Hazelcast Deep Dive

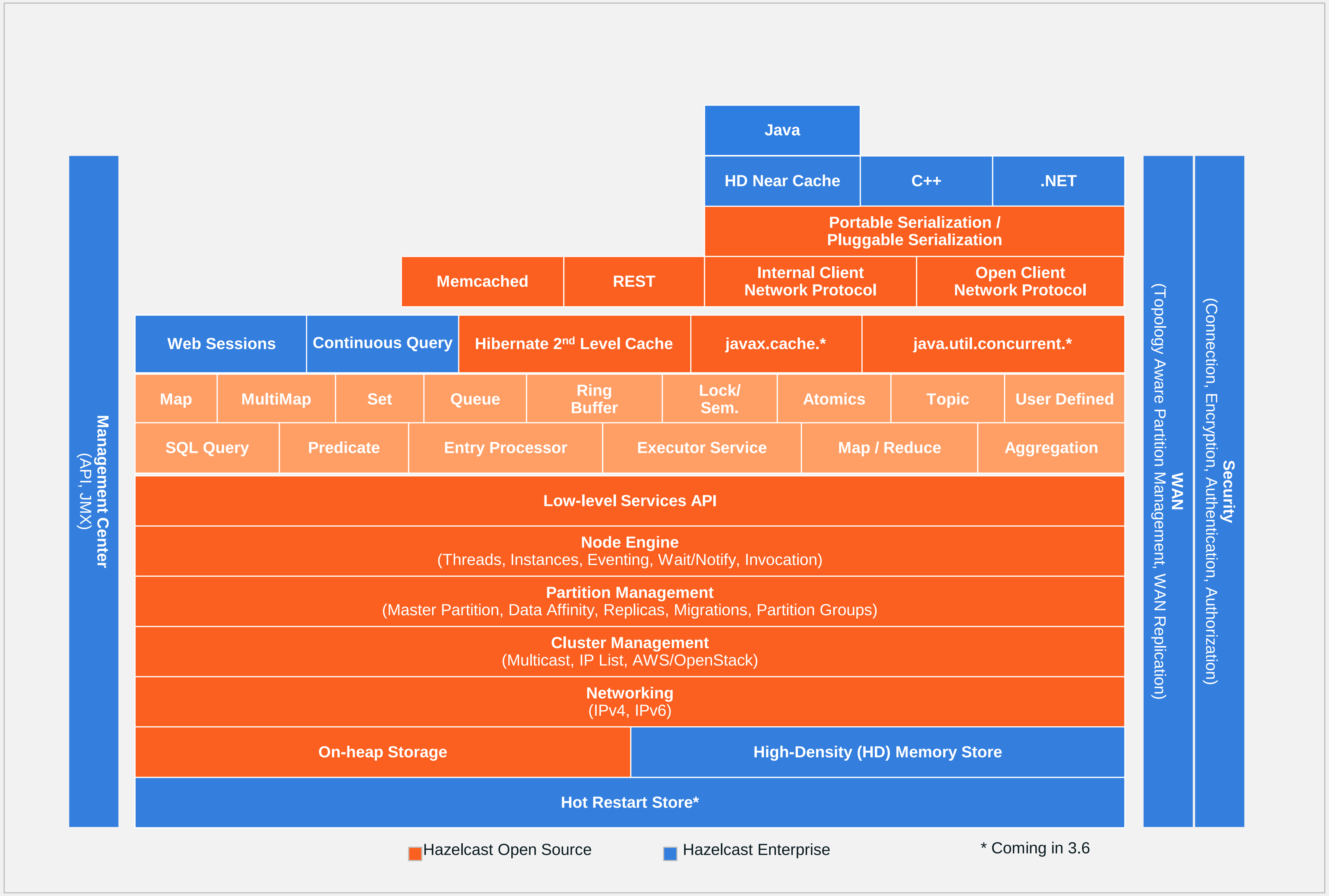
Emrah Kocaman

Software Engineer @ Hazelcast
emrahkocaman
emrah@hazelcast.com

Agenda

- Hazelcast Architecture
- Hazelcast Configuration Options
- Hazelcast IMap Configuration Details
- Hazelcast Clients
- Hazelcast Serialization
- Q/A Session

Hazelcast Architecture

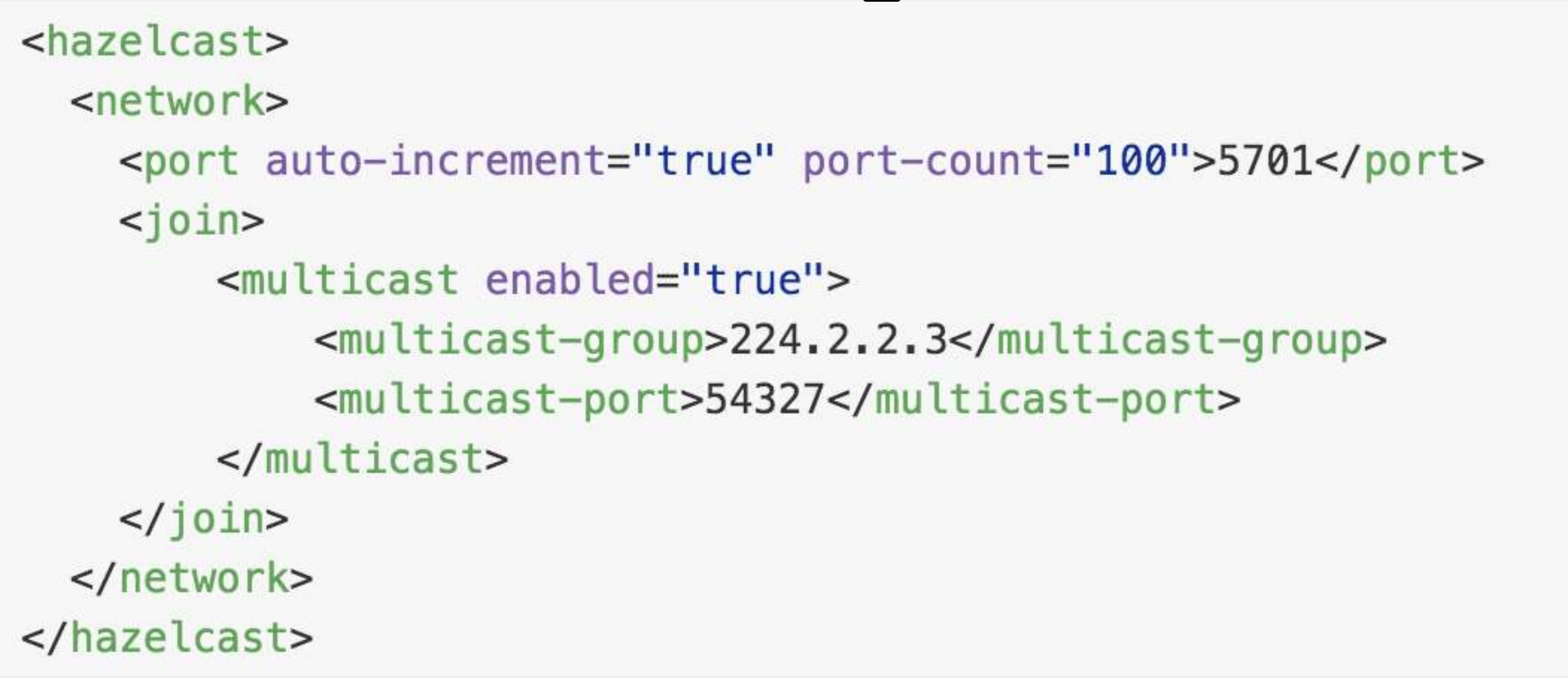

Java
HD Near Cache
C++
.NET
Portable Serialization / Pluggable Serialization
Memcached
REST
Internal Client Network Protocol
Open Client Network Protocol
Web Sessions
Continuous Query
Hibernate 2nd Level Cache
javax.cache.*
java.util.concurrent.*
Map
MultiMap
Set
Queue
Ring Buffer
Lock/ Sem.
Atomics
Topic
User Defined
SQL Query
Predicate
Entry Processor
Executor Service
Map / Reduce
Aggregation
Low-level Services API
Node Engine (Threads, Instances, Eventing, Wait/Notify, Invocation)
Partition Management (Master Partition, Data Affinity, Replicas, Migrations, Partition Groups)
Cluster Management (Multicast, IP List, AWS/OpenStack)
Networking (IPv4, IPv6)
On-heap Storage
High-Density (HD) Memory Store
Hot Restart Store*
Management Center (API, JMX)
WAN (Topology Aware Partition Management, WAN Replication)
Security (Connection, Encryption, Authentication, Authorization)
🟧 Hazelcast Open Source
🟦 Hazelcast Enterprise
* Coming in 3.6

hazelcast

Hazelcast Configuration Options

- XML Configuration
- Programmatic configuration
- Spring Configuration

XML Configuration



```xml
<hazelcast>
  <network>
    <port auto-increment="true" port-count="100">5701</port>
    <join>
        <multicast enabled="true">
            <multicast-group>224.2.2.3</multicast-group>
            <multicast-port>54327</multicast-port>
        </multicast>
    </join>
  </network>
</hazelcast>
```

XML Configuration

```xml
<map name="testmap*">
   <time-to-live-seconds>10</time-to-live-seconds>
</map>
```

XML Configuration

```xml
<executor-service name="exec">
   <pool-size>${pool.size}</pool-size>
</executor-service>
```

```java
Properties properties = new Properties();
properties.setProperty("pool.size","10");
Config config = new XmlConfigBuilder()
    .setProperties(properties)
        .build();
HazelcastInstance hz = Hazelcast.newHazelcastInstance(config);
```

XML Configuration

- `hazelcast.config` system property
- Working directory
- Classpath
- Default from Hazelcast.jar

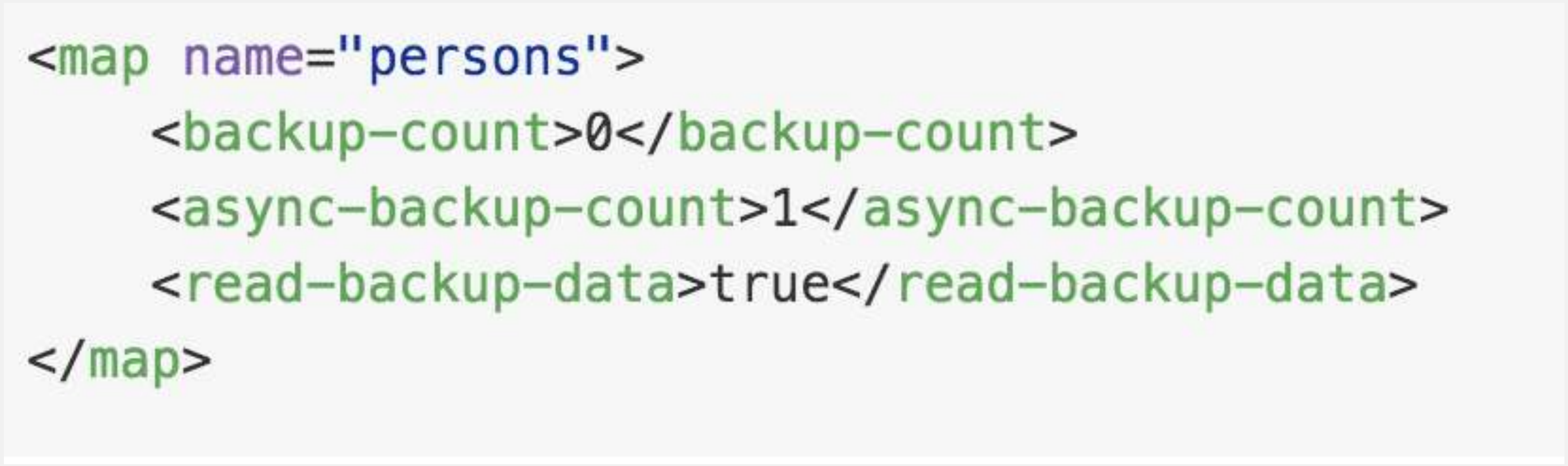
XML Configuration

- ClasspathXmlConfig
- FileSystemXmlConfig
- InMemoryXmlConfig
- UrlXmlConfig

Programmatic Configuration

```java
public class Main {
    public static void main(String[] args){
        ExecutorConfig executorConfig = new ExecutorConfig()
          .setName("someExecutor")
          .setPoolSize(10);
        Config config = new Config().addExecutorConfig(executorConfig);     (1)
        HazelcastInstance hz = Hazelcast.newHazelcastInstance(config);      (2)
    }
}
```

Distributed Map Configuration



```xml
<map name="persons">
    <backup-count>0</backup-count>
    <async-backup-count>1</async-backup-count>
    <read-backup-data>true</read-backup-data>
</map>
```

Distributed Map Configuration

```xml
<hazelcast>
    <map name="objectMap">
        <in-memory-format>OBJECT</in-memory-format>
    </map>
    <map name="binaryMap">
        <in-memory-format>BINARY</in-memory-format>
    </map>
</hazelcast>
```

IT’S DEMO TIME

Hazelcast Clients

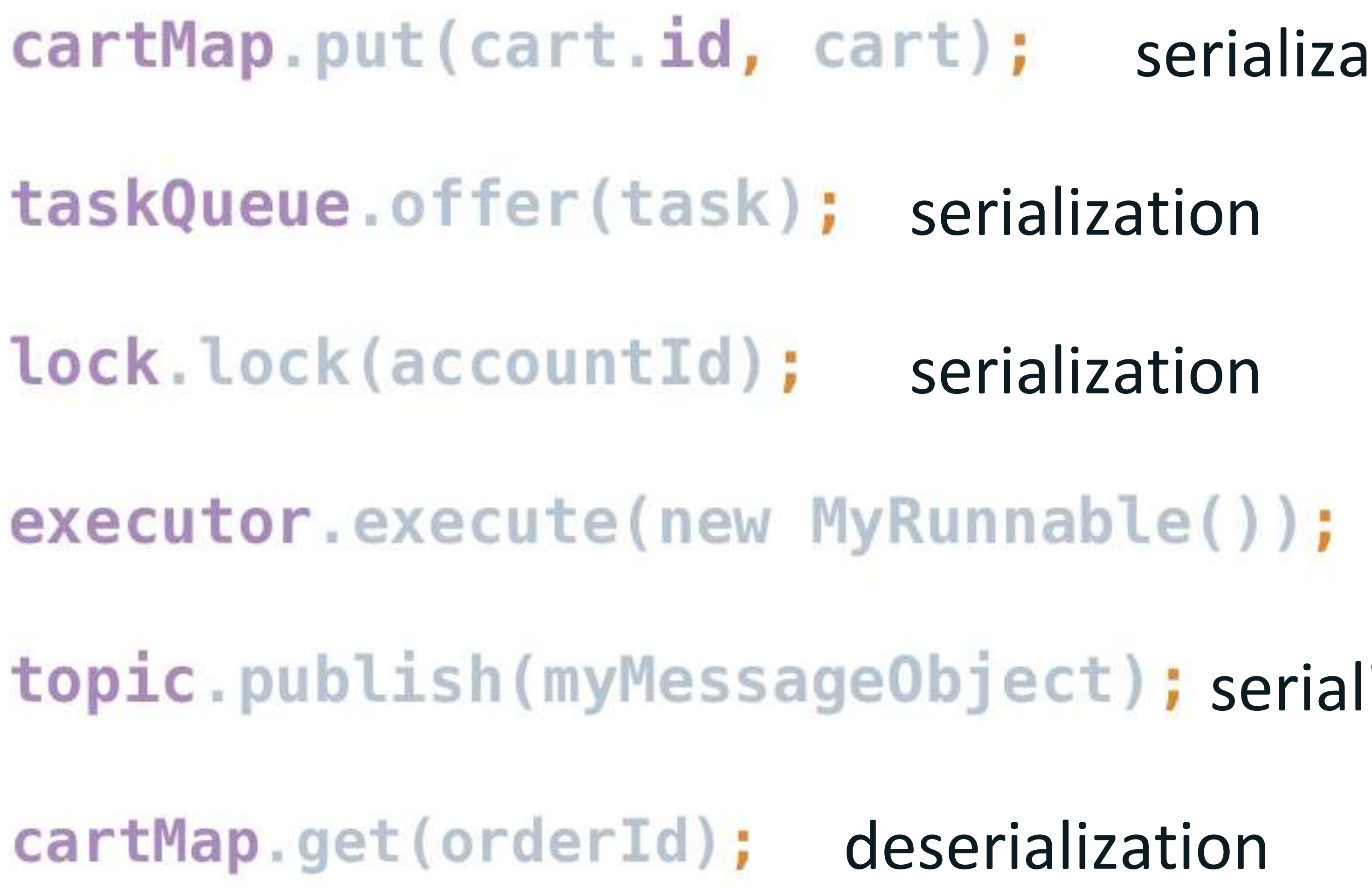
- Same intuitive API
- Similar configuration approach
- SSL Support
- Hazelcast Client Protocol
- Java, C/C++, .NET, REST, memcache

IT’S DEMO TIME

Hazelcast Serialization

- Map, Cache, Queue, Set, List
- Executor Service
- Entry Processors
- Lock
- Topic

Where it is used?



`cartMap.put(cart.id, cart);`	serialization + deserialization
`taskQueue.offer(task);`	serialization
`lock.lock(accountId);`	serialization
`executor.execute(new MyRunnable());`	serialization
`topic.publish(myMessageObject);`	serialization
`cartMap.get(orderId);`	deserialization

hazelcast

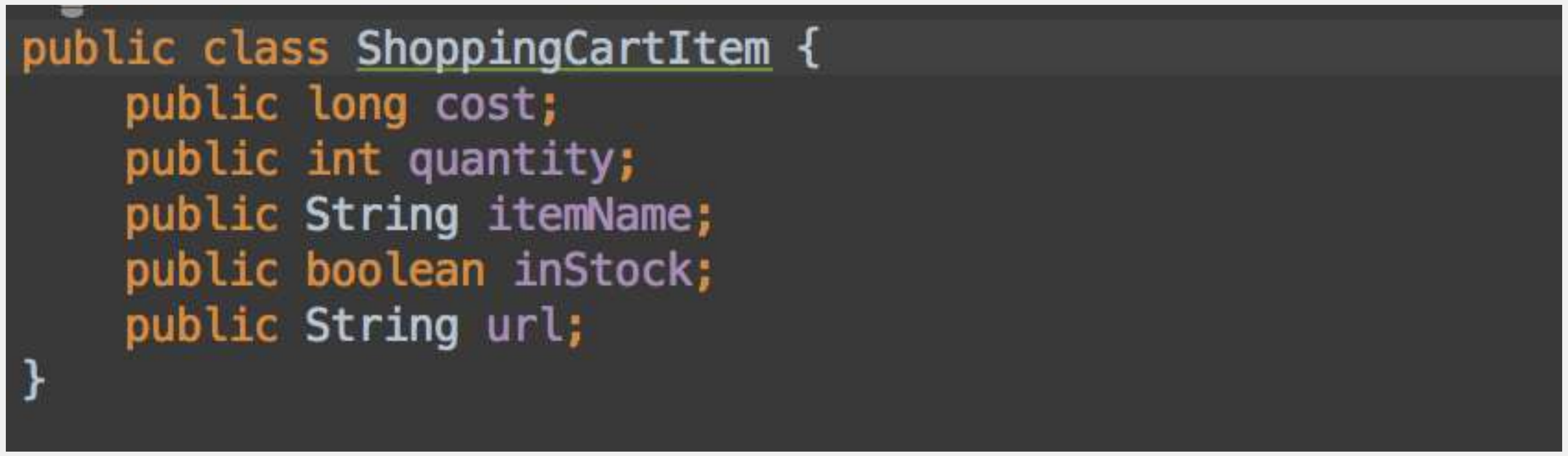
Optimised Types

Byte	byte[]	String
Boolean	char[]	Date
Character	short[]	BigInteger
Short	int[]	BigDecimal
Integer	long[]	Class
Long	float[]	Enum
Float	double[]	
Double		

Hazelcast Serialization

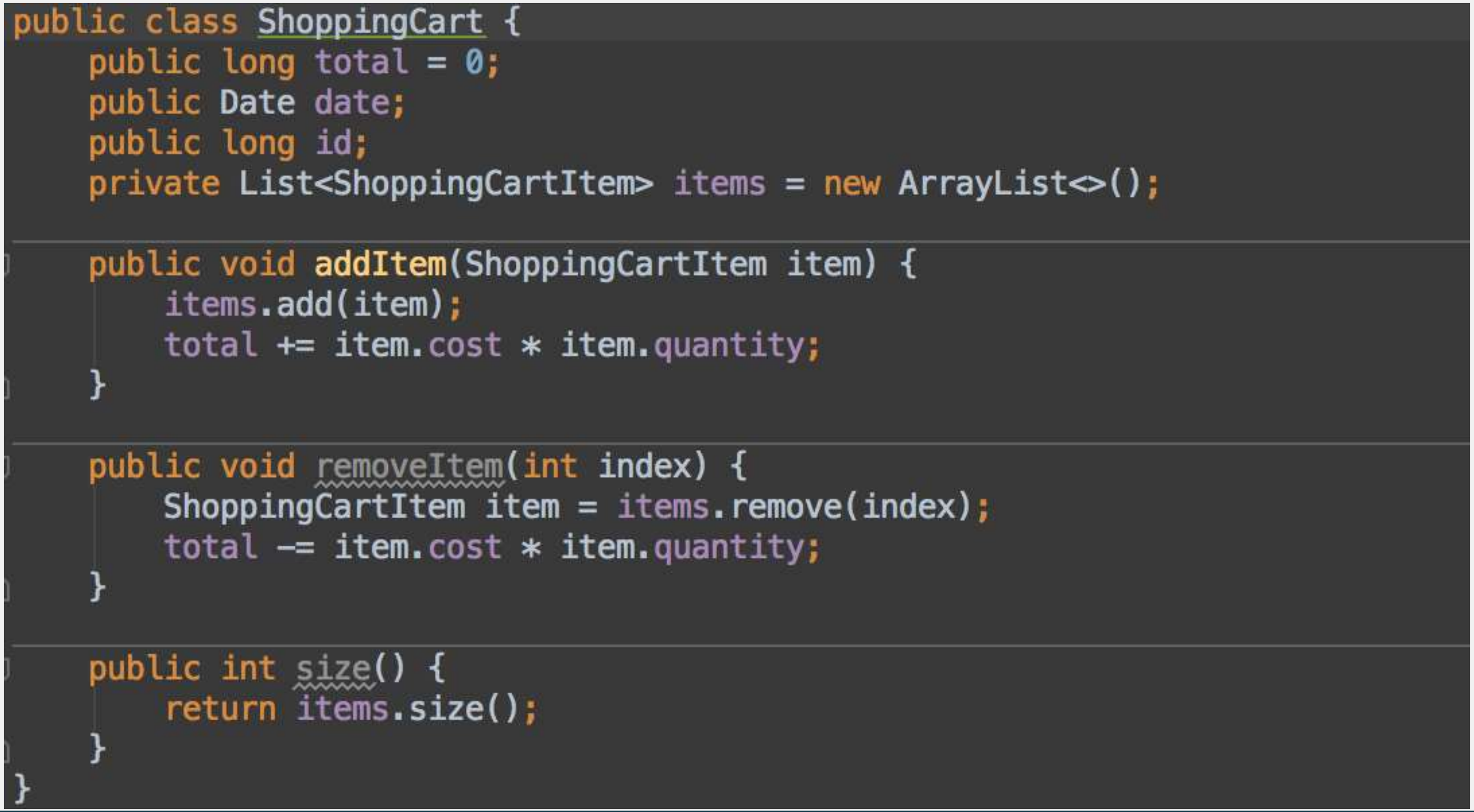
- Serializable
- DataSerializable
- IdentifiedDataSerializable
- Portable
- Pluggable

Shopping Cart Item



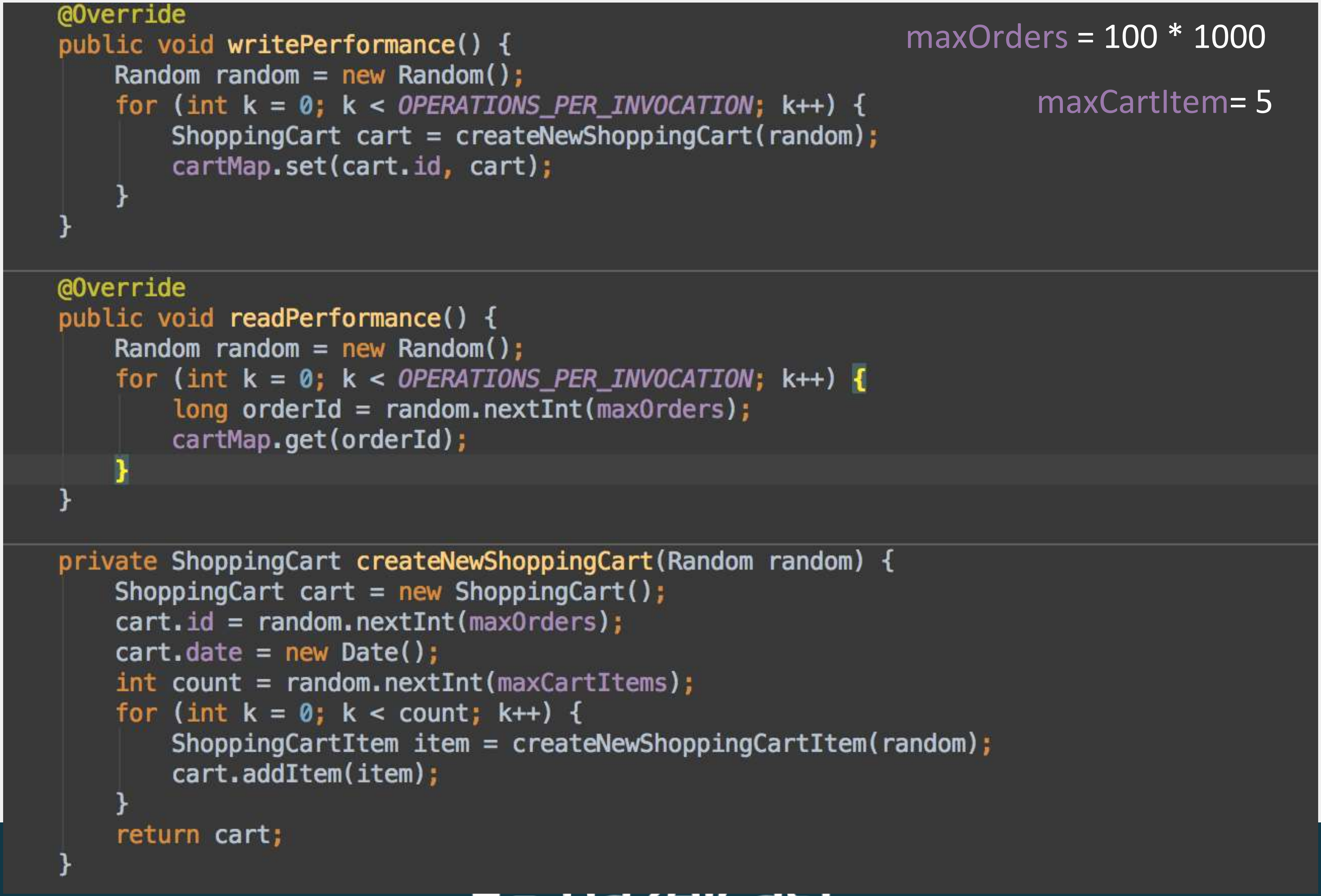
```java
public class ShoppingCartItem {
    public long cost;
    public int quantity;
    public String itemName;
    public boolean inStock;
    public String url;
}
```

Shopping Cart



```java
public class ShoppingCart {
    public long total = 0;
    public Date date;
    public long id;
    private List<ShoppingCartItem> items = new ArrayList<>();

    public void addItem(ShoppingCartItem item) {
        items.add(item);
        total += item.cost * item.quantity;
    }

    public void removeItem(int index) {
        ShoppingCartItem item = items.remove(index);
        total -= item.cost * item.quantity;
    }

    public int size() {
        return items.size();
    }
}
```

Benchmark – 100,000 times



maxOrders = 100 * 1000

maxCartItem= 5

```java
@Override
public void writePerformance() {
    Random random = new Random();
    for (int k = 0; k < OPERATIONS_PER_INVOCATION; k++) {
        ShoppingCart cart = createNewShoppingCart(random);
        cartMap.set(cart.id, cart);
    }
}

@Override
public void readPerformance() {
    Random random = new Random();
    for (int k = 0; k < OPERATIONS_PER_INVOCATION; k++) {
        long orderId = random.nextInt(maxOrders);
        cartMap.get(orderId);
    }
}

private ShoppingCart createNewShoppingCart(Random random) {
    ShoppingCart cart = new ShoppingCart();
    cart.id = random.nextInt(maxOrders);
    cart.date = new Date();
    int count = random.nextInt(maxCartItems);
    for (int k = 0; k < count; k++) {
        ShoppingCartItem item = createNewShoppingCartItem(random);
        cart.addItem(item);
    }
    return cart;
}
```

java.io.Serializable

Pros

- Standard
- Doesn’t require any implementation

Cons

- Takes more time and cpu
- Occupies more space

java.io.Serializable - Results

Read Performance

- 31 ops in ms

Write Performance

- 46 ops in ms

Binary object size

- 525 bytes

java.io.Externalizable

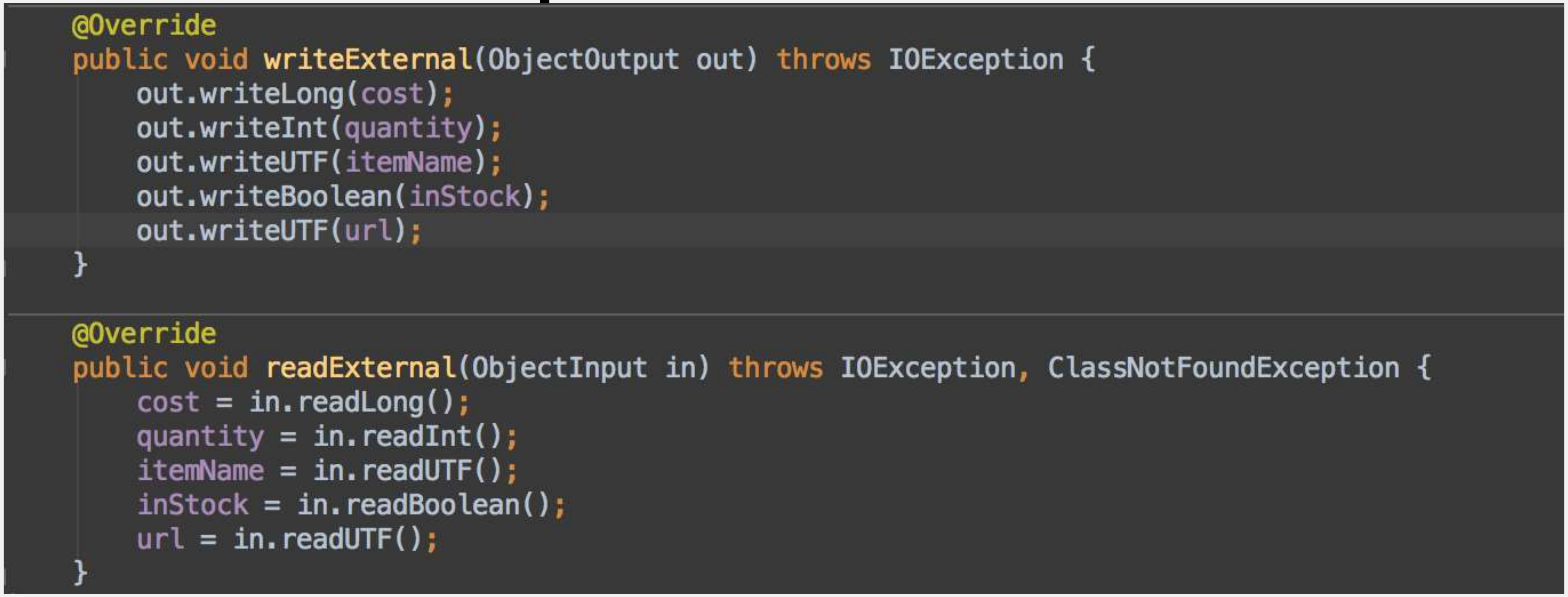
Pros

- Standard
- Efficient than Serializable in terms of CPU and Memory

Cons

- Requires to implement the actual serialization

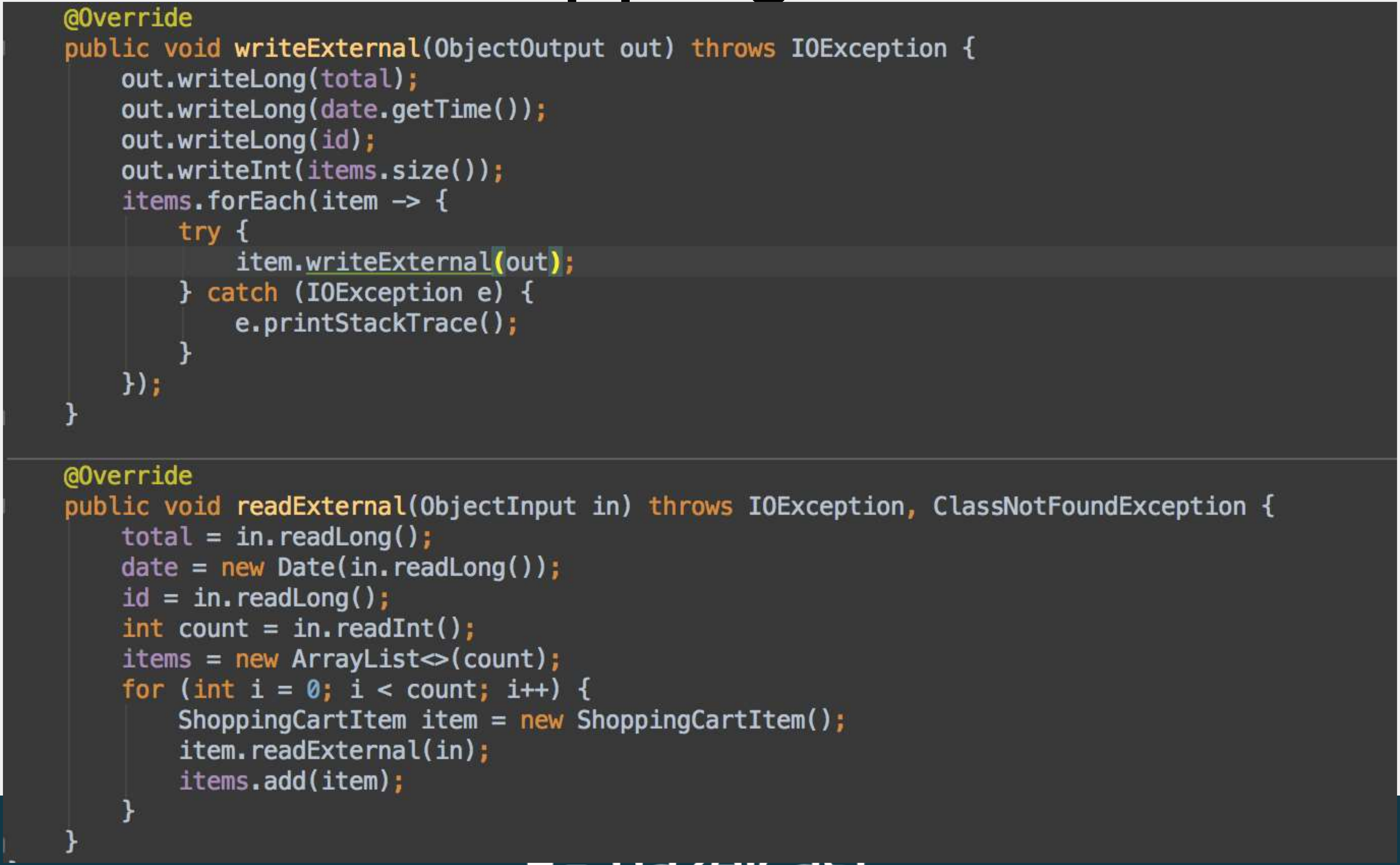
ShoppingCartItem - implementation



```
@Override
public void writeExternal(ObjectOutput out) throws IOException {
    out.writeLong(cost);
    out.writeInt(quantity);
    out.writeUTF(itemName);
    out.writeBoolean(inStock);
    out.writeUTF(url);
}

@Override
public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
    cost = in.readLong();
    quantity = in.readInt();
    itemName = in.readUTF();
    inStock = in.readBoolean();
    url = in.readUTF();
}
```

hazelcast

ShoppingCart-



```java
@Override
public void writeExternal(ObjectOutput out) throws IOException {
    out.writeLong(total);
    out.writeLong(date.getTime());
    out.writeLong(id);
    out.writeInt(items.size());
    items.forEach(item -> {
        try {
            item.writeExternal(out);
        } catch (IOException e) {
            e.printStackTrace();
        }
    });
}

@Override
public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
    total = in.readLong();
    date = new Date(in.readLong());
    id = in.readLong();
    int count = in.readInt();
    items = new ArrayList<>(count);
    for (int i = 0; i < count; i++) {
        ShoppingCartItem item = new ShoppingCartItem();
        item.readExternal(in);
        items.add(item);
    }
}
```

java.io.Externalizable - Results

Read Performance

- 61 ops in ms

Write Performance

- 60 ops in ms

Binary object size

- 235 bytes

DataSerializable

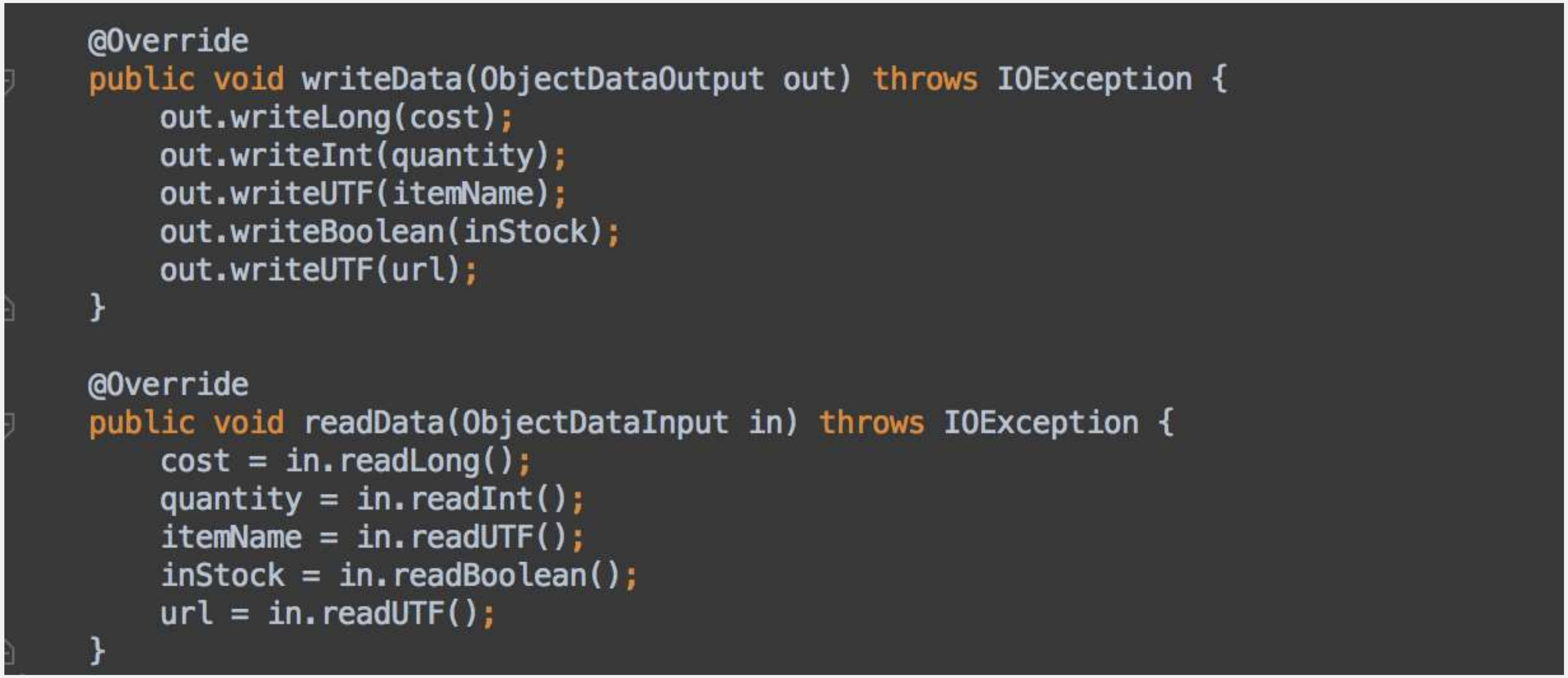
Pros

- Efficient than Serializable in terms of CPU and Memory

Cons

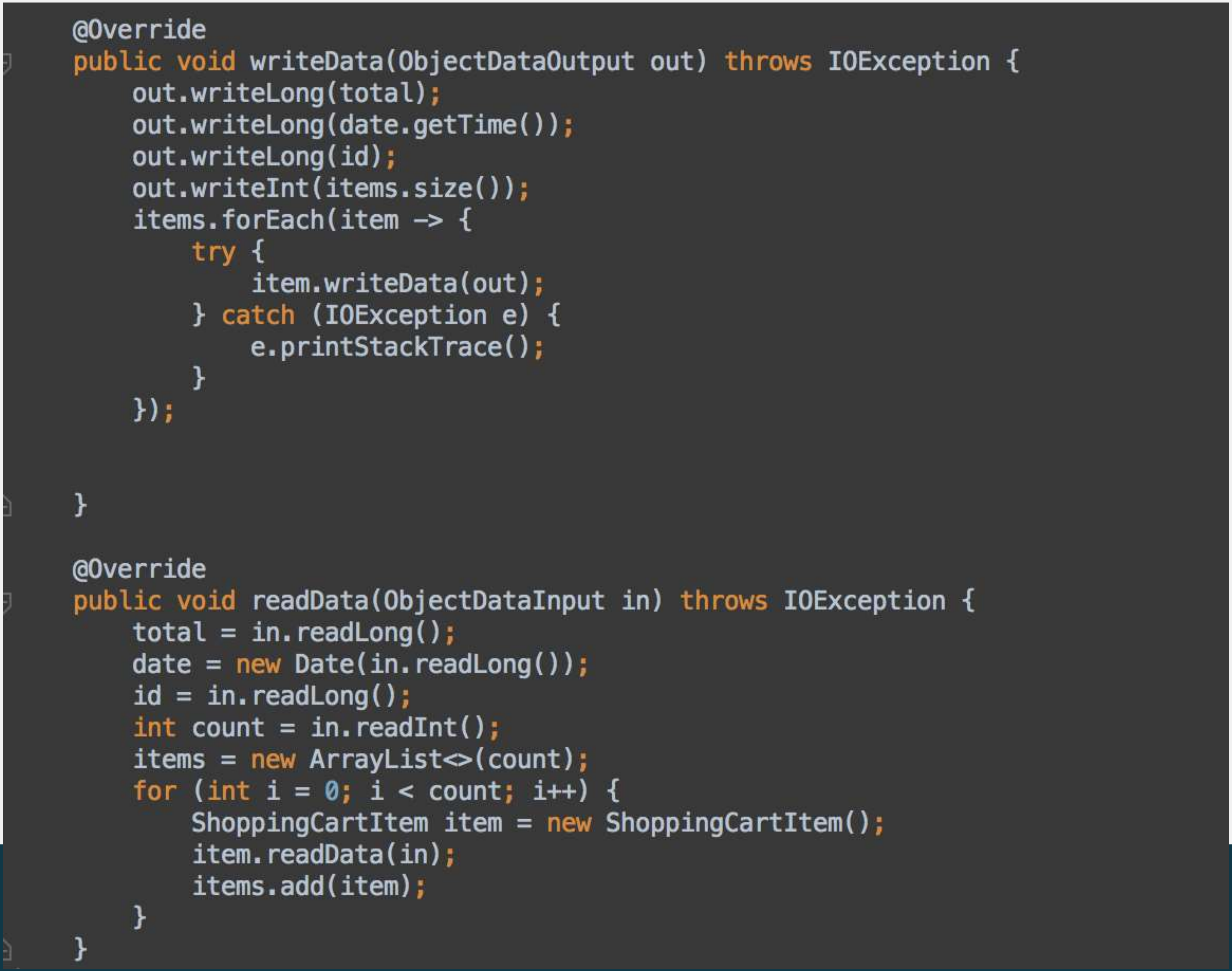
- Hazelcast specific
- Requires to implement the actual serialization
- Uses Reflection while de-serializing

ShoppingCartItem - implementation



```
@Override
public void writeData(ObjectDataOutput out) throws IOException {
    out.writeLong(cost);
    out.writeInt(quantity);
    out.writeUTF(itemName);
    out.writeBoolean(inStock);
    out.writeUTF(url);
}

@Override
public void readData(ObjectDataInput in) throws IOException {
    cost = in.readLong();
    quantity = in.readInt();
    itemName = in.readUTF();
    inStock = in.readBoolean();
    url = in.readUTF();
}
```

ShoppingCart-



```java
@Override
public void writeData(ObjectDataOutput out) throws IOException {
    out.writeLong(total);
    out.writeLong(date.getTime());
    out.writeLong(id);
    out.writeInt(items.size());
    items.forEach(item -> {
        try {
            item.writeData(out);
        } catch (IOException e) {
            e.printStackTrace();
        }
    });


}

@Override
public void readData(ObjectDataInput in) throws IOException {
    total = in.readLong();
    date = new Date(in.readLong());
    id = in.readLong();
    int count = in.readInt();
    items = new ArrayList<>(count);
    for (int i = 0; i < count; i++) {
        ShoppingCartItem item = new ShoppingCartItem();
        item.readData(in);
        items.add(item);
    }
}
```

DataSerializable- Results

Read Performance

- 64 ops in ms

Write Performance

- 59 ops in ms

Binary object size

- 231 bytes

IdentifiedDataSerializable

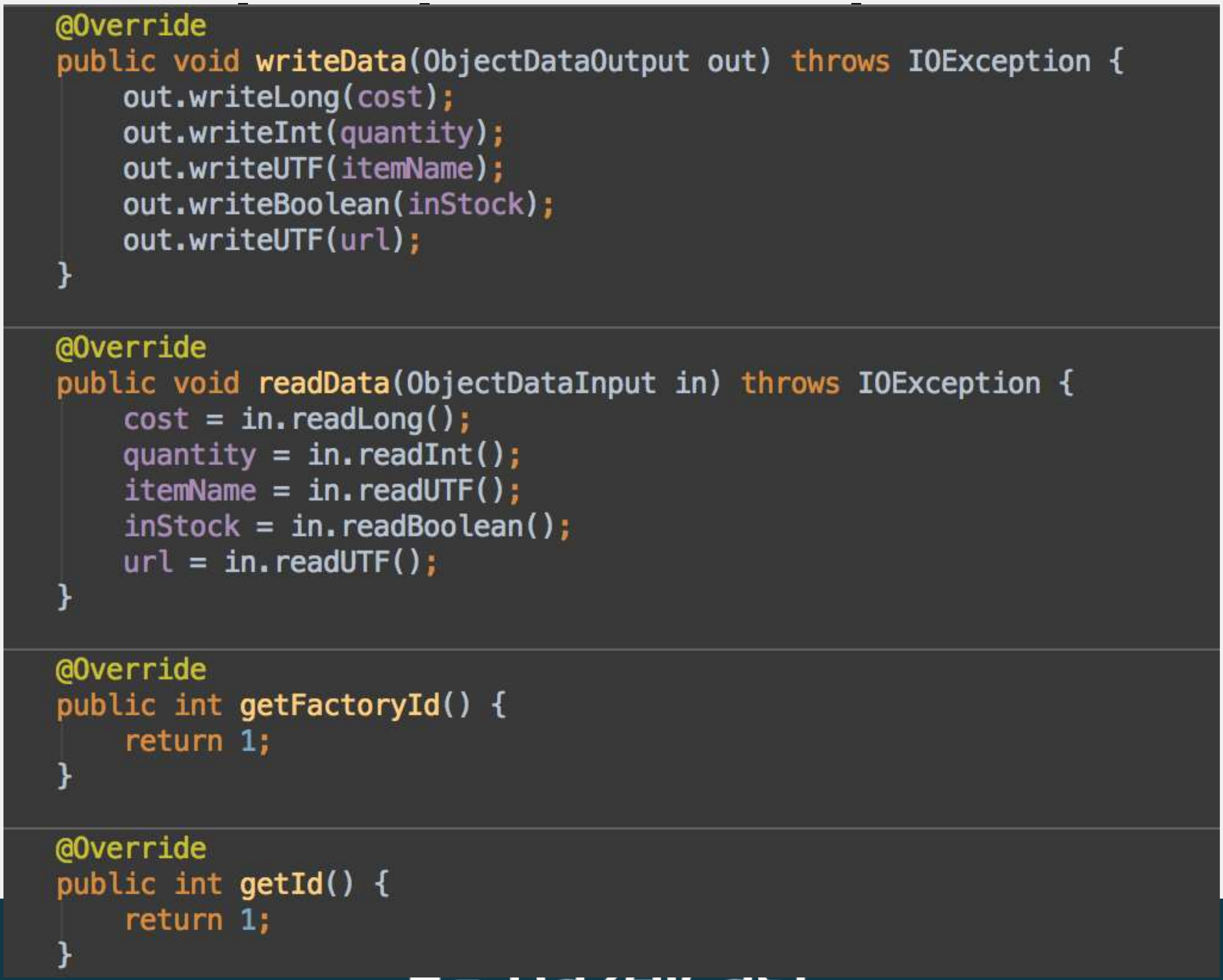
Pros

- Efficient than Serializable in terms of CPU and Memory
- Doesn’t use Reflection while de-serializing

Cons

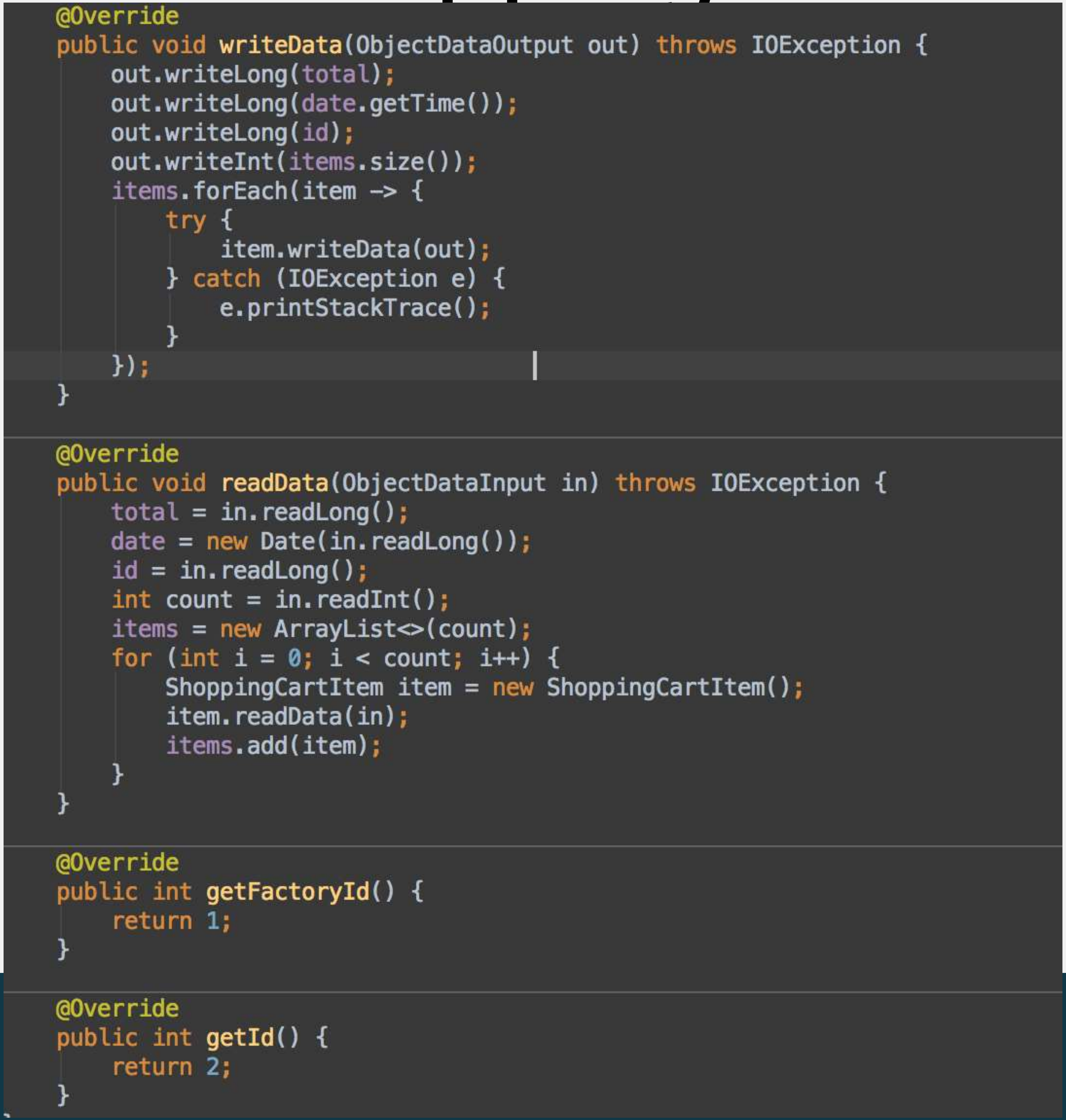
- Hazelcast specific
- Requires to implement the actual serialization
- Requires to implement a Factory and configuration

ShoppingCartItem -



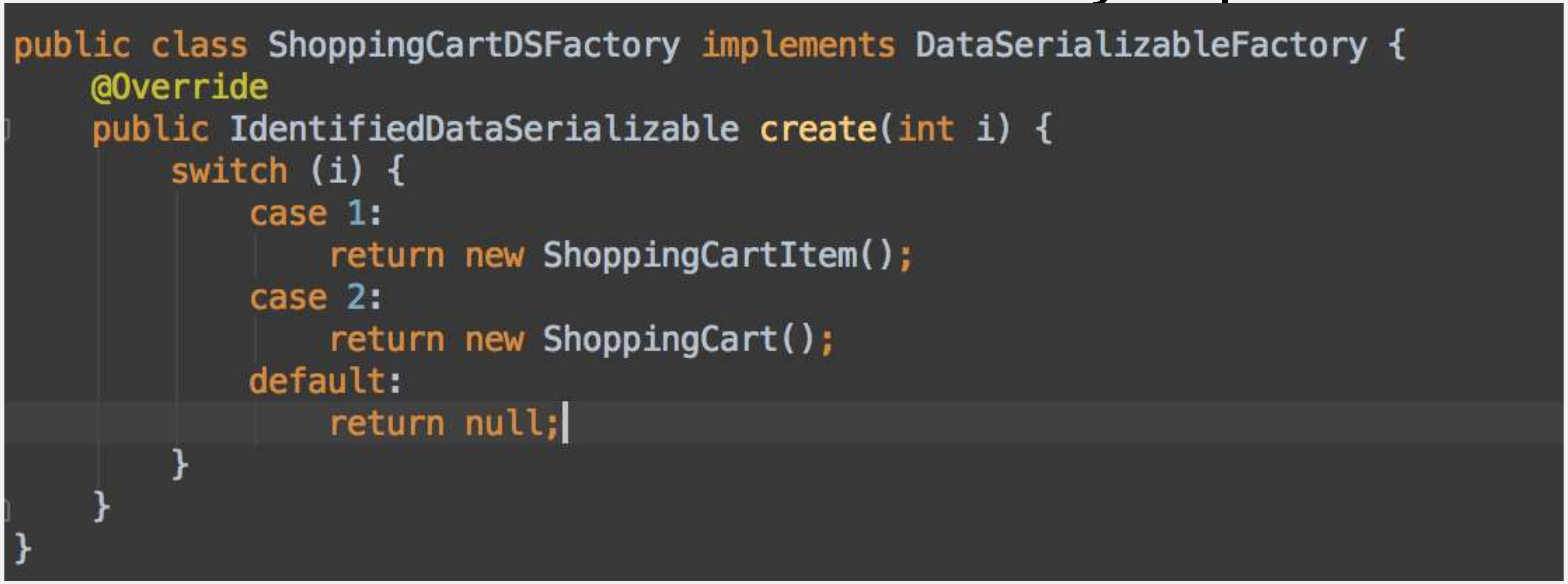
```
@Override
public void writeData(ObjectDataOutput out) throws IOException {
    out.writeLong(cost);
    out.writeInt(quantity);
    out.writeUTF(itemName);
    out.writeBoolean(inStock);
    out.writeUTF(url);
}

@Override
public void readData(ObjectDataInput in) throws IOException {
    cost = in.readLong();
    quantity = in.readInt();
    itemName = in.readUTF();
    inStock = in.readBoolean();
    url = in.readUTF();
}

@Override
public int getFactoryId() {
    return 1;
}

@Override
public int getId() {
    return 1;
}
```

ShoppingCart-

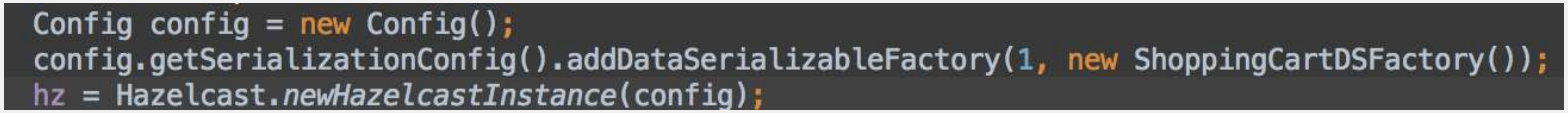


```java
@Override
public void writeData(ObjectDataOutput out) throws IOException {
    out.writeLong(total);
    out.writeLong(date.getTime());
    out.writeLong(id);
    out.writeInt(items.size());
    items.forEach(item -> {
        try {
            item.writeData(out);
        } catch (IOException e) {
            e.printStackTrace();
        }
    });
}

@Override
public void readData(ObjectDataInput in) throws IOException {
    total = in.readLong();
    date = new Date(in.readLong());
    id = in.readLong();
    int count = in.readInt();
    items = new ArrayList<>(count);
    for (int i = 0; i < count; i++) {
        ShoppingCartItem item = new ShoppingCartItem();
        item.readData(in);
        items.add(item);
    }
}

@Override
public int getFactoryId() {
    return 1;
}

@Override
public int getId() {
    return 2;
}
```

IdentifiedDataSerializable – Factory Implementation



```java
public class ShoppingCartDSFactory implements DataSerializableFactory {
    @Override
    public IdentifiedDataSerializable create(int i) {
        switch (i) {
            case 1:
                return new ShoppingCartItem();
            case 2:
                return new ShoppingCart();
            default:
                return null;
        }
    }
}
```



```java
Config config = new Config();
config.getSerializationConfig().addDataSerializableFactory(1, new ShoppingCartDSFactory());
hz = Hazelcast.newHazelcastInstance(config);
```

hazelcast

IdentifiedDataSerializable- Results

Read Performance

68 ops in ms

Write Performance

60 ops in ms

Binary object size

186 bytes

Portable

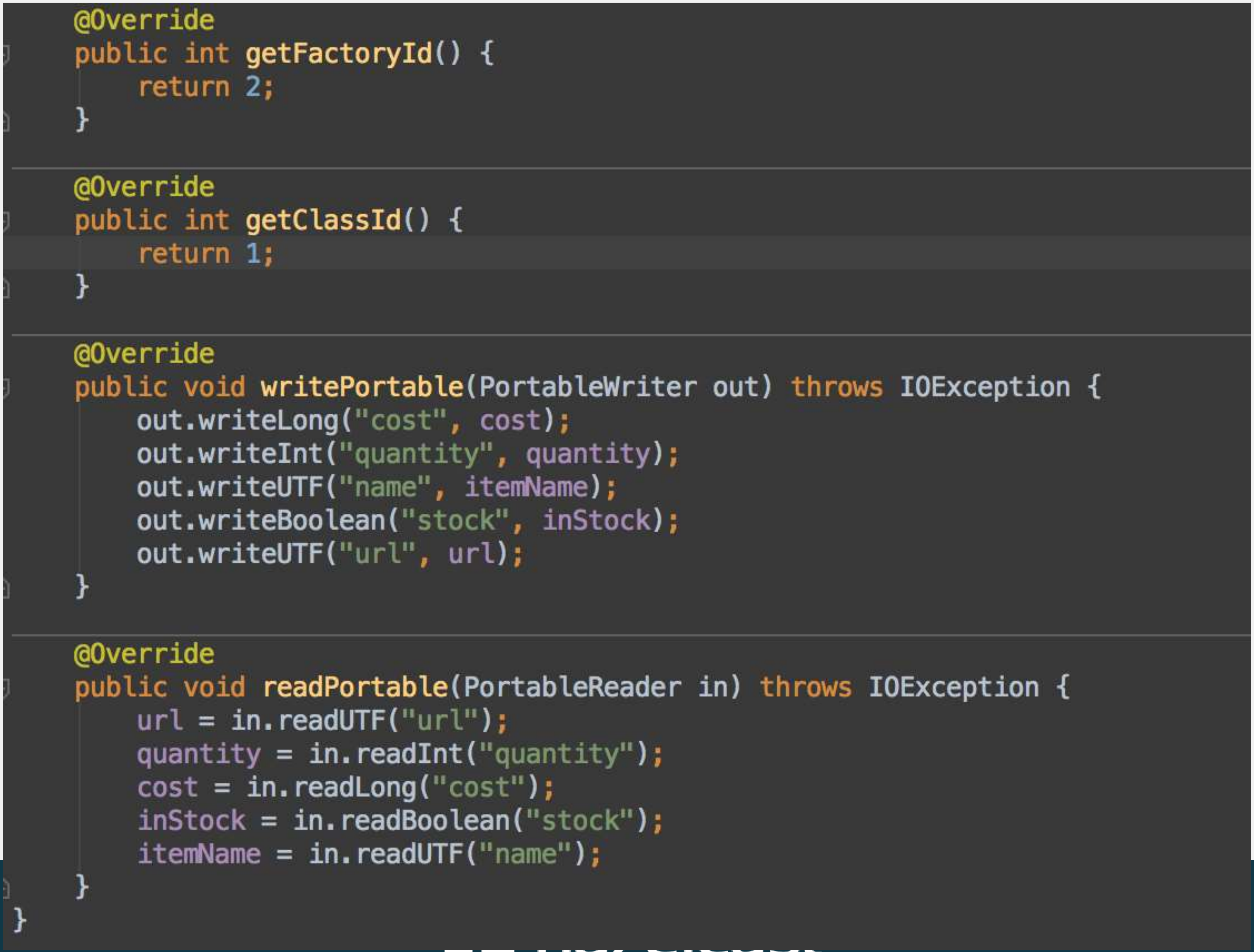
Pros

- Efficient than Serializable in terms of CPU and Memory
- Doesn’t use Reflection while de-serializing
- Supports versioning
- Supports partial de-serialization during Queries

Cons

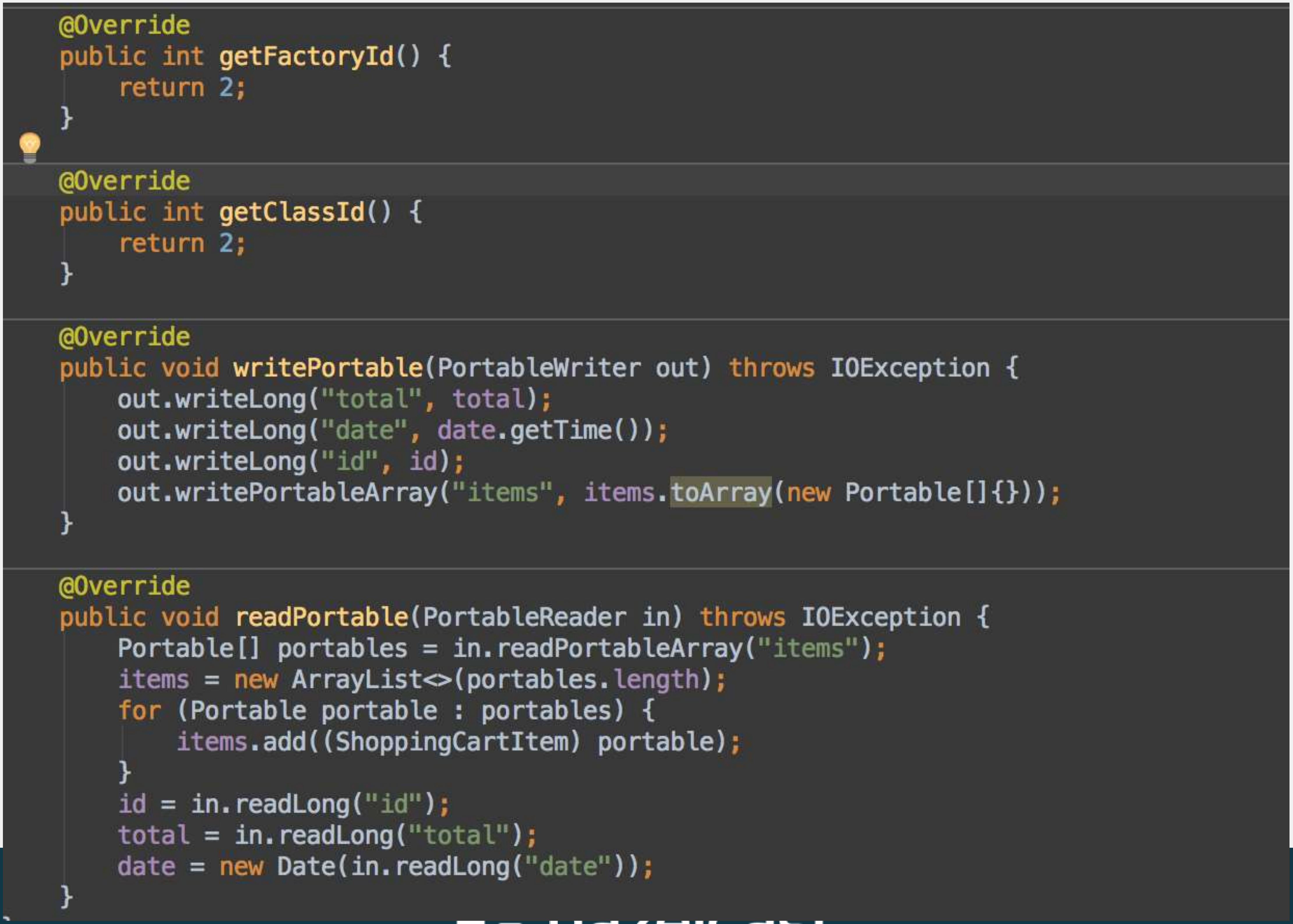
- Hazelcast specific
- Requires to implement the actual serialization
- Requires to implement a Factory and Class Definition
- Class definition is also sent together with Data but stored only once per class

ShoppingCartItem -



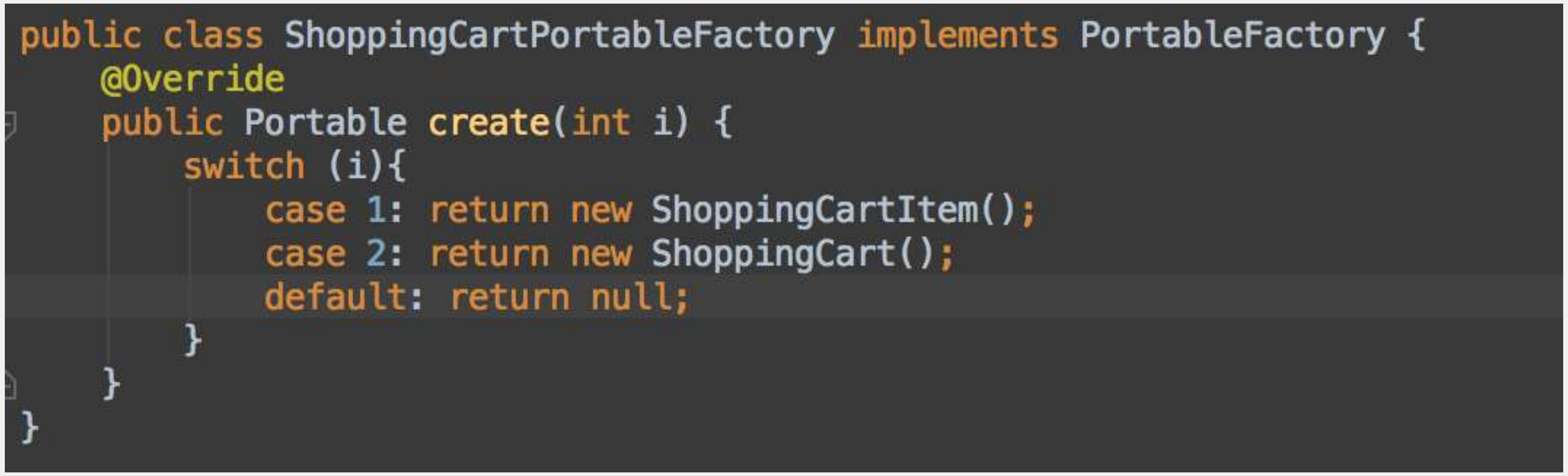
```
    @Override
    public int getFactoryId() {
        return 2;
    }

    @Override
    public int getClassId() {
        return 1;
    }

    @Override
    public void writePortable(PortableWriter out) throws IOException {
        out.writeLong("cost", cost);
        out.writeInt("quantity", quantity);
        out.writeUTF("name", itemName);
        out.writeBoolean("stock", inStock);
        out.writeUTF("url", url);
    }

    @Override
    public void readPortable(PortableReader in) throws IOException {
        url = in.readUTF("url");
        quantity = in.readInt("quantity");
        cost = in.readLong("cost");
        inStock = in.readBoolean("stock");
        itemName = in.readUTF("name");
    }
}
```

ShoppingCart-

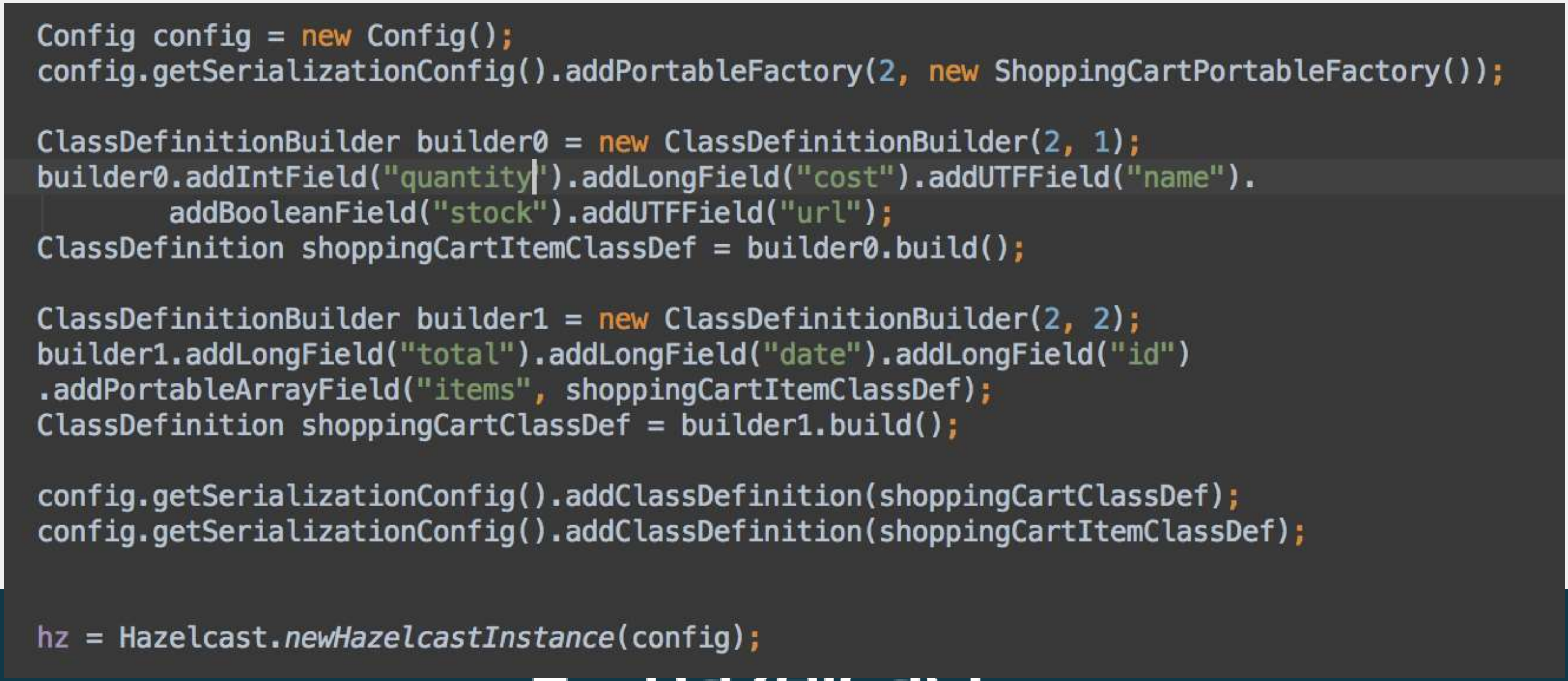


```java
@Override
public int getFactoryId() {
    return 2;
}

@Override
public int getClassId() {
    return 2;
}

@Override
public void writePortable(PortableWriter out) throws IOException {
    out.writeLong("total", total);
    out.writeLong("date", date.getTime());
    out.writeLong("id", id);
    out.writePortableArray("items", items.toArray(new Portable[]{}));
}

@Override
public void readPortable(PortableReader in) throws IOException {
    Portable[] portables = in.readPortableArray("items");
    items = new ArrayList<>(portables.length);
    for (Portable portable : portables) {
        items.add((ShoppingCartItem) portable);
    }
    id = in.readLong("id");
    total = in.readLong("total");
    date = new Date(in.readLong("date"));
}
```

Portable – Factory



```java
public class ShoppingCartPortableFactory implements PortableFactory {
    @Override
    public Portable create(int i) {
        switch (i){
            case 1: return new ShoppingCartItem();
            case 2: return new ShoppingCart();
            default: return null;
        }
    }
}
```



```java
Config config = new Config();
config.getSerializationConfig().addPortableFactory(2, new ShoppingCartPortableFactory());

ClassDefinitionBuilder builder0 = new ClassDefinitionBuilder(2, 1);
builder0.addIntField("quantity").addLongField("cost").addUTFField("name").
        addBooleanField("stock").addUTFField("url");
ClassDefinition shoppingCartItemClassDef = builder0.build();

ClassDefinitionBuilder builder1 = new ClassDefinitionBuilder(2, 2);
builder1.addLongField("total").addLongField("date").addLongField("id")
.addPortableArrayField("items", shoppingCartItemClassDef);
ClassDefinition shoppingCartClassDef = builder1.build();

config.getSerializationConfig().addClassDefinition(shoppingCartClassDef);
config.getSerializationConfig().addClassDefinition(shoppingCartItemClassDef);


hz = Hazelcast.newHazelcastInstance(config);
```

Portable- Results

Read Performance

65 ops in ms

Write Performance

54 ops in ms

Binary object size

386 bytes

Pluggable – (ex: Kryo)

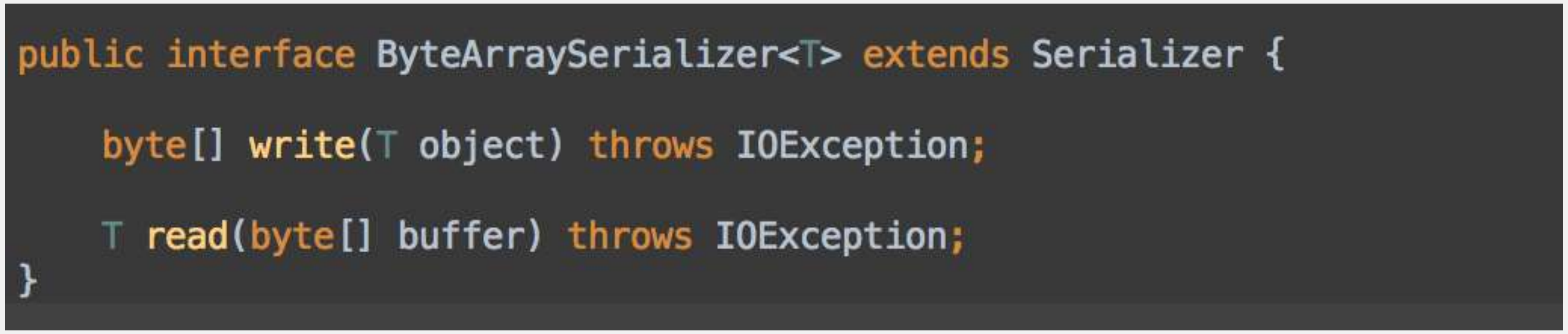
Pros

- Doesn’t require class to implement an interface
- Very convenient and flexible
- Can be stream based or byte array based

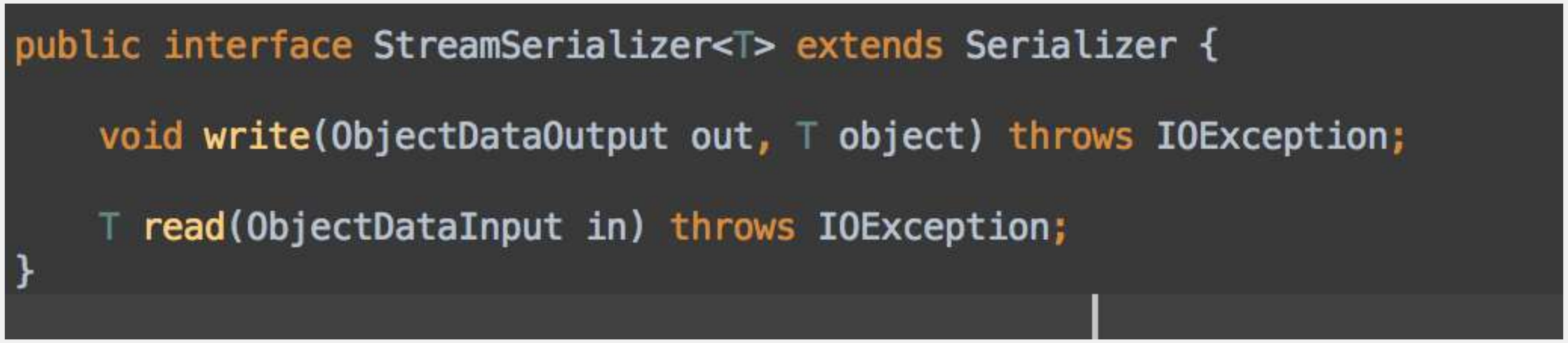
Cons

- Requires to implement the actual serialization
- Requires to plug and configure

Stream and ByteArray Serializers



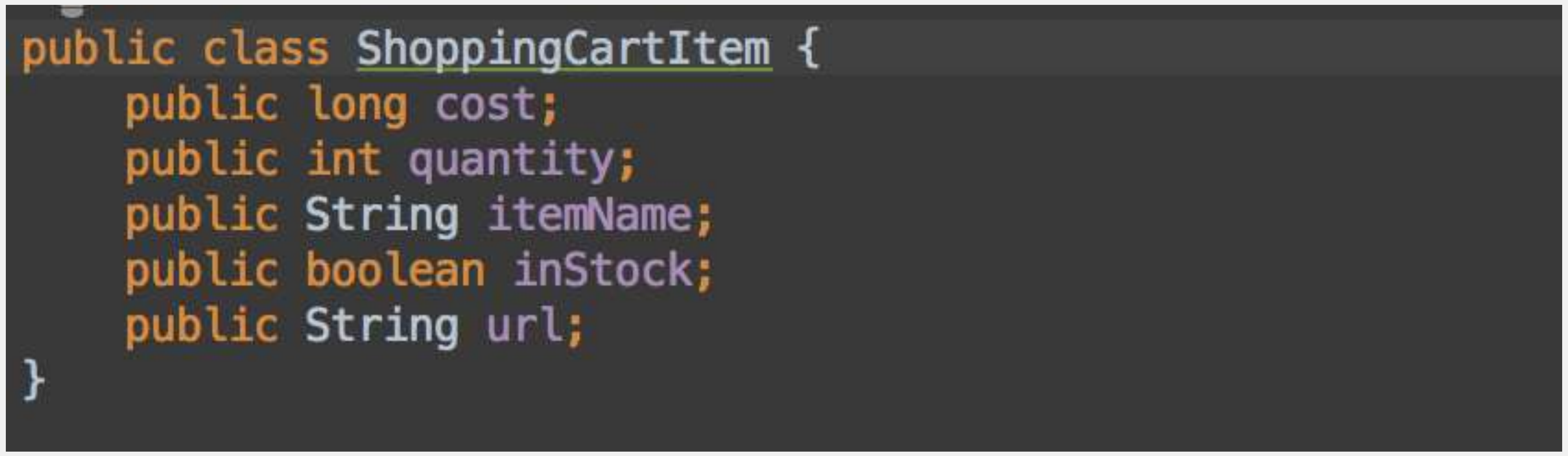
```java
public interface ByteArraySerializer<T> extends Serializer {

    byte[] write(T object) throws IOException;

    T read(byte[] buffer) throws IOException;
}
```



```java
public interface StreamSerializer<T> extends Serializer {

    void write(ObjectDataOutput out, T object) throws IOException;

    T read(ObjectDataInput in) throws IOException;
}
```

hazelcast

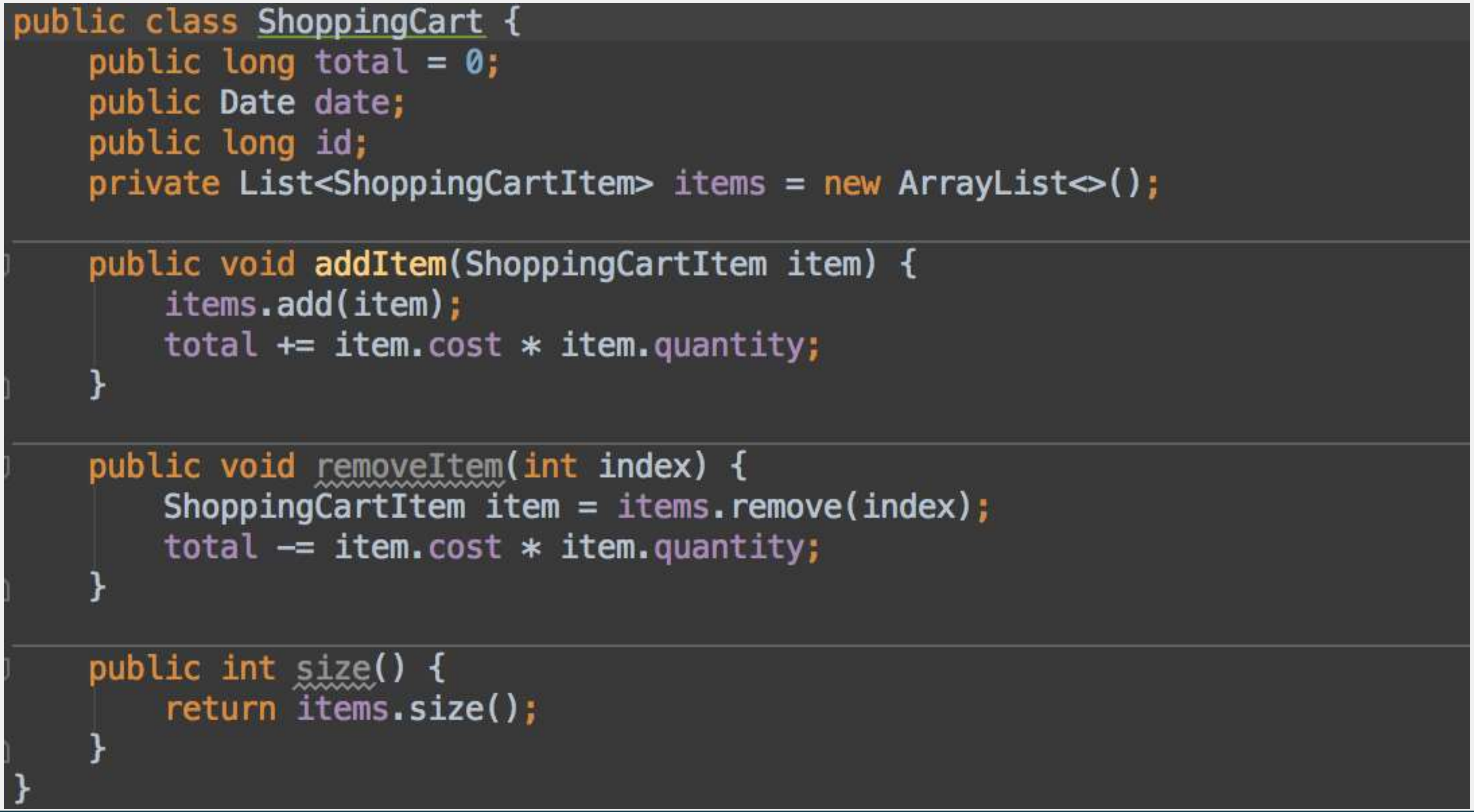
ShoppingCartItem - implementation



```
public class ShoppingCartItem {
    public long cost;
    public int quantity;
    public String itemName;
    public boolean inStock;
    public String url;
}
```

hazelcast

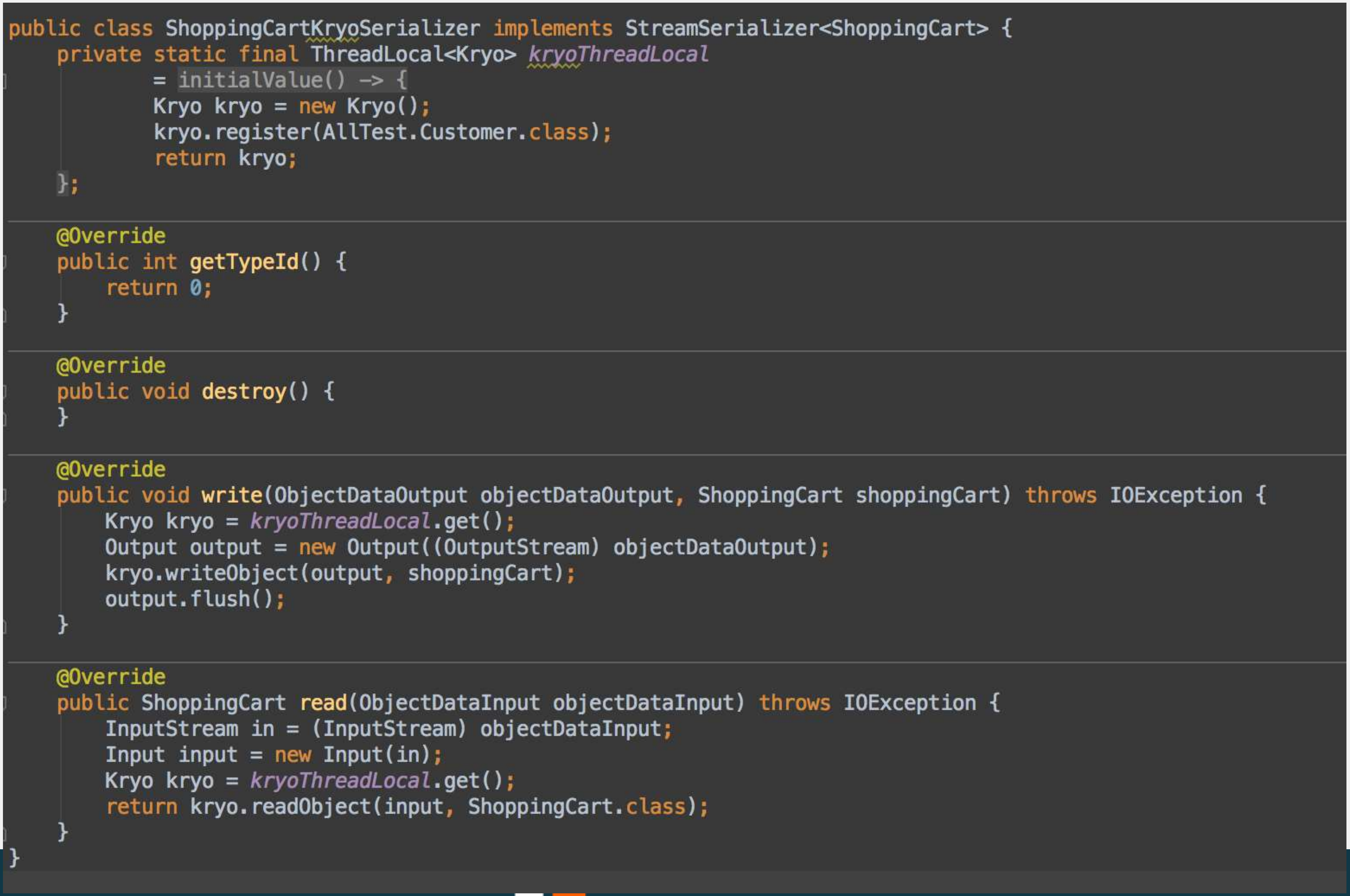
ShoppingCart-implementation



```
public class ShoppingCart {
    public long total = 0;
    public Date date;
    public long id;
    private List<ShoppingCartItem> items = new ArrayList<>();

    public void addItem(ShoppingCartItem item) {
        items.add(item);
        total += item.cost * item.quantity;
    }

    public void removeItem(int index) {
        ShoppingCartItem item = items.remove(index);
        total -= item.cost * item.quantity;
    }

    public int size() {
        return items.size();
    }
}
```

hazelcast

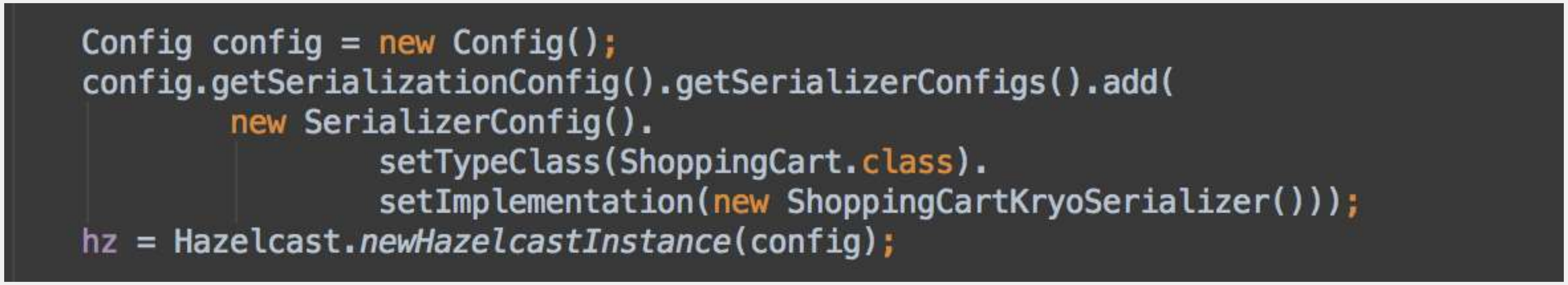
ShoppingCart Kryo



```java
public class ShoppingCartKryoSerializer implements StreamSerializer<ShoppingCart> {
    private static final ThreadLocal<Kryo> kryoThreadLocal
            = initialValue() -> {
            Kryo kryo = new Kryo();
            kryo.register(AllTest.Customer.class);
            return kryo;
    };

    @Override
    public int getTypeId() {
        return 0;
    }

    @Override
    public void destroy() {
    }

    @Override
    public void write(ObjectDataOutput objectDataOutput, ShoppingCart shoppingCart) throws IOException {
        Kryo kryo = kryoThreadLocal.get();
        Output output = new Output((OutputStream) objectDataOutput);
        kryo.writeObject(output, shoppingCart);
        output.flush();
    }

    @Override
    public ShoppingCart read(ObjectDataInput objectDataInput) throws IOException {
        InputStream in = (InputStream) objectDataInput;
        Input input = new Input(in);
        Kryo kryo = kryoThreadLocal.get();
        return kryo.readObject(input, ShoppingCart.class);
    }
}
```

hazelcast

Pluggable Serialization Configuration



```java
Config config = new Config();
config.getSerializationConfig().getSerializerConfigs().add(
        new SerializerConfig().
                setTypeClass(ShoppingCart.class).
                setImplementation(new ShoppingCartKryoSerializer()));
hz = Hazelcast.newHazelcastInstance(config);
```

Kryo- Results

Read Performance

60 ops in ms

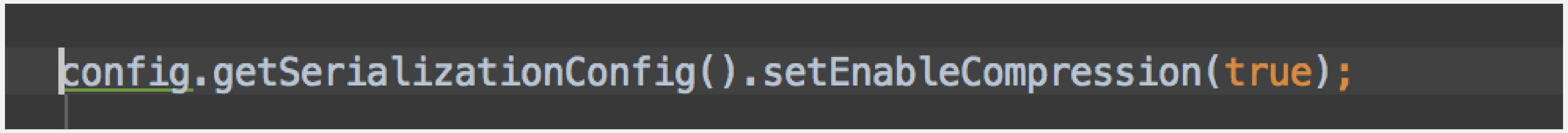
Write Performance

51 ops in ms

Binary object size

210 bytes

Compression



```
config.getSerializationConfig().setEnableCompression(true);
```

Compresses the data.

Can be applied to Serializable and Externalizable only.

Very slow (~1000 times) and CPU consuming.

Can reduce 525 bytes to 26 bytes.

Summary

Serializable
  R:31 ops/ms, W: 46 ops/ms, Size: 525 bytes

Externalizable
  R:61 ops/ms, W: 60 ops/ms, Size: 235 bytes

DataSerializable
  R:64 ops/ms, W: 59 ops/ms, Size: 231 bytes

IdentifiedDataSerializable
  R:68 ops/ms, W: 60 ops/ms, Size: 186 bytes

Portable
  R:65 ops/ms, W: 54 ops/ms, Size: 386 bytes

Kryo
  R:60 ops/ms, W: 51 ops/ms, Size: 210 bytes

Thank you ! :)

any questions ?

✉ emrah@hazelcast.com

hazelcast

http://www.zenika.com/formation-hazelcast-essentials.html

30Th November - Free Training

hazelcast