

Persisting Objects with Serialization



Jim Wilson

Mobile Solutions Developer & Architect

@hedgehogjim | jwhh.com

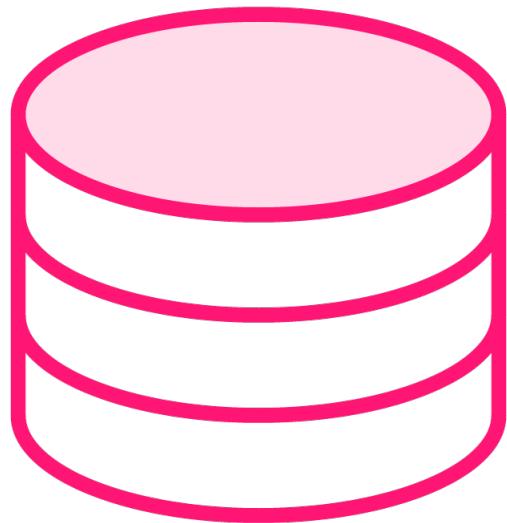
Overview



- 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

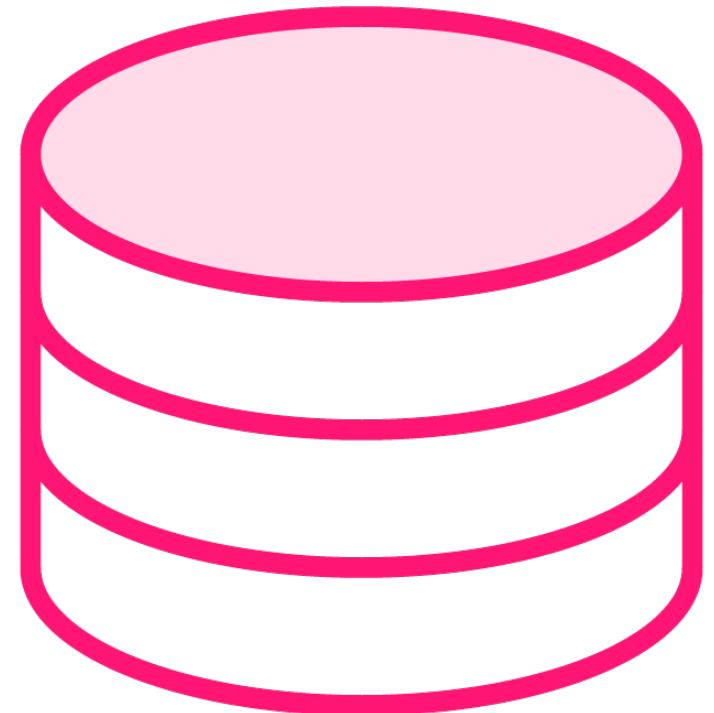


**Most cases require
very little programming**

Leverages reflection
Operates only on instance members
Customizable



Persisting Java Objects



Opens up many possibilities

Save to file system

Save in a database

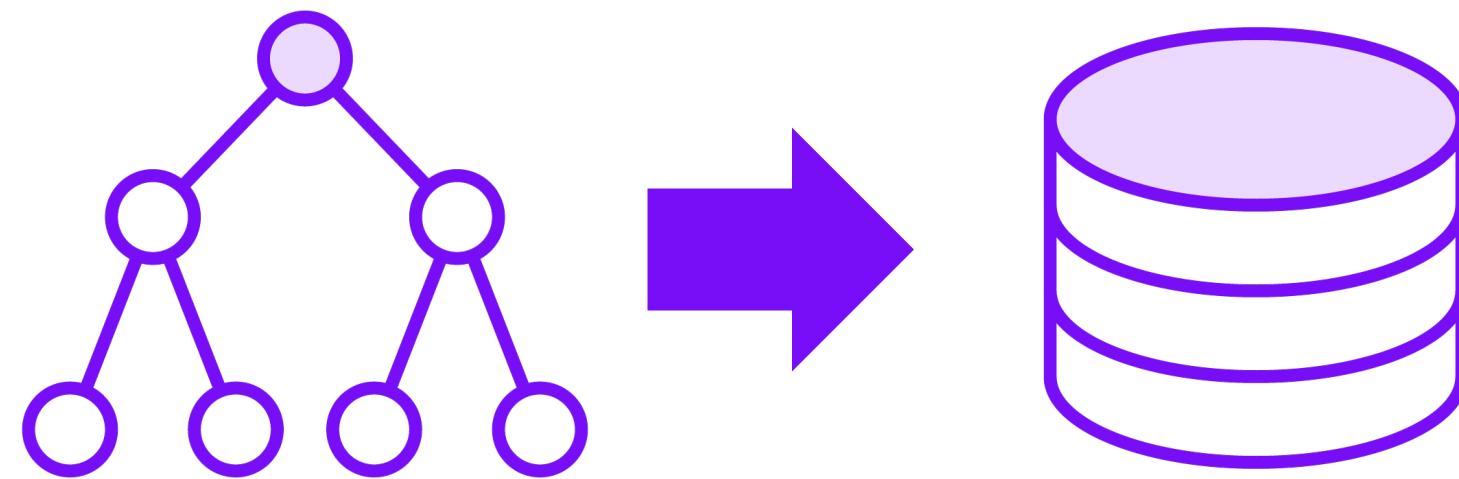
- RDBMS can store as blob
- OODBMS can often store directly

Pass across memory address boundaries

Pass over network

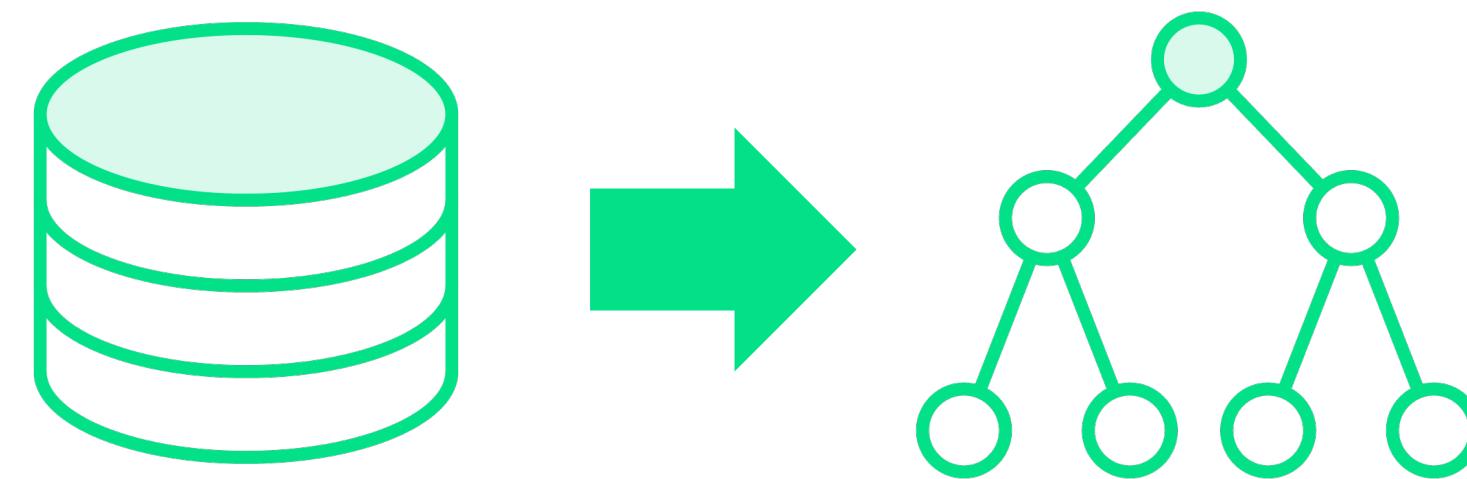


Serialization



Serializing

Storing an object-graph to a byte stream

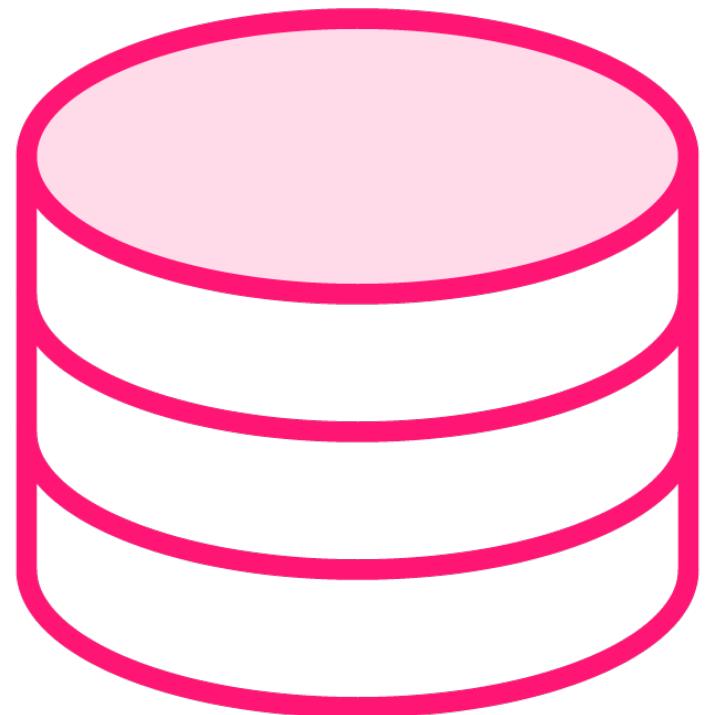


Deserializing

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



Requirements to be serializable



Implement Serializable

Members are serializable

- Primitive types are serializable
- Others must implement Serializable



Being Serializable

```
public class BankAccount {  
    private final String id;  
    private  
    public B  
    public B  
    // other  
}
```

The screenshot shows a Java API documentation page for the `String` class. The top navigation bar includes links for OVERVIEW, PACKAGE, CLASS (which is highlighted in orange), USE, TREE, DEPRECATED, INDEX, and HELP. The right side of the header displays "Java™ Platform Standard Ed. 8". Below the header, there are links for PREV CLASS, NEXT CLASS, FRAMES, NO FRAMES, and ALL CLASSES. The SUMMARY section lists NESTED, FIELD, CONSTR, and METHOD. The DETAIL section lists FIELD, CONSTR, and METHOD. The main content area shows the `String` class definition, its inheritance from `Object` and `String`, and its implementation of the `Serializable` interface. A red box highlights the `Serializable` interface in the `All Implemented Interfaces:` section.

new

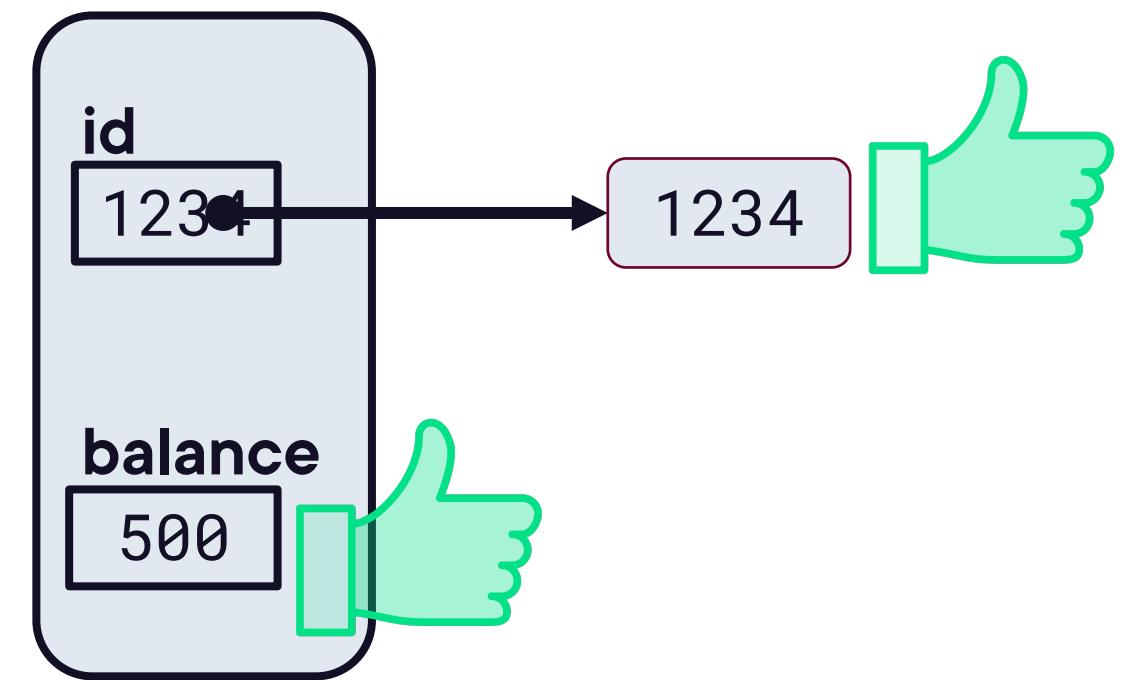
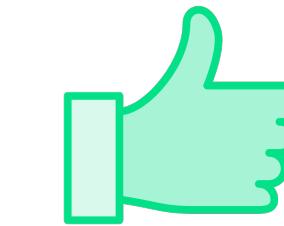
500



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) {  
    ObjectOutputStream objectStream =  
        new ObjectOutputStream(  
            new FileOutputStream(filename));  
    objectStream.writeObject(ba);  
}  
// . . .  
}
```



Deserializing an Object

```
BankAccount loadAccount(String filename) {  
    BankAccount ba = null;  
    ObjectInputStream objectStream =  
        new ObjectInputStream(  
            objectStream.readObject());  
    } catch (IOException e) {  
        // . . .  
    }  
    // . . .  
    return ba;  
}
```

```
loadAccount("account.dat");  
System.out.println(acct.getId() + " : " + acct.getBalance());
```

1234 : 750



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");
```

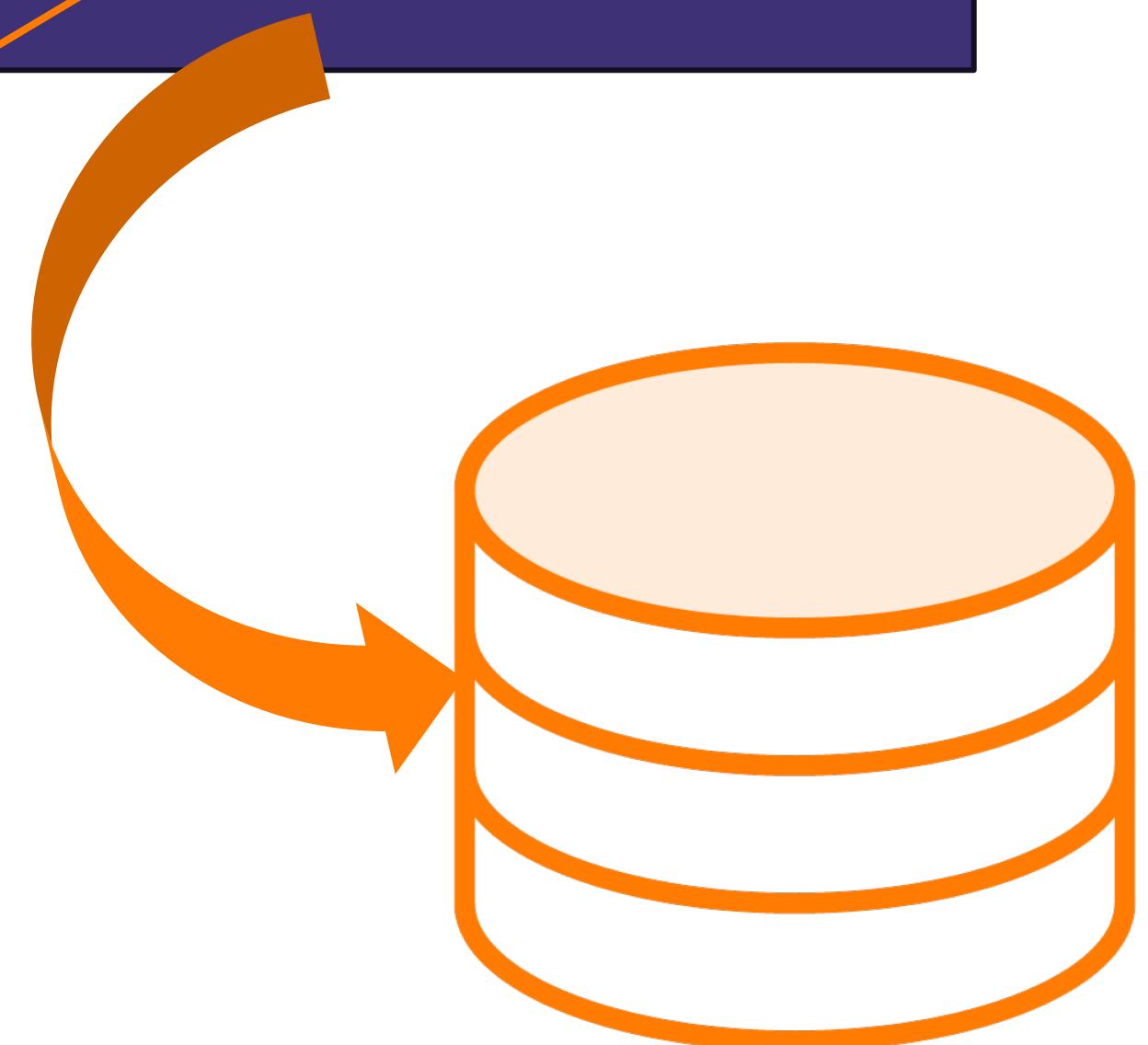


Class Version Incompatibility

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

-6328564483941980673

Serial version
unique identifier

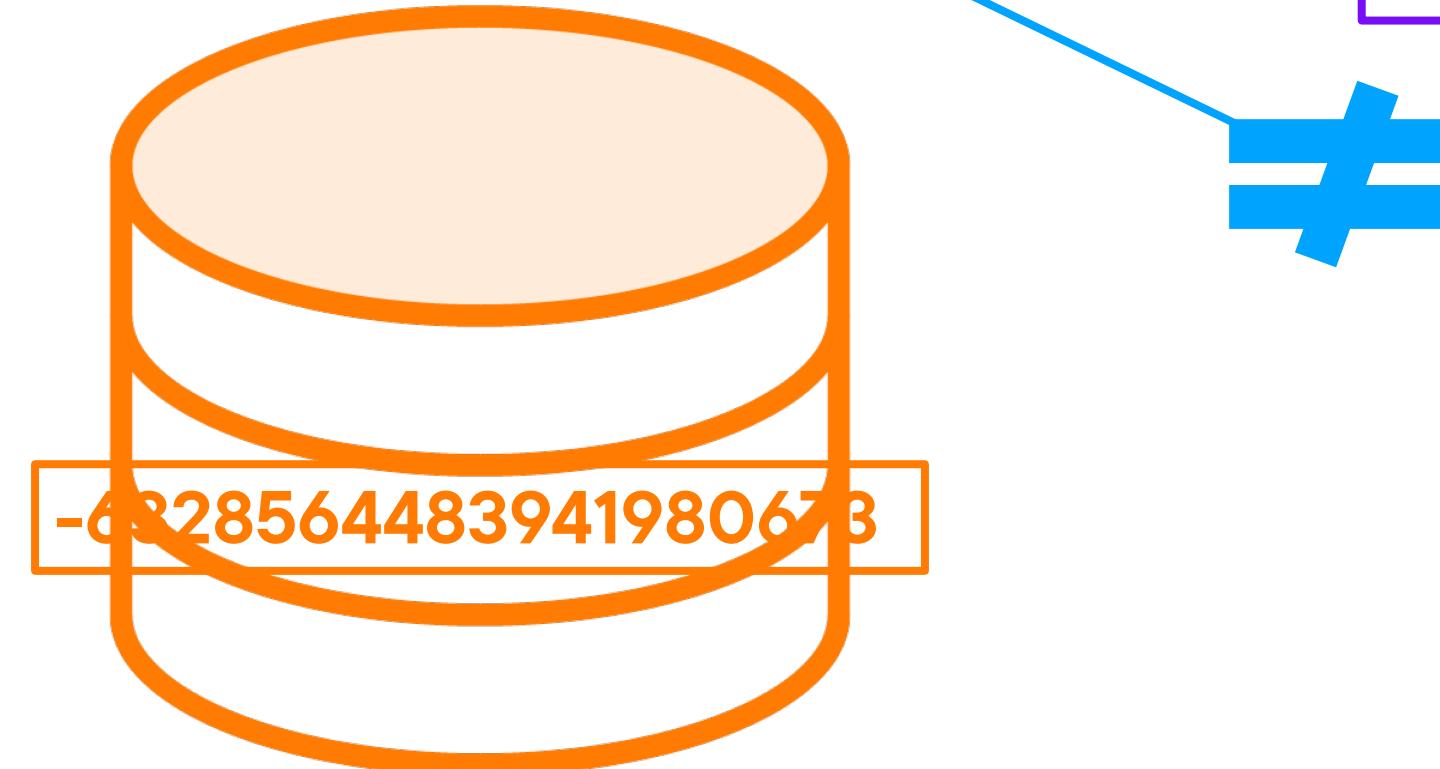


Class Version Incompatibility

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

InvalidClassException

-5887805305264759995



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

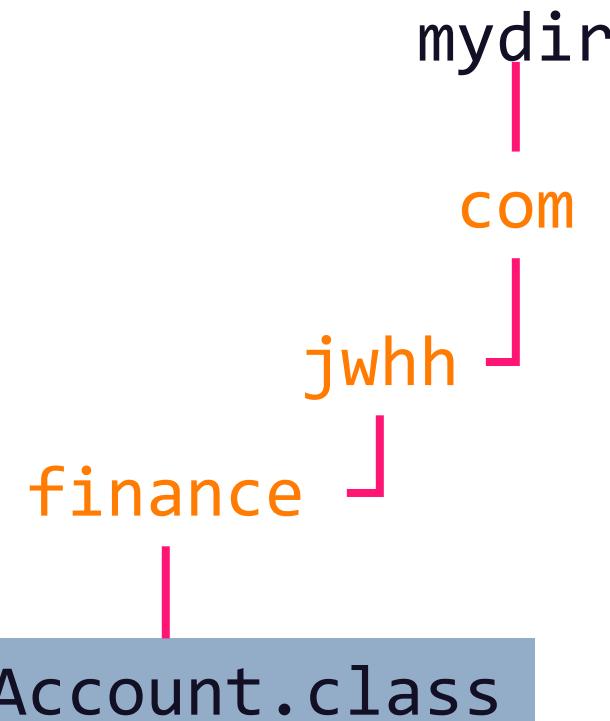
Using serialver utility

- 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



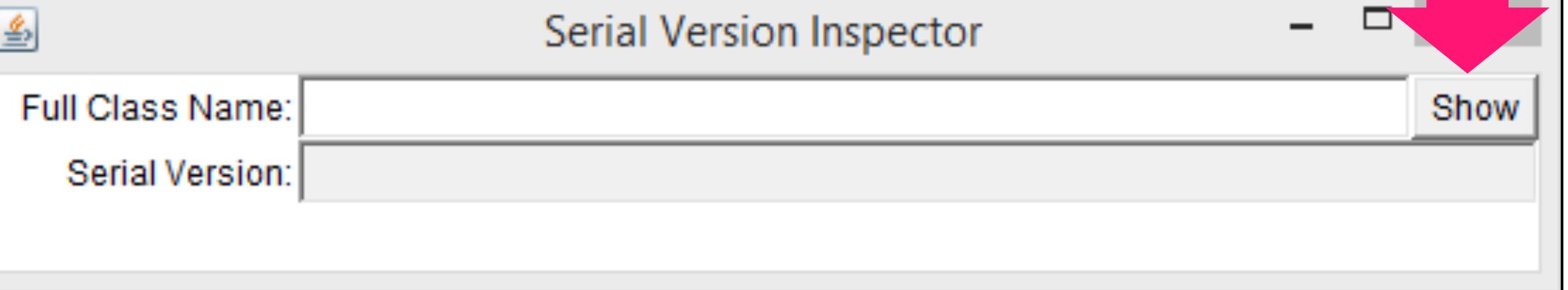
Creating Class Version Compatibility

```
package com.jwhh.finance;  
public class BankAccount implements Serializable {  
    private final String id;  
    private int balance = 0;  
    // other members elided  
}
```



C:\mydir>

C:\mydir>



Creating Class Version Compatibility

```
public class BankAccount implements Serializable {  
    serialVersionUID -6328564483941980673  
  
    private final String id;  
    private int balance = 0;  
    // other members elided  
}
```

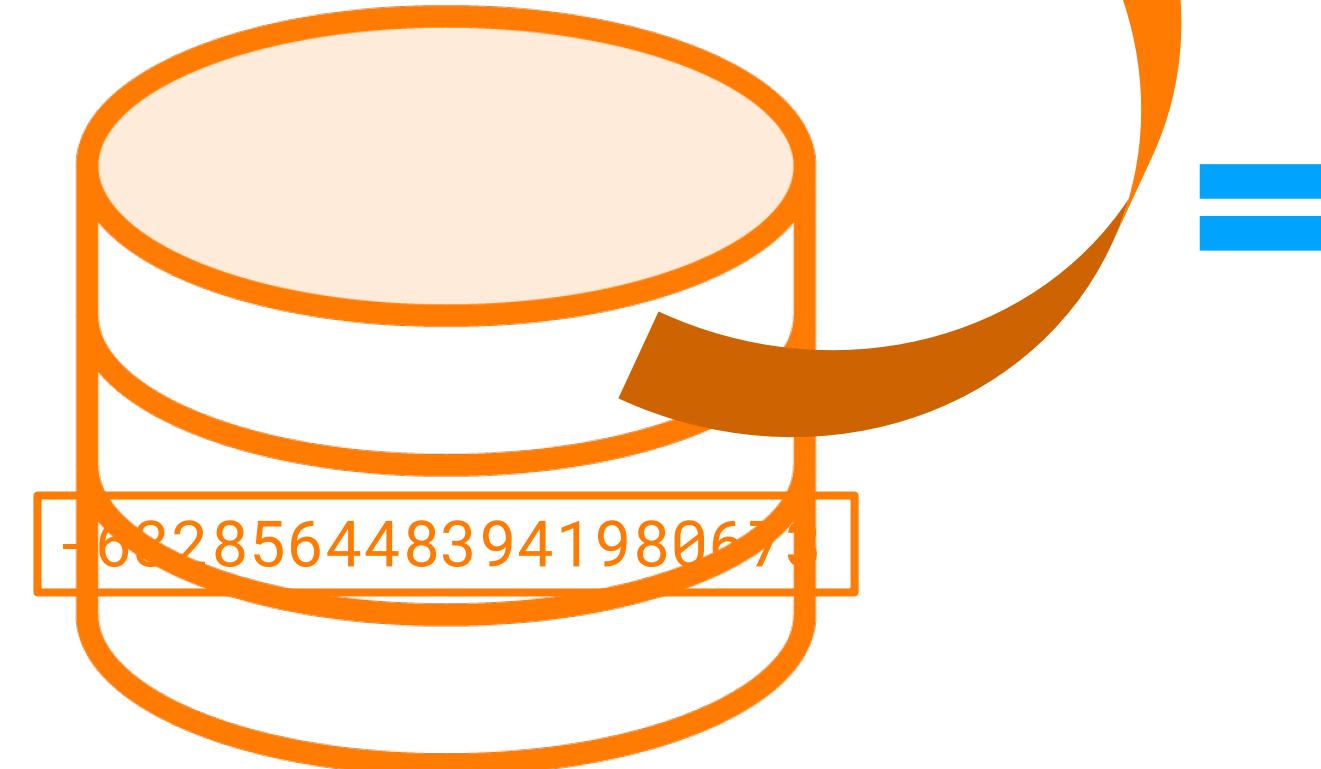
From serialver
utility



Creating Class Version Compatibility

```
public class BankAccount implements Serializable {  
    private static final long serialVersionUID = -6328564483941980673;  
    private final String id;  
    private int balance = 0;  
    private char lastTxType;  
    private int lastTxAmount;  
    // other members elided  
}
```

Default value for an
uninitialized member



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 acct = new BankAccount("1234", 500);  
acct.deposit(250);  
saveAccount(acct, "account.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;  
    // constructors & getters elided  
}
```

lastTxType = '\0'
lastTxAmount = 0

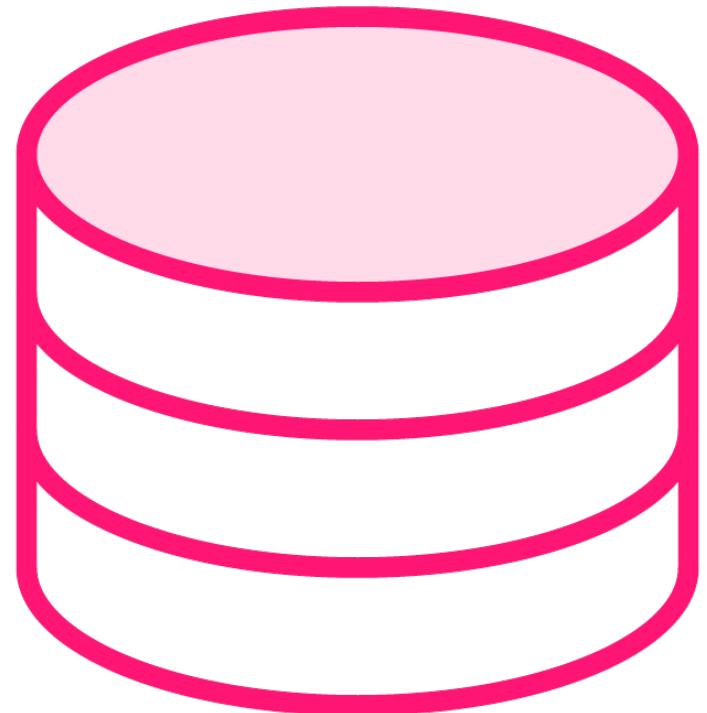
BankAccount acct2 = loadAccount("account2.dat");

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

lastTxType = '\0'
lastTxAmount = 0



Can Add Custom Serialization Handling



To handle serializing

- Add `writeObject` method to type

To handle deserializing

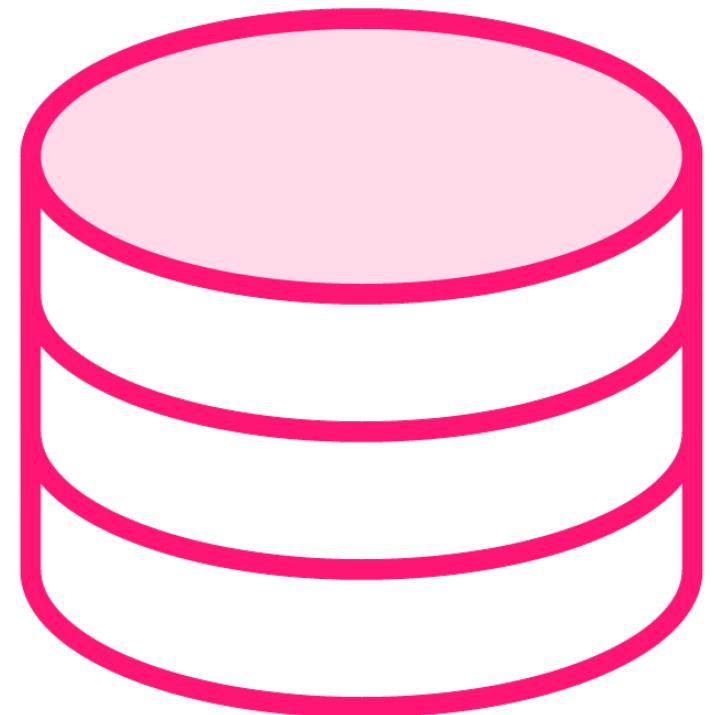
- Add `readObject` method to type

Methods called through reflection

- Methods normally marked private



Implementing writeObject Method



Return type of void

Include throws clause

- IOException

Accepts ObjectOutputStream

- Use to write values
- defaultWriteObject for default behavior

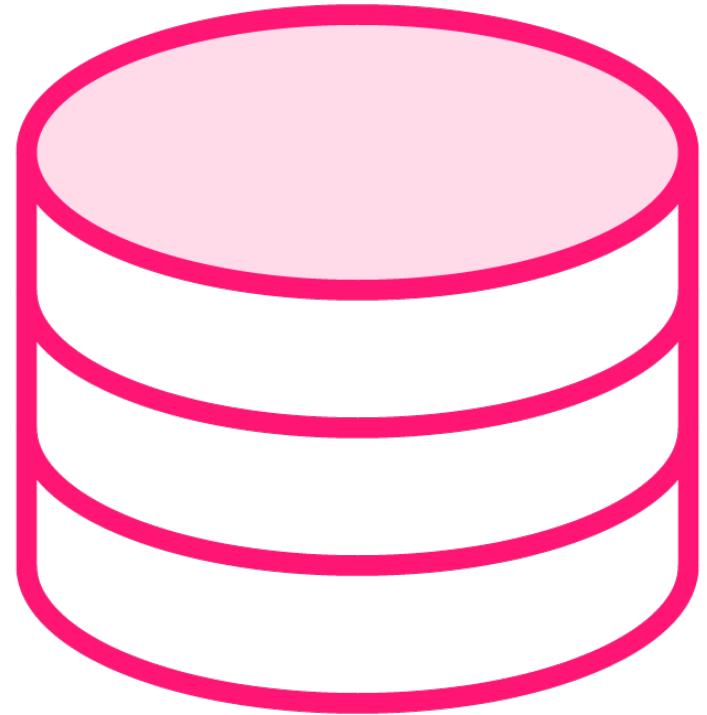


Customizing Serialization

```
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( )  
        out.defaultWriteObject();  
}  
}  
}
```



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( )  
        in.readFields();  
        fields.get( );  
        fields.get("balance", 0);  
        fields.get( );  
        fields.get( );  
    }  
}
```



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) { ... }  
    // constructors & getters elided  
}
```

lastTxType = 'u'
lastTxAmount = -1

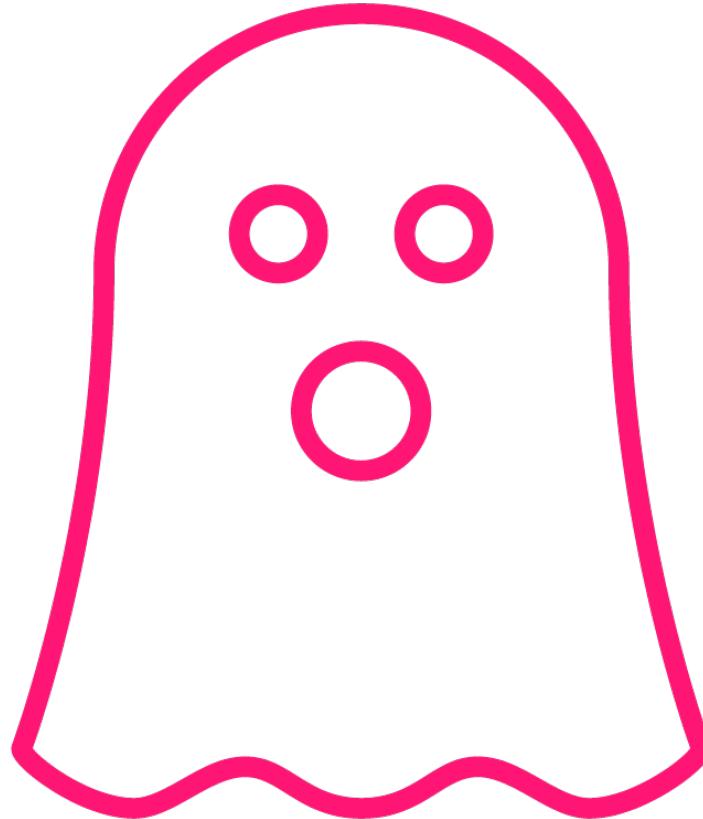
BankAccount acct1 = loadAccount("account1.dat");

lastTxType = '\0'
lastTxAmount = 0

BankAccount acct2 = loadAccount("account2.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 {  
    private Map<String, BankAccount> accountMap = new HashMap<>();  
  
    private int totalBalance;  
  
    public int getTotalBalance() { return totalBalance; }  
  
    public void addAccount(BankAccount account) {  
        totalBalance += account.getBalance();  
        accountMap.put(account.getId(), account);  
    }  
}
```



Transient Fields

```
public class AccountGroup implements Serializable {  
    private Map<String, BankAccount> accountMap = new HashMap<>();  
    private int totalBalance;  
    public int getTotalBalance() { return totalBalance; }  
    public void addAccount(BankAccount account) {  
        totalBalance += account.getBalance();  
        accountMap.put(account.getId(), account);  
    }  
}
```



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 = loadGroup("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

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

1250



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 w/ 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>

