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*

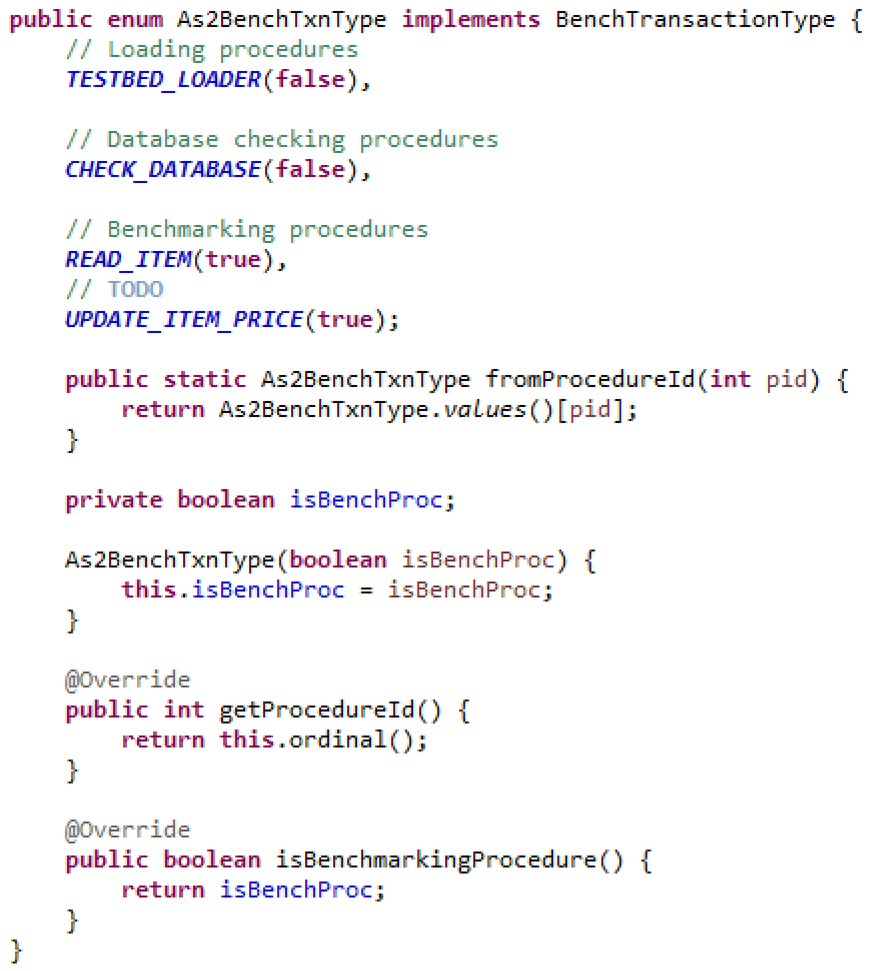
Modified/Added Classes

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

Modified/Added Classes

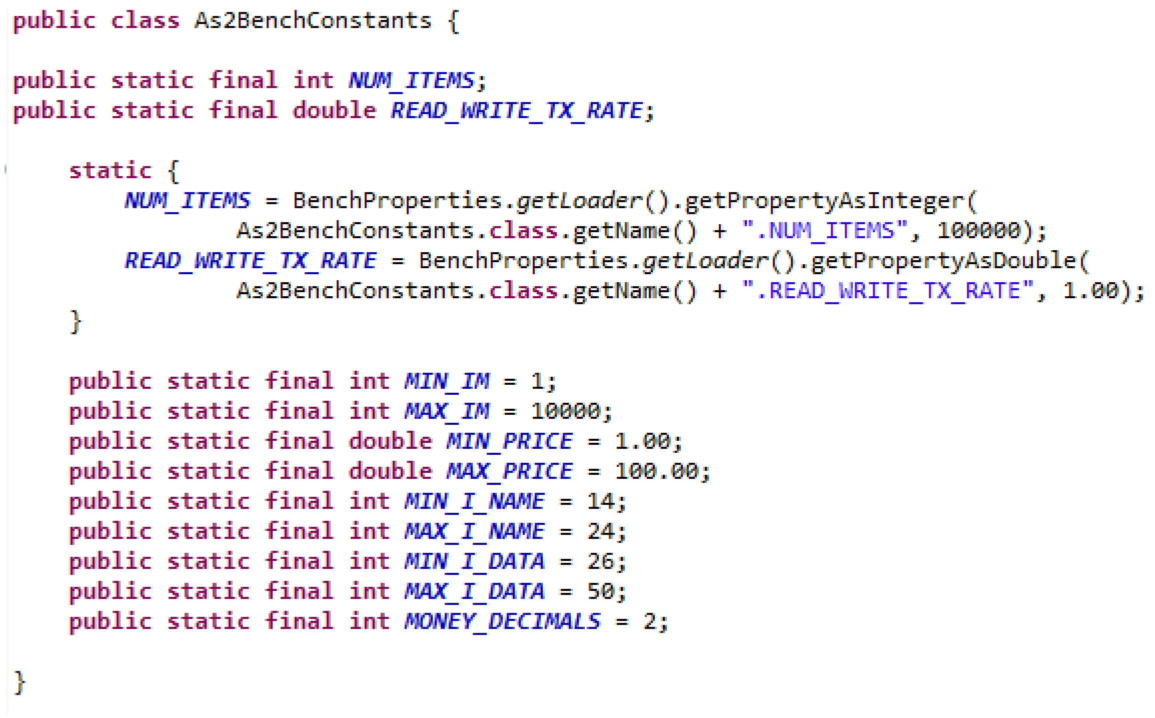
- Shared class
  - *As2BenchTxnType*
  - *As2BenchConstants*
- Client-side classes
  - *As2BenchRte*
  - *As2UpdateItemPriceParamGen*
  - *As2BenchJdbcExecutor*
  - *UpdateItemPriceTxnJdbcJob*
- Server-side classes
  - *As2BenchStoredProcFactory*
  - *UpdateItemPriceProcParamHelper*
  - *UpdateItemPriceTxnProc*

New Transaction Type



```java
public enum As2BenchTxnType implements BenchTransactionType {
    // Loading procedures
    TESTBED_LOADER(false),

    // Database checking procedures
    CHECK_DATABASE(false),

    // Benchmarking procedures
    READ_ITEM(true),
    // TODO
    UPDATE_ITEM_PRICE(true);

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

    private boolean isBenchProc;

    As2BenchTxnType(boolean isBenchProc) {
        this.isBenchProc = isBenchProc;
    }

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

    @Override
    public boolean isBenchmarkingProcedure() {
        return isBenchProc;
    }
}
```

READ_WRITE_TX_RATE



```java
public class As2BenchConstants {

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", 1.00);
    }

    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;

}
```

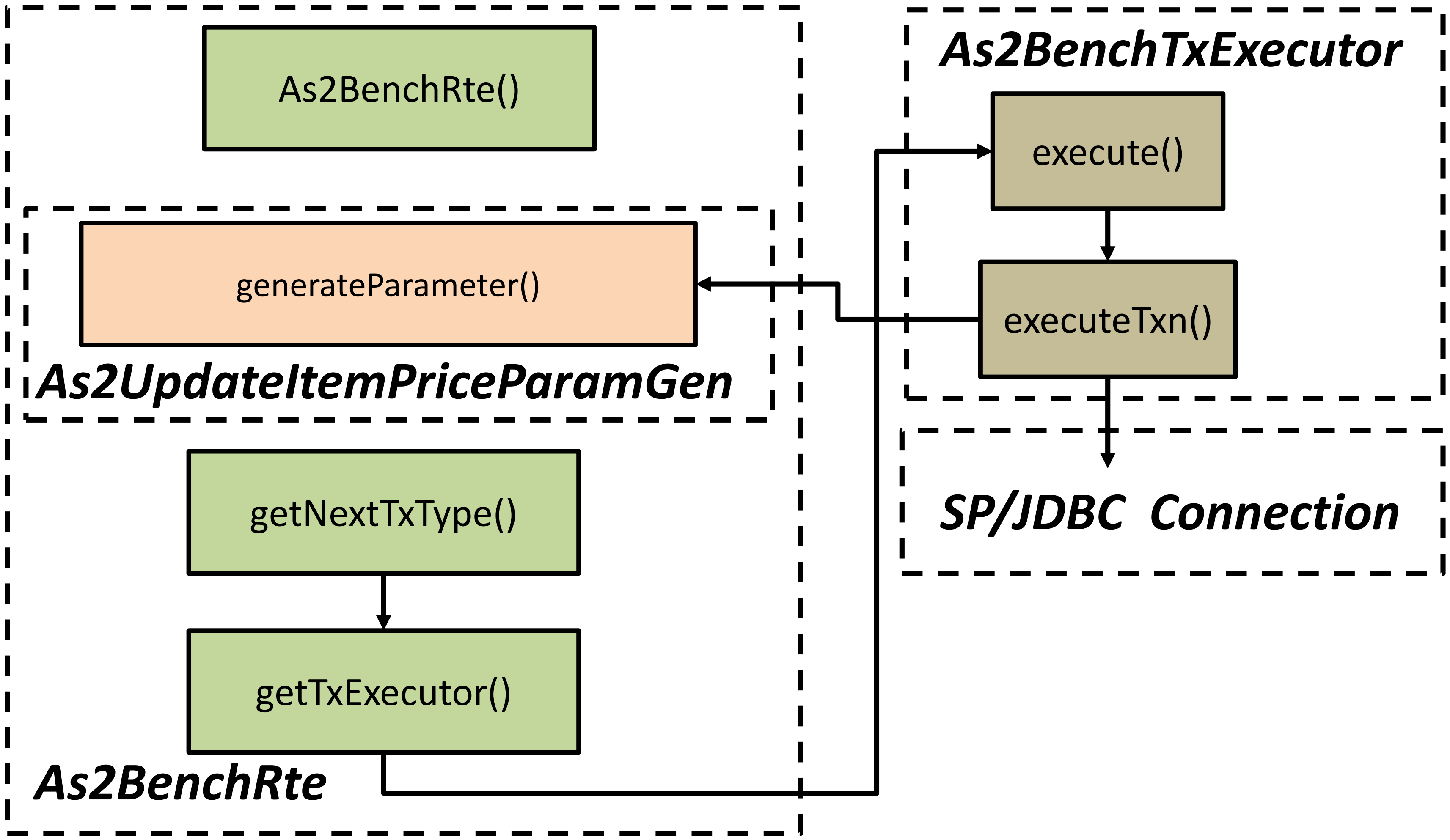
Modified/Added Classes

- 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*

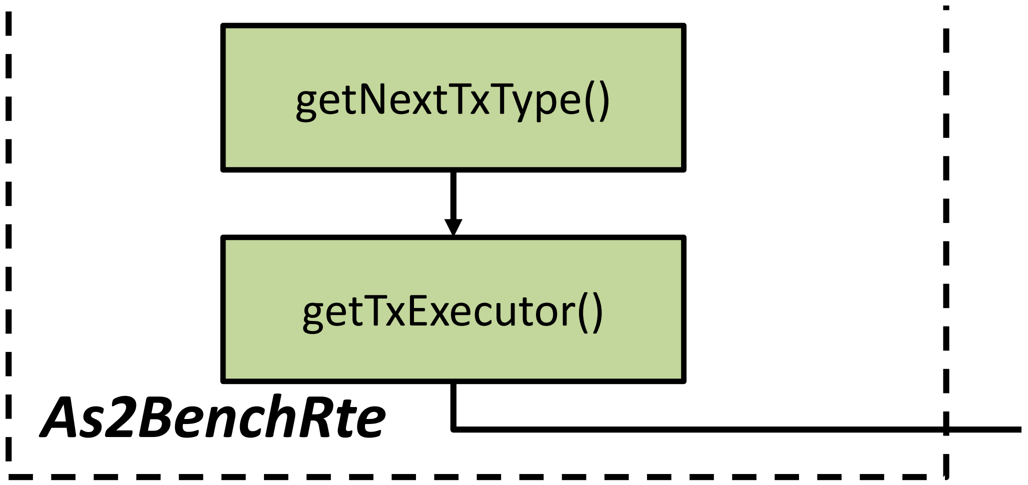
Workflow of As2BenchRte



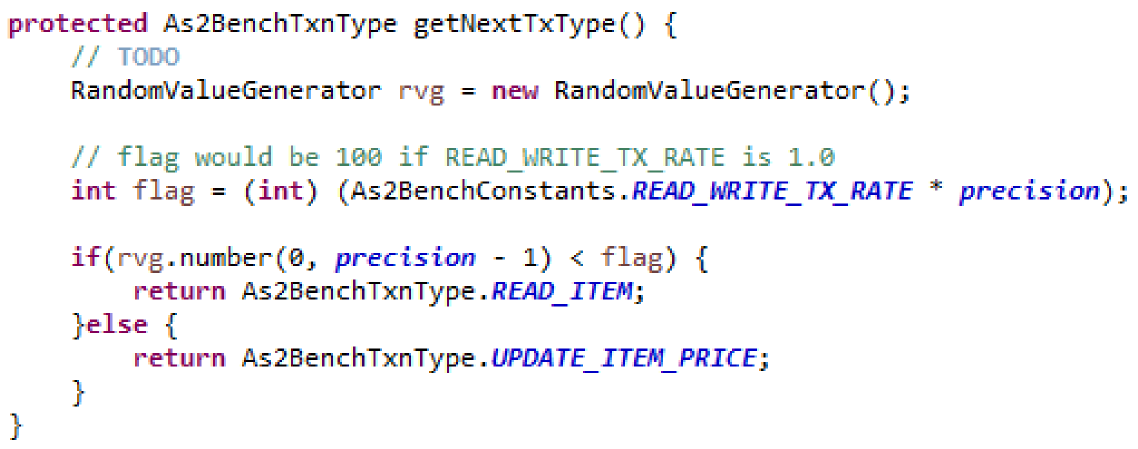
As2BenchRte

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

    private As2BenchTxExecutor executor;
    private static final int precision = 100;

    public As2BenchRte(SutConnection conn, StatisticMgr statMgr) {
        super(conn, statMgr);
    }

    protected As2BenchTxnType getNextTxType() {
        // TODO
        RandomValueGenerator rvg = new RandomValueGenerator();

        // flag would be 100 if READ_WRITE_TX_RATE is 1.0
        int flag = (int) (As2BenchConstants.READ_WRITE_TX_RATE * precision);

        if(rvg.number(0, precision - 1) < flag) {
            return As2BenchTxnType.READ_ITEM;
        }else {
            return As2BenchTxnType.UPDATE_ITEM_PRICE;
        }
    }

    protected As2BenchTxExecutor getTxExeutor(As2BenchTxnType type) {
        // TODO
        TxParamGenerator<As2BenchTxnType> paraGen;
        switch (type) {
        case READ_ITEM:
            paraGen = new As2ReadItemParamGen();
            break;

        case UPDATE_ITEM_PRICE:
            paraGen = new As2UpdateItemPriceTxnParamGen();
            break;

        default:
            paraGen = new As2ReadItemParamGen();
            break;
        }
        executor = new As2BenchTxExecutor(paraGen);
        return executor;
    }
}
```

Choose a Transaction

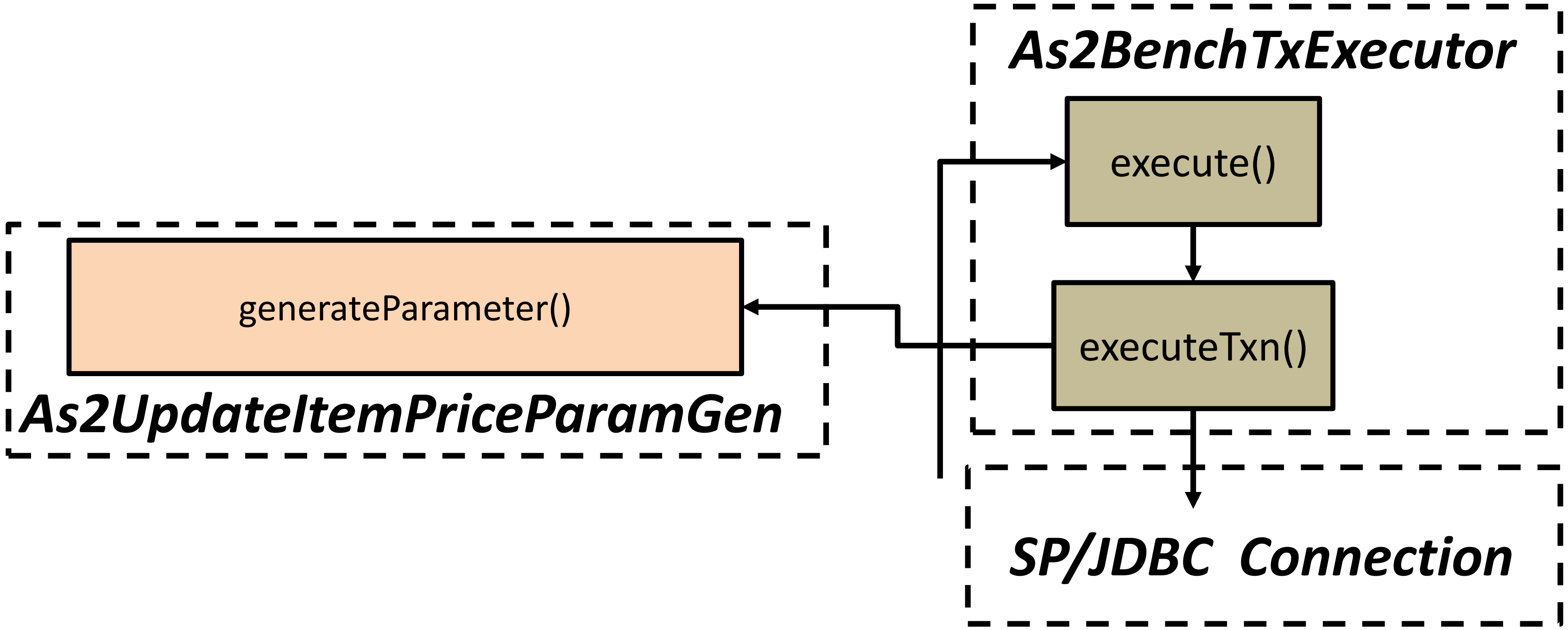


Choose a Transaction

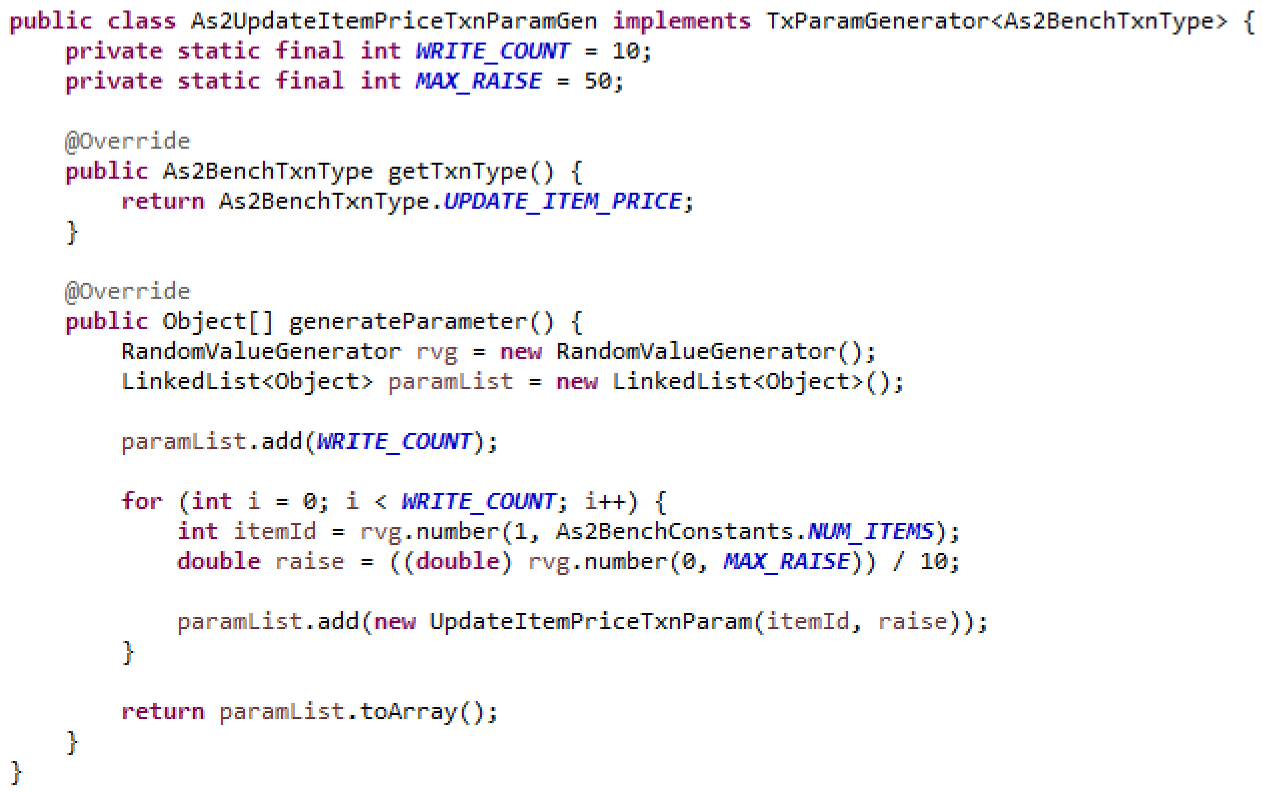


```
protected As2BenchTxnType getNextTxType() {
    // TODO
    RandomValueGenerator rvg = new RandomValueGenerator();

    // flag would be 100 if READ_WRITE_TX_RATE is 1.0
    int flag = (int) (As2BenchConstants.READ_WRITE_TX_RATE * precision);

    if(rvg.number(0, precision - 1) < flag) {
        return As2BenchTxnType.READ_ITEM;
    }else {
        return As2BenchTxnType.UPDATE_ITEM_PRICE;
    }
}
```

Generate and Send Parameters



Generate Parameters

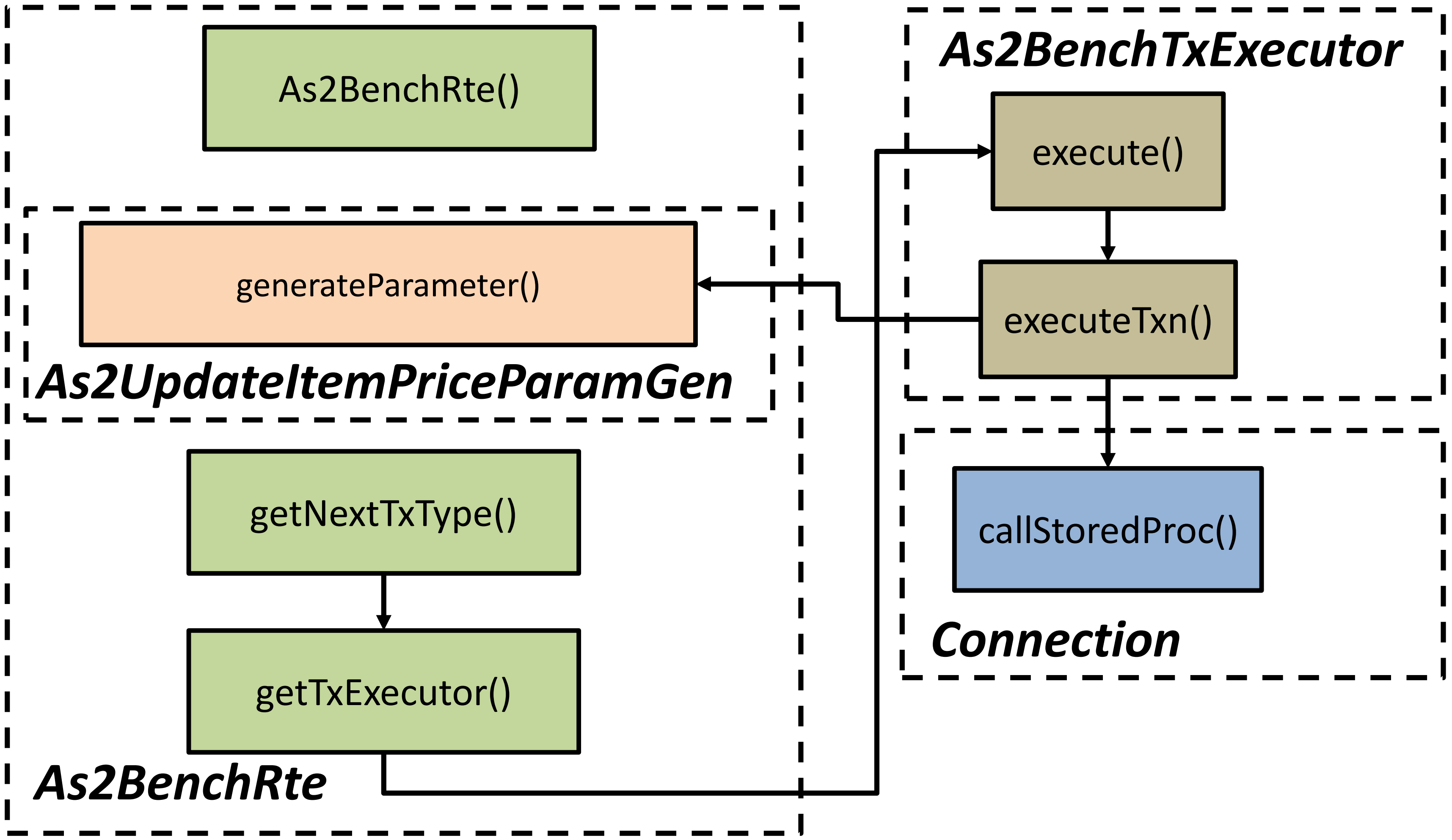


```
public class As2UpdateItemPriceTxnParamGen implements TxParamGenerator<As2BenchTxnType> {
    private static final int WRITE_COUNT = 10;
    private static final int MAX_RAISE = 50;

    @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(WRITE_COUNT);

        for (int i = 0; i < WRITE_COUNT; i++) {
            int itemId = rvg.number(1, As2BenchConstants.NUM_ITEMS);
            double raise = ((double) rvg.number(0, MAX_RAISE)) / 10;

            paramList.add(new UpdateItemPriceTxnParam(itemId, raise));
        }

        return paramList.toArray();
    }
}
```

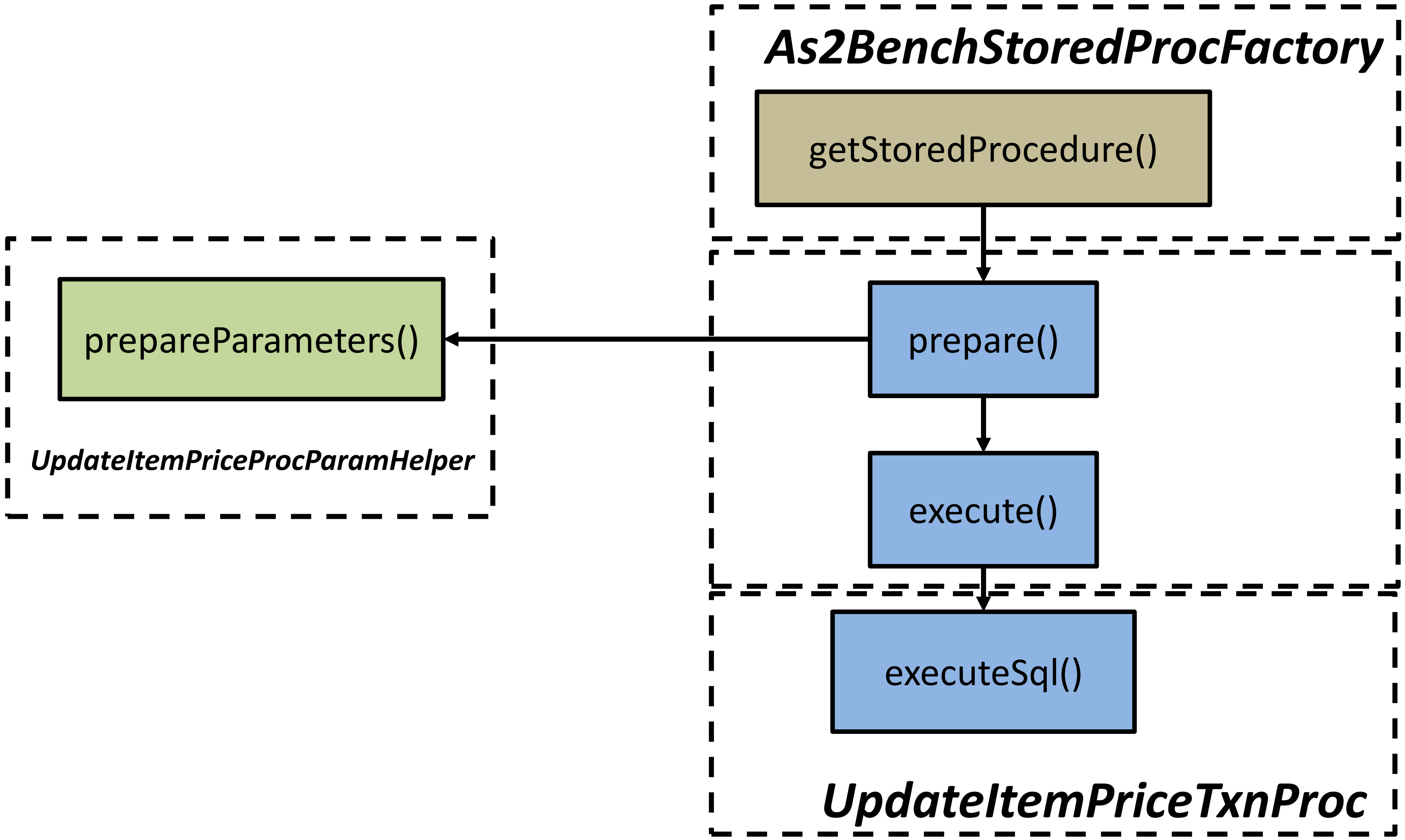
Modified/Added Classes (SP)

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

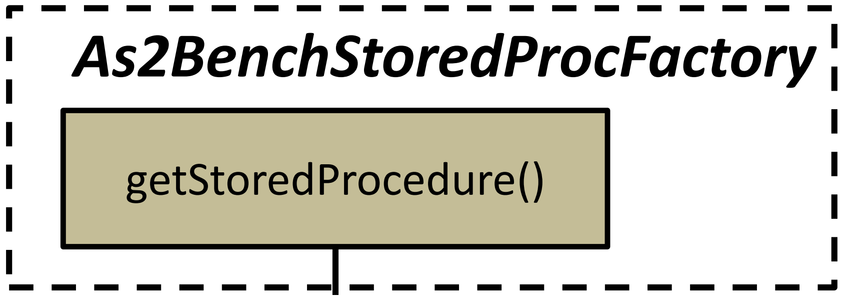
Inquiry via SP



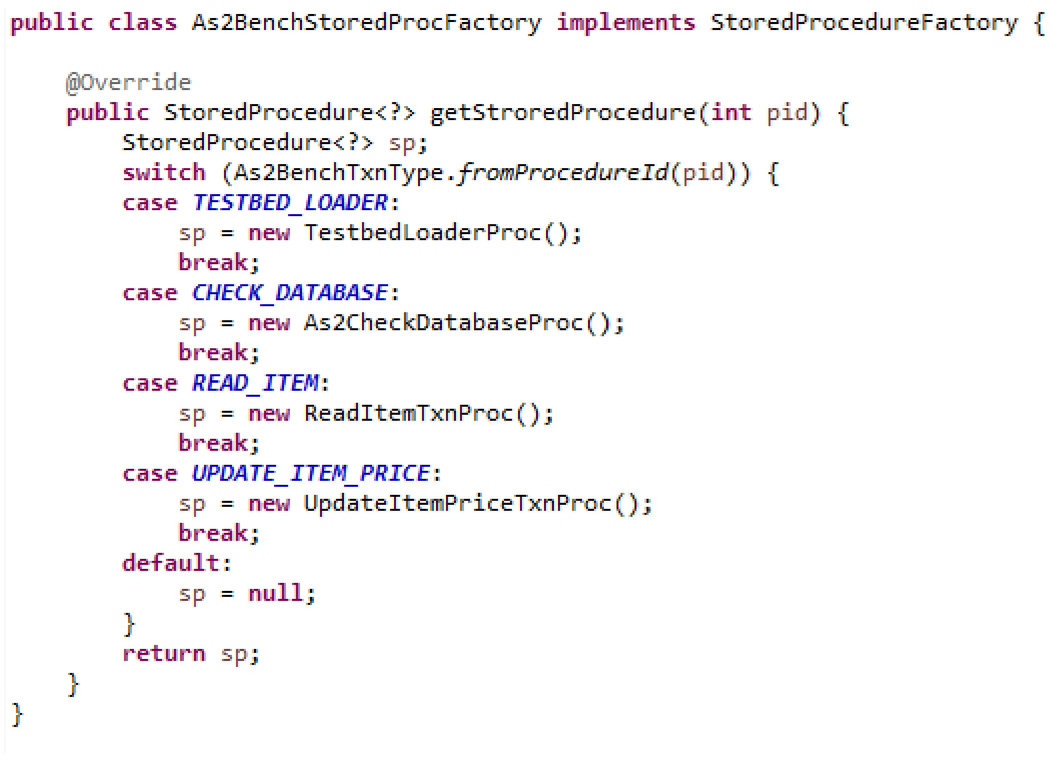
Execute a Stored Procedure



Get the Specified SP

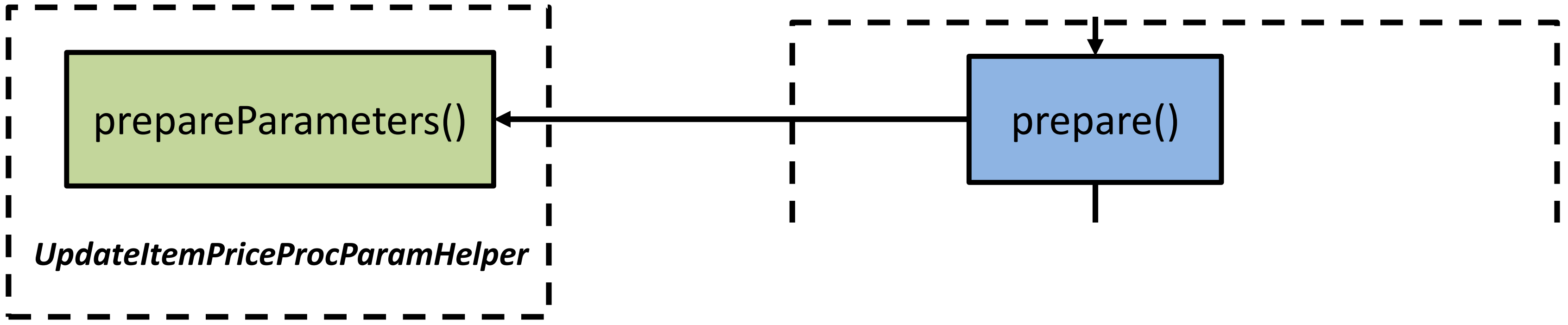


Get the Specified SP

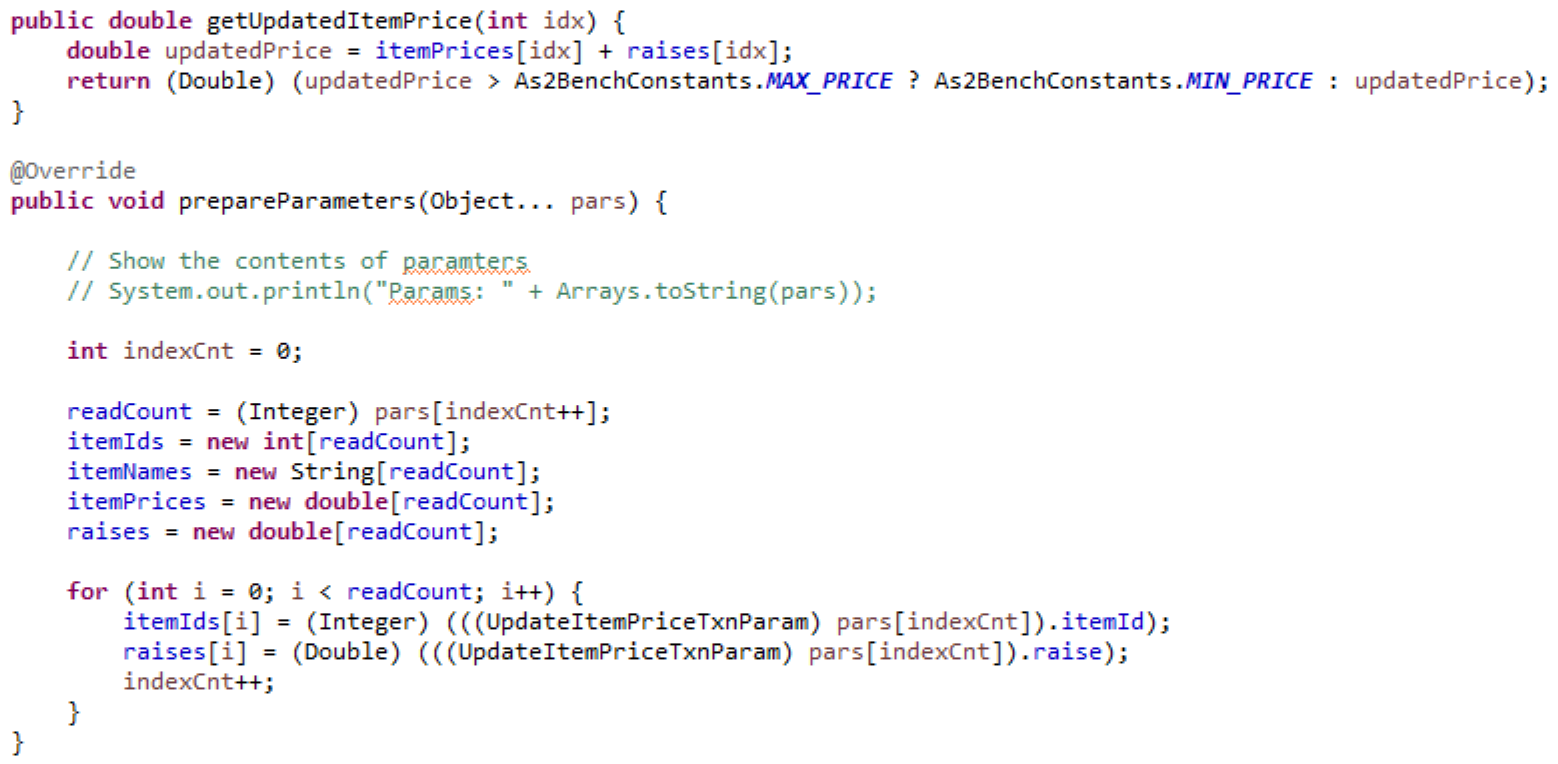


```java
public class As2BenchStoredProcFactory implements StoredProcedureFactory {

    @Override
    public StoredProcedure<?> getStroredProcedure(int pid) {
        StoredProcedure<?> sp;
        switch (As2BenchTxnType.fromProcedureId(pid)) {
        case TESTBED_LOADER:
            sp = new TestbedLoaderProc();
            break;
        case CHECK_DATABASE:
            sp = new As2CheckDatabaseProc();
            break;
        case READ_ITEM:
            sp = new ReadItemTxnProc();
            break;
        case UPDATE_ITEM_PRICE:
            sp = new UpdateItemPriceTxnProc();
            break;
        default:
            sp = null;
        }
        return sp;
    }
}
```

Preprocess Parameters

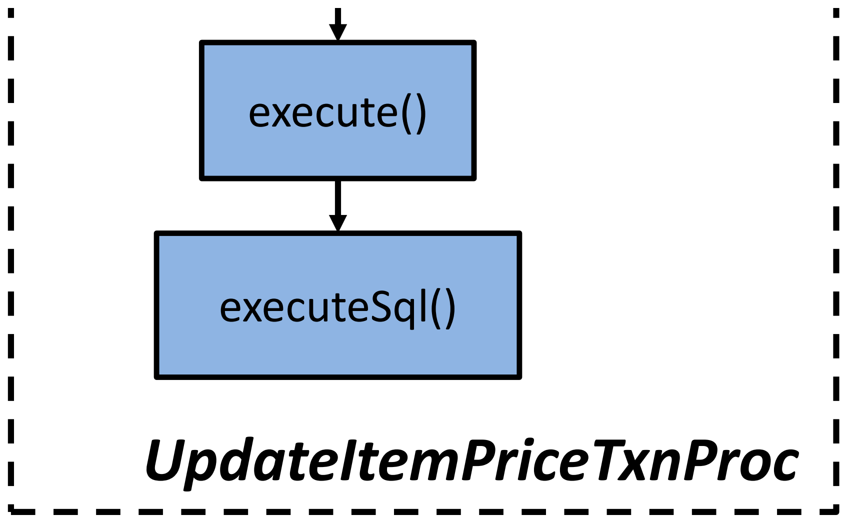


Preprocess Parameters



```
public double getUpdatedItemPrice(int idx) {
    double updatedPrice = itemPrices[idx] + raises[idx];
    return (Double) (updatedPrice > As2BenchConstants.MAX_PRICE ? As2BenchConstants.MIN_PRICE : updatedPrice);
}

@Override
public void prepareParameters(Object... pars) {

    // Show the contents of paramters
    // System.out.println("Params: " + Arrays.toString(pars));

    int indexCnt = 0;

    readCount = (Integer) pars[indexCnt++];
    itemIds = new int[readCount];
    itemNames = new String[readCount];
    itemPrices = new double[readCount];
    raises = new double[readCount];

    for (int i = 0; i < readCount; i++) {
        itemIds[i] = (Integer) (((UpdateItemPriceTxnParam) pars[indexCnt]).itemId);
        raises[i] = (Double) (((UpdateItemPriceTxnParam) pars[indexCnt]).raise);
        indexCnt++;
    }
}
```

Execute Queries

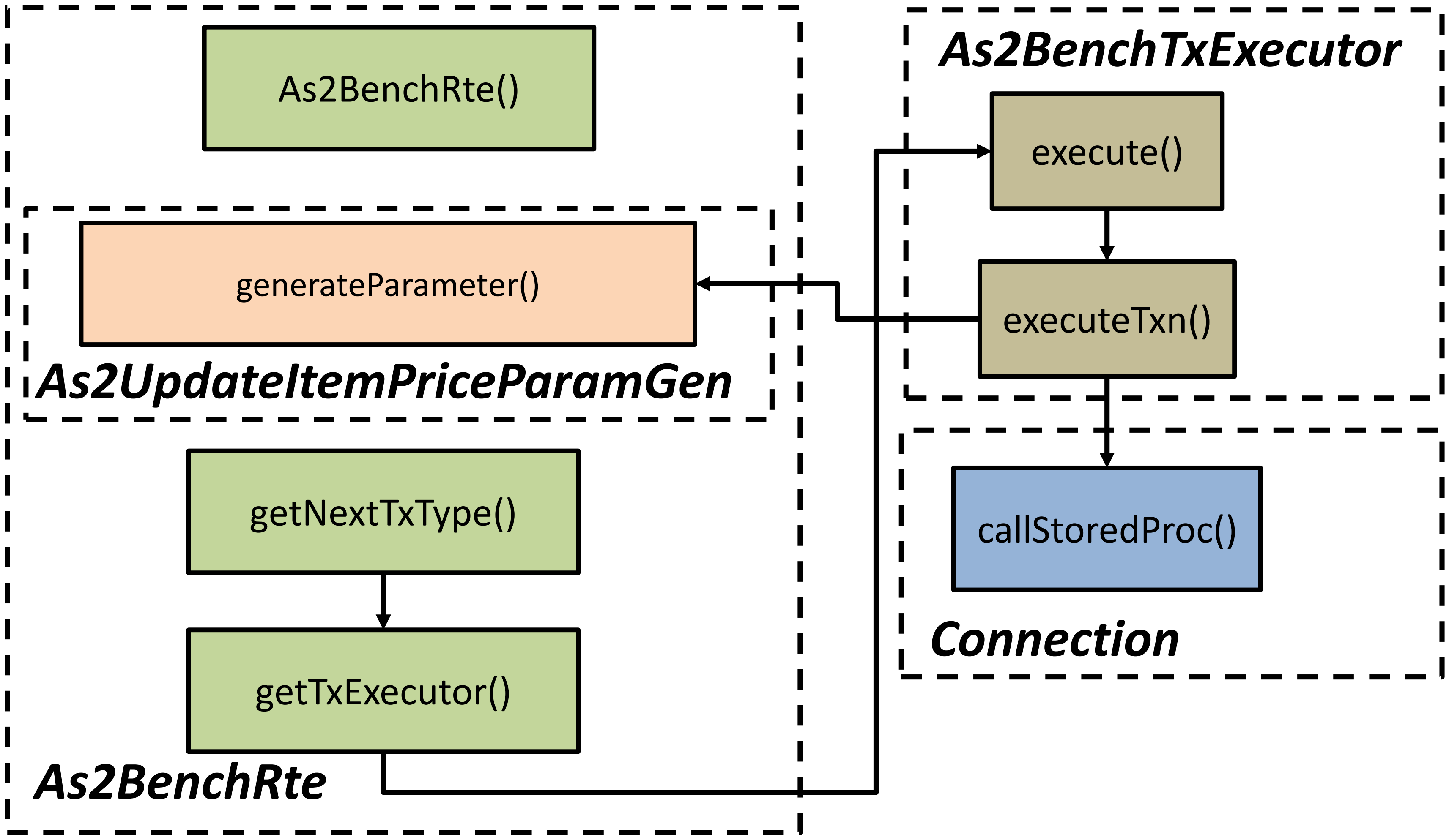


```
@Override
protected void executeSql() {
    UpdateItemPriceProcParamHelper paramHelper = getParamHelper();
    Transaction tx = getTransaction();

    for (int idx = 0; idx < paramHelper.getReadCount(); idx++) {
        int iid = paramHelper.getItemId(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()) {
            String name = (String) s.getVal("i_name").asJavaVal();
            double 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();
        // Update part
        int result = VanillaDb.newPlanner()
                .executeUpdate("UPDATE item SET i_price = " + paramHelper.getUpdatedItemPrice(idx) + " WHERE i_id = " + iid, tx);
        if (result == 0) {
            throw new RuntimeException("Could not update item record with i_id = " + iid);
        }
    }
}
```

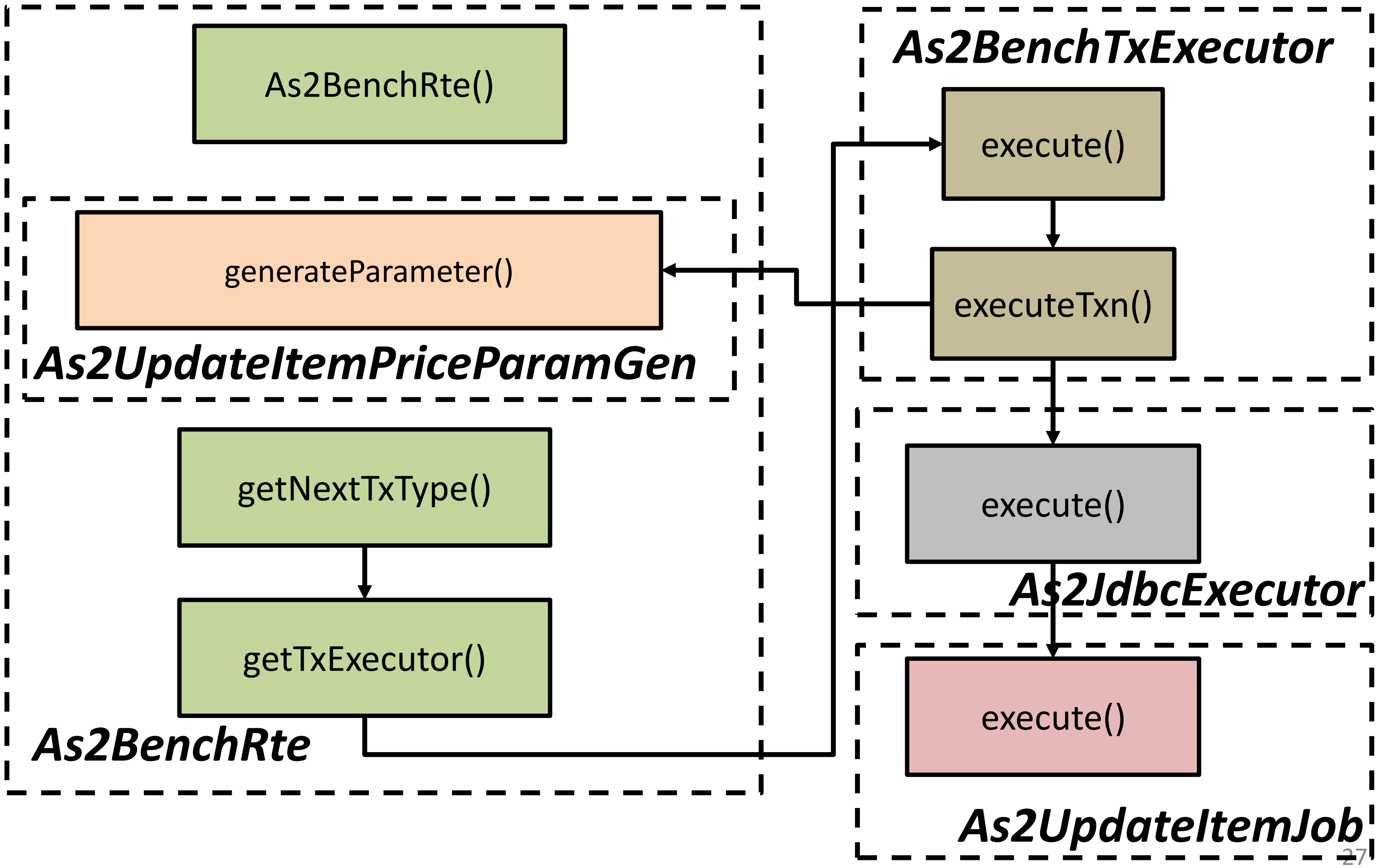
Modified/Added Classes (JDBC)

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

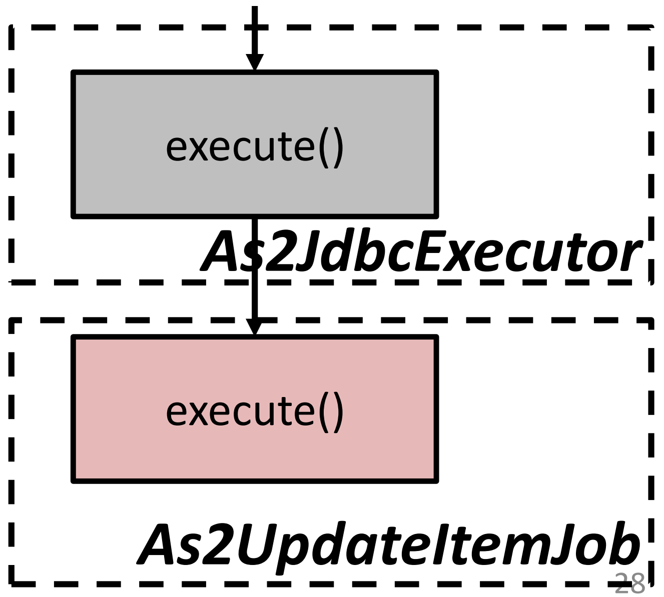
Inquiry via SP



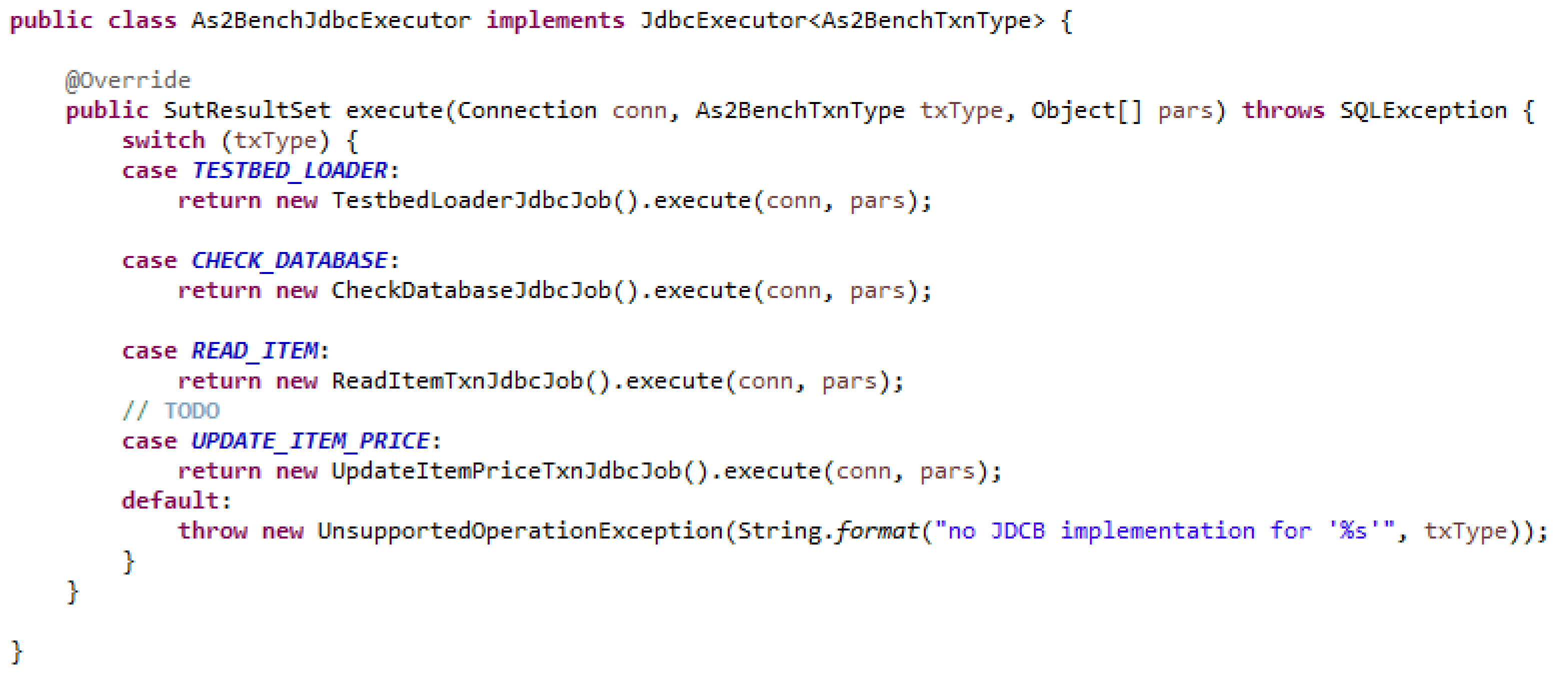
Inquiry via JDBC



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 CHECK_DATABASE:
            return new CheckDatabaseJdbcJob().execute(conn, pars);

        case READ_ITEM:
            return new ReadItemTxnJdbcJob().execute(conn, pars);
        // TODO
        case UPDATE_ITEM_PRICE:
            return new UpdateItemPriceTxnJdbcJob().execute(conn, pars);
        default:
            throw new UnsupportedOperationException(String.format("no JDCB implementation for '%s'", txType));
        }
    }

}
```

```
 @Override
 public SutResultSet execute(Connection conn, Object[] pars) throws SQLException {
     // Parse parameters
     int readCount = (Integer) pars[0];
     int[] itemIds = new int[readCount];
     double[] raises = new double[readCount];

     for (int i = 0; i < readCount; i++) {
         itemIds[i] = (Integer) (((UpdateItemPriceTxnParam) pars[i + 1]).itemId);
         raises[i] = (Double) (((UpdateItemPriceTxnParam) pars[i + 1]).raise);
     }
```

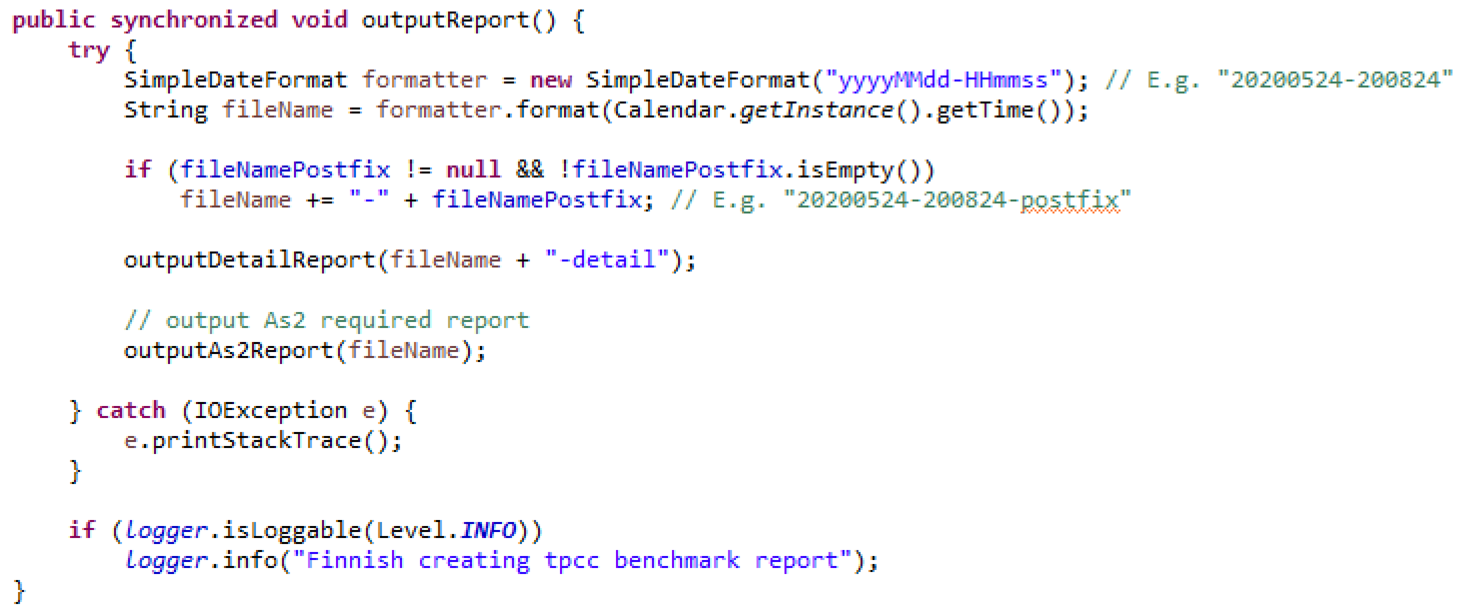
```
Statement statement = conn.createStatement();
ResultSet rs = null;

for (int i = 0; i < 10; i++) {
    double 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();

    Double updatedPrice = updatePrice(price, raises[i]);
    sql = "UPDATE item SET i_price = " + updatedPrice + " WHERE i_id = " + itemIds[i];

    int result = statement.executeUpdate(sql);
    if (result == 0) {
        throw new RuntimeException("cannot update the record with i_id = " + itemIds[i]);
    }

}
conn.commit();
```

Outline

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

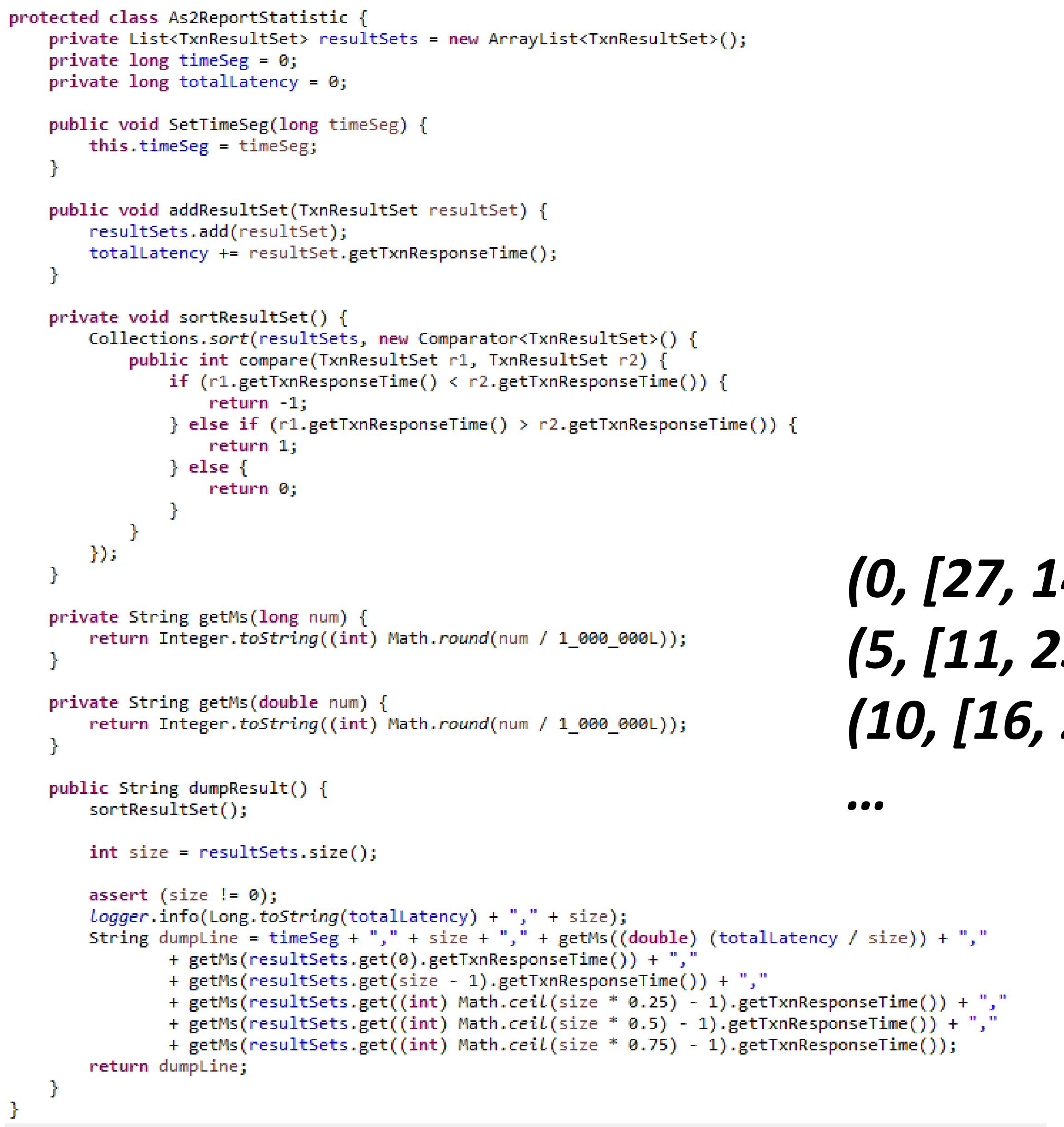
Modified Class

- *StatisticMgr*



```java
public synchronized void outputReport() {
    try {
        SimpleDateFormat formatter = new SimpleDateFormat("yyyyMMdd-HHmmss"); // E.g. "20200524-200824"
        String fileName = formatter.format(Calendar.getInstance().getTime());

        if (fileNamePostfix != null && !fileNamePostfix.isEmpty())
            fileName += "-" + fileNamePostfix; // E.g. "20200524-200824-postfix"

        outputDetailReport(fileName + "-detail");

        // output As2 required report
        outputAs2Report(fileName);

    } catch (IOException e) {
        e.printStackTrace();
    }

    if (logger.isLoggable(Level.INFO))
        logger.info("Finnish creating tpcc benchmark report");
}
```

Add Class



```java
protected class As2ReportStatistic {
    private List<TxnResultSet> resultSets = new ArrayList<TxnResultSet>();
    private long timeSeg = 0;
    private long totalLatency = 0;

    public void SetTimeSeg(long timeSeg) {
        this.timeSeg = timeSeg;
    }

    public void addResultSet(TxnResultSet resultSet) {
        resultSets.add(resultSet);
        totalLatency += resultSet.getTxnResponseTime();
    }

    private void sortResultSet() {
        Collections.sort(resultSets, new Comparator<TxnResultSet>() {
            public int compare(TxnResultSet r1, TxnResultSet r2) {
                if (r1.getTxnResponseTime() < r2.getTxnResponseTime()) {
                    return -1;
                } else if (r1.getTxnResponseTime() > r2.getTxnResponseTime()) {
                    return 1;
                } else {
                    return 0;
                }
            }
        });
    }

    private String getMs(long num) {
        return Integer.toString((int) Math.round(num / 1_000_000L));
    }

    private String getMs(double num) {
        return Integer.toString((int) Math.round(num / 1_000_000L));
    }

    public String dumpResult() {
        sortResultSet();

        int size = resultSets.size();

        assert (size != 0);
        logger.info(Long.toString(totalLatency) + "," + size);
        String dumpLine = timeSeg + "," + size + "," + getMs((double) (totalLatency / size)) + ","
                + getMs(resultSets.get(0).getTxnResponseTime()) + ","
                + getMs(resultSets.get(size - 1).getTxnResponseTime()) + ","
                + getMs(resultSets.get((int) Math.ceil(size * 0.25) - 1).getTxnResponseTime()) + ","
                + getMs(resultSets.get((int) Math.ceil(size * 0.5) - 1).getTxnResponseTime()) + ","
                + getMs(resultSets.get((int) Math.ceil(size * 0.75) - 1).getTxnResponseTime());
        return dumpLine;
    }
}
```

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

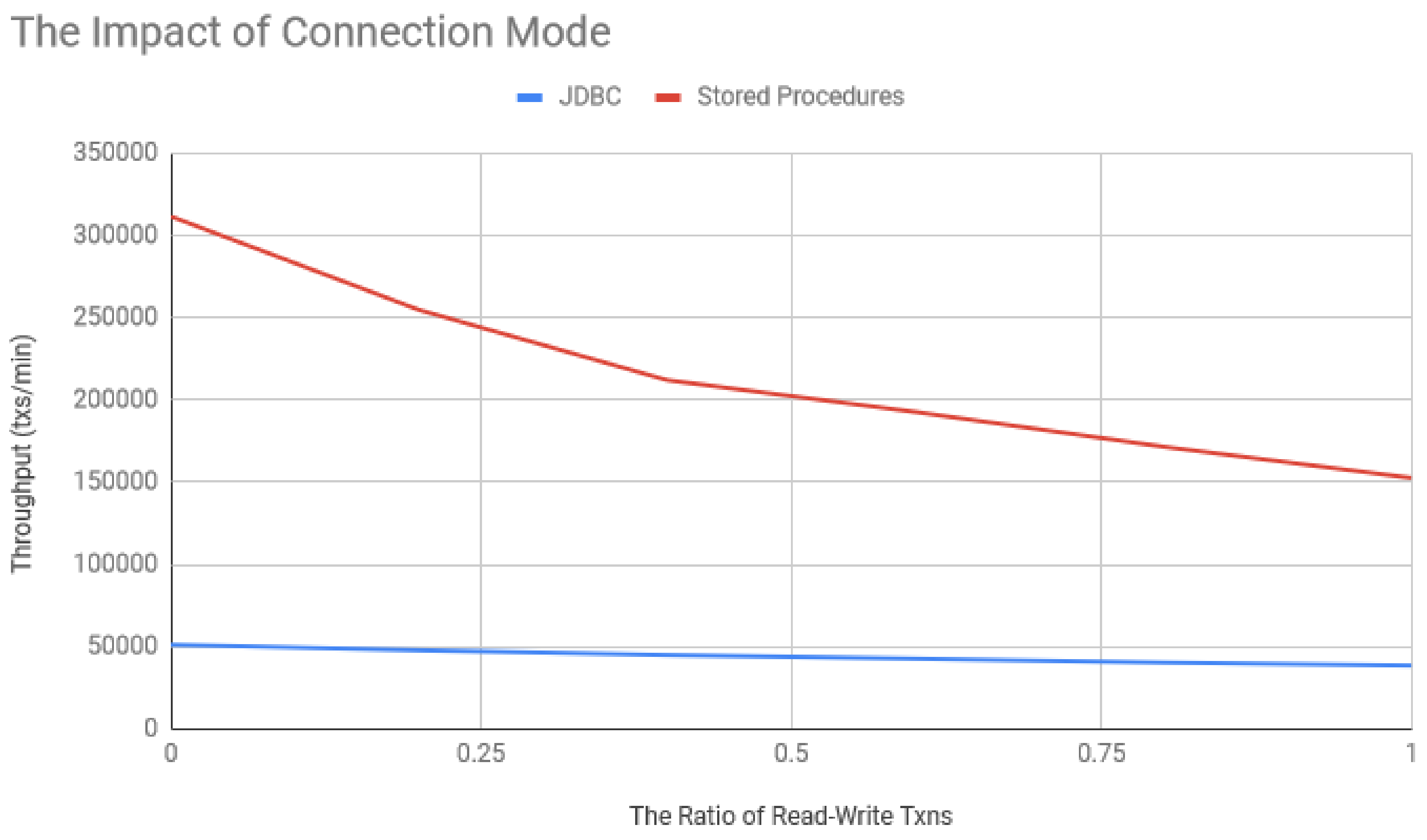
Add Method

```java
private void outputAs2Report(String fileName) throws IOException {
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(new File(OUTPUT_DIR, fileName + ".csv")))) {
        writer.write(
                "time(sec), throughput(txs), avg_latency(ms), min(ms), max(ms), 25th_lat(ms), median_lat(ms), 75th_lat(ms)");
        writer.newLine();
        long timeStart = 0;
        long timeSeg = 5;
        boolean segFirst = true;
        As2ReportStatistic as2St = new As2ReportStatistic();

        for (TxnResultSet resultSet : resultSets) {
            if (segFirst) {
                timeStart = resultSet.getTxnEndTime();
                as2St.SetTimeSeg(timeSeg);
                segFirst = false;
                timeSeg += 5;
            }
            as2St.addResultSet(resultSet);
            if (!(resultSet.getTxnEndTime() < (timeStart + 5_000_000_000L))) {
                writer.write(as2St.dumpResult());
                writer.newLine();
                as2St = new As2ReportStatistic();
                segFirst = true;
            }
        }
    }
}
```

Outline

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

An Example of Experiments


The Impact of Connection Mode
JDBC
Stored Procedures
Throughput (txs/min)
The Ratio of Read-Write Txns