Implementing java.io.Expernalizable for Cache Performance and Low Byte Footprint

Introduction

- Cacheonix an open source clustered cache and distributed data management framework that allows organizations to predictably scale their mission-critical applications
- This presentation introduces a best practice of implementing java.io.Externalizable by cache keys and values for performance and low memory footprint

Problem: Default Java Serialization is too slow

- Is done by simply declaring a signature interface java.io.Serializable
- Very easy to implement, but
- Does a lot of useless things, automatically

Best Practice: Implement java.io.Externalizable

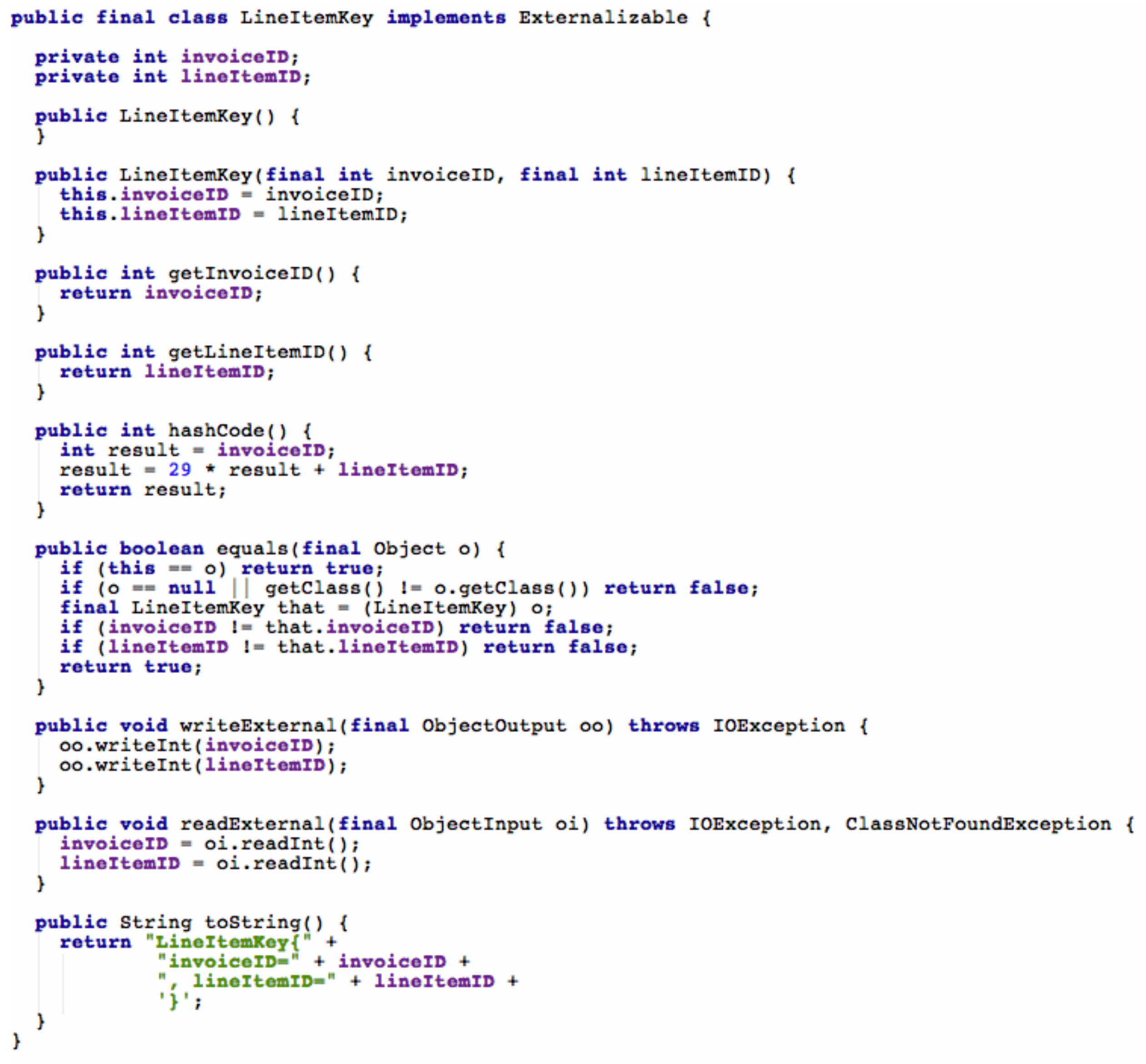
java.io.Externalizable

- Can be significantly faster (2-8 times than default serialization)
- 2-4 times smaller byte footprint – higher network throughput

Implementing java.io.Externalizable

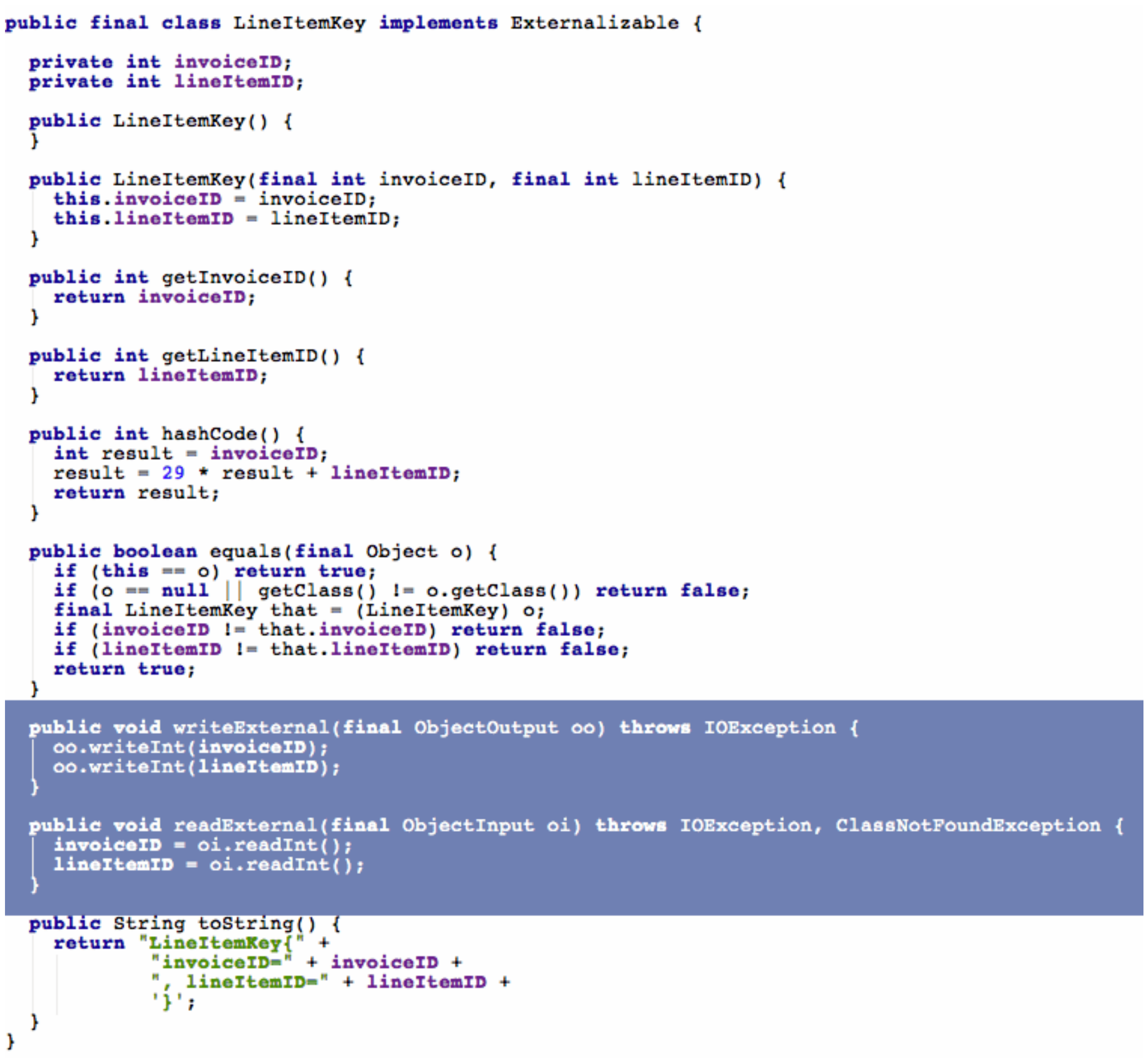
- Keys and values must provide a default public constructor
- Keys and values must implement methods writeExternal() and readExternal() and

Externalizable Example



```java
public final class LineItemKey implements Externalizable {

  private int invoiceID;
  private int lineItemID;

  public LineItemKey() {
  }

  public LineItemKey(final int invoiceID, final int lineItemID) {
    this.invoiceID = invoiceID;
    this.lineItemID = lineItemID;
  }

  public int getInvoiceID() {
    return invoiceID;
  }

  public int getLineItemID() {
    return lineItemID;
  }

  public int hashCode() {
    int result = invoiceID;
    result = 29 * result + lineItemID;
    return result;
  }

  public boolean equals(final Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    final LineItemKey that = (LineItemKey) o;
    if (invoiceID != that.invoiceID) return false;
    if (lineItemID != that.lineItemID) return false;
    return true;
  }

  public void writeExternal(final ObjectOutput oo) throws IOException {
    oo.writeInt(invoiceID);
    oo.writeInt(lineItemID);
  }

  public void readExternal(final ObjectInput oi) throws IOException, ClassNotFoundException {
    invoiceID = oi.readInt();
    lineItemID = oi.readInt();
  }

  public String toString() {
    return "LineItemKey{" +
            "invoiceID=" + invoiceID +
            ", lineItemID=" + lineItemID +
            '}';
  }
}
```

Externalizable Example



```java
public final class LineItemKey implements Externalizable {

  private int invoiceID;
  private int lineItemID;

  public LineItemKey() {
  }

  public LineItemKey(final int invoiceID, final int lineItemID) {
    this.invoiceID = invoiceID;
    this.lineItemID = lineItemID;
  }

  public int getInvoiceID() {
    return invoiceID;
  }

  public int getLineItemID() {
    return lineItemID;
  }

  public int hashCode() {
    int result = invoiceID;
    result = 29 * result + lineItemID;
    return result;
  }

  public boolean equals(final Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    final LineItemKey that = (LineItemKey) o;
    if (invoiceID != that.invoiceID) return false;
    if (lineItemID != that.lineItemID) return false;
    return true;
  }

  public void writeExternal(final ObjectOutput oo) throws IOException {
    oo.writeInt(invoiceID);
    oo.writeInt(lineItemID);
  }

  public void readExternal(final ObjectInput oi) throws IOException, ClassNotFoundException {
    invoiceID = oi.readInt();
    lineItemID = oi.readInt();
  }

  public String toString() {
    return "LineItemKey{" +
           "invoiceID=" + invoiceID +
           ", lineItemID=" + lineItemID +
           '}';
  }
}
```

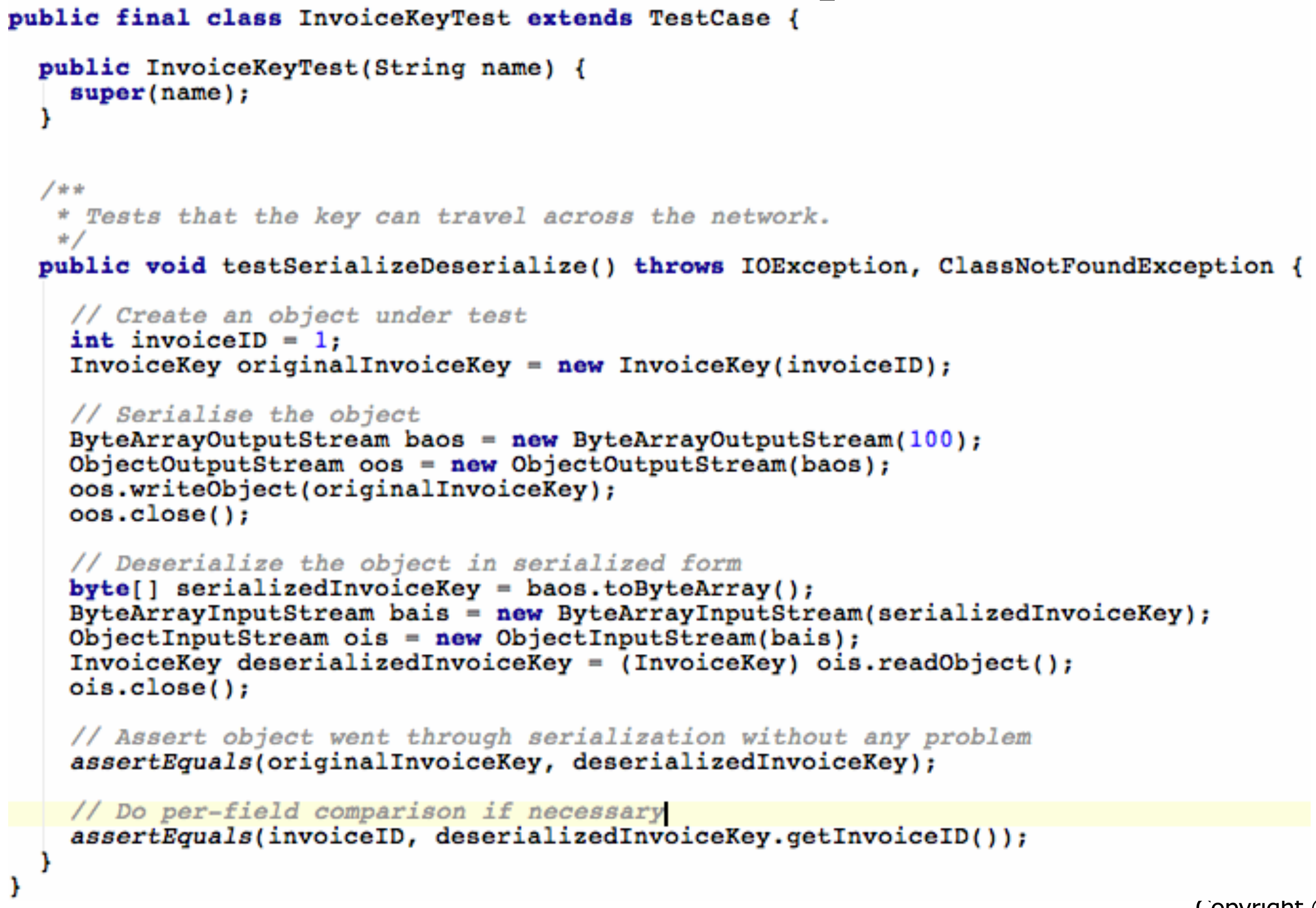
Externalizable Example

```
public void writeExternal(final ObjectOutput oo)
       throws IOException {

  oo.writeInt(invoiceID);
  oo.writeInt(lineItemID);
}

public void readExternal(final ObjectInput oi)
       throws IOException, ClassNotFoundException {

  invoiceID = oi.readInt();
  lineItemID = oi.readInt();
}
```

Best Practice: Test for Serializability

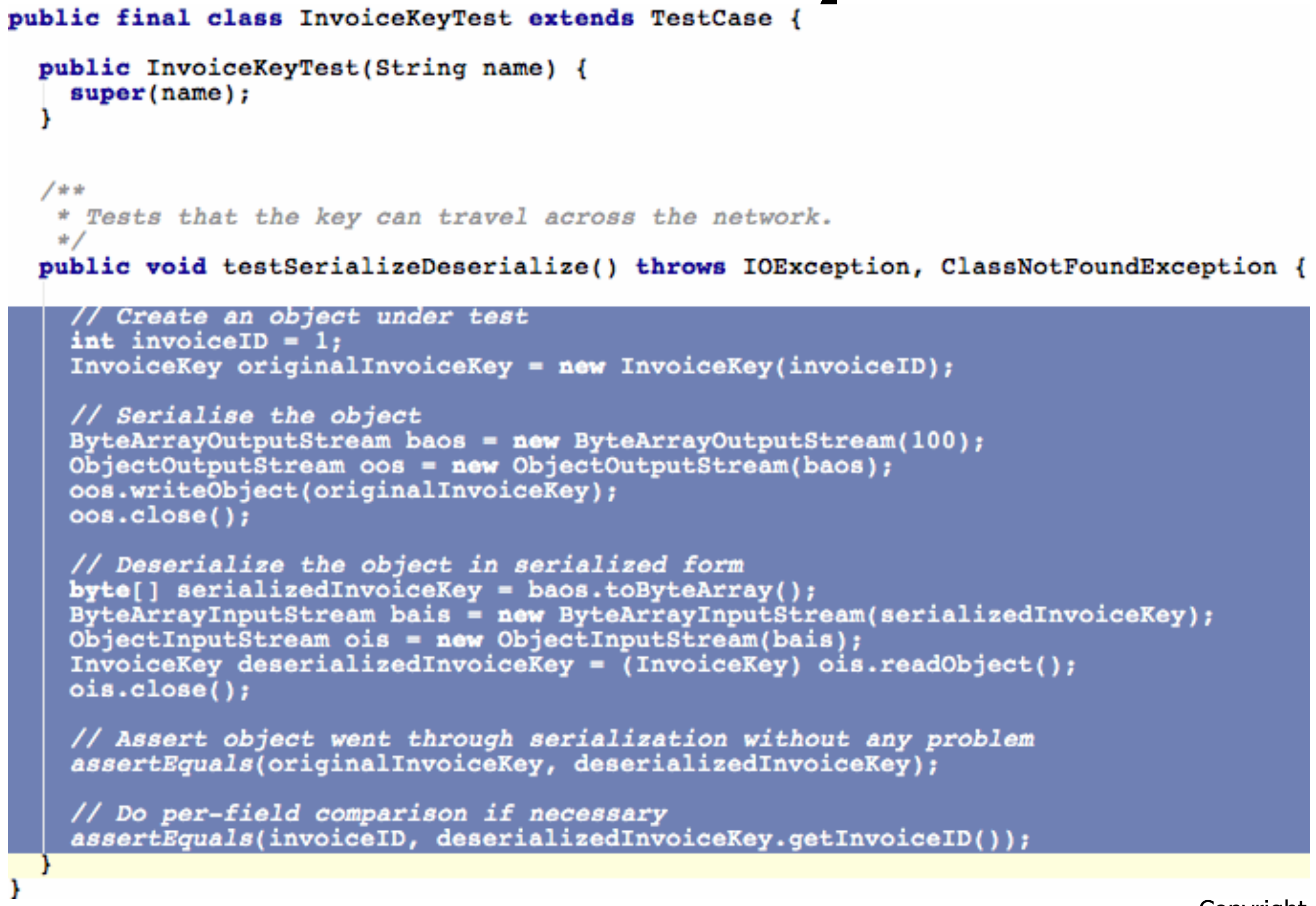
- You must ensure that the object that was received at another end is the object that was sent
- Cache keys AND cached values routinely travel across the network
- It is critical to write proper serialization tests for keys and values
- Test pattern: Serialize, deserialize, compare

Testing for Serializability Example



```java
public final class InvoiceKeyTest extends TestCase {

  public InvoiceKeyTest(String name) {
    super(name);
  }


  /**
   * Tests that the key can travel across the network.
   */
  public void testSerializeDeserialize() throws IOException, ClassNotFoundException {

    // Create an object under test
    int invoiceID = 1;
    InvoiceKey originalInvoiceKey = new InvoiceKey(invoiceID);

    // Serialise the object
    ByteArrayOutputStream baos = new ByteArrayOutputStream(100);
    ObjectOutputStream oos = new ObjectOutputStream(baos);
    oos.writeObject(originalInvoiceKey);
    oos.close();

    // Deserialize the object in serialized form
    byte[] serializedInvoiceKey = baos.toByteArray();
    ByteArrayInputStream bais = new ByteArrayInputStream(serializedInvoiceKey);
    ObjectInputStream ois = new ObjectInputStream(bais);
    InvoiceKey deserializedInvoiceKey = (InvoiceKey) ois.readObject();
    ois.close();

    // Assert object went through serialization without any problem
    assertEquals(originalInvoiceKey, deserializedInvoiceKey);

    // Do per-field comparison if necessary
    assertEquals(invoiceID, deserializedInvoiceKey.getInvoiceID());
  }
}
```

Testing for Serializability Example



```java
public final class InvoiceKeyTest extends TestCase {

  public InvoiceKeyTest(String name) {
    super(name);
  }


  /**
   * Tests that the key can travel across the network.
   */
  public void testSerializeDeserialize() throws IOException, ClassNotFoundException {

    // Create an object under test
    int invoiceID = 1;
    InvoiceKey originalInvoiceKey = new InvoiceKey(invoiceID);

    // Serialise the object
    ByteArrayOutputStream baos = new ByteArrayOutputStream(100);
    ObjectOutputStream oos = new ObjectOutputStream(baos);
    oos.writeObject(originalInvoiceKey);
    oos.close();

    // Deserialize the object in serialized form
    byte[] serializedInvoiceKey = baos.toByteArray();
    ByteArrayInputStream bais = new ByteArrayInputStream(serializedInvoiceKey);
    ObjectInputStream ois = new ObjectInputStream(bais);
    InvoiceKey deserializedInvoiceKey = (InvoiceKey) ois.readObject();
    ois.close();

    // Assert object went through serialization without any problem
    assertEquals(originalInvoiceKey, deserializedInvoiceKey);

    // Do per-field comparison if necessary
    assertEquals(invoiceID, deserializedInvoiceKey.getInvoiceID());
  }
}
```

Downloading Cacheonix

http://downloads.cacheonix.org

Questions or Suggestions?

Contact us
at
simeshev@cacheonix.org
or
www.cacheonix.org