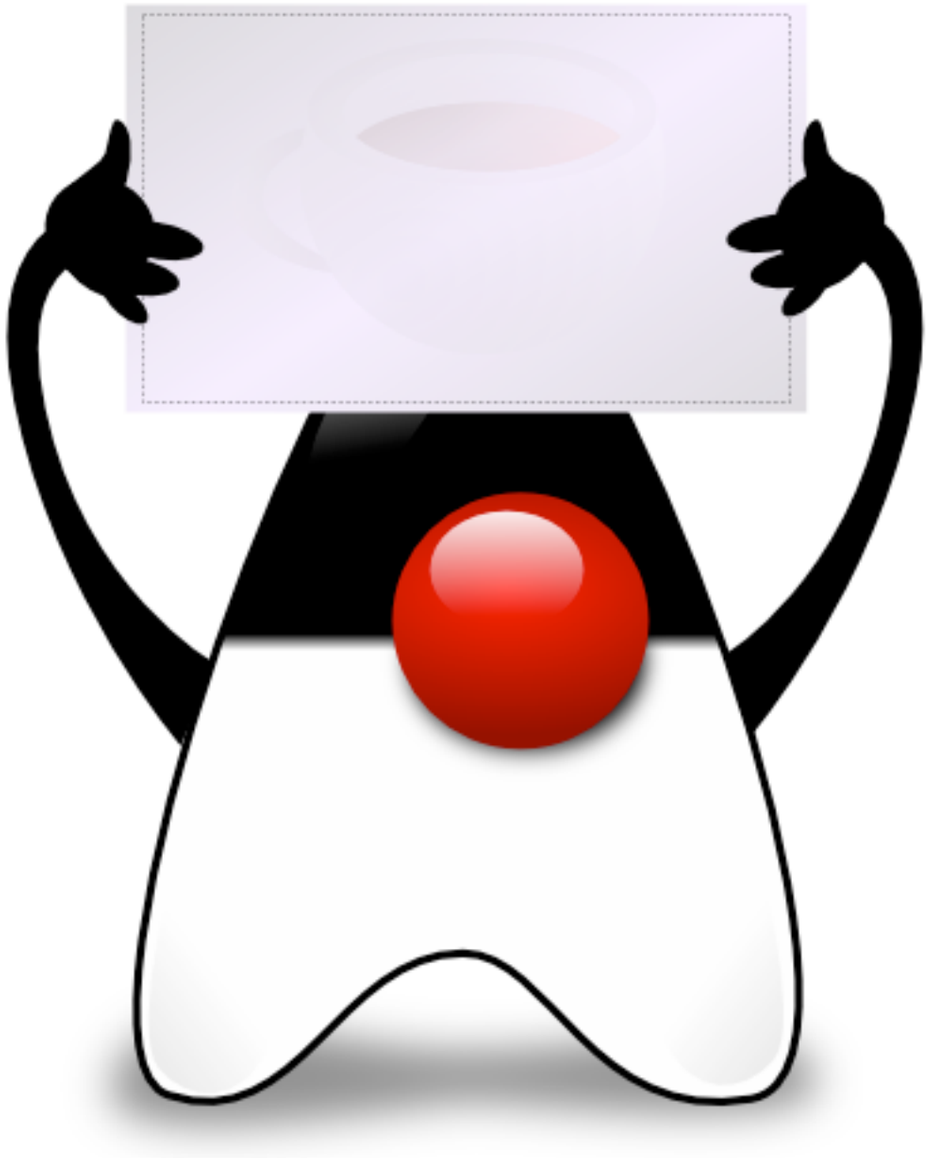
Lecture 6: Transactions in practice

Olivier Liechti
AMT

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

heig-vd

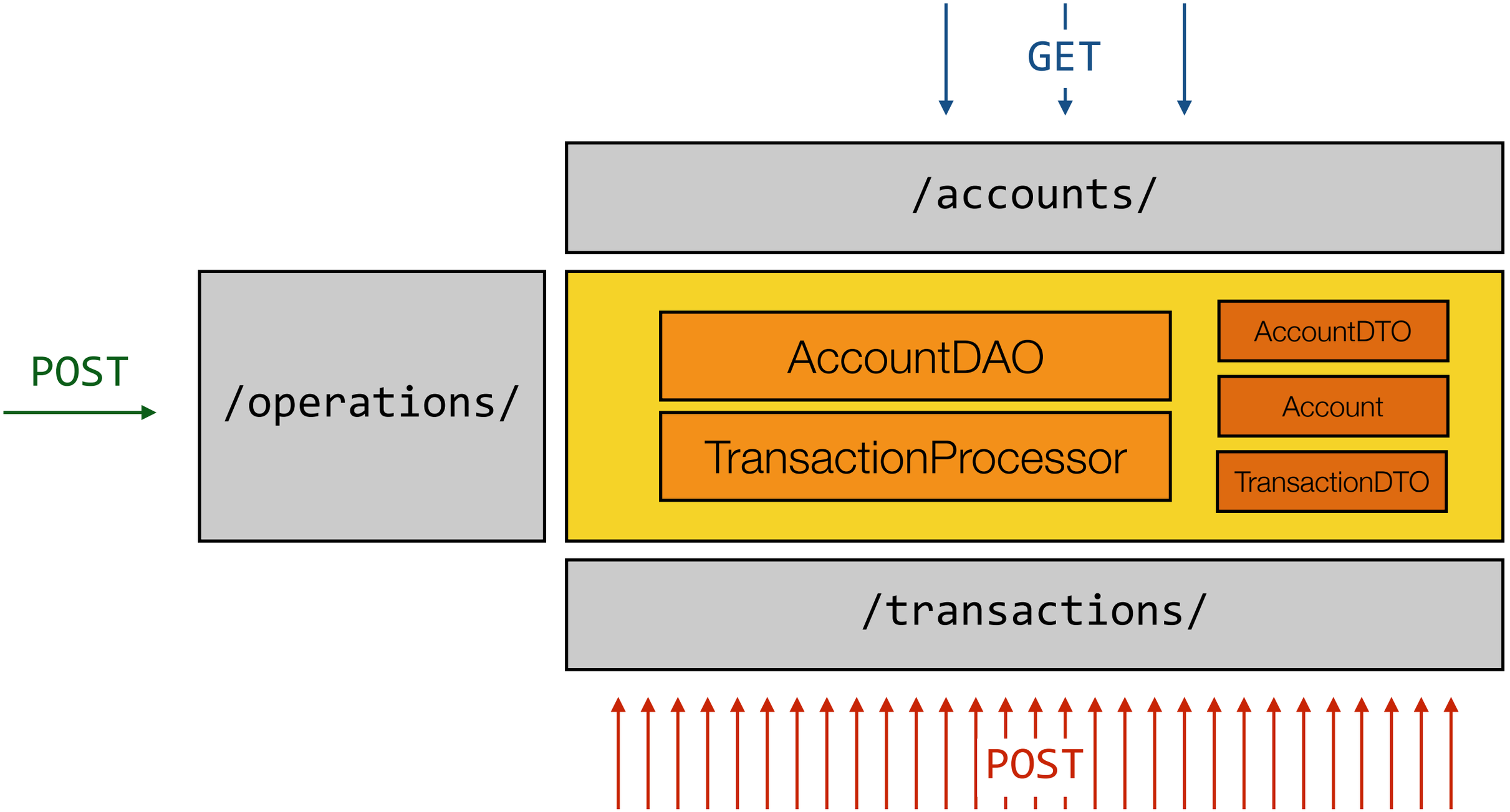
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



Transactions

System overview: REST APIs

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



System overview: REST APIs

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Account

id : long
balance : double
numberOfTransactions : long
holderName : String

Transaction

accountId : long
amount : double

Concurrent creation & unique constraints

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- In the system, we do not want to create accounts in advance.
- Instead, **we want to create them "on the fly"**: when we process a financial transaction, we check if the related account already exists:
  - If **no**, we create and initialize it.
  - If **yes**, we update it.
- **Let's try to implement this behavior!**

Concurrent creation & unique constraints

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
$ git clone git@github.com:SoftEng-HEIGVD/Teaching-HEIGVD-AMT-ConcurrentTransactions.git
```

```
$ git checkout step1-validating-on-the-fly-account-creation
```

Configure your **JDBC data source** (see `persistence.xml`): `jdbc/AMTDatabase`

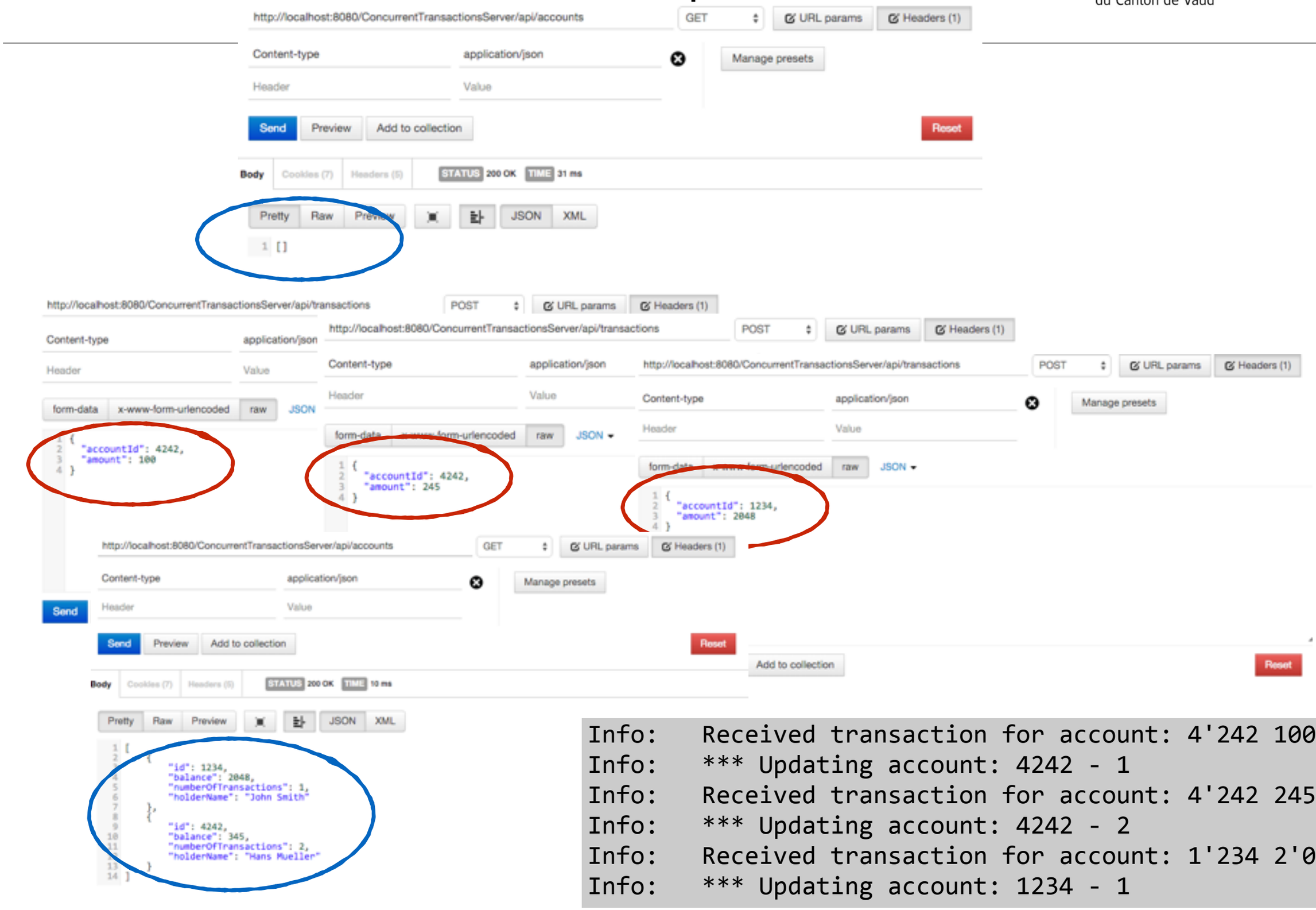
After deploying the application, POST a number of transactions on /api/transactions. Then validate with GET /accounts/.
Check the Glassfish logs.
Looks good!

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Take 15’ to build, deploy and run on your machine

Concurrent creation & unique constraints

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud




```
Info:    Received transaction for account: 4'242 100
Info:    *** Updating account: 4242 - 1
Info:    Received transaction for account: 4'242 245
Info:    *** Updating account: 4242 - 2
Info:    Received transaction for account: 1'234 2'048
Info:    *** Updating account: 1234 - 1
```

Concurrent creation & unique constraints

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Are we really **safe**?

```
$ git checkout step2-really-validating-on-the-fly-account-creation
```

You now have 2 test projects:

ConcurrentUpdateDemoClient (Java with JAX-RS client)
ConcurrentUpdateDemoClientNode (JavaScript)

Let's see what happens "out-of-the-box"

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Take 15’ to read the Java and the JavaScript test clients. You can play with them, too.

ConcurrentUpdateDemoClientNode

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
$ npm install
$ node client.js
```

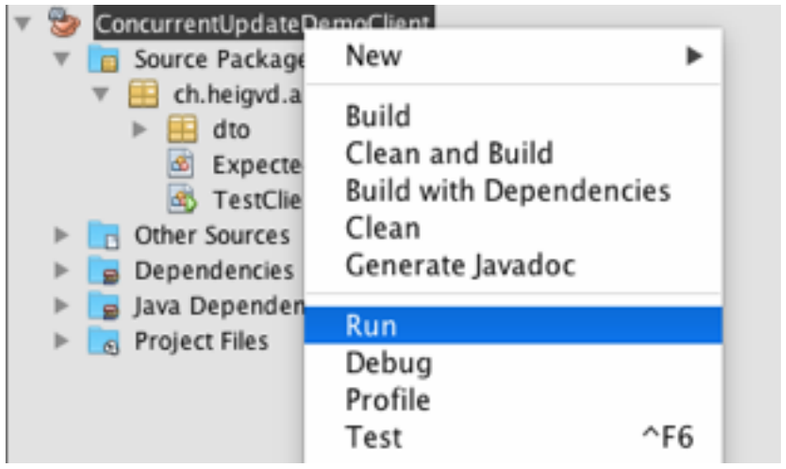
```
==========================================
Comparing client-side and server-side stats
------------------------------------------
Number of accounts on the client side: 10
Number of accounts on the server side: 10


==========================================
Summary
------------------------------------------
[ 'The RESET operation has been processed (status code: 204)',
  '200 transaction POSTs have been sent. 0 have failed.',
  'The client side and server side values have been compared. Number of corrupted accounts: 0' ]
```

```
Info:   Received transaction for account: 1 33
Info:   *** Updating account: 1 - 1
Info:   Received transaction for account: 1 74
Info:   *** Updating account: 1 - 2
Info:   Received transaction for account: 1 85
Info:   *** Updating account: 1 - 3
Info:   Received transaction for account: 1 1
Info:   *** Updating account: 1 - 4
Info:   Received transaction for account: 1 118
Info:   *** Updating account: 1 - 5
Info:   Received transaction for account: 1 -11
Info:   *** Updating account: 1 - 6
Info:   Received transaction for account: 1 61
Info:   *** Updating account: 1 - 7
Info:   Received transaction for account: 1 -3
Info:   *** Updating account: 1 - 8
Info:   Received transaction for account: 1 126
Info:   *** Updating account: 1 - 9
Info:   Received transaction for account: 1 -28
Info:   *** Updating account: 1 - 10
```

ConcurrentUpdateDemoClient

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud




```
10:50:54 INFO Expected vs actual number of transactions for account 18: 20/20
10:50:54 INFO Expected vs actual balance for account 18: 20/20
10:50:54 INFO Expected vs actual number of transactions for account 19: 20/20
10:50:54 INFO Expected vs actual balance for account 19: 20/20
10:50:54 INFO Expected vs actual number of transactions for account 20: 20/20
10:50:54 INFO Expected vs actual balance for account 20: 20/20
10:50:54 INFO Errors: []
10:50:54 INFO Done.
```

```
Info:   *** Updating account: 20 - 1
Info:   Received transaction for account: 20 1
Info:   *** Updating account: 20 - 2
Info:   Received transaction for account: 20 1
Info:   *** Updating account: 20 - 3
Info:   Received transaction for account: 20 1
Info:   *** Updating account: 20 - 4
Info:   Received transaction for account: 20 1
Info:   *** Updating account: 20 - 5
Info:   Received transaction for account: 20 1
Info:   *** Updating account: 20 - 6
Info:   Received transaction for account: 20 1
Info:   *** Updating account: 20 - 7
```

Concurrent creation & unique constraints

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Still looks good... Are we really **safe**?

Change the experiment parameters, so that we have concurrent requests!

There are parameters in the Java and the JavaScript test client

ConcurrentUpdateDemoClientNode

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
$ node client.js
```

```
Result 162: 500
Result 181: 500


==========================================
Comparing client-side and server-side stats
------------------------------------------
Number of accounts on the client side: 10
Number of accounts on the server side: 10
Account 1 --> Server/Client balance: 276/908  X
Account 2 --> Server/Client balance: 573/1161  X
Account 3 --> Server/Client balance: 478/1007  X
Account 4 --> Server/Client balance: 280/532  X
Account 5 --> Server/Client balance: 612/923  X
Account 6 --> Server/Client balance: 192/942  X
Account 7 --> Server/Client balance: 342/722  X
Account 8 --> Server/Client balance: 555/800  X
Account 9 --> Server/Client balance: 354/1107  X
Account 10 --> Server/Client balance: 407/1264  X
```

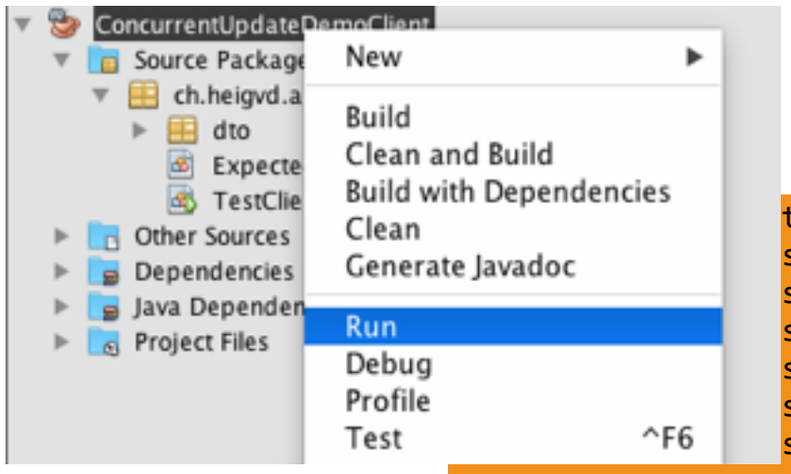
Some POST requests fail (the client is aware of a problem)

Worse: money has vanished without anyone being aware of it!

```
Caused by: javax.persistence.PersistenceException: Exception [EclipseLink-4002] (Eclipse Persistence Services - 2.5.2.v20140319-9ad6abd):
org.eclipse.persistence.exceptions.DatabaseException
Internal Exception: com.mysql.jdbc.exceptions.jdbc4.MySQLIntegrityConstraintViolationException: Duplicate entry '3' for key 'PRIMARY'
Error Code: 1062
Call: INSERT INTO ACCOUNT (ID, BALANCE, HOLDERNAME, NUMBEROFTRANSACTIONS) VALUES (?, ?, ?, ?)
        bind => [4 parameters bound]
Query: InsertObjectQuery(ch.heigvd.amt.demo.model.Account@7512b0e3)
        at org.eclipse.persistence.internal.jpa.EntityManagerImpl.flush(EntityManagerImpl.java:868)
        at com.sun.enterprise.container.common.impl.EntityManagerWrapper.flush(EntityManagerWrapper.java:437)
        at ch.heigvd.amt.demo.services.dao.AccountDAO.create(AccountDAO.java:26)
```

ConcurrentUpdateDemoClient

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud




```
tasks have been submitted to the executor and will be processed by 5 concurrent threads.
server was not able to process the transaction: 500 Internal Server Error
server was not able to process the transaction: 500 Internal Server Error
server was not able to process the transaction: 500 Internal Server Error
server was not able to process the transaction: 500 Internal Server Error
server was not able to process the transaction: 500 Internal Server Error
server was not able to process the transaction: 500 Internal Server Error

11:04:37 INFO Expected vs actual number of transactions for account 15: 20/5
11:04:37 INFO Expected vs actual balance for account 15: 20/5
11:04:37 INFO Expected vs actual number of transactions for account 16: 20/5
11:04:37 INFO Expected vs actual balance for account 16: 20/5
11:04:37 INFO Expected vs actual number of transactions for account 17: 20/5
11:04:37 INFO Expected vs actual balance for account 17: 20/5
11:04:37 INFO Expected vs actual number of transactions for account 18: 20/5
11:04:37 INFO Expected vs actual balance for account 18: 20/5
11:04:37 INFO Expected vs actual number of transactions for account 19: 20/5
11:04:37 INFO Expected vs actual balance for account 19: 20/5
11:04:37 INFO Expected vs actual number of transactions for account 20: 20/6
11:04:37 INFO Expected vs actual balance for account 20: 20/6
11:04:37 INFO Errors: [The number of transactions for account 1 is not the one expected: 7 vs 17, The balance
for account 1 is not the one expected: 7.0 vs 17.0
```

```
Caused by: javax.persistence.PersistenceException: Exception [EclipseLink-4002] (Eclipse Persistence Services - 2.5.2.v20140319-9ad6abd):
org.eclipse.persistence.exceptions.DatabaseException
Internal Exception: com.mysql.jdbc.exceptions.jdbc4.MySQLIntegrityConstraintViolationException: Duplicate entry '1' for key 'PRIMARY'
Error Code: 1062
Call: INSERT INTO ACCOUNT (ID, BALANCE, HOLDERNAME, NUMBEROFTRANSACTIONS) VALUES (?, ?, ?, ?)
        bind => [4 parameters bound]
Query: InsertObjectQuery(ch.heigvd.amt.demo.model.Account@178ebe08)
        at org.eclipse.persistence.internal.jpa.EntityManagerImpl.flush(EntityManagerImpl.java:868)
        at com.sun.enterprise.container.common.impl.EntityManagerWrapper.flush(EntityManagerWrapper.java:437)
        at ch.heigvd.amt.demo.services.dao.AccountDAO.create(AccountDAO.java:26)
```

What is the **account creation** problem?

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
@Override
public void createAccountIfNotExists(long id) {
  Account account = accountDAO.findById(id);
  if (account == null) {
    account = new Account();
    account.setId(id);
    account.setBalance(0);
    account.setNumberOfTransactions(0);
    account.setHolderName(generateRandomHolderName());
    accountDAO.create(account);
  }
}
```

Thread T1 on EJB 1

```
Account account = accountDAO.findById(id);
 if (account == null) {
   account = new Account();
   account.setId(id);
   account.setBalance(0);
   account.setNumberOfTransactions(0);
   account.setHolderName(generateRandomHolderName());
   accountDAO.create(account);
  }
}
```

Thread T2 on EJB2

```
Account account = accountDAO.findById(id);
  if (account == null) {
    account = new Account();
    account.setId(id);
    account.setBalance(0);
    account.setNumberOfTransactions(0);
    account.setHolderName(generateRandomHolderName());
    accountDAO.create(account);
  }
}
```

Fixing the problem: approach 1 (new tx)

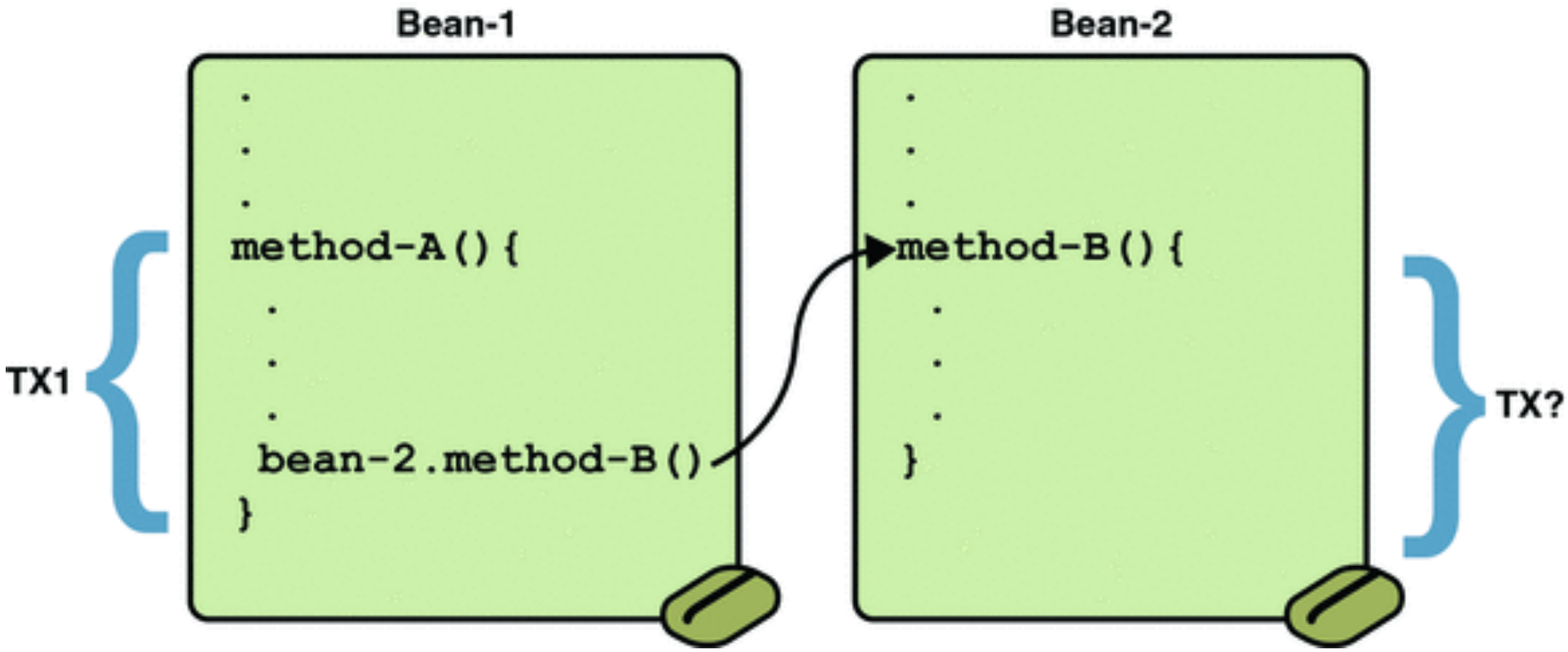
heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- Last week, we have seen that it is possible:
  - to divide one “use case” into multiple sub-transactions
  - to decide whether all sub-transactions should be rolled back or only some of them in the case of errors
- We have seen that there is a special annotation (@TransactionAttribute) for specifying the behavior (by default, the container rolls back everything).

```
$ git checkout step3-fix-account-creation-with-try-catch
```

Transaction Scope

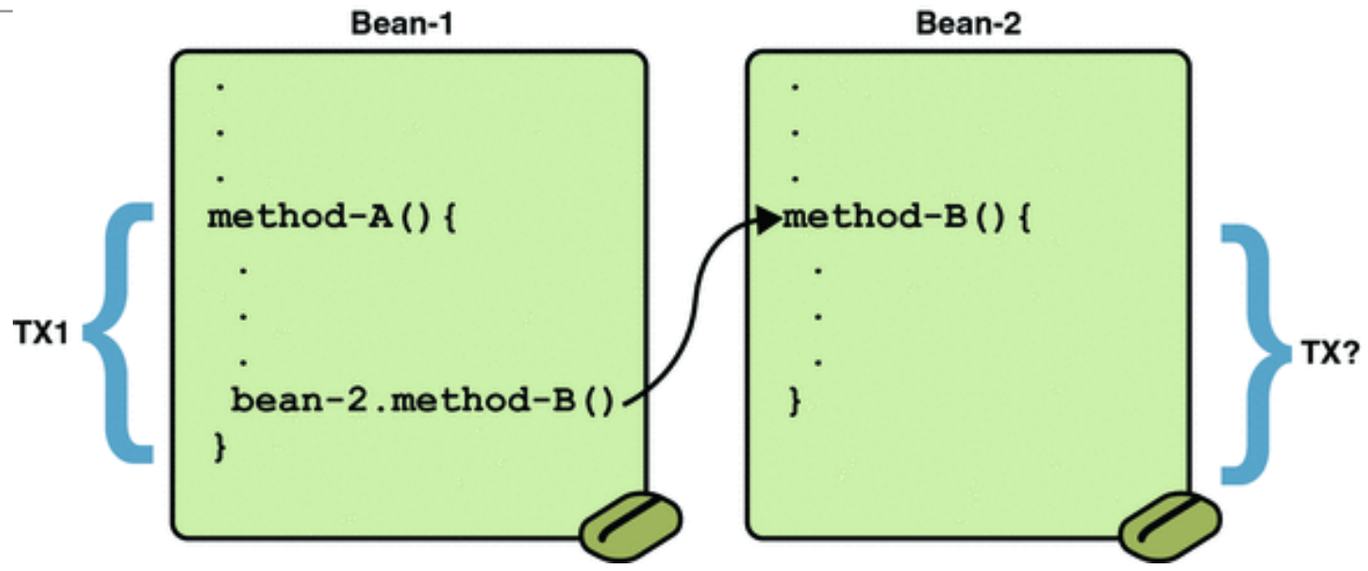
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



http://java.sun.com/javaee/5/docs/tutorial/doc/bncij.html

Transaction Scope

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud




```
@TransactionAttribute(NOT_SUPPORTED)
@Stateless
public class TransactionBean implements
Transaction {
...
    @TransactionAttribute(REQUIRES_NEW)
    public void firstMethod() {...}

    @TransactionAttribute(REQUIRED)
    public void secondMethod() {...}

    public void thirdMethod() {...}

    public void fourthMethod() {...}
}
```

Transaction Attribute	Client's Transaction	Business Method's Transaction
Required	None	T2
	T1	T1
RequiresNew	None	T2
	T1	T2
Mandatory	None	error
	T1	T1
NotSupported	None	None
	T1	None
Supports	None	None
	T1	T1
Never	None	None
	T1	Error

Fixing the problem: approach 1 (new tx)

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

If we want to capture failed transactions, we need to go via the container

If an exception occurs in this block, we don't want to rollback everything!

```
@Stateless
public class TransactionProcessor implements TransactionProcessorLocal {

  private static final Logger LOG = Logger.getLogger(TransactionProcessor.class.getName());

  @EJB
  AccountDAOLocal accountDAO;

  @EJB
  TransactionProcessorLocal selfViaContainer;

  @Override
  public void processTransaction(TransactionDTO transaction) {
    try {
      selfViaContainer.createAccountIfNotExists(transaction.getAccountId());
    } catch (Exception e) {
      LOG.info("*** Maybe a DUPLICATE KEY that would not be a real problem..." + e.getMessage());
    }
...
  }

  @TransactionAttribute(TransactionAttributeType.REQUIRES_NEW)
  public void createAccountIfNotExists(long id) {
    Account account = accountDAO.findById(id);
    if (account == null) {
      account = new Account();
      account.setId(id);
      account.setBalance(0);
      account.setNumberOfTransactions(0);
      account.setHolderName(generateRandomHolderName());
      accountDAO.create(account);
    }
  }
```

ConcurrentUpdateDemoClientNode

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
$ node client.js
```

```
==========================================
Summary
------------------------------------------
[ 'The RESET operation has been processed (status code: 204)',
  '200 transaction POSTs have been sent. 0 have failed.',
  'The client side and server side values have been compared. Number of corrupted accounts: 10' ]
```

We have resolved one issue: the client does not receive any error when the first two financial transactions for one account are sent simultaneously.

However, we still have a problem with data corruption (unrelated to account creation).

We also have ugly stack traces in our logs and assuming that the exception thrown during account creation is harmless is not very robust...

```
Caused by: javax.persistence.PersistenceException: Exception [EclipseLink-4002] (Eclipse Persistence Services - 2.5.2.v20140319-9ad6abd):
org.eclipse.persistence.exceptions.DatabaseException
Internal Exception: com.mysql.jdbc.exceptions.jdbc4.MySQLIntegrityConstraintViolationException: Duplicate entry '3' for key 'PRIMARY'
Error Code: 1062
Call: INSERT INTO ACCOUNT (ID, BALANCE, HOLDERNAME, NUMBEROFTRANSACTIONS) VALUES (?, ?, ?, ?)
        bind => [4 parameters bound]
Query: InsertObjectQuery(ch.heigvd.amt.demo.model.Account@7512b0e3)
        at org.eclipse.persistence.internal.jpa.EntityManagerImpl.flush(EntityManagerImpl.java:868)
        at com.sun.enterprise.container.common.impl.EntityManagerWrapper.flush(EntityManagerWrapper.java:437)
        at ch.heigvd.amt.demo.services.dao.AccountDAO.create(AccountDAO.java:26)
```

Fixing the problem: approach 2 (upsert)

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- Many databases support a special type of operation, often called an “upsert”
- With this operation, you can specify that when you can update a record if it already exists in the database, or create it if does not exist yet.
- MySQL supports this feature with the **INSERT ... ON DUPLICATE KEY UPDATE syntax**

```
$ git checkout step4-fix-account-creation-with-upsert
```

Fixing the problem: approach 2 (upsert)

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
@Entity
@NamedQueries({
    @NamedQuery(name="Account.findAll", query="SELECT a FROM Account a"),
    @NamedQuery(name="Account.deleteAll", query="DELETE FROM Account")
})
@NamedNativeQuery(name = "Account.upsert", query = "INSERT INTO Account (ID, HOLDERNAME, BALANCE, NUMBEROFTRANSACTIONS) VALUES
(?1, ?2, ?3, ?4) ON DUPLICATE KEY UPDATE BALANCE=BALANCE+?4, NUMBEROFTRANSACTIONS=NUMBEROFTRANSACTIONS+0")
public class Account { ... }
```

This is a **proprietary** feature provided by MySQL

```
 @Stateless
public class TransactionProcessor implements TransactionProcessorLocal {

  private static final Logger LOG = Logger.getLogger(TransactionProcessor.class.getName());

  @EJB
  AccountDAOLocal accountDAO;

 public void createAccountIfNotExists(long id) {
  @Override
  public void createAccountIfNotExists(long id) {
   Query query = em.createNamedQuery("Account.upsert");
    query.setParameter(1, id);
    query.setParameter(2, generateRandomHolderName());
    query.setParameter(3, 0);
    query.setParameter(4, 0);
    long result = query.executeUpdate();
  }

}
```

ConcurrentUpdateDemoClientNode

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
$ node client.js
```

```
==========================================
Comparing client-side and server-side stats
------------------------------------------
Number of accounts on the client side: 20
Number of accounts on the server side: 20


==========================================
Summary
------------------------------------------
[ 'The RESET operation has been processed (status code: 204)',
  '800 transaction POSTs have been sent. 0 have failed.',
  'The client side and server side values have been compared. Number of corrupted accounts: 0' ]
```

We don’t have any 500 response sent to the client (no problem with duplicate accounts)

As an additional benefit, we don’t have any data corruption! That is because the special MySQL requests locks the row in the database.

We have also got rid of the exceptions!

```
Info:   Received transaction for account: 20 104
Info:   *** Updating account: 20 - 22
Info:   Received transaction for account: 20 21
Info:   *** Updating account: 20 - 23
Info:   Received transaction for account: 20 46
Info:   *** Updating account: 20 - 24
Info:   *** Updating account: 20 - 25
Info:   Received transaction for account: 20 143
Info:   *** Updating account: 20 - 26
Info:   Received transaction for account: 20 74
Info:   *** Updating account: 20 - 27
Info:   Received transaction for account: 20 12
```

What is the **data corruption** problem?

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
public void processTransaction(TransactionDTO transaction) {
  try {
    selfViaContainer.createAccountIfNotExists(transaction.getAccountId());
  } catch (Exception e) {
    LOG.info("*** Maybe a DUPLICATE KEY that would not be a real problem..." + e.getMessage());
  }

  Account account = accountDAO.findById(transaction.getAccountId());
  double bal = account.getBalance();
  bal = bal + transaction.getAmount();
  account.setBalance(bal);
  account.setNumberOfTransactions(account.getNumberOfTransactions() + 1);
}
```

Thread T1 on EJB 1

```
Account account = accountDAO.findById(transaction.getAccountId());
double bal = account.getBalance();



bal = bal + transaction.getAmount();
account.setBalance(bal);
account.setNumberOfTransactions(account.getNumberOfTransactions() + 1);
```

Thread T2 on EJB2

```
Account account = accountDAO.findById(transaction.getAccountId());
double bal = account.getBalance();
bal = bal + transaction.getAmount();
account.setBalance(bal);
account.setNumberOfTransactions(account.getNumberOfTransactions() + 1);
```

Optimistic vs Pessimistic Locking

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- To fix this issue, we have the choice between a pessimistic and an optimistic locking strategy:
  - If we believe that there is a high probability to have a conflict (we are pessimistic), then we should lock the record before modifying it (the other transaction will have to wait that we release it).
  - If we believe that there is a little probability to have a conflict, then we can look at the version number of the record when we read it, check that it is still the same and increment it when we update the record.
  - If someone has modified the record in the meantime, then the version number will have changed and we will be aware of the issue.
- JPA provides support for both pessimistic and optimistic locking strategies.
- Pessimistic Locking has a performance cost (and may introduce deadlocks). Optimistic locking may require some extra work (dealing with exceptions).

Pessimistic locking solution

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
$ git checkout step5-fix-account-creation-with-try-catch-pessimistic-lock
```

```
@Stateless
public class AccountDAO implements AccountDAOLocal {

...
  @Override
  public Account findByIdForUpdate(long id) {
    return em.find(Account.class, id, LockModeType.PESSIMISTIC_WRITE);
  }
...
}
```

```
@Stateless
public class TransactionProcessor implements TransactionProcessorLocal {
  ...
  @Override
  public void processTransaction(TransactionDTO transaction) {
    ...
    Account account = accountDAO.findByIdForUpdate(transaction.getAccountId());
    ...
  }

}
```

Optimistic locking solution

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
$ git checkout step6-fix-account-creation-with-try-catch-optimistic-lock
```

```
@Entity
public class Account {

  @Id
  private long id;

  @Version
  private long version;
```