and APIs to general architecture decisions, this Refcard will help you get started setting up your own e-commerce offering. [Read](#) ►

Presented by **Shopify**





Introduction to Cloud-Native Java

Learn the basic explanations of cloud-native development on Java and explore everything you need to make your project cloud-native. [Read more](#)

[\(/users/login.html\)](#)

[\(/search\)](#)

[REFCARDZ \(/refcardz\)](#) [RESEARCH \(/research\)](#) [WEBINARS \(/webinars\)](#) [ZONES](#) ▾



Getting Started With Feature Flags

As a core component of CD, feature flagging empowers developers to release software faster, more reliably, and with more control. [Get started ▶](#)

Presented by **CloudBees**

ABOUT US

[About DZone \(/pages/about\)](#)

[Send feedback](#)

[\(/mailto:support@dzone.com\)](mailto:support@dzone.com)

[Careers \(https://devada.com/careers/\)](https://devada.com/careers/)

[Sitemap \(/sitemap\)](#)

CONTRIBUTE ON DZONE

[Article Submission Guidelines](#)

[\(/articles/dzones-article-submission-guidelines\)](#)

[MVB Program \(/pages/mvb\)](#)

[Become a Contributor](#)

[\(/pages/contribute\)](#)

[Visit the Writers' Zone \(/writers-zone\)](#)

LEGAL

[Terms of Service \(/pages/tos\)](#)

[Privacy Policy \(/pages/privacy\)](#)

ADVERTISE

[Developer Marketing Blog](#)

[\(/https://devada.com/blog/developer-marketing\)](https://devada.com/blog/developer-marketing)

[Advertise with DZone \(/pages/advertise\)](#)

[+1 \(919\) 238-7100 \(tel:+19192387100\)](#)

CONTACT US

600 Park Offices Drive

Suite 150

Research Triangle Park, NC 27709

[support@dzone.com](mailto:support@dzone.com)

[\(/mailto:support@dzone.com\)](mailto:support@dzone.com)

[+1 \(919\) 678-0300 \(tel:+19196780300\)](#)

Let's be friends:



[\(/pages/feed\)](#) [https://www.dzone.com/dzonecompany/dzone/](#)

DZone.com is powered by



[\(https://devada.com/answerhub/\)](https://devada.com/answerhub/)