Review Payment Calculator 1.0 Component Specification

1. Design

TopCoder attempts to improve the competition review process and is in need for updating current systems for integrating with new review process outlined in separate Competition Review Process Conceptualization document. Currently there are two applications involved in review process. TopCoder Website application is used by the reviewers for signing up for the review positions and listing the statistics for projects and users. Online Review application is actually used for managing the review process providing the users with abilities to upload submissions for contests, fill the screening and review scorecards, submitting and resolving appeals, performing contest results aggregation, submitting final fixes and approving the results of contests.

This component provides interfaces and implementations for calculating the review payments based on some computed statistics.

1.1 Design Patterns

Strategy pattern – ReviewPaymentCalculator together with its implementation can be used in some external strategy context.

1.2 Industry Standards

None

1.3 Required Algorithms

1.3.1 Logging

This component must perform logging in computeReviewPayment() method of DefaultReviewPaymentCalculator.

All information must be logged using log:Log attribute. If this attribute is equal to null, then logging is not required to be performed.

In the mentioned method entrance with input arguments, method exit with return value and call duration time must be logged at DEBUG level. It’s not required to log method exit when method throws an exception.

All errors (for all thrown exceptions) must be logged at ERROR level with exception message and stack trace.

1.4 Component Class Overview

DefaultReviewPaymentCalculator

This class is an implementation of ReviewPaymentCalculator that uses the following logic when calculating the payment amounts: 1) payment amount for the primary reviewer is calculated as (base primary reviewer payment * timeline reliability coefficient of this reviewer); 2) secondary reviewer with the highest total evaluation coefficient receives a bonus equal to the configurable part of the secondary payment pool (if secondary reviewers have equal total evaluation coefficients, the bonus is distributed equally); 3) the rest of the secondary payment pool is distributed among secondary reviewers proportionally to their total evaluation coefficients.

ReviewPaymentCalculator [interface]

This interface represents a review payment calculator. It provides a method for calculation payments of primary and secondary reviewers for specific project based on base primary payment amount, secondary payment pool and reviewer statistics for this contest. Additionally it provides configure() method to support consistent configuration mechanism with use Configuration API component among all the implementations.

ReviewerPayment

This class is a container for information about a single reviewer payment. It holds reviewer ID, project ID and payment amount. It is a simple JavaBean (POJO) that provides getters and setters for all private attributes and performs no argument validation in the setters.

1.5 Component Exception Definitions

InvalidReviewersStatisticsException

This exception is thrown by implementations of ReviewPaymentCalculator when the provided reviewer statistics is invalid (e.g. number of stats for primary reviewer is not equal to 1, coefficient values are out of the expected range, etc).

ReviewPaymentCalculatorConfigurationException

This exception is thrown by implementations of ReviewPaymentCalculator when some error occurs while initializing an instance using the given configuration.

ReviewPaymentCalculatorException

This exception is thrown by implementations of ReviewPaymentCalculator when some unexpected error occurs. Also this exception is used as a base class for other specific custom exceptions.

1.6 Thread Safety

This component is thread safe.

Implementations of ReviewPaymentCalculator must be thread safe. It's assumed that configure() method will be called just once right after instantiation, before calling computeReviewPayment().

ReviewerPayment is mutable and not thread safe entity; it's used in thread safe manner in this component.

DefaultReviewPaymentCalculator is immutable and thread safe assuming that configure() method will be called just once right after instantiation, before calling computeReviewPayment() method. The Log instance used by this class is thread safe.

2. Environment Requirements

2.1 Environment

Development language: Java1.5
Compile target: Java1.5
QA Environment: Java 1.5, Solaris 7, RedHat Linux 7.1, Windows 2000, Windows 2003, Oracle, JBoss 4.2.2

2.2 TopCoder Software Components

Base Exception 2.0 – is used by custom exceptions defined in this component.

Logging Wrapper 1.2 – is used for logging errors and debug information in this component.

Configuration API 1.0 – is used for providing configuration for this component.

Resource Management 1.3 – defines ReviewerStatistics entity used in this component.

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.cronos.onlinereview.review.payment

com.cronos.onlinereview.review.payment.impl

3.2 Configuration Parameters

3.2.1 Configuration of DefaultReviewPaymentCalculator

The following table describes the structure of ConfigurationObject passed to the configure() method of DefaultReviewPaymentCalculator class (angle brackets are used for identifying child configuration objects).

Parameter	Description	Values
loggerName	The logger name passed to LogFactory when creating a Log instance to be used by this class. When not provided, logging is not performed.	String. Not empty. Optional.
bonusPercentage	The percentage of the secondary payment pool that is used for bonus awarded to a secondary reviewer with the highest total evaluation coefficient (must be in the range [0 .. 1]). Default is “0.2” (means that bonus equals to 20% of the secondary payment pool).	String representation of double in the range [0 .. 1]. Optional.

3.3 Dependencies Configuration

Please see docs of Logging Wrapper component to configure it properly.

4. Usage Notes

4.1 Required steps to test the component

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

4.2 Required steps to use the component

Please see the demo.

4.3 Demo

4.3.1 Sample DefaultReviewPaymentCalculator configuration

This XML configuration can be read to ConfigurationObject instance using Configuration Persistence component. Please see its docs for details.

```
<?xml version="1.0"?>
<CMConfig>
  <Config name="com.cronos.onlinereview.review.payment.impl.DefaultReviewPaymentCalculator">
    <Property name="loggerName">
      <Value>myLogger</Value>
    </Property>
    <Property name="bonusPercentage">
      <Value>0.2</Value>
    </Property>
  </Config>
</CMConfig>
```

4.3.2 API usage and sample input/output

```
// Create an instance of reviewPaymentCalculator
ReviewPaymentCalculator reviewPaymentCalculator = new DefaultReviewPaymentCalculator();

// Get configuration for DefaultReviewPaymentCalculator
```

```
ConfigurationObject config = ...

// Configure review payment calculator
reviewPaymentCalculator.configure(config);

// Prepare review statistics data
ReviewerStatistics[] statistics = new ReviewerStatistics[3];
ReviewerStatistics primaryReviewerStats = new ReviewerStatistics();
primaryReviewerStats.setProjectId(1);
primaryReviewerStats.setReviewerId(1001);
primaryReviewerStats.setTimelineReliability(0.85);
statistics[0] = primaryReviewerStats;
ReviewerStatistics firstSecondaryReviewerStats = new ReviewerStatistics();
firstSecondaryReviewerStats.setProjectId(1);
firstSecondaryReviewerStats.setReviewerId(1002);
firstSecondaryReviewerStats.setTimelineReliability(1);
firstSecondaryReviewerStats.setCoverage(0.61);
firstSecondaryReviewerStats.setAccuracy(0.95);
statistics[1] = firstSecondaryReviewerStats;
ReviewerStatistics secondSecondaryReviewerStats = new ReviewerStatistics();
secondSecondaryReviewerStats.setProjectId(1);
secondSecondaryReviewerStats.setReviewerId(1003);
secondSecondaryReviewerStats.setTimelineReliability(0.74);
secondSecondaryReviewerStats.setCoverage(0.39);
secondSecondaryReviewerStats.setAccuracy(1);
statistics[2] = secondSecondaryReviewerStats;

// Compute review payment for project with ID=1
// Use $120 as a base primary payment
// Use $200 as a secondary payment pool
ReviewerPayment[] reviewerPayments =
    reviewPaymentCalculator.computeReviewPayment(1, 120, 200, statistics);
// reviewerPayments.length must be 3
// reviewerPayments[0].getProjectId() must be 1
// reviewerPayments[0].getReviewerId() must be 1001
// reviewerPayments[0].getPaymentAmount() must be 102
// reviewerPayments[1].getProjectId() must be 1
// reviewerPayments[1].getReviewerId() must be 1002
// reviewerPayments[1].getPaymentAmount() must be 142.78775...
// reviewerPayments[2].getProjectId() must be 1
// reviewerPayments[2].getReviewerId() must be 1003
// reviewerPayments[2].getPaymentAmount() must be 57.21224...
```

The performed calculations are described below.

For the primary review the payment is calculated as $120 * 0.85 = $102.
For secondary reviewers first total evaluation coefficients are calculated:

First secondary reviewer:	1 * 0.61 * 0.85 = 0.5185.
Second secondary reviewer:	0.74 * 0.39 * 1 = 0.2886.

The first secondary reviewer who has a higher total evaluation coefficient obtains a bonus payment that is equal to $200 * 0.2 = $40 (according to the provided sample configuration).

The rest of the payment pool ($200 - $40 = $160) is distributed between secondary reviewers:

First secondary reviewer:	$160 * 0.5185 / (0.5185 + 0.2886) = $102.79 (approx.).
Second secondary reviewer:	$160 * 0.2886 / (0.5185 + 0.2886) = $57.21 (approx.).

The total payment for the first secondary reviewer will be $40 + $102.79 = $142.79.

5. Future Enhancements

None