Compatibility testing Hazelcast

Vassilis Bekiaris
Software Engineer, Core IMDG
@v_bekiaris

About me

- First computer:
  Amstrad 6128 green screen

- Favourite languages I never used in production: Ada, CLISP

- Freelancer (-2008)
- Software Architect, Team Leader, Jack of all Trades (2008-2016)
- Software Engineer, Hazelcast (2016-)

Hazelcast

- Open-source Distributed In-memory Object Store
- Data structures: Map, Cache, Set, List, MultiMap, RingBuffer, HyperLogLog, …
- Distributed compute: Executor framework
  - + Hazelcast Jet for stream processing

- + Enterprise Edition (Off-heap storage, Hot Restart persistence, Security, WAN replication, …)
- + Managed Service https://cloud.hazelcast.com/sign-up

The problem

- Hazelcast compatibility guarantees
  - patch-version compatibility (Hazelcast Open Source & Enterprise)
  - minor-version compatibility (Hazelcast Enterprise)

- Typical minor-version compatibility test:
  - Start a cluster at X.Y
  - Verify X.Y cluster
  - Partial members upgrade to X.Y+1
  - Verify X.Y features work on mixed cluster
  - Complete rolling upgrade to X.Y+1
  - Verify X.Y & X.Y+1 cluster

- Issues:
  - Resource intensive
  - Disconnect between developing new minor version features and writing the compatibility tests
  - Slow feedback from commit to compatibility-broken detection

Motivation

- Compatibility tests that I can run on my laptop, co-existing in same (pull-request|repository) as code being tested
- Massive body of Hazelcast JUnit tests: can I reuse existing tests as compatibility tests on mixed clusters?

```java
@RunWith(HazelcastParallelClassRunner.class)
@Category({QuickTest.class, ParallelJVMTest.class})
public class AtomicLongBasicTest extends AbstractAtomicLongBasicTest {

    @Override
    protected HazelcastInstance[] createInstances() { return newInstan

    @Override
    protected String getName() { return "long@group"; }

    @Override
    protected IAtomicLong createAtomicLong(String name) {
        HazelcastInstance instance = instances[RandomPicker.getInt(ins
        return instance.getCPSubsystem().getAtomicLong(name);
    }

    @Test
    public void testCreate_withDefaultGroup() {
```

```java
/**
 * Compatibility test for IAtomicLong.
 */
@RunWith(EnterpriseSerialJUnitClassRunner.class)
@Category(CompatibilityTest.class)
public class AtomicLongCompatibilityTest extends AtomicLongBasicTest {

}
```

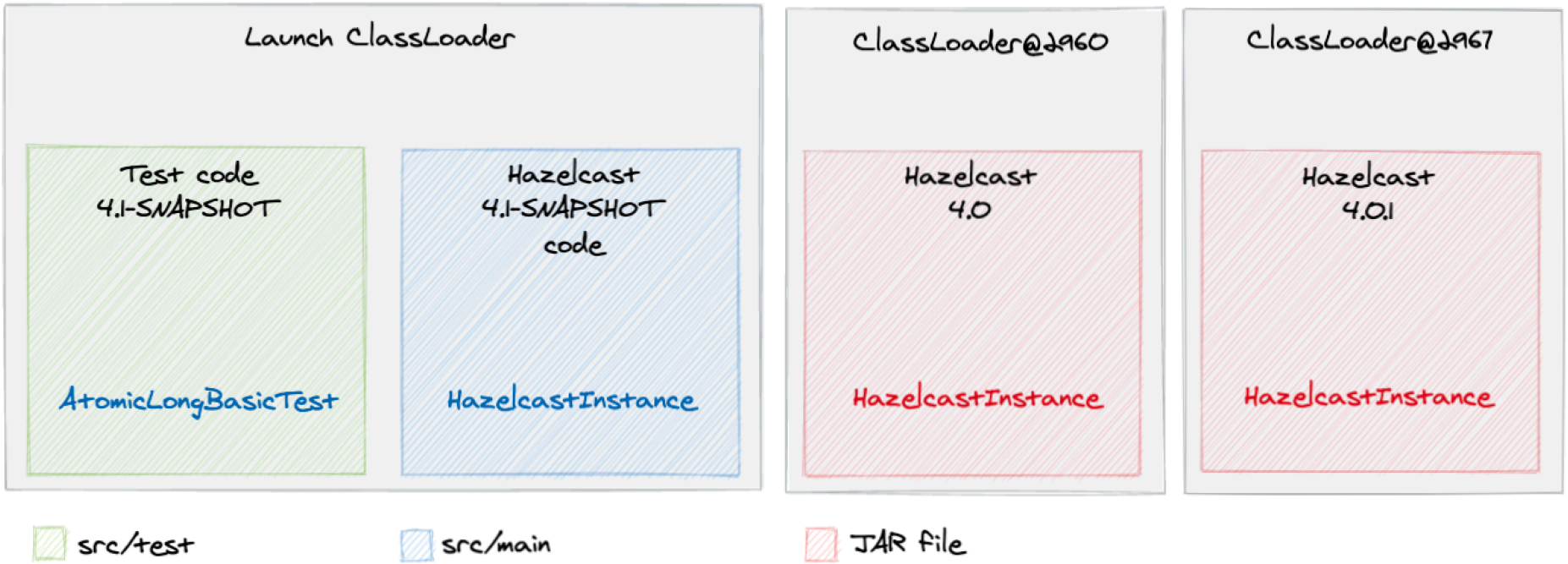
Structure of a simple test

```
public abstract class AbstractAtomicLongBasicTest extends HazelcastRaftTestSupport {

    protected HazelcastInstance[] instances;
    protected IAtomicLong atomicLong;
    protected String name;

    @Before
    public void setup() {
        instances = createInstances();
        name = getName();
        atomicLong = createAtomicLong(name);
        assertNotNull(atomicLong);
    }

    @Test
    public void testIncrementAndGet() {
        assertEquals(1, atomicLong.incrementAndGet());
        assertEquals(2, atomicLong.incrementAndGet());
    }
}
```

Problems

- Can I run N `HazelcastInstances` of different versions in the same JVM?
  - Same class names (mostly), different versions
- How will my test code interface with previous Hazelcast versions?

Enter ClassLoaders



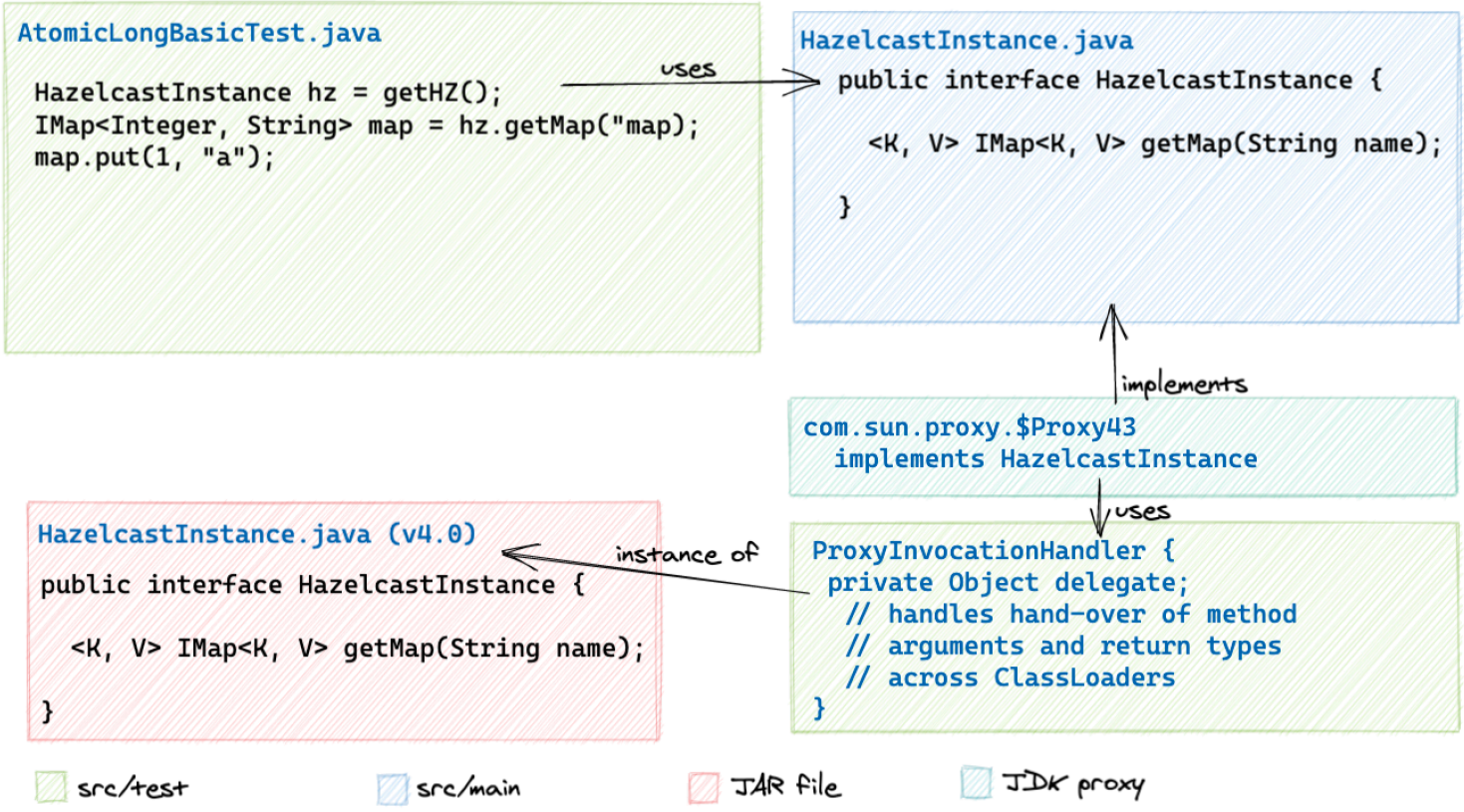
Loading Hazelcast versions in separate ClassLoaders

- Entry point is HazelcastStarter#newHazelcastInstance(String version)
- HazelcastVersionLocator locates requested binary locally or from remote maven repository
- HazelcastAPIDelegatingClassloader prioritizes specific version's JAR classes
  - when requested to load a Hazelcast production class, it will load it from the version-specific JAR
  - other classes are loaded by the parent class loader

Crossing ClassLoader boundaries

- Test code can only interact with current-version code
  - HazelcastInstance.class (current) != HazelcastInstance(v4.0).class
  - `IMap map = hazelcastInstance.getMap("map"):`
    `ClassCastException: IMap cannot be cast to IMap` 🙀

Crossing ClassLoader boundaries



Crossing ClassLoader boundaries

- JDK dynamic proxies are awesome for proxying interfaces

- What about concrete classes?
  - Reflection: given a source object, locate class in target classloader, instantiate object in target and copy fields from source to target
  - Mockito + subclass proxying + custom Answers
  - More troubles: `final` classes / methods
    - remove `final` modifier at class load time with agent

Are we there yet?

- Yes! Just adding `@Category(CompatibilityTest.class)` works! (most of the time*)

- Compatibility tests are in Hazelcast Enterprise code repositories, but the infrastructure is in open source repository.

Contributors welcome!

- Pick an issue and join: https://github.com/hazelcast/hazelcast/contribute

Thank You