O'REILLY
OSCON
Open Source Convention

PORTLAND, OREGON | JULY 21-25, 2008

open YOUR mind

spring source

Simple JDBC with Spring 2.5

Thomas Risberg
SpringSource Inc.

Introduction

- Committer on the Spring Framework project since 2003, supporting the JDBC and Data Access code
- Co-author of “Professional Java Development with the Spring Framework” from Wrox
- Currently work for SpringSource working on Advanced Pack for Oracle Database and Spring Batch projects

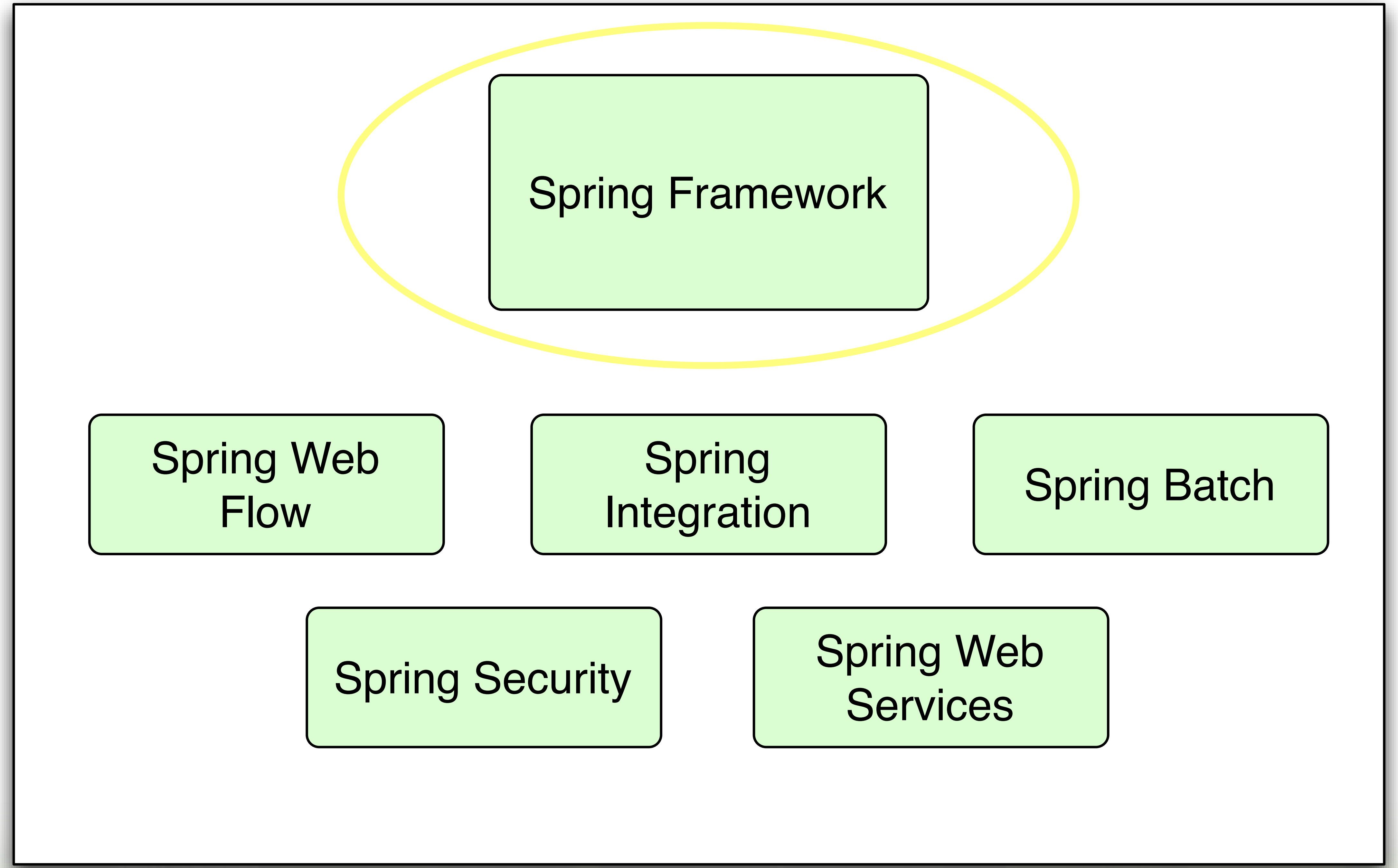
Agenda

- Spring overview
- JDBC development doesn’t have to be difficult
- When is JDBC more appropriate than an ORM solution
- Look at new Spring 2.5 JDBC features
- A few other 2.5 features that are handy for data access development and testing

Spring Overview

- Spring Framework project started early 2003 on SourceForge
- Based on code from *J2EE Design and Development* by Rod Johnson late 2002
- Widely adopted, most popular Java Enterprise framework
- Combined with Tomcat and Hibernate, Spring provides a popular alternative to Java EE

Spring Portfolio



Spring Framework

- Packages:
  - beans
  - context
  - dao
  - jdbc
  - orm
  - transaction
  - jms
  - web
  - webmvc
  - ...

Who does what?

Task	Spring	You
Connection Management	√	
SQL		√
Statement Management	√	
ResultSet Management	√	
Row Data Retrieval		√
Parameter Declaration		√
Parameter Setting	√	
Transaction Management	√	
Exception Handling	√	

Plain JDBC vs. Spring JDBC

JDBC	Spring
DriverManager / DataSource	DataSource
Statement / PreparedStatement / CallableStatement	JdbcTemplate / SimpleJdbcTemplate, SimpleJdbcCall, SimpleJdbcInsert MappingSqlQuery / StoredProcedure
ResultSet / RowSet	POJOs / List of POJOs or Maps / SqlRowSet

Yuck!!

```
public class JdbcDaoImpl {

    public int getCustomerCount() {
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        int count = 0;
        Properties properties = new Properties();
        try {
            properties.load(new FileInputStream("jdbc.properties"));
        } catch (IOException e) {
            throw new MyDataAccessException("I/O Error", e);
        }
        try {
            Class.forName(properties.getProperty("driverClassName"));
            conn = DriverManager.getConnection(properties.getProperty("url"), properties);
            stmt = conn.createStatement();
            rs = stmt.executeQuery("select count(*) from customers");
            if (rs.next()) {
                count = rs.getInt(1);
            }
        }
        catch (ClassNotFoundException e) {
            throw new MyDataAccessException("JDBC Error", e);
        }
        catch (SQLException se) {
            throw new MyDataAccessException("JDBC Error", se);
        }
        finally {
            if (rs != null) {
                try {
                    rs.close();
                }
                catch (SQLException ignore) {}
            }
            if (stmt != null) {
                try {
                    stmt.close();
                }
                catch (SQLException ignore) {}
            }
            if (conn != null) {
                try {
                    conn.close();
                }
                catch (SQLException ignore) {}
            }
        }
        return count;
    }
}
```

Using straight JDBC code

Much better!

```
@Repository
public class SpringDaoImpl {
    private JdbcTemplate jdbcTemplate;

    @Autowired
    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }

    public int getCustomerCount() {
        return jdbcTemplate.queryForInt("select count(*) from customers");
    }
}
```

```
<context:property-placeholder location="classpath:jdbc.properties"/>
<context:annotation-config/>
<context:component-scan base-package="com.springsource.data.sample"/>

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
      destroy-method="close">
    <property name="driverClassName" value="${db.driverClassName}"/>
    <property name="url" value="${db.url}"/>
    <property name="username" value="${db.username}"/>
    <property name="password" value="${db.password}"/>
</bean>
```

Exception Translation

SQLException is translated to a more expressive sub-class of DataAccessException like DataIntegrityViolationException or CannotAcquireLockException. This is translation is based on SQL Error codes forts and then SQL State codes. Translation is controlled by entries in sql-error-codes.xml.

sql-error-codes.xml

```
<bean id="MySQL" class="org.springframework.jdbc.support.SQLErrorCodes">
        <property name="badSqlGrammarCodes">
                <value>1054,1064,1146</value>
        </property>
        <property name="dataAccessResourceFailureCodes">
                <value>1</value>
        </property>
        <property name="dataIntegrityViolationCodes">
                <value>1062</value>
        </property>
        <property name="cannotAcquireLockCodes">
                <value>1205</value>
        </property>
        <property name="deadlockLoserCodes">
                <value>1213</value>
        </property>
</bean>
```

When should you use JDBC?

- Trivial domain model, few tables
- Complex queries reading and modifying multiple tables
- Stored procedures, db specific data types
- Bulk operations
- Tasks where you are not mapping relational data to an object graph

Mix JDBC and ORM

- It’s not an either or situation
- You can mix ORM access with JDBC
- Stored procedures are typically better supported with JDBC
- ORM and JDBC share transaction management - can participate in same transaction
- Remember to flush ORM session/entity manager before JDBC access

Simple Example

- Single Table (MySQL)

```
CREATE
   TABLE customer
   (
      id int(11) NOT NULL AUTO_INCREMENT,
      name varchar(50),
      customer_number varchar(50),
      birth_date date,
      PRIMARY KEY (id)
   )
   ENGINE= InnoDB
```

The Customer class

```
package com.springsource.data.sample.domain;

import java.util.Date;

public class Customer
 {
    private Long id;

    private String name;

    private String customerNumber;

    private Date birthDate;

    // getters and setters //

}
```

DAO/Repository Interface

```
package com.springsource.data.sample.repository;

import java.util.List;

import com.springsource.data.sample.domain.Customer;

public interface CustomerRepository {

         void add(Customer customer);

         Customer findById(Long id);

         void save(Customer customer);

         List<Customer> findAll();

}
```

Spring 2.5 and SimpleJdbc

- Spring 2.5 features for Simple JDBC:
  - SimpleJdbcTemplate (named parameter support)
  - SimpleJdbcInsert
  - SimpleJdbcCall
  - Annotation configuration
    - @Repository, @Service, @Component
    - @Autowired, @Qualifier
  - JUnit 4 support
    - @RunWith(SpringJUnit4ClassRunner.class)
    - @ContextConfiguration
    - @Transactional @Test

Simple Queries

```
Customer customer = simpleJdbcTemplate.queryForObject(
    "select id, name, customer_number, birth_date from customer where id = ?",
    ParameterizedBeanPropertyRowMapper.newInstance(Customer.class),
    id);
```

```
List<Customer> customerList = simpleJdbcTemplate.query(
    "select id, name, customer_number, birth_date from customer",
    ParameterizedBeanPropertyRowMapper.newInstance(Customer.class));
```

Simple Updates

named parameters

```
simpleJdbcTemplate.update(
    "update customer set name = :name, birth_date = :birth_date where id = :id",
        new BeanPropertySqlParameterSource(customer));
```

source of values to match parameters

Passing in parameter values

- java.util.Map
  - simple but doesn’t provide type info
- MapSqlParameterSource
  - provides option of passing in SQL type

```
values.addValue("birth_date", customer.getBirthdate(), Types.DATE)
```

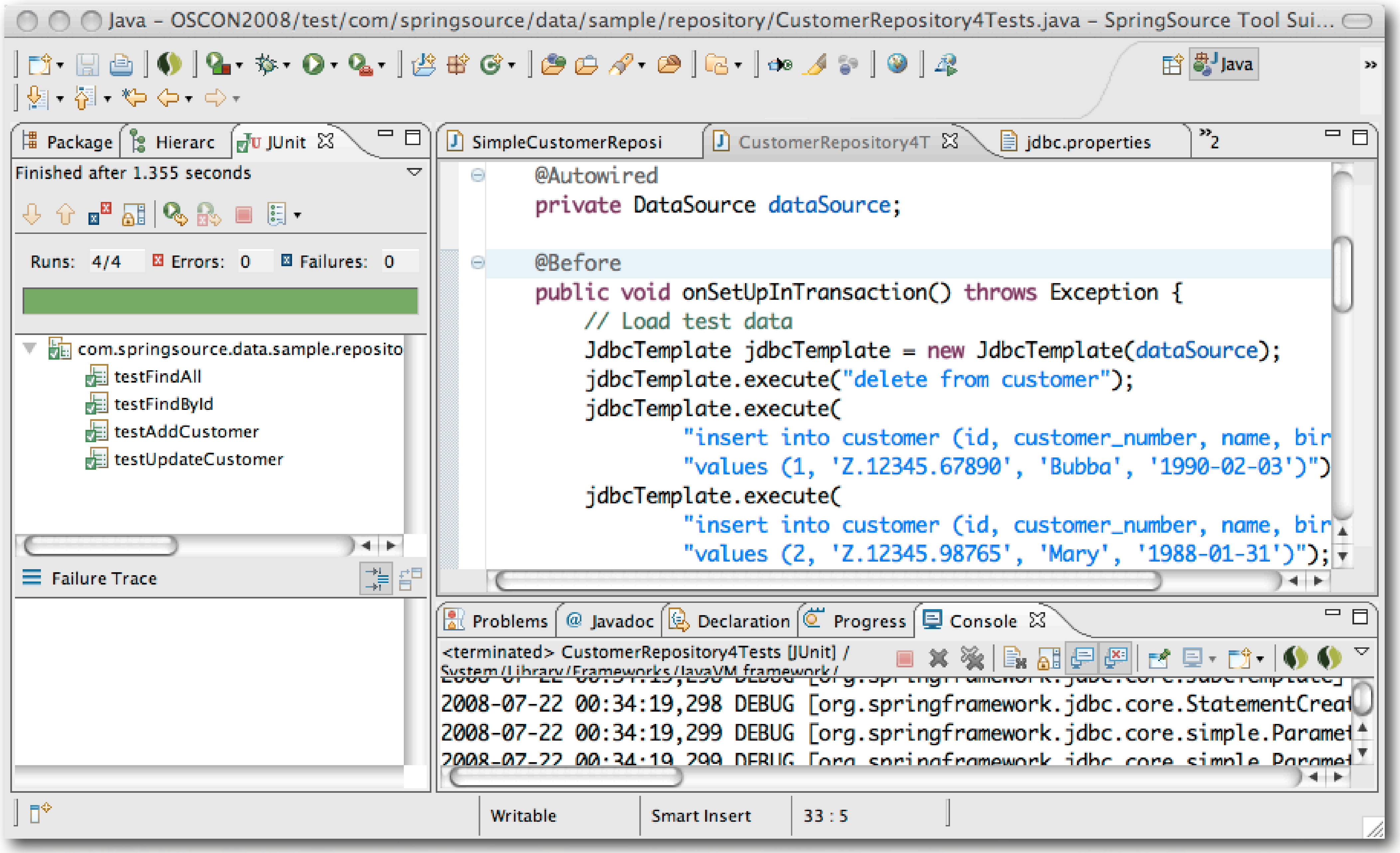
- BeanPropertySqlParameterSource
  - automatic mapping of property values from a JavaBean to parameters of same name
  - provides option of specifying SQL type

```
values.registerSqlType("birth_date", Types.DATE)
```

Mapping results

- ParameterizedRowMapper
  - provides for customized mapping from results data to domain class
- ParameterizedBeanPropertyRowMapper
  - automatic mapping from results data to a JavaBean. Column names are mapped to properties. (Use column aliases if names don’t match)

Live Code



Java – OSCON2008/test/com/springsource/data/sample/repository/CustomerRepository4Tests.java – SpringSource Tool Sui...

Finished after 1.355 seconds

Runs: 4/4   Errors: 0   Failures: 0

```
com.springsource.data.sample.reposito
    testFindAll
    testFindById
    testAddCustomer
    testUpdateCustomer
```

```
@Autowired
private DataSource dataSource;

@Before
public void onSetUpInTransaction() throws Exception {
    // Load test data
    JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);
    jdbcTemplate.execute("delete from customer");
    jdbcTemplate.execute(
            "insert into customer (id, customer_number, name, bir
            "values (1, 'Z.12345.67890', 'Bubba', '1990-02-03')")
    jdbcTemplate.execute(
            "insert into customer (id, customer_number, name, bir
            "values (2, 'Z.12345.98765', 'Mary', '1988-01-31')");
```

```
<terminated> CustomerRepository4Tests [JUnit] /System/Library/Frameworks/JavaVM.framework/
2008-07-22 00:34:19,298 DEBUG [org.springframework.jdbc.core.StatementCreat
2008-07-22 00:34:19,299 DEBUG [org.springframework.jdbc.core.simple.Paramet
2008-07-22 00:34:19,299 DEBUG [org.springframework.jdbc.core.simple.Paramet
```

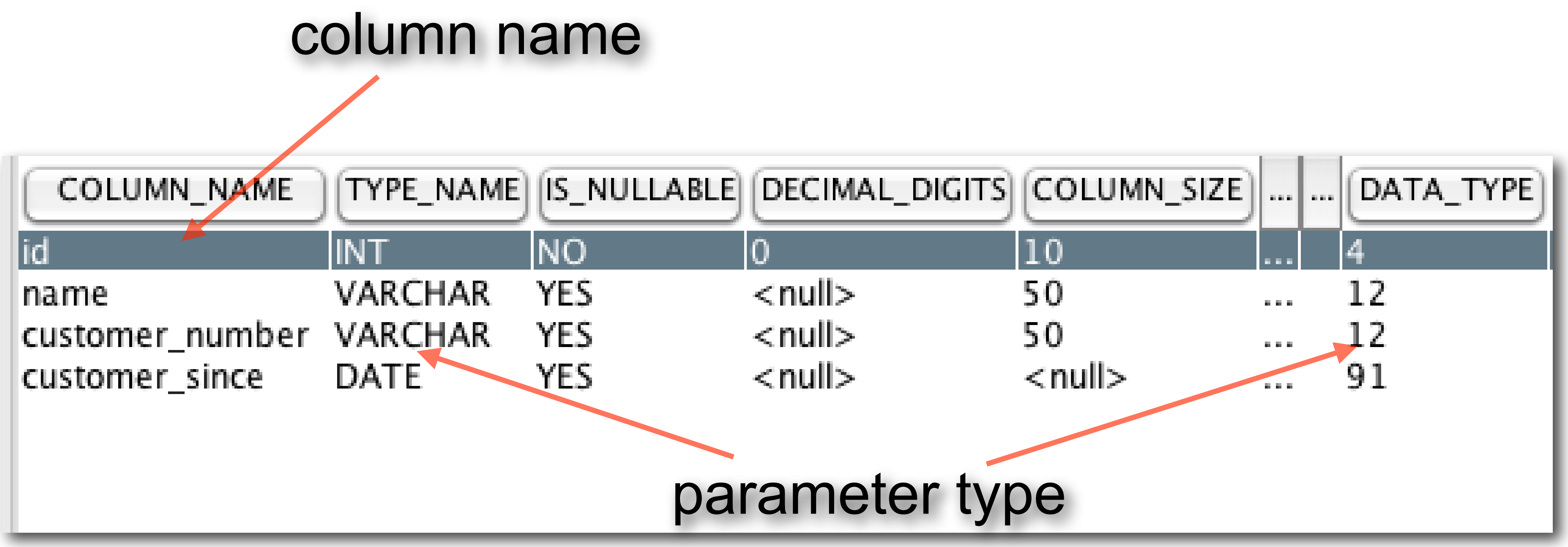
Insert data and access generated keys

- Traditionally a bit tricky - need to use PreparedStatementCreator and a KeyHolder
- What if -
  - you knew the name of the table
  - the name and type of the columns
- Maybe the JDBC framework could provide the rest ...

Create TABLE

```
CREATE
    TABLE customer
    (
        id int(11) NOT NULL AUTO_INCREMENT,
        name varchar(50),
        customer_number varchar(50),
        birth_date date,
        PRIMARY KEY (id)
    )
    ENGINE= InnoDB
```

Database Metadata customer table



COLUMN_NAME	TYPE_NAME	IS_NULLABLE	DECIMAL_DIGITS	COLUMN_SIZE	...	...	DATA_TYPE
id	INT	NO	0	10	...		4
name	VARCHAR	YES	<null>	50	...		12
customer_number	VARCHAR	YES	<null>	50	...		12
customer_since	DATE	YES	<null>	<null>	...		91

SimpleJdbcInsert

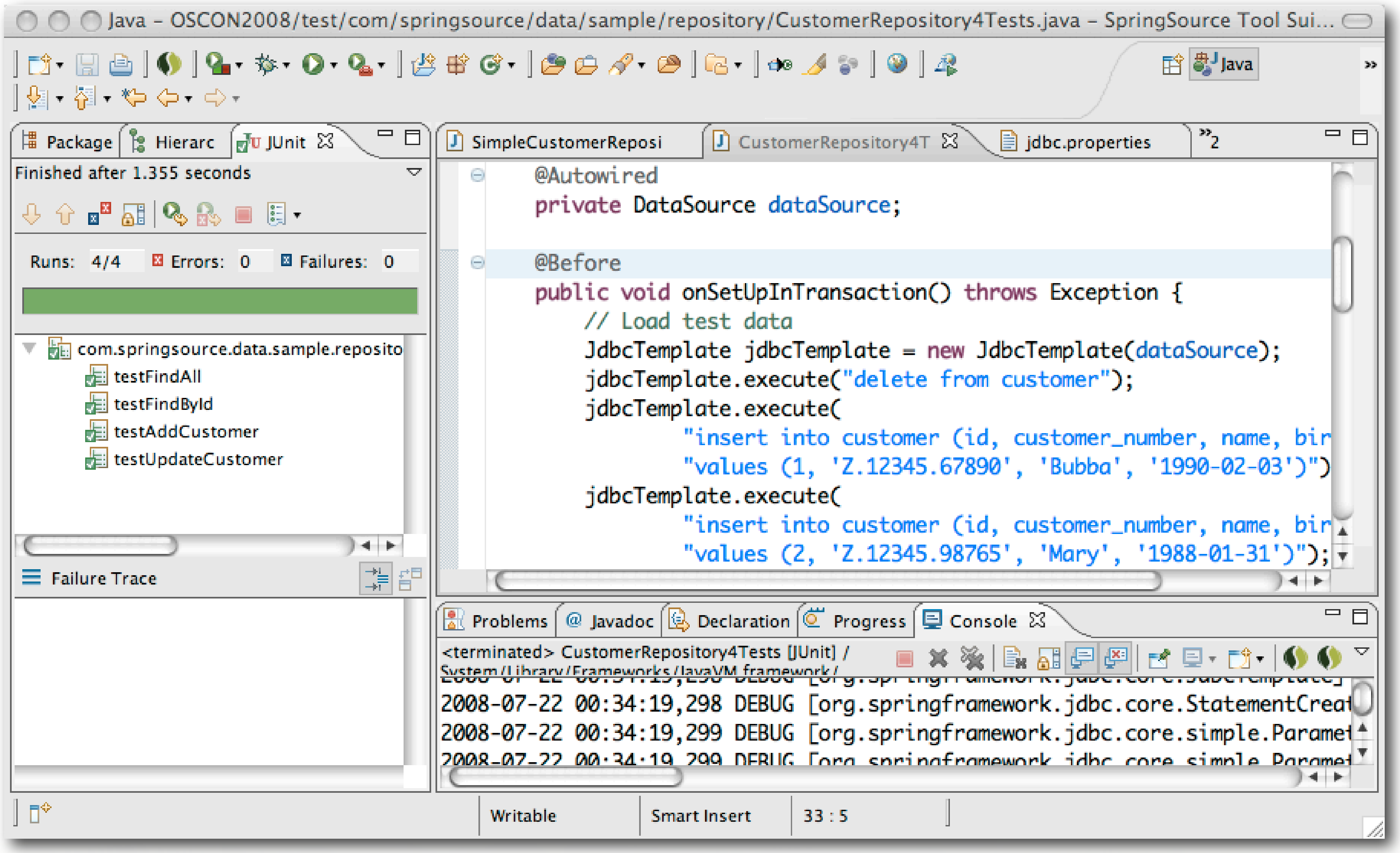
```
private SimpleJdbcInsert insertCustomer;
...
this.insertCustomer = new SimpleJdbcInsert(dataSource)
         .withTableName("customer")
         .usingGeneratedKeyColumns("id");
```

```
Number newId =
         insertCustomer.executeAndReturnKey(
                  new BeanPropertySqlParameterSource(customer));
customer.setId(newId.longValue());
```

SimpleJdbcInsert

- Simplifies insert operations
- Uses table metadata to provide automatic column configuration
- Option to provide explicit configuration
- Supports auto generated keys in a database independent fashion
- Supports batch inserts of arrays of Maps or SqlParameters

Live Code


Java – OSCON2008/test/com/springsource/data/sample/repository/CustomerRepository4Tests.java – SpringSource Tool Sui...

Finished after 1.355 seconds

Runs: 4/4 Errors: 0 Failures: 0

```
com.springsource.data.sample.reposito
    testFindAll
    testFindById
    testAddCustomer
    testUpdateCustomer
```

Failure Trace

```java
    @Autowired
    private DataSource dataSource;

    @Before
    public void onSetUpInTransaction() throws Exception {
        // Load test data
        JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);
        jdbcTemplate.execute("delete from customer");
        jdbcTemplate.execute(
                "insert into customer (id, customer_number, name, bir
                "values (1, 'Z.12345.67890', 'Bubba', '1990-02-03')")
        jdbcTemplate.execute(
                "insert into customer (id, customer_number, name, bir
                "values (2, 'Z.12345.98765', 'Mary', '1988-01-31')");
```

<terminated> CustomerRepository4Tests [JUnit] /System/Library/Frameworks/JavaVM.framework/

```
2008-07-22 00:34:19,298 DEBUG [org.springframework.jdbc.core.StatementCreat
2008-07-22 00:34:19,299 DEBUG [org.springframework.jdbc.core.simple.Paramet
2008-07-22 00:34:19,299 DEBUG [org.springframework.jdbc.core.simple.Paramet
```

Writable | Smart Insert | 33 : 5

Stored Procedures

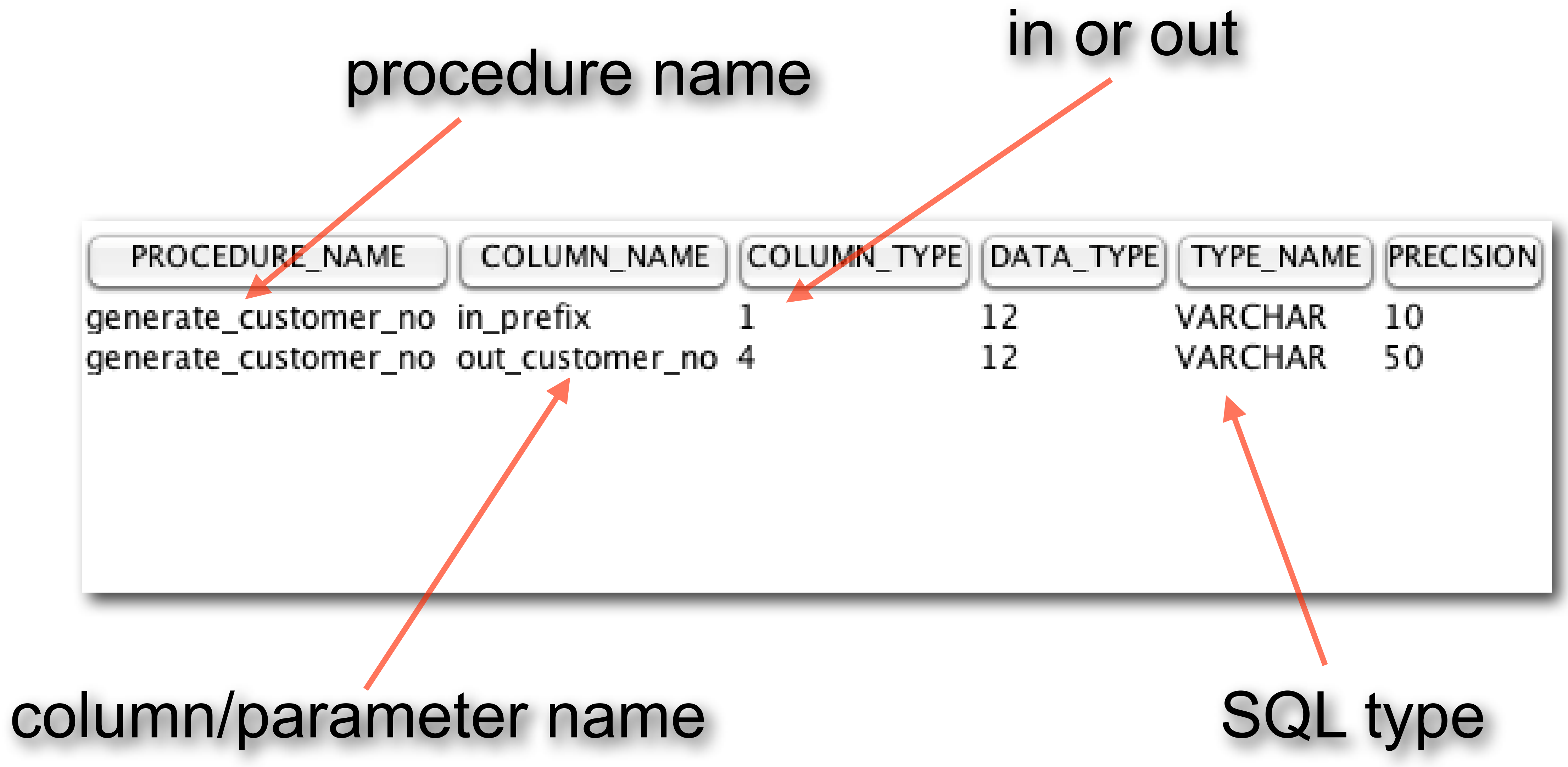
- Are they really that hard to use?
- What if -
  - you knew the name of the procedure
  - the name and type of the parameters

- Maybe the JDBC framework could provide the rest ...

generate_customer_no declaration

```
CREATE PROCEDURE generate_customer_no(
  IN in_prefix varchar(10),
  OUT out_customer_no varchar(50))
BEGIN
  SELECT CONCAT(in_prefix, '.',
        (RAND() * 10000000))
     INTO out_customer_no;
END
```

Database Metadata generate_customer_no



PROCEDURE_NAME	COLUMN_NAME	COLUMN_TYPE	DATA_TYPE	TYPE_NAME	PRECISION
generate_customer_no	in_prefix	1	12	VARCHAR	10
generate_customer_no	out_customer_no	4	12	VARCHAR	50

SimpleJdbcCall generate_customer_no

```
private SimpleJdbcCall generateCustomerNumberCall;
...
this.generateCustomerNumberCall = new SimpleJdbcCall(dataSource)
        .withProcedureName("generate_customer_no");
```

```
String customerNumber = generateCustomerNumberCall.executeObject(
        String.class,
        Collections.singletonMap("in_prefix", "XYZ"));
```

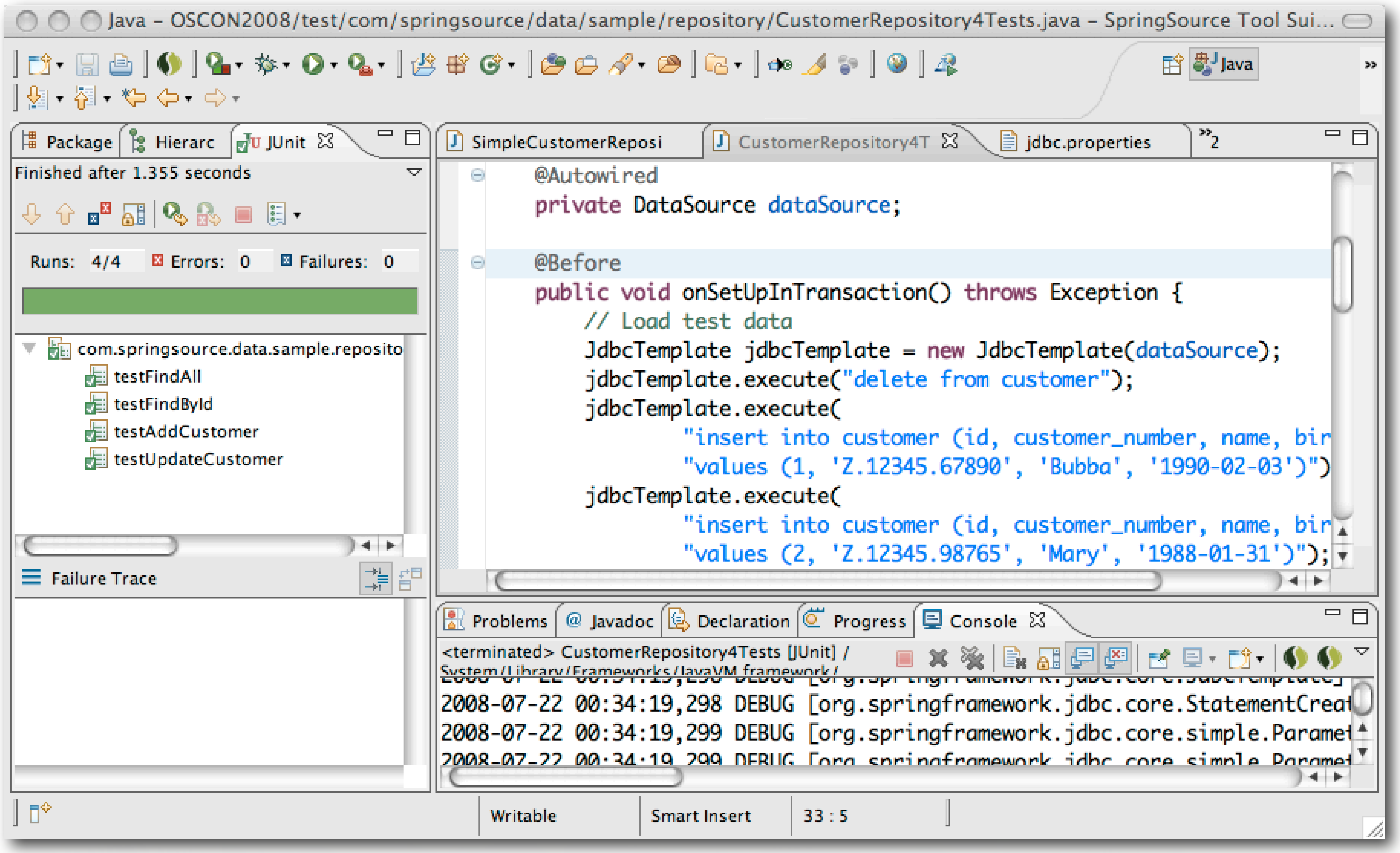
SimpleJdbcCall DEBUG output

```
DEBUG [org.springframework.jdbc.core.simple.SimpleJdbcCall] - JdbcCall call not compiled before execution - invoking compile
DEBUG [org.springframework.jdbc.core.metadata.CallMetaDataProviderFactory] - Using
org.springframework.jdbc.core.metadata.GenericCallMetaDataProvider
DEBUG [org.springframework.jdbc.core.metadata.CallMetaDataProvider] - Retrieving metadata for null/null/generate_customer_no
DEBUG [org.springframework.jdbc.core.metadata.CallMetaDataProvider] - Retrieved metadata: in_prefix 1 12 VARCHAR false
DEBUG [org.springframework.jdbc.core.metadata.CallMetaDataProvider] - Retrieved metadata: out_customer_no 4 12 VARCHAR false
DEBUG [org.springframework.jdbc.core.metadata.CallMetaDataContext] - Added metadata in parameter for: in_prefix
DEBUG [org.springframework.jdbc.core.metadata.CallMetaDataContext] - Added metadata out parameter for: out_customer_no
DEBUG [org.springframework.jdbc.core.simple.SimpleJdbcCall] - Compiled stored procedure. Call string is [{call
generate_customer_no(?, ?)}]
DEBUG [org.springframework.jdbc.core.simple.SimpleJdbcCall] - SqlCall for procedure [generate_customer_no] compiled
DEBUG [org.springframework.jdbc.core.metadata.CallMetaDataContext] - Matching {in_prefix=XYZ} with
{out_customer_no=out_customer_no, in_prefix=in_prefix}
DEBUG [org.springframework.jdbc.core.simple.SimpleJdbcCall] - The following parameters are used for call {call
generate_customer_no(?, ?)} with: {in_prefix=XYZ}
DEBUG [org.springframework.jdbc.core.simple.SimpleJdbcCall] - 1: in_prefix SQL Type 12 Type Name null
org.springframework.jdbc.core.SqlParameter
DEBUG [org.springframework.jdbc.core.simple.SimpleJdbcCall] - 2: out_customer_no SQL Type 12 Type Name null
org.springframework.jdbc.core.SqlOutParameter
DEBUG [org.springframework.jdbc.core.JdbcTemplate] - Calling stored procedure [{call generate_customer_no(?, ?)}]
DEBUG [org.springframework.jdbc.core.StatementCreatorUtils] - Setting SQL statement parameter value: column index 1, parameter value
[XYZ], value class [java.lang.String], SQL type 12
DEBUG [org.springframework.jdbc.core.JdbcTemplate] - CallableStatement.execute() returned 'false'
DEBUG [org.springframework.jdbc.core.JdbcTemplate] - CallableStatement.getUpdateCount() returned 0
INFO [org.springframework.jdbc.core.JdbcTemplate] - Added default SqlReturnUpdateCount parameter named #update-count-1
DEBUG [org.springframework.jdbc.core.JdbcTemplate] - CallableStatement.getUpdateCount() returned -1
```

SimpleJdbcCall

- Simplifies access to stored procedures
- Any database can be used with explicit parameter configuration
- Supported databases provides automatic parameter configuration:
  - Apache Derby
  - DB2
  - MySQL
  - Microsoft SQL Server
  - Oracle
  - Sybase

Live Code



```
    @Autowired
    private DataSource dataSource;

    @Before
    public void onSetUpInTransaction() throws Exception {
        // Load test data
        JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);
        jdbcTemplate.execute("delete from customer");
        jdbcTemplate.execute(
                "insert into customer (id, customer_number, name, bir
                "values (1, 'Z.12345.67890', 'Bubba', '1990-02-03')")
        jdbcTemplate.execute(
                "insert into customer (id, customer_number, name, bir
                "values (2, 'Z.12345.98765', 'Mary', '1988-01-31')");
```

```
<terminated> CustomerRepository4Tests [JUnit] /
System/Library/Frameworks/JavaVM.framework/
2008-07-22 00:34:19,298 DEBUG [org.springframework.jdbc.core.StatementCreat
2008-07-22 00:34:19,299 DEBUG [org.springframework.jdbc.core.simple.Paramet
2008-07-22 00:34:19,299 DEBUG [org.springframework.jdbc.core.simple.Paramet
```

Q & A

Questions?

thomas.risberg@springsource.com

Source Code

Customer.java

```
package com.springsource.data.sample.domain;

import java.util.Date;

public class Customer
 {
    private Long id;

    private String name;

    private String customerNumber;

    private Date birthDate;

    public Long getId() {
        return id;
     }

    public void setId(Long id) {
        this.id = id;
     }
```

(continued)

Source Code

Customer.java (continued)

```
    public String getName() {
        return name;
     }

    public void setName(String name) {
        this.name = name;
     }

    public String getCustomerNumber() {
        return customerNumber;
     }

    public void setCustomerNumber(String customerNumber) {
        this.customerNumber = customerNumber;
     }

    public Date getBirthDate() {
        return birthDate;
    }

    public void setBirthDate(Date birthDate) {
        this.birthDate = birthDate;
    }

}
```

Source Code

SimpleCustomerRepository.java

```
package com.springsource.data.sample.repository;

import java.util.Collections;
import java.util.List;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.namedparam.BeanPropertySqlParameterSource;
import org.springframework.jdbc.core.simple.ParameterizedBeanPropertyRowMapper;
import org.springframework.jdbc.core.simple.SimpleJdbcCall;
import org.springframework.jdbc.core.simple.SimpleJdbcInsert;
import org.springframework.jdbc.core.simple.SimpleJdbcTemplate;
import org.springframework.stereotype.Repository;

import com.springsource.data.sample.domain.Customer;


@Repository
public class SimpleCustomerRepository implements CustomerRepository {

    private SimpleJdbcTemplate simpleJdbcTemplate;

    private SimpleJdbcInsert insertCustomer;

    private SimpleJdbcCall generateCustomerNumberCall;
```

(continued)

Source Code

SimpleCustomerRepository.java (continued)

```java
@Autowired
public void init(DataSource dataSource) {

    this.simpleJdbcTemplate = new SimpleJdbcTemplate(dataSource);

    this.insertCustomer = new SimpleJdbcInsert(dataSource)
        .withTableName("customer")
        .usingGeneratedKeyColumns("id");

    this.generateCustomerNumberCall = new SimpleJdbcCall(dataSource)
        .withProcedureName("generate_customer_no");

}

public void add(Customer customer) {
    String customerNumber = generateCustomerNumberCall.executeObject(
            String.class,
            Collections.singletonMap("in_prefix", "XYZ"));
    customer.setCustomerNumber(customerNumber);
    Number newId =
        insertCustomer.executeAndReturnKey(
                new BeanPropertySqlParameterSource(customer));
        customer.setId(newId.longValue());
}
```

(continued)

Source Code

SimpleCustomerRepository.java (continued)

```
    public Customer findById(Long id) {
        Customer customer = simpleJdbcTemplate.queryForObject(
                "select id, name, customer_number, birth_date from customer where id = ?",
                 ParameterizedBeanPropertyRowMapper.newInstance(Customer.class),
                 id);
        return customer;
    }

    public void save(Customer customer) {
        simpleJdbcTemplate.update(
                "update customer set name = :name, birth_date = :birthDate where id = :id",
                 new BeanPropertySqlParameterSource(customer));
    }

    public List<Customer> findAll() {
         List<Customer> customerList = simpleJdbcTemplate.query(
            "select id, name, customer_number, birth_date from customer",
            ParameterizedBeanPropertyRowMapper.newInstance(Customer.class));
         return customerList;
    }

}
```

Source Code

CustomerRepository4Tests.java

```
package com.springsource.data.sample.repository;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotNull;
import static org.junit.Assert.assertTrue;

import java.util.Date;
import java.util.List;

import javax.sql.DataSource;

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.transaction.annotation.Transactional;

import com.springsource.data.sample.domain.Customer;
```

(continued)

Source Code

CustomerRepository4Tests.java (continued)

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration
public class CustomerRepository4Tests {

    @Autowired
    private CustomerRepository customerRepository;

    @Autowired
    private DataSource dataSource;

    @Before
    public void onSetUpInTransaction() throws Exception {
        // Load test data
        JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);
        jdbcTemplate.execute("delete from customer");
        jdbcTemplate.execute(
                "insert into customer (id, customer_number, name, birth_date) " +
                "values (1, 'Z.12345.67890', 'Bubba', '1990-02-03')");
        jdbcTemplate.execute(
                "insert into customer (id, customer_number, name, birth_date) " +
                "values (2, 'Z.12345.98765', 'Mary', '1988-01-31')");
    }
```

(continued)

Source Code

CustomerRepository4Tests.java (continued)

```
    @Transactional @Test
    public void testFindAll() {
        List<Customer> l = customerRepository.findAll();
        assertEquals("Wrong number of customers returned", 2, l.size());
    }

    @Transactional @Test
    public void testFindById() {
        Customer c = customerRepository.findById(2L);
        assertNotNull("Customer not found", c);
    }

    @Transactional @Test
    public void testAddCustomer() {
        Customer c = new Customer();
        c.setBirthDate(new Date(104400000L));
        c.setName("Sven");
        customerRepository.add(c);
        assertNotNull("Customer id not assigned", c.getId());
        assertTrue("Bad customer id", 3 <= c.getId());
        assertNotNull("Customer number not assigned", c.getCustomerNumber());
    }
```

(continued)

Source Code

CustomerRepository4Tests.java (continued)

```
    @Transactional @Test
    public void testUpdateCustomer() {
        Customer c = customerRepository.findById(2L);
        c.setBirthDate(new Date(18000000L));
        customerRepository.save(c);
        Customer c2 = customerRepository.findById(2L);
        assertEquals("BirthDate not updated", 18000000L, c2.getBirthDate().getTime());
    }
}
```

Source Code

CustomerRepository4Tests-context.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.springframework.org/schema/tx
            http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
        http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context-2.5.xsd">

    <context:annotation-config/>

    <context:component-scan base-package="com.springsource.data.sample"/>

    <context:property-placeholder location="classpath:jdbc.properties"/>

    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close"
        p:driverClassName="${jdbc.driverClassName}"
        p:url="${jdbc.url}"
        p:username="${jdbc.username}"
        p:password="${jdbc.password}"/>

    <tx:annotation-driven/>

    <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"
        p:dataSource-ref="dataSource"/>

</beans>
```