Review Score Calculator 1.0 Component Specification

1. Design

Given a scorecard and a review, the component will be able to evaluate the review answers and calculate the overall score. Different question types will have different mechanisms to resolve the answer into scores. Simple caching strategy is provided so that calculators don't need to be created for the same scorecard.
Pluggable mechanisms to resolve question answers to scores are represented by ScoreCalculator entities. This component release supports only String-represented answers. In particular, the following question types are supported:

1. Yes/No questions. The positive question answer (Yes) will be scored as 1.0 and the negative question answer will be scored 0.0.
2. Fixed scale questions. Each question answer will be represented by an integer value in the range [1, Scale]. The answer will be scored as Value/Scale. Fixed scale questions can be used to represent standard scorecard items like scale 4 and scale 10 questions.
3. Dynamic scale questions. Each question answer will be represented as an integer value + "/" + the scale value. The answer will be scored as Value/Scale. Dynamic scale questions can be used to represent TestCase items where the question is scored as the number of passed test cases divided by the total number of test cases.

1.1 Design Patterns

- Each ScoreCalculator and ScorecardMatrixBuilder is a pluggable **Strategy**.
- CalculationManager provides a simple **Façade** to the component.
- AbstractScoreCalculator provides **Template** for evaluation of weighted answers represented in String format.
- ScorecardMatrixBuilder is a **Factory** for ScorecardMatrix objects.

1.2 Industry Standards

None

1.3 Required Algorithms

There are two simple algorithms that should be provided. Note that algorithms are described in pseudo-code and all exception handling was omitted. Refer to the method docs for more details.

1.3.1 Constructing ScorecardMatrix for a Scorecard (DefaultScorecardMatrixBuilder)

This algorithm will create a MathMatrix implementation for the given Scorecard. Both Scorecard and MathMatrix represent weighted matrix entities with similar APIs so the conversion is straightforward:

```
// max scorecard weight is set to 100
MathMatrix m = new MathMatrix(scorecard.getName(), 100);

ScorecardMatrix sm = new ScorecardMatrix(sm);

// note that the normalization might be needed
// if group weights don't sum to one
double totGroupW = 0;
Foreach Group g in scorecard.getAllGroups()
    totGroupW += g.getWeight();

// iterate over all scorecard groups and represent each as a section
Foreach Group g in scorecard.getAllGroups()
{
    // we should use fully qualified names because of naming conflict
    // between weighted calculator Section and scorecard Section
    com.topcoder.util.weightedcalculator.Section groupSection = new
        com.topcoder.util.weightedcalculator.Section(g.getWeight()/totGroupW);

    // add group section to the weighted calculator
    m.addItem(groupSection);

    // note that the normalization might be needed
    // if section weights don't sum to one
    double totSectW = 0;
    Foreach com.topcoder.management.scorecard.data.Section s in g.getAllSections()
        totSectW += s.getWeight();

    // iterate over all subsections and represent each as a section
    Foreach com.topcoder.management.scorecard.data.Section s in g.getAllSections()
    {
        com.topcoder.util.weightedcalculator.Section sectionSection = new
            com.topcoder.util.weightedcalculator.Section(s.getWeight()/totSectW);

        // add subsection to the group section
        groupSection.addItem(sectionSection);

        // note that the normalization might be needed
        // if question weights don't sum to one
        double totQuestW = 0;
        Foreach Question q in s.getAllQuestions()
            totQuestW += q.getWeight();


        // finally iterate over all questions
        Foreach Question q in s.getAllQuestions()
        {

             // create line item for the question
             // we use 1.0 as the max un-weighted question score
             LineItem li = new LineItem(q.getDecsription(),
                                        q.getWeight()/totQuestW, 1.0);


             // add line item to the subsection
             sectionSection.addItem(li);

             // add question mapping entry to the scorecard matrix
             sm.addEntry(q.getId(),li,q);
         }
    }
}
```

1.3.2 *Evaluating a review*

This algorithm will evaluate the given review against its scorecard. It assumes that the ScorecardMatrix for the scorecard was already constructed (see the previous item). All that remains is to process all review items according to the configured ScoreCalculators, fill the MathMatrix appropriately and use it to calculate the score. The algorithm to do this is given below.

```
// iterate over all reviewed items
Foreach Item item in review.getAllItems()
{
    // get question object
    Question q = scorecardMatrix.getQuestion(item.getQuestion());

    // get question type
    Long questionType = new Long(q.getQuestionType().getId());

    // get the calculator to apply
    ScoreCalculator calculator = (ScoreCalculator)calculators.get(questionType);

    // evaluate the item
    float score = calculator.evaluateItem(item, q);

    // set the line item score
    scorecardMatrix.getLineItem(item.getQuestion()).setActualScore(score);
}

// get the math matrix
MathMatrix m = scorecardMatrix.getMathMatrix();

// evaluate it to get the final score
float finalScore = m.getWeightedScore();
```

1.4 Component Class Overview

CalculationManager:
This class provides a simple facade to the component. It provides API to evaluate the given review item against the given scorecard structure. To do this it manages a set of score calculators to use for review evaluation. One score calculator will be registered for each used question type occurring in the scorecard. For performance reasons CalculationManager may (optionally) cache intermediate results (ScorecardMatrix objects representing the result of scorecard conversion to the MathMatrix compatible format).

ScoreCalculator (interface):
This interface represents a simple score calculator. It will evaluate the given review item against the given question and return a score in the range 0..item.getWeight().

AbstractScoreCalculator (abstract):
This class provides simple template for evaluation of review items. It replaces evaluateItem method by a simpler evaluateAnswer which introduces additional limitation for representing all answers as strings. The class also ensures appropriate weighting of the score.

ScaledScoreCalculator:
Represents a simple score calculator for scaled items i.e. items accepting integer answers scaled in the preconfigured range. The answer score will be determined as answerValue/scaleValue. The scale will always be positive and it is configurable. The answer will always be formatted as a string value: "value" or "value/scale". Note that in the second case, the preconfigured scale will be replaced by the scale provided inside the answer string.

BinaryScoreCalculator:
Represents a simple score calculator for binary items i.e. items accepting two answers (for example, Yes/No). One of these items will always be scored as 1.0 (positiveAnswer) while another one will be scored as 0.0 (negativeAnswer). Positive and negative answers are configurable, the default settings are "Yes" and "No". Note that the comparison is case sensitive.

ScorecardMatrix:
This class is used as a bridge object between Scorecard and its MathMatrix representation used for score calculations. It keeps a reference to the weighted calculator to use. It can also be used to quickly locate particular question and it’s LineItem by question identifier. Instances of this class may optionally be cached by CalculationManager.

ScorecardMatrixBuilder (interface):
This interface represents a builder that can construct a ScorecardMatrix representation for the given Scorecard.

DefaultScorecardMatrixBuilder:
This class represents the default scorecard matrix builder. It will create a weighted calculator with

one section for each scorecard group, one subsection for each scorecard section in the group and one LineItem for each question in the scorecard section.

1.5 Component Exception Definitions

All exceptions thrown by the component are listed below. For the details on exceptions thrown by a particular method refer to the method documentation.

java.lang.IllegalArgumentException:
This exception will be thrown when a null reference is passed to a method that does not accept it as a valid argument.

com.topcoder.management.review.scorecalculator.CalculationException:
This is the base component exception. All custom exceptions defined by the component will subclass it. It is also used as the wrapper for all "foreign" exceptions, for example, caught from the weighted calculator.

com.topcoder.management.review.scorecalculator.ConfigurationException:
This exception will be thrown to indicate all static configuration related issues (missing namespace, missing required property, invalid property value etc)

com.topcoder.management.review.scorecalculator.ScorecardStructureException:
This exception will be thrown to indicate that the scorecard is incorrect/inconsistent and can't be processed.

com.topcoder.management.review.scorecalculator.ReviewStructureException:
This exception will be thrown to indicate that the review is incorrect/inconsistent with the scorecard and can't be processed.

com.topcoder.management.review.scorecalculator.ScoreCalculatorException:
This exception will be thrown by ScoreCalculator to indicate any problem while processing a review item (for example, unrecognized review answer).

1.6 Thread Safety

The component is required to be thread-safe. All component classes except for ScorecardMatrix and CalculationManager are immutable and therefore thread-safe. ScorecardMatrix provides inherently not thread-safe API (exposes MathMatrix reference for modifications), therefore all modifications to instances of this class (and the underlying weighted calculator) should be performed under the lock on the class instance. Fortunately, the application will never need to modify ScorecardMatrix directly but only inside the CalculationManager.getScore() method which will perform appropriate locking. CalculationManager itself is mutable however its thread-safety is based on the following three reasons:

- It uses thread-safe hashtable to keep calculators.
- It uses Simple Cache component which is thread-safe.
- The class will lock on the ScorecardMatrix instance during score evaluation.

2. Environment Requirements

2.1 Environment

Java 1.4

2.2 TopCoder Software Components

- **Java Configuration Manager** 2.1.4 is used to load static component configuration
- **Java Scorecard Management** is used to manage scorecard data
- **Java Review Management** is used to manage review data
- **Java Simple Cache** 2.0 is used to cache weighted calculator data for scorecards
- **Java Weighted Calculator** 1.0 is used to perform score calculations
- **Java Base Exception 1.0** is used to support custom exceptions

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3 Third Party Components

None.

3. Installation and Configuration

3.1 Package Name

com.topcoder.management.review.scorecalculator – main package
com.topcoder.management.review.scorecalculator.calculators – implementations of the score calculation logic
com.topcoder.management.review.scorecalculator.builders – implementations of scorecard processing logic (building the weighted calculator)

3.2 Configuration Parameters

This section gives a simple example of the component configuration.

```
<?xml version="1.0" encoding="UTF-8"?>
<ConfigManager>
<!— The default namespace for configuration of the CalculationManager -->
<namespace name="com.topcoder.management.review.scorecalculator.CalculationManager">

<!— This property defines whether the caching should be used -->
<property name="use_caching">
    <value>true</value>
</property>

<!— This property defines the ScorecardMatrixBuilder class to use -->
<property name="builder_class">
    <value>
        com.topcoder.management.review.scorecalculator.builders.DefaultScorecardMatrixBuilder
    </value>
</property>


<!— This property defines configured calculators -->
<property name="calculators">
```

```
    <!—This is the configuration for the yes/no question type-->
    <property name="binaryCalculator">
        <!—This sub-property describes the question type id -->
        <property name="question_type">
            <value>1</value>
        </property>

        <!—This sub-property describes the class name for the calculator-->
        <property name="class">
            <value>
                 com.topcoder.management.review.scorecalculator.calculators.BinaryScoreCalculator
            </value>
        </property>
    </property>

    <!—This is the configuration for the scale 4 question type-->
    <property name="scale4Calculator">
        <!—This sub-property describes the question type id -->
        <property name="question_type">
            <value>2</value>
        </property>

        <!—This sub-property describes the class name for the calculator-->
        <property name="class">
            <value>
                 com.topcoder.management.review.scorecalculator.calculators.ScaledScoreCalculator
            </value>
        </property>

        <!—This sub-property describes the namespace to use for the calculator-->
        <property name="namespace">
            <value>
                 com.topcoder.management.review.scorecalculator.calculators.ScaledScoreCalculator1
            </value>
        </property>
    </property>

    <!—This is the configuration for the scale 10 question type-->
    <property name="scale10Calculator">
        <!—This sub-property describes the question type id -->
        <property name="question_type">
            <value>3</value>
        </property>

        <!—This sub-property describes the class name for the calculator-->
        <property name="class">
            <value>
                 com.topcoder.management.review.scorecalculator.calculators.ScaledScoreCalculator
            </value>
        </property>

        <!—This sub-property describes the namespace to use for the calculator-->
        <property name="namespace">
            <value>
                 com.topcoder.management.review.scorecalculator.calculators.ScaledScoreCalculator2
            </value>
        </property>
    </property>

    <!—This is the configuration for the dynamic question type-->
    <property name="scale10Calculator">
        <!—This sub-property describes the question type id -->
        <property name="question_type">
            <value>4</value>
        </property>

        <!—This sub-property describes the class name for the calculator-->
        <property name="class">
            <value>
                 com.topcoder.management.review.scorecalculator.calculators.ScaledScoreCalculator
            </value>
        </property>
   </property>
</property>
```

```
</namespace>


<!— This namespace contains additional configuration for scale 4 calculator -->
<namespace name="com.topcoder.management.review.scorecalculator.calculators.ScaledScoreCalculator1">

<!—This property defines the scale to use-->
<property name="default_scale">
    <value>4</value>
</property>

</namespace>

<!— This namespace contains additional configuration for scale 10 calculator -->
<namespace name="com.topcoder.management.review.scorecalculator.calculators.ScaledScoreCalculator1">

<!—This property defines the scale to use-->
<property name="default_scale">
    <value>10</value>
</property>

</namespace>

</ConfigManager>
```

3.3 Dependencies Configuration

None.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Execute ‘ant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

Provide a valid configuration file. The configuration file is not required when using a programming API to configure CalculationManager.

4.3 Demo

The following demo should be used with the configuration file provided in the section 3.2
Assume that the following simple scorecard is given:

Group	Group Weight	Section	Section Weight	Question	Question Type	Question Weight
Design	0.3	A	0.4	1	1	0.5
				2	1	0.5
		B	0.6	3	2	1.0
Documentation	0.7	C	1.0	4	4	0.4
				5	3	0.3
				6	3	0.3

It has two groups (Design and Documentation). The first group has two sections (A and B), the second group has one section (C). Section A has two question of type 1 (yes/no), section B has one question of type 2 (scale 4), section C has one question of type 4 (dynamic scale) and two questions of type 3 (scale 10). Weights for all groups, sections and questions are given in the table.

Assume that the following review answers are given:

1. Question 1: ‘Yes’.
2. Question 2: ‘No’.
3. Question 3: ‘2’.
4. Question 4: ‘60/100’.
5. Question 5: ‘3’.
6. Question 6: ’10’.

To calculate the score one should simply create CalculationManager and call getScore() method.

```
CalculationManager m = new CalculationManager();

System.out.println(“Review score is: “ + m.getScore(review,scorecard));
```

The method will print the following score:
100 * (0.3*0.4*0.5*1.0 + 0.3*0.4*0.5*0.0 + 0.3*0.6*1.0*0.5 + 0.7*1.0*0.4*0.6 + 0.7*1.0*0.3*0.3 + 0.7*1.0*0.3*1.0) i.e. **59.1**

The component provides API to manage score calculators.

```
// for example, we can remove a score calculator for type 1 questions (yes/no)
m.removeScoreCalculator(1);

// now we can create a new score calculator with negative and positive
// answers swapped
ScoreCalculator acceptNo = new BinaryScoreCalculator(“No”, “Yes”);

// set it for usage
m.addScoreCalculator(1, acceptNo);

// in our case the score will not change because questions 1 and 2
// have the same cost
System.out.println(“Review score is: “ + m.getScore(review,scorecard));
```

5. Future Enhancements

Additional question types can be supported, including questions with non-string answers.