Assignment 2 Solution

Introduction to Database Systems

DataLab

CS, NTHU

Outline

- *UpdateItemPrice* transaction (SP/JDBC implementations)
- *StatisticManager*
- *An example of Experiment Results*

Outline

- *UpdateItemPrice* transaction (SP/JDBC implementations)
- *StatisticManager*
- *An example of Experiment Results*

Modified/Added Classes

- Shared class
  - *As2BenchTxnType*
  - *As2BenchConstants*
- Client-side classes
  - *As2BenchRte*
  - *As2UpdateItemPriceParamGen*
  - *As2BenchJdbcExecutor*
  - *UpdateItemPriceTxnJdbcJob*
- Server-side classes
  - *As2BenchStoredProcFactory*
  - *UpdateItemPriceProcParamHelper*
  - *UpdateItemPriceTxnProc*

Modified/Added Classes

- Shared class
  - *As2BenchTxnType*
  - *As2BenchConstants*
- Client-side classes
  - *As2BenchRte*
  - *As2UpdateItemPriceParamGen*
  - *As2BenchJdbcExecutor*
  - *UpdateItemPriceTxnJdbcJob*
- Server-side classes
  - *As2BenchStoredProcFactory*
  - *UpdateItemPriceProcParamHelper*
  - *UpdateItemPriceTxnProc*

New Transaction Type

```
public enum As2BenchTxnType implements TransactionType {
    // Loading procedures
    TESTBED_LOADER,

    // Benchmarking procedures
    READ_ITEM;
    READ_ITEM, UPDATE_ITEM_PRICE;

    public static As2BenchTxnType fromProcedureId(int pid) {
        return As2BenchTxnType.values()[pid];
    }

    public int getProcedureId() {
        return this.ordinal();
    }

    public boolean isBenchmarkingTx() {
        if (this == READ_ITEM)
        if (this == READ_ITEM || this == UPDATE_ITEM_PRICE)
            return true;
        return false;
    }
}
```

READ_WRITE_TX_RATE

```
public class As2BenchConstants {

public static final int NUM_ITEMS;
    public static final int NUM_ITEMS;
    public static final double READ_WRITE_TX_RATE;

    static {
        NUM_ITEMS = BenchProperties.getLoader().getPropertyAsInteger(
                As2BenchConstants.class.getName() + ".NUM_ITEMS", 100000);
        READ_WRITE_TX_RATE = BenchProperties.getLoader().getPropertyAsDouble(
                As2BenchConstants.class.getName() + ".READ_WRITE_TX_RATE", 0.0);
    }

    public static final int MIN_IM = 1;
    public static final int MAX_IM = 10000;
    public static final double MIN_PRICE = 1.00;
    public static final double MAX_PRICE = 100.00;
    public static final int MIN_I_NAME = 14;
    public static final int MAX_I_NAME = 24;
    public static final int MIN_I_DATA = 26;
    public static final int MAX_I_DATA = 50;
    public static final int MONEY_DECIMALS = 2;

}
```

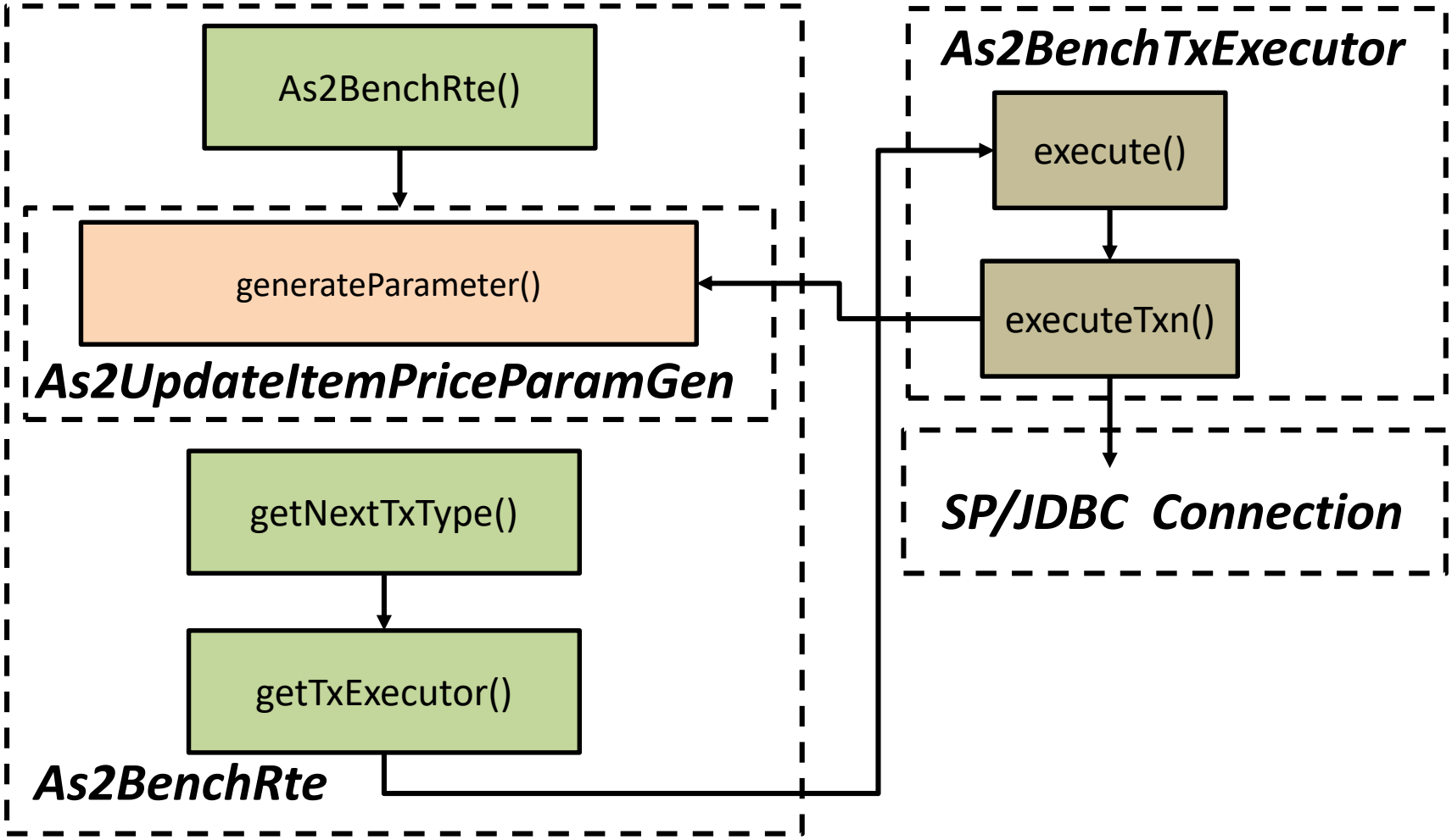
Modified/Added Classes

- Shared class
  - *As2BenchTxnType*
  - *As2BenchConstants*
- Client-side classes
  - *As2BenchRte*
  - *As2UpdateItemPriceParamGen*
  - *As2BenchJdbcExecutor*
  - *UpdateItemPriceTxnJdbcJob*
- Server-side classes
  - *As2BenchStoredProcFactory*
  - *UpdateItemPriceProcParamHelper*
  - *UpdateItemPriceTxnProc*

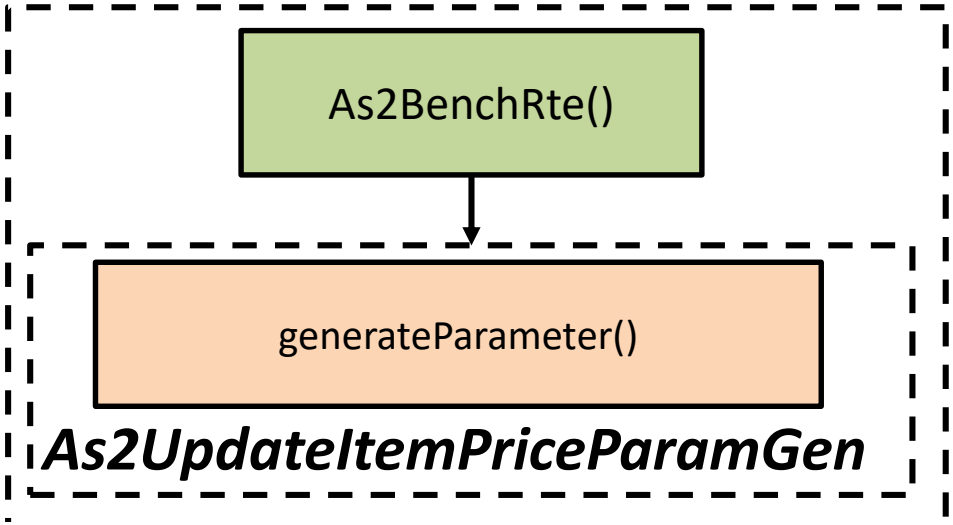
Modified/Added Classes (Shared)

- Shared class
  - *As2BenchTxnType*
  - *As2BenchConstants*
- Client-side classes
  - *As2BenchRte*
  - *As2UpdateItemPriceParamGen*
  - *As2BenchJdbcExecutor*
  - *UpdateItemPriceTxnJdbcJob*
- Server-side classes
  - *As2BenchStoredProcFactory*
  - *UpdateItemPriceProcParamHelper*
  - *UpdateItemPriceTxnProc*

Workflow of As2BenchRte



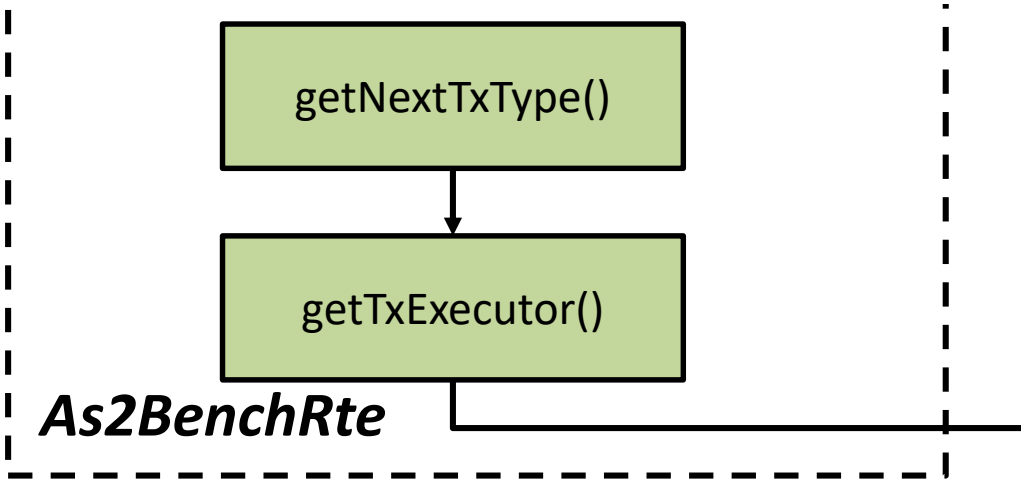
Initialize RTE



Initialize RTE

```java
public class As2BenchRte extends RemoteTerminalEmulator<As2BenchTxnType> {

    private As2BenchTxExecutor executor;
    private Map<As2BenchTxnType, As2BenchTxExecutor> executors;
    private RandomValueGenerator rvg;

    public As2BenchRte(SutConnection conn, StatisticMgr statMgr) {
        super(conn, statMgr);
        executor = new As2BenchTxExecutor(new As2ReadItemParamGen());
        executors = new HashMap<As2BenchTxnType, As2BenchTxExecutor>();
        rvg = new RandomValueGenerator();
        executors.put(As2BenchTxnType.READ_ITEM, new As2BenchTxExecutor(new As2ReadItemParamGen()));
        executors.put(As2BenchTxnType.UPDATE_ITEM_PRICE, new As2BenchTxExecutor(new As2UpdateItemPriceParamGen()));
    }
```

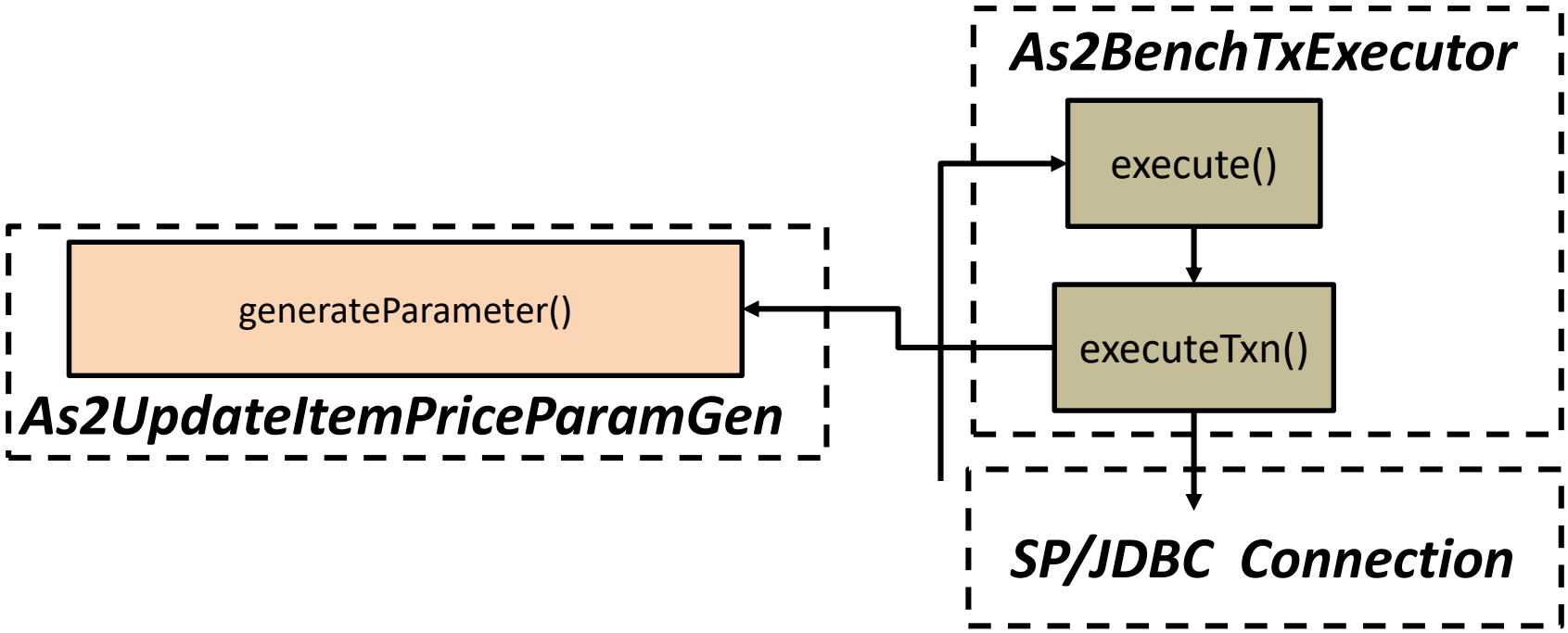
Choose a Transaction



Choose a Transaction

```
    protected As2BenchTxnType getNextTxType() {
        if (rvg.fixedDecimalNumber(4, 0.0001, 1.0000) <= As2BenchConstants.READ_WRITE_TX_RATE)
            return As2BenchTxnType.UPDATE_ITEM_PRICE;
        else
        return As2BenchTxnType.READ_ITEM;
            return As2BenchTxnType.READ_ITEM;
    }

    protected As2BenchTxExecutor getTxExeutor(As2BenchTxnType type) {
        return executor;
        return executors.get(type);
    }
}
```

Generate and Send Parameters



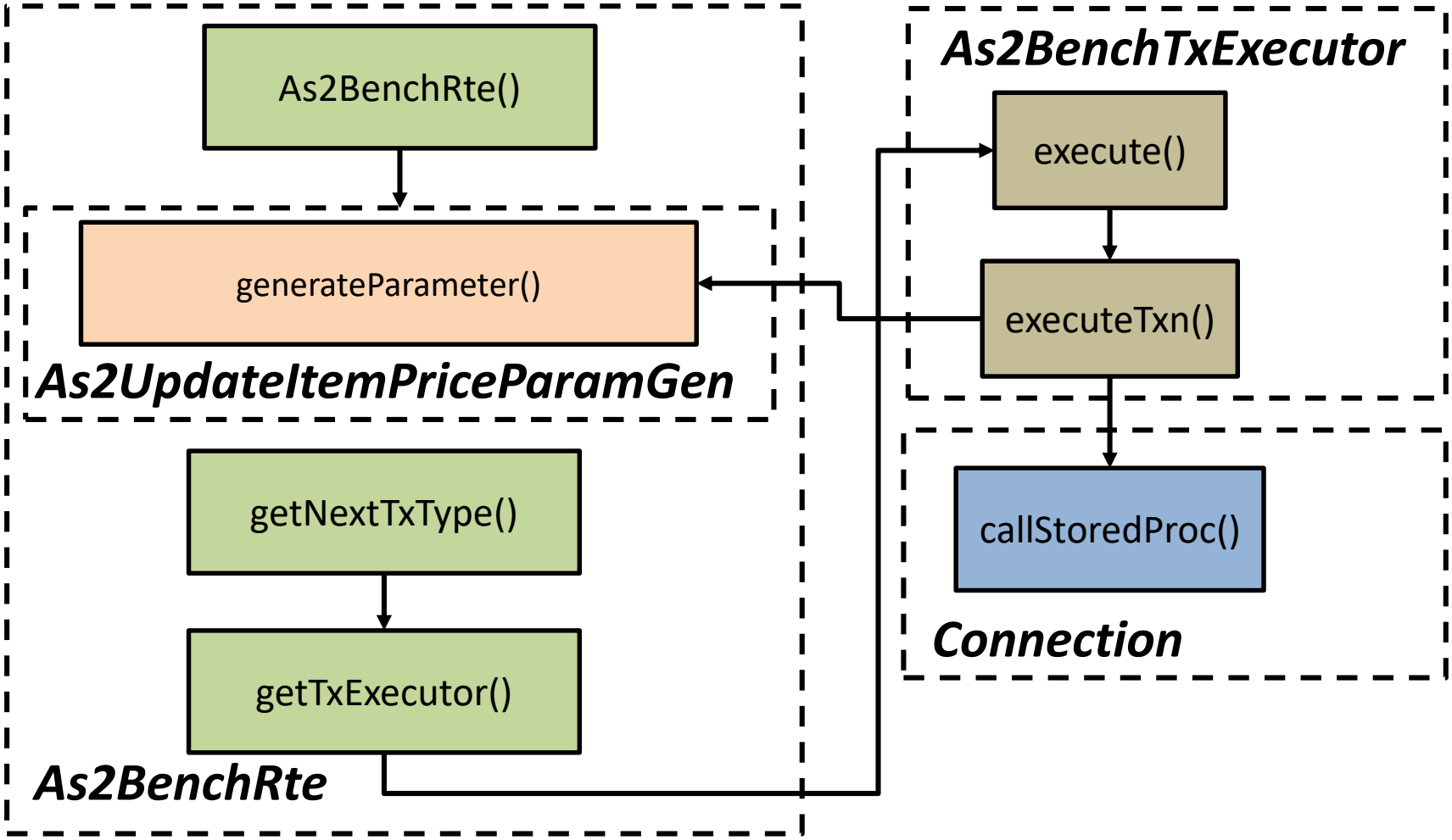
Generate Parameters

```java
public class As2UpdateItemPriceParamGen implements TxParamGenerator<As2BenchTxnType> {

    private static final int UPDATE_COUNT = 10;

    @Override
    public As2BenchTxnType getTxnType() {
        return As2BenchTxnType.UPDATE_ITEM_PRICE;
    }

    @Override
    public Object[] generateParameter() {
        RandomValueGenerator rvg = new RandomValueGenerator();
        LinkedList<Object> paramList = new LinkedList<Object>();

        paramList.add(UPDATE_COUNT);
        for (int i = 0; i < UPDATE_COUNT; i++)
            paramList.add(rvg.number(1, As2BenchConstants.NUM_ITEMS));
        for (int i = 0; i < UPDATE_COUNT; i++)
            paramList.add(rvg.fixedDecimalNumber(2, 0.0, 5.0));

        return paramList.toArray();
    }

}
```

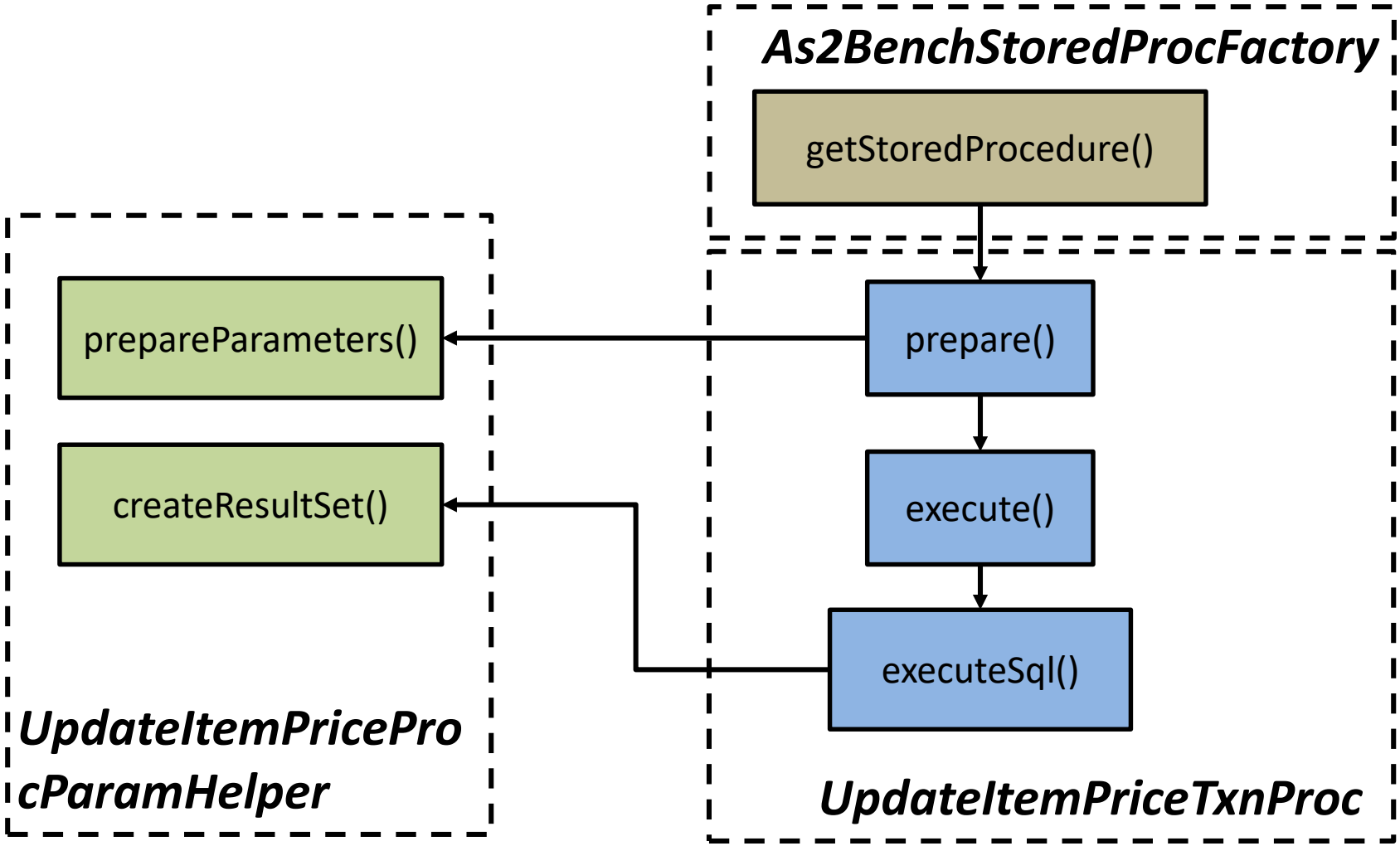
Modified/Added Classes (SP)

- Shared class
  - *As2BenchTxnType*
  - *As2BenchConstants*
- Client-side classes
  - *As2BenchRte*
  - *As2UpdateItemPriceParamGen*
  - *As2BenchJdbcExecutor*
  - *UpdateItemPriceTxnJdbcJob*
- Server-side classes
  - *As2BenchStoredProcFactory*
  - *UpdateItemPriceProcParamHelper*
  - *UpdateItemPriceTxnProc*

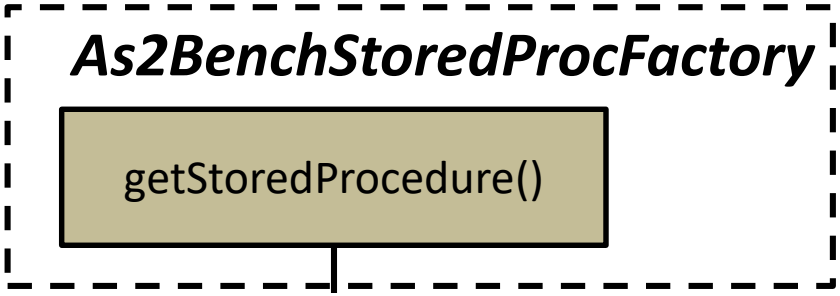
Inquiry via SP



Execute a Stored Procedure



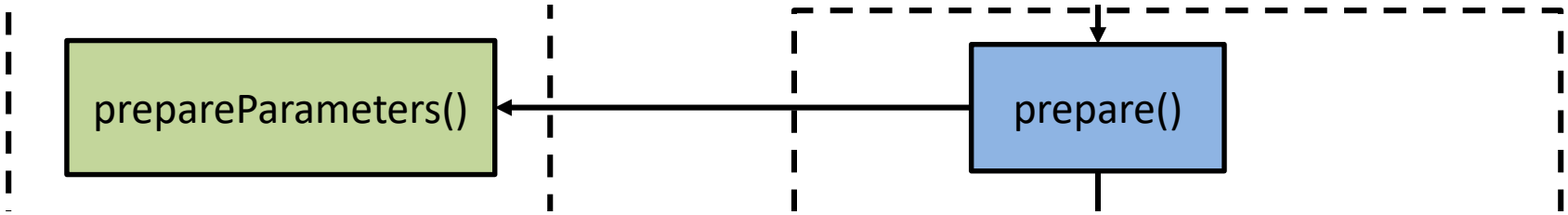
Get the Specified SP



Get the Specified SP

```java
public class As2BenchStoredProcFactory implements StoredProcedureFactory {

    @Override
    public StoredProcedure getStroredProcedure(int pid) {
        StoredProcedure sp;
        switch (As2BenchTxnType.fromProcedureId(pid)) {
        case TESTBED_LOADER:
            sp = new TestbedLoaderProc();
            break;
        case READ_ITEM:
            sp = new ReadItemTxnProc();
            break;
        case UPDATE_ITEM_PRICE:
            sp = new UpdateItemPriceTxnProc();
            break;
        default:
            sp = null;
        }
        return sp;
    }
}
```

Preprocess Parameters

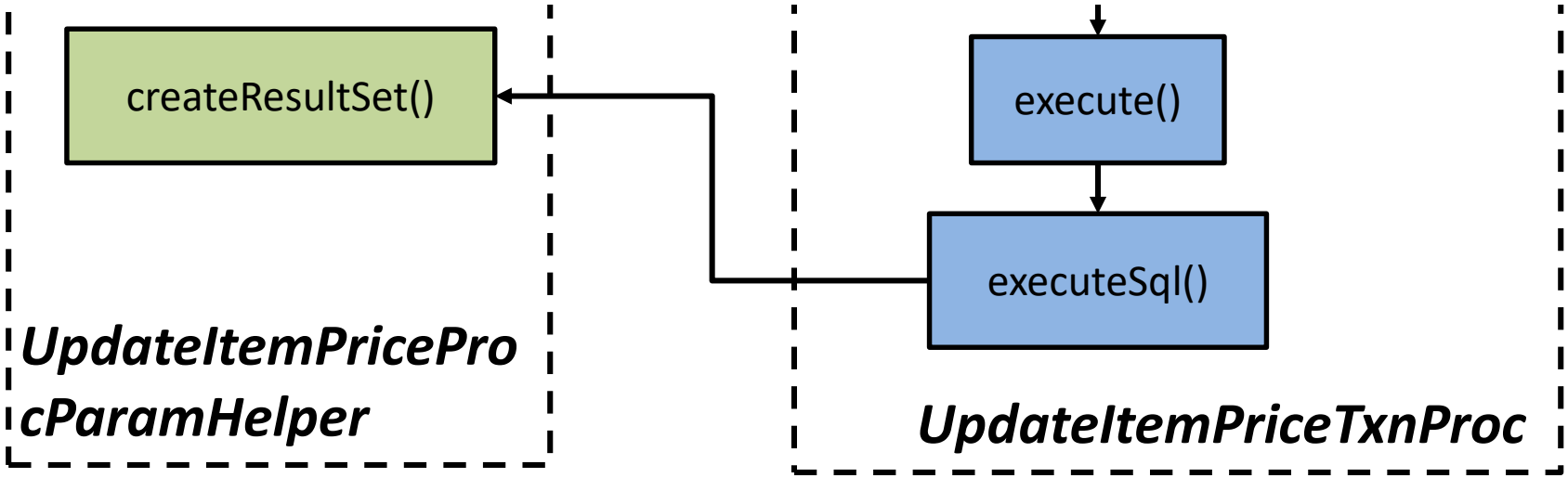


Preprocess Parameters

```
public double getUpdateItemPrice(int index) {
    return updateItemPrice[index];
}
```

```
@Override
public void prepareParameters(Object... pars) {
    int indexCnt = 0;

    updateCount = (Integer) pars[indexCnt++];
    updateItemId = new int[updateCount];
    updateItemPrice = new double[updateCount];
    itemName = new String[updateCount];
    itemPrice = new double[updateCount];

    for (int i = 0; i < updateCount; i++)
        updateItemId[i] = (Integer) pars[indexCnt++];
    for (int i = 0; i < updateCount; i++)
        updateItemPrice[i] = (Double) pars[indexCnt++];
}
```

Execute Queries



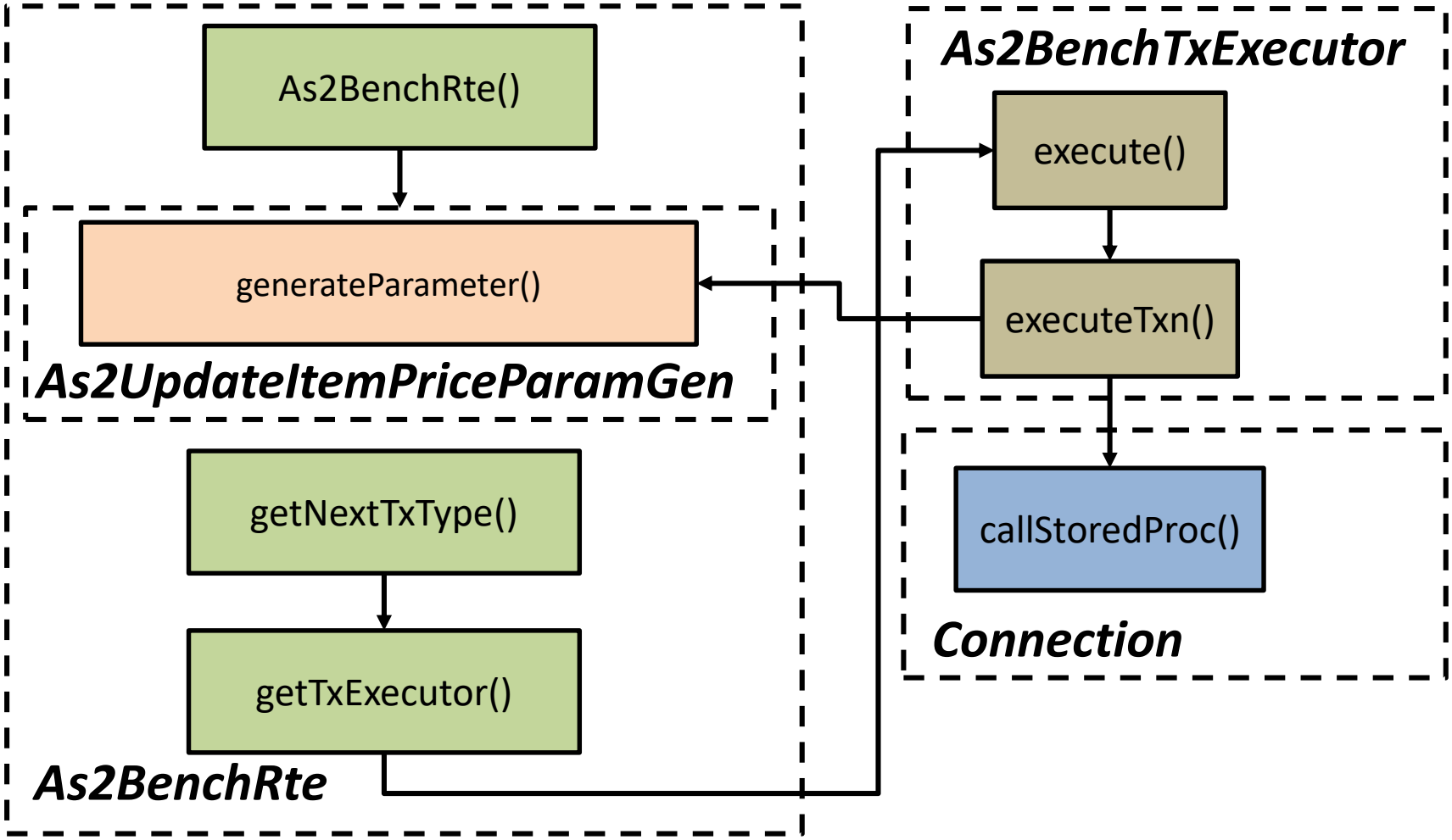
Execute Queries

```java
@Override
protected void executeSql() {
    String name;
    double price;

    for (int idx = 0; idx < paramHelper.getUpdateCount(); idx++) {
        int iid = paramHelper.getUpdateItemId(idx);
        Plan p = VanillaDb.newPlanner().createQueryPlan(
                "SELECT i_name, i_price FROM item WHERE i_id = " + iid, tx);
        Scan s = p.open();
        s.beforeFirst();
        if (s.next()) {
            name = (String) s.getVal("i_name").asJavaVal();
            price = (Double) s.getVal("i_price").asJavaVal();

            paramHelper.setItemName(name, idx);
            paramHelper.setItemPrice(price, idx);
        } else
            throw new RuntimeException("Cloud not find item record with i_id = " + iid);
        s.close();

        // Check the original price
        if (price > As2BenchConstants.MAX_PRICE) {
            price = As2BenchConstants.MIN_PRICE;
        } else {
            price += paramHelper.getUpdateItemPrice(idx);
        }
        paramHelper.setItemPrice(price, idx);

        // Update the price
        int result = VanillaDb.newPlanner().executeUpdate(
                "UPDATE item SET i_price = " + price + " WHERE i_id = " + iid, tx);
        if (result == 0)
            throw new RuntimeException("Could not change the price of item with i_id = " + iid);
    }
}
```

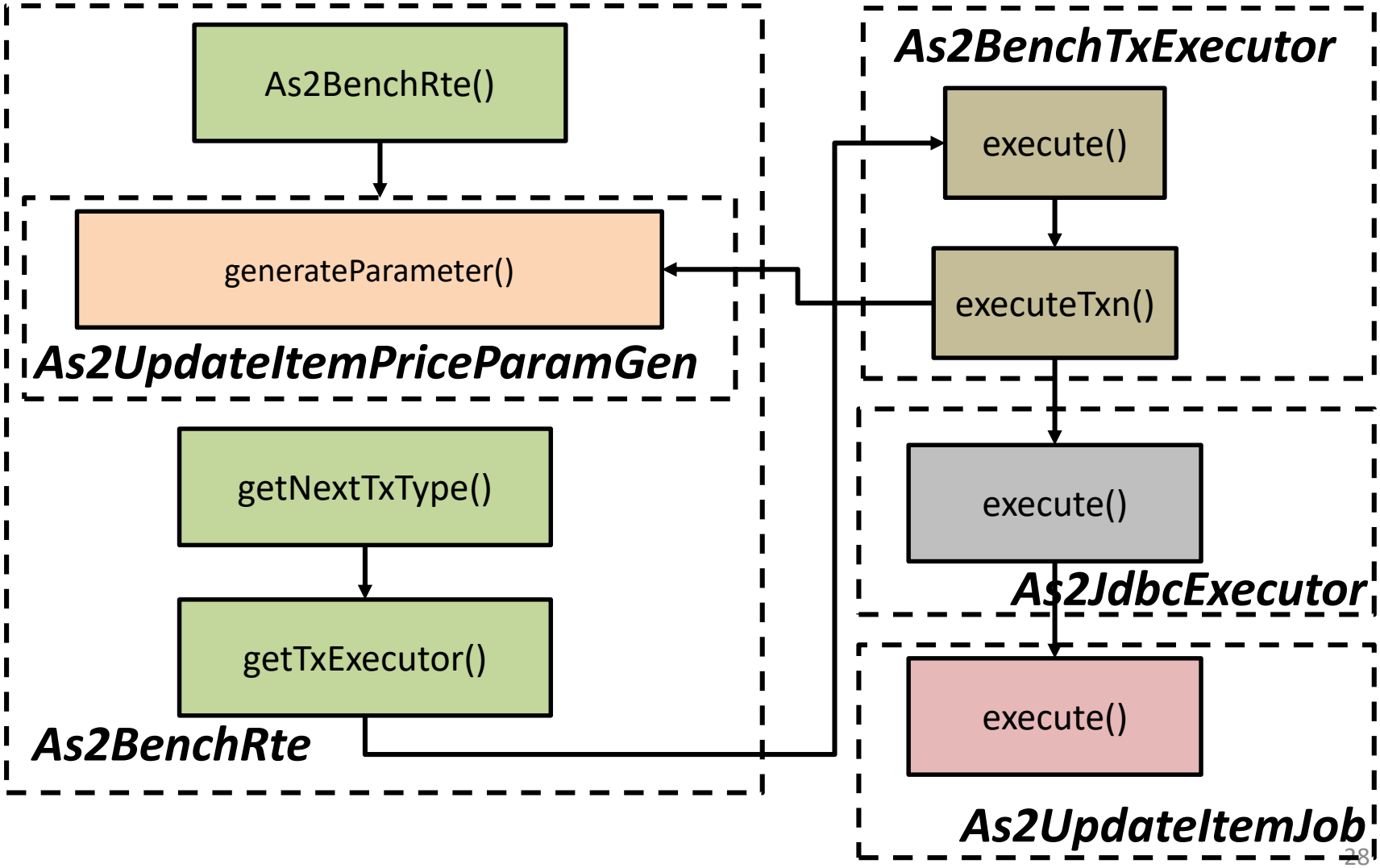
Modified/Added Classes (JDBC)

- Shared class
  - *As2BenchTxnType*
  - *As2BenchConstants*
- Client-side classes
  - *As2BenchRte*
  - *As2UpdateItemPriceParamGen*
  - *As2BenchJdbcExecutor*
  - *UpdateItemPriceTxnJdbcJob*
- Server-side classes
  - *As2BenchStoredProcFactory*
  - *UpdateItemPriceProcParamHelper*
  - *UpdateItemPriceTxnProc*

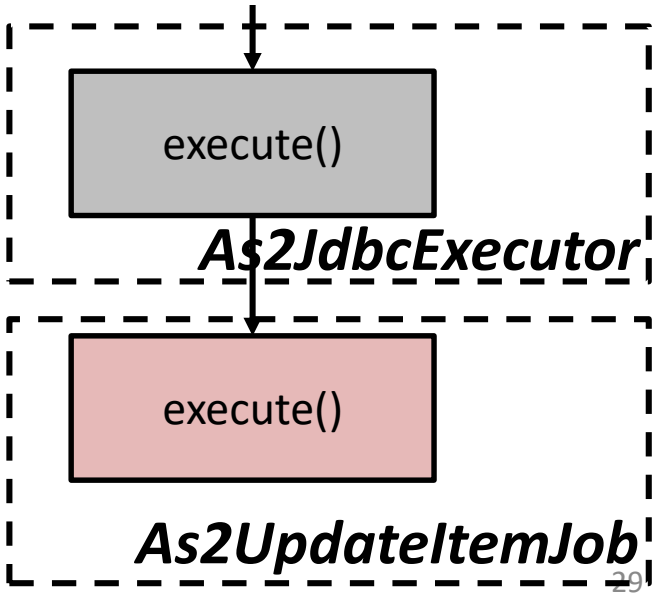
Inquiry via SP



Inquiry via JDBC



Inquiry via JDBC



Inquiry via JDBC

```java
public class As2BenchJdbcExecutor implements JdbcExecutor<As2BenchTxnType> {

    @Override
    public SutResultSet execute(Connection conn, As2BenchTxnType txType, Object[] pars)
            throws SQLException {
        switch (txType) {
        case TESTBED_LOADER:
            return new TestbedLoaderJdbcJob().execute(conn, pars);
        case READ_ITEM:
            return new ReadItemTxnJdbcJob().execute(conn, pars);
        case UPDATE_ITEM_PRICE:
            return new UpdateItemPriceTxnJdbcJob().execute(conn, pars);
        default:
            throw new UnsupportedOperationException(
                    String.format("no JDCB implementation for '%s'", txType));
        }
    }

}
```

Execute Queries

```java
@Override
public SutResultSet execute(Connection conn, Object[] pars) throws SQLException {
    // Parse parameters
    int updateCount = (Integer) pars[0];
    int[] itemIds = new int[updateCount];
    double[] itemPrices = new double[updateCount];
    for (int i = 0; i < updateCount; i++)
        itemIds[i] = (Integer) pars[i + 1];
    for (int i = 0; i < updateCount; i++)
        itemPrices[i] = (Double) pars[i + updateCount + 1];

        Statement statement = conn.createStatement();
        ResultSet rs = null;
        double price = 0.0;
        for (int i = 0; i < 10; i++) {
            // Select the name and the price
            String sql = "SELECT i_name, i_price FROM item WHERE i_id = " + itemIds[i];
            rs = statement.executeQuery(sql);
            rs.beforeFirst();
            if (rs.next()) {
                outputMsg.append(String.format("'%s': ", rs.getString("i_name")));
                price = rs.getDouble("i_price");
            } else
                throw new RuntimeException("cannot find the record with i_id = " + itemIds[i]);
            rs.close();

            // Check the original price
            if (price > As2BenchConstants.MAX_PRICE) {
                price = As2BenchConstants.MIN_PRICE;
            } else {
                price += itemPrices[i];
            }
            outputMsg.append(String.format("'%f', ", price));

            // Update the price
            sql = "UPDATE item SET i_price = " + price + " WHERE i_id = " + itemIds[i];
            int result = statement.executeUpdate(sql);
            if (result == 0)
                throw new RuntimeException("cannot change the price of item with i_id = " + itemIds[i]);
        }
        conn.commit();
```

Outline

- *UpdateItemPrice* transaction (SP/JDBC implementations)
- *StatisticManager*
- *An example of Experiment Results*

Modified Class

- *StatisticMgr*

Latency History

```
private void addTxnLatency(TxnResultSet rs) {
    long elapsedTime = TimeUnit.NANOSECONDS.toMillis(rs.getTxnEndTime() - Benchmarker.BENCH_START_TIME);
    long timeSlotBoundary = (elapsedTime / GRANULARITY) * GRANULARITY / 1000; // in seconds

    ArrayList<Long> timeSlot = latencyHistory.get(timeSlotBoundary);
    if (timeSlot == null) {
        timeSlot = new ArrayList<Long>();
        latencyHistory.put(timeSlotBoundary, timeSlot);
    }
    timeSlot.add(TimeUnit.NANOSECONDS.toMillis(rs.getTxnResponseTime()));
}
```

(0, [145, 27, 33, …])
(5, [23, 11, 150, …])
(10, [28, 16, 50, …])
…

```java
private String makeStatString(long timeSlotBoundary, List<Long> timeSlot) {
    Collections.sort(timeSlot);

    // Transfer it to unmodifiable in order to prevent modification
    // when we use a sublist to access it.
    timeSlot = Collections.unmodifiableList(timeSlot);

    int count = timeSlot.size();
    int middleOffset = timeSlot.size() / 2;
    long lowerQ, upperQ, median;
    double mean;

    median = calcMedian(timeSlot);
    mean = calcMean(timeSlot);

    if (count < 2) { // Boundary case: there is only one number in the list
        lowerQ = median;
        upperQ = median;
    } else if (count % 2 == 0) { // Even
        lowerQ = calcMedian(timeSlot.subList(0, middleOffset));
        upperQ = calcMedian(timeSlot.subList(middleOffset, count));
    } else { // Odd
        lowerQ = calcMedian(timeSlot.subList(0, middleOffset));
        upperQ = calcMedian(timeSlot.subList(middleOffset + 1, count));
    }

    Long min = Collections.min(timeSlot);
    Long max = Collections.max(timeSlot);

    return String.format("%d, %d, %f, %d, %d, %d, %d, %d",
            timeSlotBoundary, count, mean, min, max, lowerQ, median, upperQ);
}
```

(0, [145, 27, 33, …])
(5, [23, 11, 150, …])
(10, [28, 16, 50, …])
…

Outline

- *UpdateItemPrice* transaction (SP/JDBC implementations)
- *StatisticManager*
- *An example of Experiment Results*

An Example of Experiments