**Checkout Application Design Document**

**Introduction**

This document describes the high level design of the Checkout software application. Application developers should follow the design outlined in this document in conjunction with the details contained in the functional specification document to implement Checkout.

This design divides the implementation of Checkout into components, with each component focused on one part of the overall functionality. Components interact with each other to perform the overall task of generating a Rental Agreement for the tool being rented. Each component is described in the Components section below.

While Checkout is a simple application created for demonstration purposes, it is possible that it will be developed further. Therefore, to the extent reasonable, and allowed by the schedule, components should be implemented to not need refactoring if the application evolves in the future.

**Components**

**Application Log File**

Use log4j or similar logging library to create and maintain a log file for the application. The log file will only be used to store error messages and stack traces to aid in diagnosing issues with the application. Normal application execution does not need to be logged. The log file should be rolling file(s), never consuming more than 10MB of storage space.

**Tool and Charge Info Data Files**

The Tool Info and Rental Charge Info defined in the tables in the Tools section of the functional specification will be stored in separate CSV files. While XML and JSON are more typical data formats, CSV can be less intimidating to non-technical users. It is editable using Excel, or other spreadsheet applications, and there are free Java libraries such as Apache Commons CSV to assist in writing code to process them.

**Properties File**

Configurable parameters used by Checkout will be stored in a properties file. Values used in the application that may change over time, and should not require a recompilation of the application to change them, should be included as a property in this file.

Possible properties include the format that dates are entered and displayed in, and the names of the files holding the tool and charge information.

**Error Handling**

This component is distributed among most of the other components, rather than being self contained like all other components are. All components, other than Main, should handle errors by throwing an exception that is passed up to Main. The purpose of this is to confine all dependencies on the fact that this application is a command line application to the Main component. Main will handle the exception by printing a message to the command shell, or the Application Log File, as appropriate.

**Main**

The Checkout class will have the public static method main that is the starting point for this command line application. Main should:

1) Call the Properties File Reader component.

2) Parse and error check command line arguments. The required arguments and valid values for them are defined in the Technical Specification document.

Additionally, while the rental date can be in the past or the future, it seems likely that the rental date will be the current date the vast majority of the time. Therefore, there should be a shortcut of some type to set the rental date as the current date. If that shortcut isn’t used, then the rental date should be entered in the format specified in the Properties File.

The parser will also support a --help, or similar argument that will display a brief description of what this application does, and list all of the supported command line arguments along with a brief description of each one.

3) Call the RentalAgreement component to generate the rental agreement for the tool that is being rented, and then print the finished rental agreement as described in the functional specification.

4) Handle exceptions. Errors that the user could possibly correct, such as entering an invalid value on the command line, should be printed to the command shell with a description of the error and how to correct it. Errors that probably require a software developer to diagnose and correct should write the error and stack trace to the application log file, and display a short message to the user telling them to contact technical support.

**Rental Agreement**

Rental Agreement will take the input values provided by the user, and go through all of the steps necessary to produce an accurate and complete rental agreement for the tool being rented, as described in the functional specification.

Rental Agreement will call the following components as part of generating the rental agreement:

Get Tool Info

Get Tool Charge Info

Is Holiday

Round Up To Cents

This component will also have a method to print the completed Rental Agreement to the command shell, as described in the functional specification.

**Properties File Reader**

This component encapsulates all of the code required to read the Properties File and allow access to the value of each property.

This component should gracefully handle missing, incomplete, or invalid properties files by displaying a user friendly message with information on how the file might be repaired.

**Get Tool Info**

Given a Tool Code, this component will return a fully populated Tool object that contains the Tool Code, Tool Type, and Brand, as defined in the functional specification.

This component should gracefully handle missing, incomplete, or invalid data files by displaying a user friendly message with some information on how the file might be repaired.

**Get Tool Charge Info**

Given a Tool Type, the component will return a fully populated ToolChargeInfo object that contains the Daily Charge, Weekday Charge, Weekend Charge, and Holiday Charge values as defined in the functional specification.

This component should gracefully handle missing, incomplete, or invalid data files by displaying a user friendly message with some information on how the file might be repaired.

**Is Holiday**

Given a date, this component will return a boolean that is true if the date is one of the holidays defined in the functional specification, or false if the date is not one of those holidays.

**Round Up To Cents**

This component will round floating point numbers that represent a dollar amount half up to cents as required in the technical specification. While this rounding can be done with one line of code, it is being called out as a separate component because this seems like a utility that might be useful to any application that deals with monetary values. Please reuse an existing class that already performs this function, if one exists. If one does not already exist, please write this component so it can easily be reused.