Persisting Objects with Serialization

Jim Wilson
MOBILE SOLUTIONS DEVELOPER & ARCHITECT
@hedgehogjim   blog.jwhh.com

Overview

Purpose and capabilities of serialization

Making a type serializable

Serializing/deserializing an object

Creating class version compatibility

Custom serialization

Transient fields

Persisting Java Objects

Java has built-in ability to persist objects

- Store from runtime into a byte stream
- Restore from byte stream into runtime

Most cases require very little programming

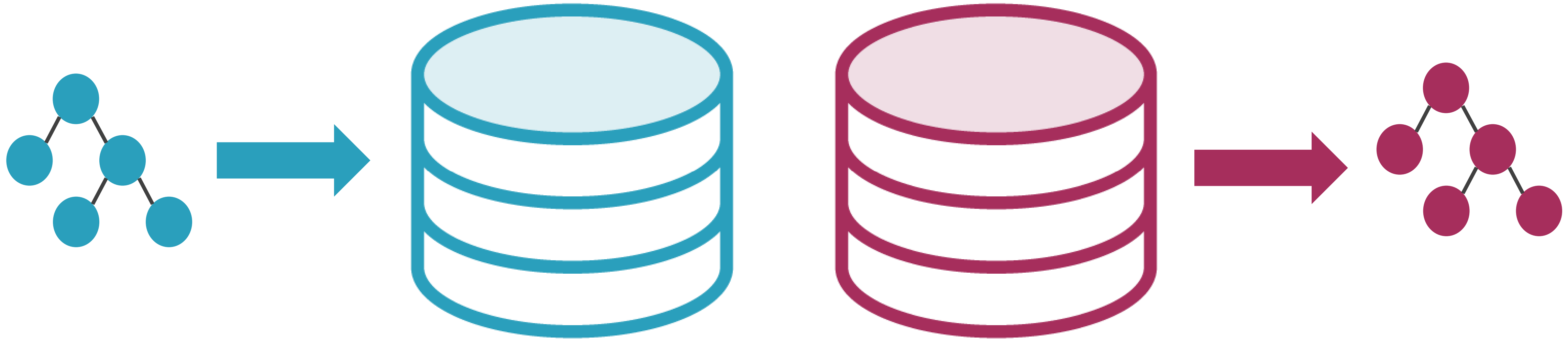
- Leverages reflection
- Operates only on instance members
- Customizable

Persisting Java Objects

Opens up many possibilities

- Save to file system
- Save in a database
    - RDBMS can store as blob
    - OODBMS can often store directly
- Pass across memory address boundaries
- Pass over network

Serialization



Serializing

Storing an object-graph to a byte stream

Deserializing

Restoring an object-graph from a byte stream

Serialization Types

Serializable

- Implemented by serializable types
- Indicates that type supports serialization
- Has no methods

ObjectOutputStream

- Serializes object-graph to stream

ObjectInputStream

- Deserializes stream to object-graph

Being Serializable

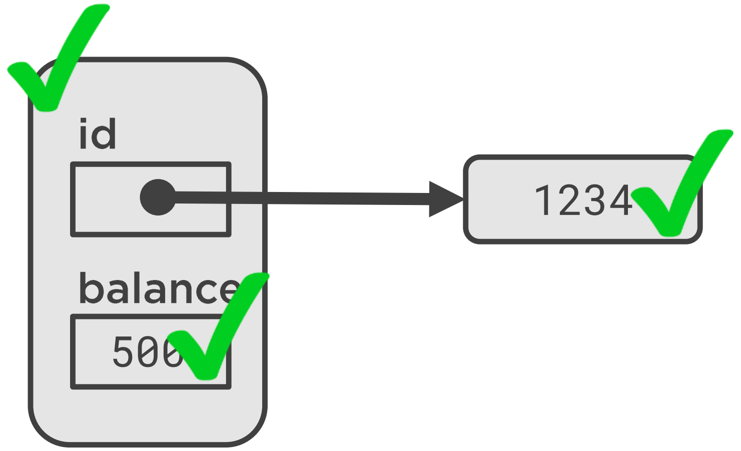
Requirements to be serializable

- Implement Serializable
- Members are serializable
  - Primitive types are serializable
  - Others must implement Serializable

Being Serializable

```
public class BankAccount implements Serializable {

  private final String id;
  private int balance = 0;

  public BankAccount(String id) {...}
  public BankAccount(String id, int balance) {...}

  // other members elided
}
```

```
new BankAccount(“1234”, 500);
```



Serializing an Object

```
BankAccount acct = new BankAccount(“1234”, 500);
acct.deposit(250);
saveAccount(acct, “account.dat”);
```

```
void saveAccount(BankAccount ba, String filename) {
  try(ObjectOutputStream objectStream =
      new ObjectOutputStream(Files.newOutputStream(Paths.get(filename)))) {
    objectStream.writeObject(ba);
  } catch (IOException e) {
    // . . .
  }
}
```

Deserializing an Object

```
BankAccount loadAccount(String filename) {

  BankAccount ba = null;

      ObjectInputStream objectStream =
      new ObjectInputStream(Files.newInputStream(Paths.get(filename)))

    ba = (BankAccount) objectStream.readObject();

  } catch (IOException e) {
    // . . .
  } catch (ClassNotFoundException e) {
    // . . .
  }
  return ba;
}
```

```
1234 : 750
```

```
BankAccount acct = loadAccount(“account.dat”);

System.out.println(acct.getId() + “ : ” + acct.getBalance());
```

Class Version Incompatibility

```
public class BankAccount implements Serializable {

  private final String id;
  private int balance = 0;
  // constructors & getters elided

  public synchronized void deposit(int amount) {
    balance += amount;
  }

  public synchronized void withdrawal(int amount) {
    balance -= amount;
  }
}
```

```
BankAccount acct = new BankAccount(“1234”, 500);
acct.deposit(250);
saveAccount(acct, “account.dat”);
```

Class Version Incompatibility

```
public class BankAccount implements Serializable {

  private final String id;
  private int balance = 0;
  private char lastTxType;
  private int lastTxAmount;

  // constructors & getters elided

  public synchronized void deposit(int amount) {
    balance += amount;
    lastTxType = 'd';
    lastTxAmount = amount;
  }

  public synchronized void withdrawal(int amount) {
    balance -= amount;
    lastTxType = 'w';
    lastTxAmount = amount;
  }
}
```

InvalidClassException

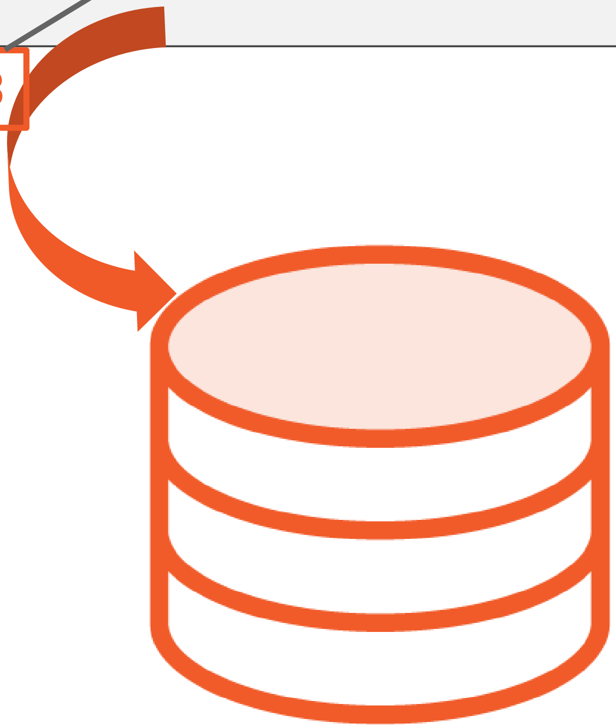
```
BankAccount acct = loadAccount("account.dat");
```

Class Version Incompatibility

```
public class BankAccount implements Serializable {

  private final String id;
  private int balance = 0;

  // other members elided
}
```

```
-6328564483941980673
```

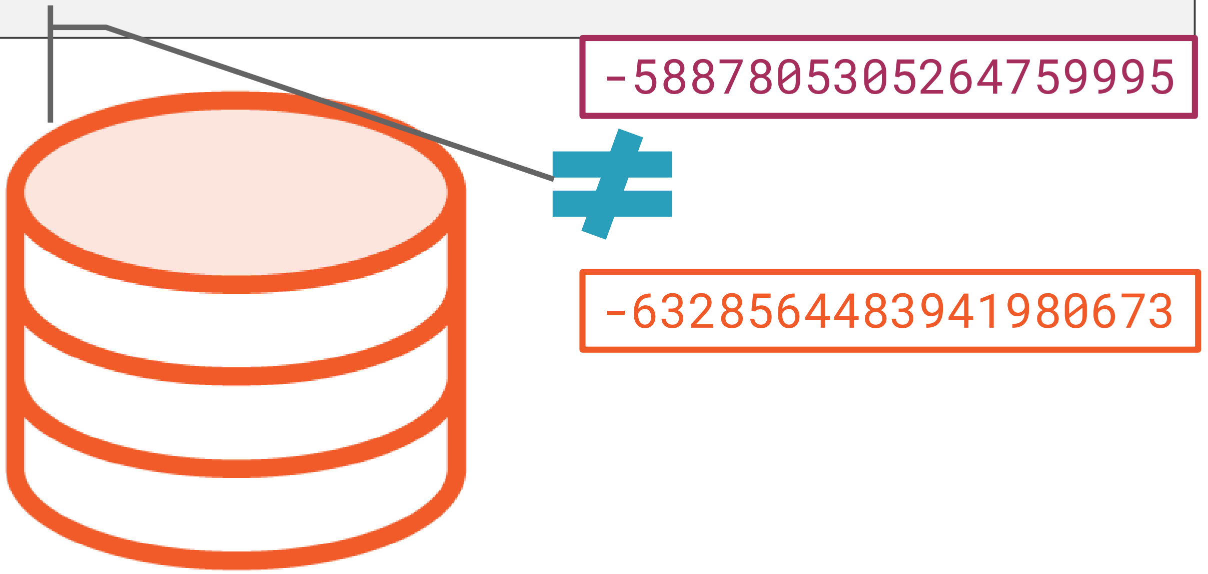
Serial version unique identifier



Class Version Incompatibility

```
public class BankAccount implements Serializable {

  private final String id;
  private int balance = 0;
  private char lastTxType;
  private int lastTxAmount;

  // other members elided
}
```

InvalidClassException



Creating Class Version Compatibility

Serial version unique identifier

- Indicates version compatibility
  - Compatible versions have same value

Java can calculate at runtime

- Value affected by a number of factors
  - Full type name
  - Implemented interfaces
  - Members
- Type content determines compatibility

Can specify as part of type definition

- Developer determines compatibility

Creating Class Version Compatibility

Specifying serial version unique identifier

- Add serialVersionUID field
  - Must be a long
  - Must be static final
  - Should be private

Calculate for initial version of type

- Use serialver utility

Use same value for future versions

- Maintains version compatibility

Creating Class Version Compatibility

The serialver utility

- Performs same calculation as Java runtime
- Found in JDK bin folder
    - IDEs often provide a plug-in

Using serialver utility

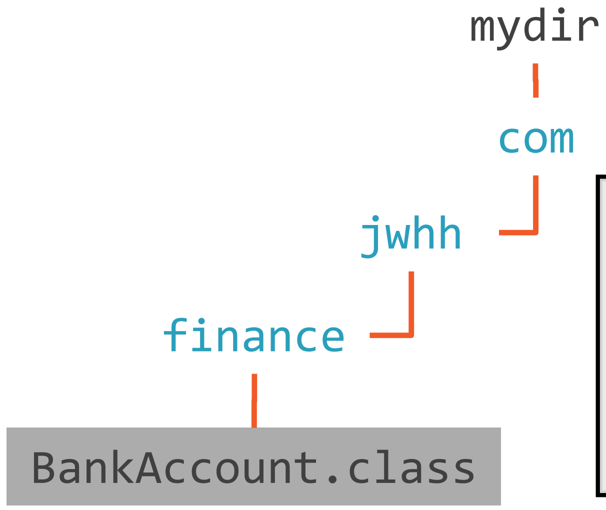
- Uses class’ class file
    - Will search in local folder
    - Can specify –classpath
- Can pass class name on command line
    - Displays value to console
- Can use –show option
    - Opens a simple graphical UI

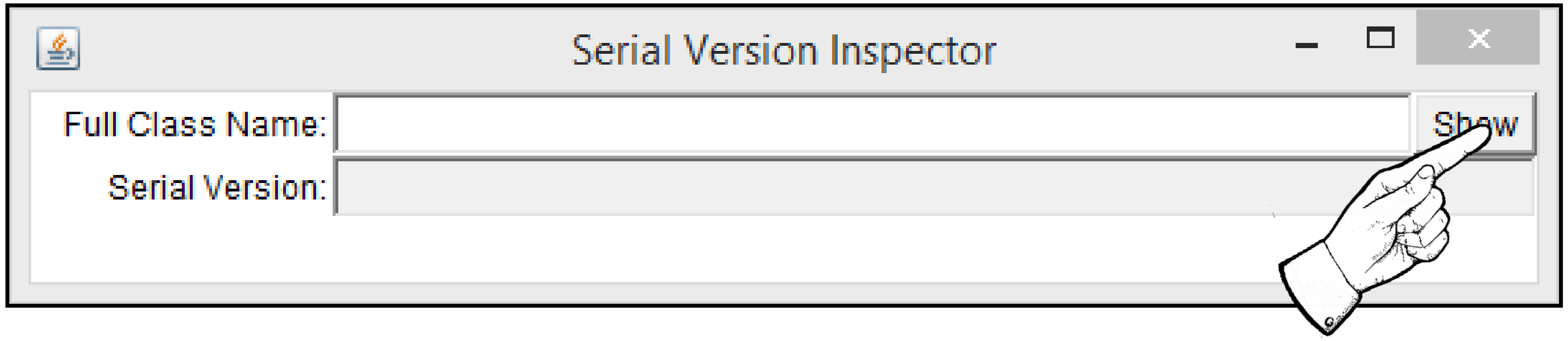
Creating Class Version Compatibility

```
package com.jwhh.finance;

public class BankAccount implements Serializable {

  private final String id;
  private int balance = 0;

  // other members elided
}
```

```
C:\mydir> serialver com.jwhh.finance.BankAccount
```

```
C:\mydir> serialver -show
```

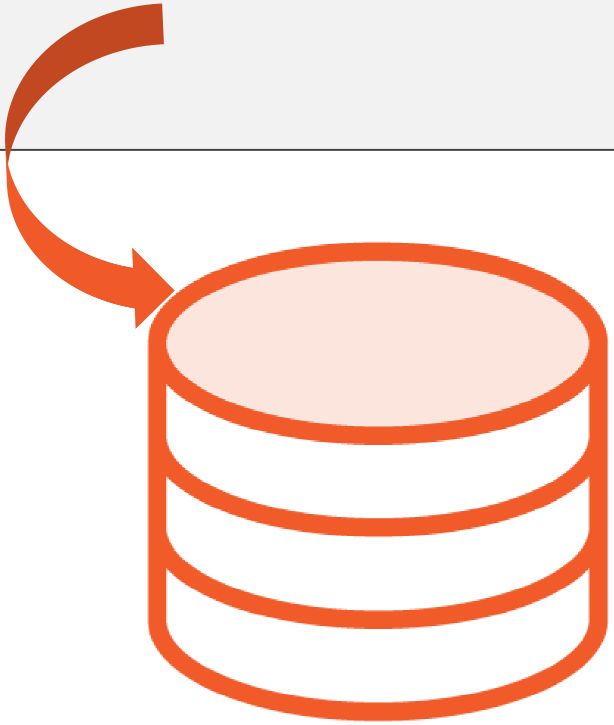




Creating Class Version Compatibility

From serialver utility

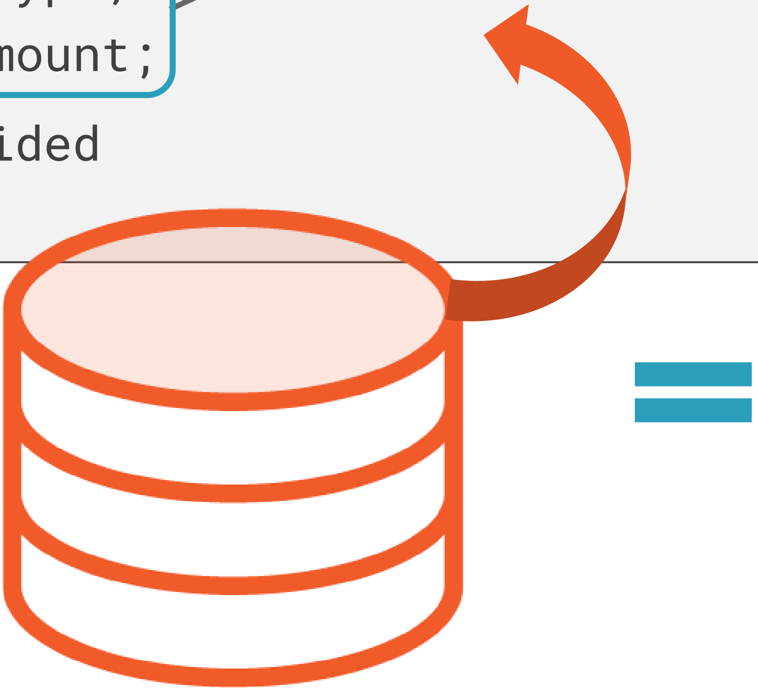
```
public class BankAccount implements Serializable {

  private static final long serialVersionUID = -6328564483941980673L;

  private final String id;
  private int balance = 0;

  // other members elided
}
```



Creating Class Version Compatibility

```
public class BankAccount implements Serializable {

  private static final long serialVersionUID = -6328564483941980673L;

  private final String id;
  private int balance = 0;
  private char lastTxType;
  private int lastTxAmount;

  // other members elided
}
```

Default value for an uninitialized member



-6328564483941980673

=

-6328564483941980673

Customizing Serialization

```
public class BankAccount implements Serializable {

  private final String id;
  private int balance = 0;
  // constructors & getters elided

  public synchronized void deposit(int amount) {
    balance += amount;
  }

  public synchronized void withdrawal(int amount) {
    balance -= amount;
  }
}
```

```
BankAccount acct1 = new BankAccount(“1234”, 500);
acct1.deposit(250);
saveAccount(acct1, “account1.dat”);
```

Customizing Serialization

```
public class BankAccount implements Serializable {

  private final String id;
  private int balance = 0;
  private char lastTxType;
  private int lastTxAmount;

  // constructors & getters elided
```

```
 public synchronized void deposit(int amount) {
   balance += amount;
   lastTxType = ‘d’;
   lastTxAmount = amount;
 }

 public synchronized void withdrawal(int amount) {
   balance -= amount;
   lastTxType = ‘w’;
   lastTxAmount = amount;
 }
}
```

```
BankAccount acct2 = new BankAccount(“9876”, 500);
saveAccount(acct2, “account2.dat”);
```

Customizing Serialization

```
public class BankAccount implements Serializable {
  private final String id;
  private int balance = 0;
  private char lastTxType;
  private int lastTxAmount;
  // other members elided
}
```

```
lastTxType = '\0'
lastTxAmount = 0
```

```
BankAccount acct2 = loadAccount("account2.dat");
```

```
lastTxType = '\0'
lastTxAmount = 0
```

```
BankAccount acct1 = loadAccount("account1.dat");
```

Customizing Serialization

Can add custom serialization handling

- To handle serializing
  - Add writeObject method to type
- To handle deserialzing
  - Add readObject method to type

Methods called through reflection

- Methods normally marked private

Customizing Serialization

Implementing writeObject method

- Return type of void
- Include throws clause
  - IOException
- Accepts ObjectOutputStream
  - Use to write values
  - defaultWriteObject for default behavior

Customizing Serialization

```java
public class BankAccount implements Serializable {

  private final String id;
  private int balance = 0;
  private char lastTxType;
  private int lastTxAmount;

  // other members elided
```

```java
  private void writeObject(ObjectOutputStream out)
      throws IOException {

    out.defaultWriteObject();
  }









}
```

Customizing Serialization

Implementing readObject method

- Return type of void
- Include throws clause
  - IOException
  - ClassNotFoundException
- Accepts ObjectInputStream
  - Use to read values
  - Use readFields to get field name info
    - Can access values by name
  - defaultReadObject for default behavior

Customizing Serialization

```
public class BankAccount implements Serializable {

  private final String id;
  private int balance = 0;
  private char lastTxType;
  private int lastTxAmount;

  // other members elided
```

```
  private void writeObject(ObjectOutputStream out)
      throws IOException {

    out.defaultWriteObject();
  }

  private void readObject(ObjectInputStream in)
      throws IOException, ClassNotFoundException {

    ObjectInputStream.GetField fields = in.readFields();

    id = (String) fields.get("id", null);
    balance = fields.get("balance", 0);
    lastTxType = fields.get("lastTxType", 'u');
    lastTxAmount = fields.get("lastTxAmount", -1);
  }
}
```

Customizing Serialization

```
public class BankAccount implements Serializable {
  private final String id;
  private int balance = 0;
  private char lastTxType;
  private int lastTxAmount;

  private void writeObject(ObjectOutputStream out) { ...}

  private void readObject(ObjectInputStream in) { ...}

  // other members elided
}
```

```
lastTxType = ‘\0’
lastTxAmount = 0
```

```
lastTxType = ‘u’
lastTxAmount = -1
```

```
BankAccount acct2 = loadAccount(“account2.dat”);
```

```
BankAccount acct1 = loadAccount(“account1.dat”);
```

Transient Fields

In some cases don’t want all fields serialized

- Useful for fields derived from another
- Avoids unnecessary use of storage

Use transient keyword

- Excludes field from serialization

Normally restore value manually

- Use custom serialization

Transient Fields

```
public class AccountGroup implements Serializable {
  private Map<String, BankAccount> accountMap = new HashMap<>();

  private transient int totalBalance;

  public int getTotalBalance() { return totalBalance; }

  public void addAccount(BankAccount account) {
    totalBalance += account.getBalance();
    accountMap.put(account.getId(), account);
  }
}
```

Serializing with a Transient Field

```
BankAccount acct1 = new BankAccount(“1234”, 500);
BankAccount acct2 = new BankAccount(“9876”, 750);
AccountGroup group = new AccountGroup();
group.add(acct1);
group.add(acct2);
saveGroup(group, “group.dat”);
```

```
void saveGroup(AccountGroup g, String filename) {
  try(ObjectOutputStream objectStream =
      new ObjectOutputStream(Files.newOutputStream(Paths.get(filename)))) {
    objectStream.writeObject(g);
  } catch(IOException e)
    // . . .
  }
}
```

Deserializing with a Transient Field

```
AccountGroup loadGroup(String filename) {

  AccountGroup g = null;

  try(ObjectInputStream objectStream =
      new ObjectInputStream(Files.newInputStream(Paths.get(filename)))) {
    g = (AccountGroup) objectStream.readObject();
  } catch (IOException e) {
    // . . .
  } catch (ClassNotFoundException e) {
    // . . .
  }
  return g;
}
```

```
AccountGroup group = loadAccount("group.dat");
System.out.println(group.getTotalBalance());
```

```
0
```

Transient Fields

```
public class AccountGroup implements Serializable {
  private Map<String, BankAccount> accountMap = new HashMap();
  private transient int totalBalance;
  public int getTotalBalance() { return totalBalance; }
  public void addAccount(BankAccount account) {
    totalBalance += account.getBalance();
    accountMap.put(account.getId(), account);
  }
 void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException {
    in.defaultReadObject();
    for(BankAccount account : accountMap.values())
      totalBalance += account.getBalance();
  }
}
```

Deserializing with a Transient Field

```
1250
```

```
AccountGroup group = loadAccount(“group.dat”);
System.out.println(group.getTotalBalance());
```

Summary

Serialization provides object persistence

- Files, databases
- Between processes, across networks

Serializable types

- Primitive types implicitly serializable
- Classes must implement Serializable
  - No methods to implement

Types that perform serializing/deserializing

- ObjectOutputStream
- ObjectInputStream

Summary

Serial version unique identifier

- Used to determine version compatibility
- Java calculates by default
  - Changes to type changes value
  - Breaks compatibility
- Can explicitly set
  - Add serialVersionUID field
  - Calculate initial value with serialver utility
  - Value maintained across versions
  - Gives developer control

Summary

Can customize serialization processes

- writeObject
  - Called to serialize object
  - Receives ObjectOutputStream
- readObject
  - Called to deserialize object
  - Receives ObjectInputStream
- Use transient to exclude fields
  - Useful when value can be derived
  - Can manually set during deserialization

Where to Go from Here

Java Skill Path

http://bit.ly/psjavaskill

Where to Go From Here

Java Collections

http://bit.ly/psjavacollections

Applying Concurrency and Multi-threading to Common Java Patterns

http://bit.ly/psjavamultithreading