|  |
| --- |
| **Tool Rental** |
| A Simple Tool Rental Checkout Application |

|  |
| --- |
| 2024 | By: Jeff L Greenwell |

****

|  |
| --- |
| Table of Contents  [Preface 2](#_Toc171636032)  [Introduction 3](#_Toc171636033)  [Purpose 3](#_Toc171636034)  [Scope 4](#_Toc171636035)  [Overview of the Solution 8](#_Toc171636036)  [Functional Requirements 10](#_Toc171636037)  [Description of the Tool Rental System 10](#_Toc171636038)  [Requirements 10](#_Toc171636039)  [Architecture and Design 13](#_Toc171636040)  [High-Level Architecture Diagram 13](#_Toc171636041)  [Detail Design 15](#_Toc171636042)  [Class Diagrams 19](#_Toc171636043)  [Sequence Diagrams 24](#_Toc171636044)  [Component Description 29](#_Toc171636045)  [Tool 29](#_Toc171636046)  [Attributes: Tool Code, Tool Type, Brand 29](#_Toc171636047)  [Methods 29](#_Toc171636048)  [Rental Agreement 29](#_Toc171636049)  [Attributes 29](#_Toc171636050)  [Methods 29](#_Toc171636051)  [Checkout Service 30](#_Toc171636052)  [Methods: Checkout Validate Input, Calculate Due Date 30](#_Toc171636053)  [Holiday Service 30](#_Toc171636054)  [Methods: Check if Date is Holiday 30](#_Toc171636055)  [Implementation Details 31](#_Toc171636056)  [Code Snippets for Key Methods 31](#_Toc171636057)  [Explanation of Algorithms Used 34](#_Toc171636058)  [Exception Handling 35](#_Toc171636059)  [Testing 36](#_Toc171636060)  [Test Cases 36](#_Toc171636061)  [Validation Tests (e.g., rental day count, discount percent) 36](#_Toc171636062)  [Functional Tests (e.g., calculating charges, generating rental agreements) 38](#_Toc171636063)  [Sample Output 41](#_Toc171636064)  [Example of Rental Agreement Output 41](#_Toc171636065)  [Conclusion 43](#_Toc171636066)  [Summary of the Solution 43](#_Toc171636067)  [Future Enhancements 43](#_Toc171636068) |

# Preface

This document provides an in-depth architectural overview of the Tool Rental Application. It is designed to be a comprehensive guide that covers all aspects of the system, from high-level design to detailed implementation specifics. The primary goal of this document is to serve as a complete reference for all stakeholders involved in the project, ensuring that everyone has a clear understanding of the system's architecture, functionality, and design principles.

In practice, it is rare to publish such an extensive and detailed architectural document in its entirety. Typically, subsets of this documentation are tailored to meet the needs of specific audiences. By segmenting the documentation, each group can focus on the information most relevant to their roles and responsibilities:

* **Product Owners and High-Level Stakeholders:** For this audience, the holistic design section, including high-level architecture diagrams and an overview of the system's functionality, will be of utmost importance. This section provides a clear picture of how the system meets business objectives and aligns with strategic goals.
* **Project Managers and Business Analysts:** These stakeholders will benefit from sections detailing the requirements, use cases, and validation tests. This information ensures that the project scope is well-defined and that the system meets all necessary business and functional requirements.
* **Developers and Engineers:** The detailed design specifications, class diagrams, sequence diagrams, and implementation details are crucial for this group. These sections provide the technical blueprint needed to build and maintain the system. Detailed explanations of algorithms, code snippets, and exception handling procedures will assist developers in understanding the system's inner workings and adhering to best practices.
* **Quality Assurance and Testers:** For QA teams, the testing section, including test cases and validation tests, is vital. This part of the documentation ensures that the system is thoroughly tested and meets the desired quality standards.
* **Operations and Maintenance Teams:** The implementation details, including information on exception handling, logging, and configuration, will be of particular interest to this group. This information helps in ensuring the system's reliability, maintainability, and ease of operation.

The decision to present a comprehensive architectural document stems from the desire to create a single, unified source of truth. This document serves as an invaluable resource for onboarding new team members, facilitating communication across departments, and ensuring consistency in understanding and implementation. It allows for a holistic view of the system while providing the granularity needed for detailed technical work.

By documenting every aspect of the Tool Rental Application in detail, we ensure that all stakeholders, regardless of their focus area, have access to the information they need. This comprehensive approach supports the successful development, deployment, and maintenance of the application, fostering a well-coordinated and informed project environment.

# Introduction

## Purpose

The purpose of this document is to provide a comprehensive guide for the implementation of a tool rental application. This document serves as a detailed blueprint for software engineers, outlining the functional requirements, architectural design, component design, and implementation details necessary to create a robust and scalable solution for managing the checkout process of tool rentals and generating rental agreements.

Specifically, this document aims to:

**Define Functional Requirements**: Clearly outline the functional specifications of the tool rental application, including the rules for rental periods, applicable charges, and discount policies.

**Present Architectural Design:** Describe the overall architecture of the application, including the high-level design and the interactions between various components. This includes the design patterns, technologies, and frameworks used.

**Detail Component Design:** Provide detailed descriptions of each component within the system, their responsibilities, and how they interact with each other. This includes class diagrams, sequence diagrams, and other relevant design artifacts.

**Explain Implementation Details:** Offer detailed guidance on the implementation of the system, including code snippets, algorithms, and logic that address specific requirements such as holiday charge rules and tool charge rules.

**Ensure Extensibility and Maintainability:** Highlight design decisions and practices that facilitate future extensions and maintenance of the application. This includes the use of interfaces and rule processors to handle holidays and tool charges.

**Support Programmers:** Provide sufficient detail and clarity to enable programmers to implement the solution accurately and efficiently. This includes handling edge cases, ensuring robustness, and maintaining code quality standards.

By adhering to this document, software engineers will be able to develop a tool rental application that meets all specified requirements, is easy to extend, and support the implementation of a seamless user experience.

# Scope

The scope of this document includes the following aspects:

**Functional Requirements:** A detailed description of the functional specifications for the tool rental application, including user interactions, rental period rules, applicable charges, discount policies, and the generation of rental agreements.

**Architectural Design:** An overview of the high-level architecture, including system components, their interactions, and the design patterns used to achieve a robust and scalable solution.

**Component Design:** In-depth descriptions of each system component, including their roles, responsibilities, interfaces, and interactions. This includes class diagrams, sequence diagrams, and other relevant design artifacts.

**Implementation Details:** Guidance on the implementation of the application, including code snippets, algorithms, and logic required to meet the functional specifications. This also covers validation, error handling, and extensibility considerations.

**Holiday and Charge Rules Processors:** Specific focus on the design and implementation of rule processors for determining applicable holidays and tool charges. This includes how these processors can be extended to support new rules and additional charge types in the future.

**Testing and Validation:** An outline of the testing strategy and example test cases to ensure the application meets its functional requirements and manages edge cases appropriately.

Exclusions

**User Interface Design:** This document does not cover the design or implementation of any user interface components, as the tool rental application can function with or without a graphical user interface in more of a service-oriented manner.

**Database Design and Integration:** This document excludes database design and integration details, focusing solely on the business logic and core functionality of the application.

**Deployment and Operations:** Deployment strategies, operational concerns, and performance monitoring are outside the scope of this documentation.

Intended Audience

This document is intended for multiple stakeholders involved in the design, development, implementing, and maintaining the tool rental application. It assumes a basic understanding of Java programming and software design principles.

Use Cases

The primary use cases addressed by this document include:

**1. Checking Out a Tool for a Specified Rental Period**

* **Actors**: Store Clerk, Customer
* **Preconditions**: The tool must be available for rent, and the store's system must be operational.
* **Main Flow**:
  1. The customer selects a tool to rent.
  2. The store clerk enters the tool code, rental days, discount percent, and check-out date into the system.
  3. The system validates the input:
     + Ensures the rental days are 1 or greater.
     + Ensures the discount percentage is between 0 and 100.
  4. The system calculates the due date based on the check-out date and rental days.
  5. The system generates a Rental Agreement.
  6. The Rental Agreement is printed or displayed for the customer.
* **Postconditions**: The tool is marked as rented, and the Rental Agreement is stored in the system.

**2. Calculating the Rental Charges, Including Applicable Discounts and Exemptions**

* **Actors**: Store Clerk, System
* **Preconditions**: The tool and rental period must be specified.
* **Main Flow**:
  1. The system retrieves the daily rental charge for the tool type.
  2. The system determines the chargeable days within the rental period, considering:
     + Weekends and holidays that may be exempt from charges for specific tool types.
  3. The system calculates the pre-discount charge (chargeable days \* daily rental charge).
  4. The system calculates the discount amount (pre-discount charge \* discount percent).
  5. The system calculates the final charge (pre-discount charge - discount amount).
* **Postconditions**: The calculated charges are included in the Rental Agreement.

**3. Generating a Rental Agreement That Summarizes the Rental Details and Charges**

* **Actors**: Store Clerk, System
* **Preconditions**: The rental charges must be calculated.
* **Main Flow**:
  1. The system compiles the rental details:
     + Tool code, type, and brand.
     + Rental days, check-out date, and due date.
     + Daily rental charge, charge days, pre-discount charge, discount percent, discount amount, and final charge.
  2. The system formats the Rental Agreement for printing or display.
  3. The Rental Agreement is presented to the customer.
* **Postconditions**: The Rental Agreement is recorded in the system, and a copy is provided to the customer.

**4. Handling Errors and Exceptions During Checkout**

* **Actors**: Store Clerk, System
* **Preconditions**: The tool and rental period must be specified.
* **Main Flow**:
  1. The system checks the rental days:
     + If less than 1, an exception is thrown with a message: "Rental day count must be 1 or greater."
  2. The system checks the discount percent:
     + If not between 0 and 100, an exception is thrown with a message: "Discount percent must be between 0 and 100."
* **Postconditions**: Errors are handled gracefully, and the store clerk is prompted to correct the input.

**5. Checking Out Multiple Tools at Once**

* **Actors**: Store Clerk, Customer
* **Preconditions**: The tools must be available for rent, and the store's system must be operational.
* **Main Flow**:
  1. The customer selects multiple tools to rent.
  2. The store clerk enters the details for each tool into the system.
  3. The system processes each tool as described in the individual checkout use case.
  4. The system generates a consolidated Rental Agreement that includes all tools.
  5. The consolidated Rental Agreement is printed or displayed for the customer.
* **Postconditions**: The tools are marked as rented, and the consolidated Rental Agreement is stored in the system.

**6. Printing and Logging Rental Agreements**

* **Actors**: Store Clerk, System
* **Preconditions**: The Rental Agreement must be generated.
* **Main Flow**:
  1. The system formats the Rental Agreement for printing.
  2. The system sends the Rental Agreement to the printer and the system log.
  3. The Rental Agreement is printed and logged with a timestamp.
* **Postconditions**: The customer receives a printed copy, and the agreement is logged for record-keeping.

# Overview of the Solution

The Tool Rental Application is a robust and flexible system designed to manage the rental process for a variety of tools at a store, such as Home Depot. The application manages tool checkouts, calculates rental charges based on a range of factors, and generates comprehensive rental agreements.

High-Level Description

The Tool Rental Application allows store clerks to efficiently manage tool rentals by customers for specified durations. It calculates rental charges according to tool type, applicable discounts, and special rules for weekends and holidays. The system produces a detailed rental agreement summarizing all rental details and charges.

Main Components

**Tool Management:** This component manages information about the tools available for rent, including attributes like tool code, type, and brand. It ensures that all tool data is easily accessible and up to date.

**Rental Agreement Generator:** This component generates rental agreements based on the checkout process. It includes all relevant details such as tool information, rental period, daily charges, discount details, and the final charge.

**Charge Calculation:** This module calculates rental charges, considering different rates for weekdays, weekends, and holidays. It ensures accurate computation of total charges based on the rental period and applicable rules.

**Holiday Processor:** A dedicated component that determines if a given date falls on a holiday. It uses configurable rules to ensure accurate identification of holidays, allowing the system to adjust charges accordingly.

**Charge Processor:** This component determines the applicable charges for each day of the rental period. The charge processor supports different charge types (e.g., daily, weekday, weekend) and is designed to be easily extensible for future charge types.

**Validation Module:** Ensures that all input data, such as rental days and discount percentages, adhere to specified constraints. It provides user-friendly error messages for invalid inputs, ensuring a smooth checkout process.

Key Features

**Configurable Charge Calculation:** Supports configurable charge rates for weekdays, weekends, and holidays. This flexibility allows for easy updates and extensions to the charging rules.

**Accurate Holiday Handling:** Utilizes a rules processor to accurately determine holidays and apply appropriate charges, ensuring compliance with holiday policies.

**Discount Application:** Allows clerks to apply discounts to the total rental charge, with the discount being calculated and displayed in the rental agreement.

**Comprehensive Rental Agreements:** Generates detailed rental agreements that include all necessary information formatted for clarity, aiding both clerks and customers.

Architectural Approach

The Tool Rental Application employs a modular architecture, ensuring that components are loosely coupled and highly cohesive. This design promotes maintainability, scalability, and ease of understanding. Key design patterns used include:

**Rule Processor Pattern:** Facilitates handling holiday and charge calculations, making it straightforward to add new rules without modifying existing code.

**Factory Pattern:** Used for creating instances of tools and charge rules, ensuring the system can easily accommodate new tool types and rules.

Interaction Flow

**Checkout Process:** A customer provides the tool code, rental period, discount percentage, and checkout date to initiate the rental process.

**Charge Calculation:** The application calculates total rental charges by evaluating each day in the rental period and applying relevant charge rules.

**Holiday Evaluation:** The holiday processor determines if any days in the rental period are holidays, adjusting charges, as necessary.

**Rental Agreement Generation:** The system generates a detailed rental agreement summarizing all rental details, including charges and discounts.

**Output:** The rental agreement output to the console, providing a clear summary of the transaction for the clerk and customer.

This solution ensures a dependable, flexible, and maintainable tool rental application that meets specified functional requirements and designed for easy extension to accommodate future needs.

# Functional Requirements

## Description of the Tool Rental System

The Tool Rental System is a point-of-sale application designed for a store that rents large tools, such as Home Depot. This system facilitates the rental process by allowing customers to rent tools for specified periods, calculates rental charges including discounts and exemptions, and generates detailed rental agreements. The system aims to streamline the rental process, ensuring accuracy in rental charge calculations and providing clear rental agreements to customers.

Key functionalities of the Tool Rental System include:

* **Tool Checkout**: Enables store clerks to process tool rentals by entering rental details such as tool code, rental days, discount percentage, and check-out date.
* **Charge Calculation**: Automatically calculates rental charges based on tool type, rental period, applicable discounts, and exemptions for weekends and holidays.
* **Rental Agreement Generation**: Produces a comprehensive rental agreement that summarizes rental details and charges, formatted for both display and printing.
* **Error Handling**: Validates input data to ensure rental days are at least one and discount percentages are within the valid range, providing user-friendly error messages for invalid entries.
* **Multiple Tool Rentals**: Allows the checkout of multiple tools in a single transaction, generating individual rental agreements for each tool as well as a consolidated agreement.

## Requirements

The functional requirements for the Tool Rental System are outlined below, detailing the necessary features and capabilities to ensure the system meets user needs:

1. **Tool Information Management**
   * Store and manage information for various tools available for rental, including tool code, type, brand, and daily charges (weekday, weekend, and holiday).
2. **Rental Checkout Process**
   * Enable store clerks to initiate a rental transaction by entering the following details:
     + Tool code (unique identifier for each tool)
     + Rental days (number of days the tool is rented for, must be 1 or greater)
     + Discount percent (percentage discount applied to the total charges, must be between 0 and 100)
     + Check-out date (date when the tool is rented out)
   * Validate the rental days and discount percent, throwing an exception for invalid values with appropriate error messages.
3. **Charge Calculation**
   * Calculate the due date based on the check-out date and rental days.
   * Determine chargeable days within the rental period, considering weekends and holidays:
     + Weekends and holidays may be exempt from charges depending on the tool type.
   * Calculate the pre-discount charge as the product of chargeable days and the daily rental charge.
   * Apply the discount to the pre-discount charge, calculating the discount amount and the final charge.
4. **Rental Agreement Generation**
   * Generate a detailed rental agreement that includes:
     + Tool code, type, and brand
     + Rental days, check-out date, and due date
     + Daily rental charge, charge days, pre-discount charge, discount percent, discount amount, and final charge
   * Format the rental agreement for display and printing, adhering to specified formatting rules (date as mm/dd/yy, currency as $9,999.99, and percent as 99%).
5. **Error Handling**
   * Provide user-friendly error messages for invalid input during the checkout process:
     + "Rental day count must be 1 or greater"
     + "Discount percent must be between 0 and 100"
6. **Multiple Tool Rentals**
   * Allow multiple tools to be rented in a single transaction.
   * Generate individual rental agreements for each tool as well as a consolidated rental agreement that summarizes all tools rented in the transaction.
7. **Printing and Logging**
   * Print individual and consolidated rental agreements to the console and log them for record-keeping.
   * Ensure the printed and logged agreements are formatted for readability and include all relevant details.
8. **Holiday Management**
   * Recognize and manage specific holidays (Independence Day and Labor Day) with the following rules:
     + Independence Day (July 4th) is observed on the closest weekday if it falls on a weekend.
     + Labor Day is on the first Monday in September.

By fulfilling these functional requirements, the Tool Rental System will effectively support the store's rental operations, providing a user-friendly experience for both store clerks and customers while ensuring accurate and efficient rental transactions.

# Architecture and Design

## High-Level Architecture Diagram

A screenshot of a computer screen

Description automatically generated

Components

1. **User Interface (UI)**
   * The entry point for users to interact with the system (e.g., CLI or a future GUI).
2. **ShoppingCart**
   * Manages the collection of tools selected for rental.
   * Handles adding tools, calculating charges, and generating rental agreements.
3. **ChargeProcessor**
   * Calculates rental charges based on predefined rules.
   * Applies discounts and exemptions for holidays and weekends.
4. **HolidayProcessor**
   * Manages holiday rules.
   * Determines if a given date is a holiday.
5. **Configuration Files**
   * JSON/YAML files that contain the charge rules and holiday definitions.
6. **Logger**
   * Logs the rental agreements and other important events.
   * Provides console and file logging capabilities.

Interactions

1. **User -> ShoppingCart**
   * Users add tools to the shopping cart specifying tool code, rental period, discount, etc.
2. **ShoppingCart -> ChargeProcessor**
   * The shopping cart requests the charge processor to calculate the rental charges for the tools.
3. **ChargeProcessor -> HolidayProcessor**
   * The charge processor consults the holiday processor to determine if certain days are holidays.
4. **ShoppingCart -> RentalAgreement**
   * The shopping cart generates rental agreements based on the processed charges and rental details.
5. **RentalAgreement -> Logger**
   * The rental agreement details are printed to the console and logged to files for record-keeping.

## Detail Design

The system is composed of several key components, each responsible for specific functionalities within the Tool Rental System. Below is a detailed description of each component, including its purpose and interactions with other components.

- Tool

**Purpose:**  
The Tool class represents a tool available for rent. It encapsulates the attributes of a tool such as its code, type, and brand.

**Attributes:**

* toolCode: A unique identifier for the tool.
* toolType: The type of tool (e.g., ladder, chainsaw, jackhammer).
* toolBrand: The brand of the tool.

**Interactions:**

public class Tool {

private final String toolCode;

private final String toolType;

private final String toolBrand;

// Constructor, Getters, and Setters

}

* Used by the ShoppingCart to manage and store details of the tools added for rental.

- ShoppingCart

**Purpose:**  
The ShoppingCart class manages the collection of tools to be rented in a single transaction. It handles adding tools, calculating charges, and generating rental agreements.

**Attributes:**

* rentals: A list of tools added to the cart for rental.
* chargeProcessor: An instance of ChargeProcessor to calculate rental charges.

**Methods:**

* addTool(): Adds a tool to the shopping cart.
* checkout(): Generates rental agreements for all tools in the cart.
* printConsolidatedAgreement(): Prints a consolidated rental agreement for all tools in the cart.

**Interactions:**

* Interacts with ChargeProcessor to calculate rental charges.
* Utilizes RentalAgreement to generate detailed rental agreements.

public class ShoppingCart {

private final List<ToolRental> rentals = new ArrayList<>();

private final ChargeProcessor chargeProcessor;

// Methods: addTool(), checkout(), printConsolidatedAgreement()

}

- ChargeProcessor

**Purpose:**  
The ChargeProcessor class calculates rental charges based on the tool type, rental period, and applicable discounts. It applies holiday and weekend exemptions using the HolidayProcessor.

**Attributes:**

* chargeRules: A list of charging rules for different tools.
* holidayProcessor: An instance of HolidayProcessor to check for holidays.

**Methods:**

* calculateTotalCharge(): Calculates the total rental charge for a given tool over a rental period.
* getChargeRules(): Retrieves the charge rules for tools.

**Interactions:**

* Utilizes ChargeRule to determine applicable charges for tools.
* Interacts with HolidayProcessor to check for holidays and apply exemptions.

public class ChargeProcessor {

private final List<ChargeRule> chargeRules;

private final HolidayProcessor holidayProcessor;

// Methods: calculateTotalCharge(), getChargeRules()

}

- HolidayProcessor

**Purpose:**  
The HolidayProcessor class determines if a given date is a holiday. It manages a list of holiday rules and checks if a date falls under any of the defined holidays.

**Attributes:**

* holidayRules: A list of holiday rules.

**Methods:**

* addHolidayRule(): Adds a holiday rule to the processor.
* isHoliday(): Checks if a given date is a holiday.

**Interactions:**

* Interacts with ChargeProcessor to apply holiday exemptions during charge calculation.

public class HolidayProcessor {

private final List<HolidayRule> holidayRules = new ArrayList<>();

// Methods: addHolidayRule(), isHoliday()

}

- RentalAgreement

**Purpose:**  
The RentalAgreement class generates a rental agreement summarizing the rental details and charges for a tool. It includes information such as rental days, daily charges, discounts, and final charges.

**Attributes:**

* toolCode, toolType, toolBrand: Attributes of the tool being rented.
* rentalDays, checkOutDate, dueDate: Rental period details.
* dailyRentalCharge, preDiscountCharge, discountPercent, discountAmount, finalCharge: Financial details of the rental.
* quantity: Number of tools rented.
* totalChargeDays: Total chargeable days excluding holidays.

**Methods:**

* printAgreement(): Prints the rental agreement.
* Private methods for calculating charges and discounts.

**Interactions:**

* Utilizes ChargeProcessor to calculate the applicable charges.
* Generated by ShoppingCart during the checkout process.

public class RentalAgreement {

private final String toolCode;

private final String toolType;

private final String toolBrand;

private final int rentalDays;

private final LocalDate checkOutDate;

private final LocalDate dueDate;

private final BigDecimal dailyRentalCharge;

private final BigDecimal preDiscountCharge;

private final int discountPercent;

private final BigDecimal discountAmount;

private final BigDecimal finalCharge;

private final int quantity;

private int totalChargeDays;

// Methods: printAgreement(), private calculation methods

}

### Class Diagrams

The following class diagrams provide a visual representation of the classes in the Tool Rental System and their relationships. Each class is depicted with its attributes and methods to give a comprehensive overview of the system's structure.

Tool Class Diagram

A black screen with white text

Description automatically generated

**Description:**

* Tool: Represents a tool available for rent, with attributes toolCode, toolType, and toolBrand.

ShoppingCart Class Diagram

A black and white rectangle with text

Description automatically generated

**Description:**

* ShoppingCart: Manages the collection of tools to be rented, calculates charges, and generates rental agreements.
* ToolRental: An inner class representing the rental details of a tool.

ChargeProcessor Class Diagram

A black screen with a white border

Description automatically generated

**Description:**

* ChargeProcessor: Calculates rental charges based on tool type, rental period, and applicable discounts.

HolidayProcessor Class Diagram

A black and white screen with text

Description automatically generated

**Description:**

* HolidayProcessor: Determines if a given date is a holiday by managing holiday rules.

RentalAgreement Class Diagram

A black background with a black square

Description automatically generated with medium confidence

**Description:**

* RentalAgreement: Generates a rental agreement summarizing the rental details and charges for a tool.

ChargeRule Class Diagram

A black screen with white text

Description automatically generated

**Description:**

* ChargeRule: Represents the charging rules for a specific tool, including weekday, weekend, and holiday charges.

LoggerConfig Class Diagram

A black and white rectangular object with black text

Description automatically generated

**Description:**

* LoggerConfig: Configures the logger for logging information, warnings, and errors in the application.

HolidayRule Interface Diagram

A black and white rectangular sign

Description automatically generated

**Description:**

* HolidayRule: Interface for defining rules to determine if a given date is a holiday.

IndependenceDayRule Class Diagram

A black and white rectangle with blue text

Description automatically generated

**Description:**

* IndependenceDayRule: Defines the rules for determining if a date is the Independence Day holiday.

LaborDayRule Class Diagram

A black and white sign with blue text

Description automatically generated

**Description:**

* LaborDayRule: Defines the rules for determining if a date is the Labor Day holiday.

ToolChargeConfig Class Diagram

A black rectangle with text

Description automatically generated

**Description:**

* ToolChargeConfig: Holds the configuration for tool charges, including weekday, weekend, and holiday charges.
* ToolCharge: An inner class representing the charge details for a tool.

### Sequence Diagrams

The following sequence diagrams illustrate the interactions between the various components of the Tool Rental System during key operations. These diagrams provide a step-by-step visualization of how the system functions.

Sequence Diagram: Adding a Tool to the ShoppingCart

A black background with white text

Description automatically generated

**Description:**

1. The user calls the addTool() method on the ShoppingCart.
2. The ShoppingCart creates a new ToolRental object to represent the rental details of the tool.

Sequence Diagram: Checking Out and Generating Rental Agreements

A black background with white text

Description automatically generated

**Description:**

1. The user calls the checkout() method on the ShoppingCart.
2. The ShoppingCart iterates through the rentals and calls calculateTotalCharge() on the ChargeProcessor to compute the charges.
3. For each rental, a RentalAgreement is created with the calculated charges.

Sequence Diagram: Printing a Rental Agreement

A screen shot of a computer

Description automatically generated

**Description:**

1. The user calls the printAgreement() method on the RentalAgreement.
2. The RentalAgreement retrieves the necessary charges using getCharge() from the ChargeProcessor.
3. The RentalAgreement logs the details of the agreement using the Logger.

Sequence Diagram: Checking if a Date is a Holiday

A black background with white text

Description automatically generated

**Description:**

1. The user calls the isHoliday() method on the HolidayProcessor.
2. The HolidayProcessor checks the date against each HolidayRule to determine if it is a holiday.

Sequence Diagram: Logging an Event

A black and white screen with white text

Description automatically generated

**Description:**

1. The user calls the logEvent() method on the Logger.
2. The Logger uses the LoggerConfig to set up and log the event.

# Component Description

## Tool

The Tool class represents a tool available for rent. It contains attributes that describe the tool and provides methods to access these attributes.

### Attributes: Tool Code, Tool Type, Brand

* Tool Code: A unique identifier for the tool.
* Tool Type: The type of tool (e.g., Ladder, Chainsaw).
* Brand: The brand of the tool.

### Methods

* getToolCode(): Returns the tool code.
* getToolType(): Returns the tool type.
* getBrand(): Returns the brand of the tool.

## Rental Agreement

The RentalAgreement class generates a rental agreement summarizing the rental details and charges. It calculates the charges based on the rental period, tool type, applicable discounts, and holidays.

### Attributes

* Tool Code: Code of the rented tool.
* Tool Type: Type of the rented tool.
* Tool Brand: Brand of the rented tool.
* Rental Days: Number of days the tool is rented.
* Check Out Date: The date the tool is checked out.
* Due Date: The date the tool is due back.
* Daily Rental Charge: Daily charge for renting the tool.
* Quantity: Number of tools rented.
* Charge Days: Number of days for which the rental charge applies.
* Pre-discount Charge: Total charge before applying discounts.
* Discount Percent: Percentage of discount applied.
* Discount Amount: Amount of discount applied.
* Final Charge: Total charge after applying discounts.

### Methods

* printAgreement(int agreementNumber): Prints the rental agreement.
* calculatePreDiscountCharge(): Calculates the total charge before applying discounts.
* calculateDiscountAmount(): Calculates the discount amount.
* calculateFinalCharge(): Calculates the final charge after applying discounts.
* findFirstChargeableDay(): Finds the first day that is chargeable.
* calculateChargeDays(): Calculates the number of chargeable days.

## Checkout Service

The ShoppingCart class acts as the checkout service, managing the collection of tools to be rented and processing the rental transactions.

### Methods: Checkout Validate Input, Calculate Due Date

Checkout:

* checkout(): Processes the tools in the shopping cart, calculates the charges, and generates rental agreements.

Validate Input:

* addTool(String toolCode, String toolType, String brand, int rentalDays, int discountPercent, int quantity, LocalDate checkOutDate): Adds a tool to the shopping cart after validating the input.

Calculate Due Date:

* calculateDueDate(LocalDate checkOutDate, int rentalDays): Calculates the due date for the rented tools based on the check-out date and rental days.

## Holiday Service

The HolidayProcessor class determines if a given date is a holiday by using the rules defined in the HolidayRule implementations.

### Methods: Check if Date is Holiday

Check if Date is Holiday:

* isHoliday(LocalDate date): Checks if the provided date is a holiday by applying the holiday rules.
* addHolidayRule(HolidayRule rule): Adds a holiday rule to the processor.

# Implementation Details

## Code Snippets for Key Methods

Tool Class

public class Tool {

private final String toolCode;

private final String toolType;

private final String toolBrand;

public Tool(String toolCode, String toolType, String toolBrand) {

this.toolCode = toolCode;

this.toolType = toolType;

this.brand = toolBrand;

}

public String getToolCode() {

return toolCode;

}

public String getToolType() {

return toolType;

}

public String getBrand() {

return brand;

}

}

RentalAgreement Class

public class RentalAgreement {

// Attributes omitted for brevity

public RentalAgreement(String toolCode, String toolType, String toolBrand, int rentalDays,

LocalDate checkOutDate, int discountPercent, int quantity,

ChargeProcessor chargeProcessor) {

// Initialization logic omitted for brevity

this.preDiscountCharge = calculatePreDiscountCharge(this.totalChargeDays, this.dailyRentalCharge, this.quantity);

this.discountAmount = calculateDiscountAmount(this.preDiscountCharge, this.discountPercent);

this.finalCharge = calculateFinalCharge(this.preDiscountCharge, this.discountAmount);

}

private BigDecimal calculatePreDiscountCharge(int chargeDays, BigDecimal dailyRentalCharge, int quantity) {

return dailyRentalCharge.multiply(BigDecimal.valueOf(chargeDays))

.multiply(BigDecimal.valueOf(quantity)).

setScale(2, BigDecimal.ROUND\_HALF\_UP);

}

private BigDecimal calculateDiscountAmount(BigDecimal preDiscountCharge, int discountPercent) {

return preDiscountCharge.multiply(BigDecimal.valueOf(discountPercent))

.divide(BigDecimal.valueOf(100))

.setScale(2, BigDecimal.ROUND\_HALF\_UP);

}

private BigDecimal calculateFinalCharge(BigDecimal preDiscountCharge, BigDecimal discountAmount) {

return preDiscountCharge.subtract(discountAmount).setScale(2, BigDecimal.ROUND\_HALF\_UP);

}

}

ChargeProcessor Class

public class ChargeProcessor {

private final List<ChargeRule> chargeRules;

private final HolidayProcessor holidayProcessor;

public ChargeProcessor(List<ChargeRule> chargeRules, HolidayProcessor holidayProcessor) {

this.chargeRules = chargeRules;

this.holidayProcessor = holidayProcessor;

}

public BigDecimal calculateTotalCharge(LocalDate startDate, LocalDate endDate, ChargeRule chargeRule) {

BigDecimal totalCharge = BigDecimal.ZERO;

for (LocalDate date = startDate; !date.isAfter(endDate); date = date.plusDays(1)) {

if (!holidayProcessor.isHoliday(date)) {

BigDecimal charge = chargeRule.getCharge(date);

totalCharge = totalCharge.add(charge);

}

}

return totalCharge.setScale(2, BigDecimal.ROUND\_HALF\_UP);

}

}

## Explanation of Algorithms Used

Rental Charge Calculation

The charge calculation algorithm in the RentalAgreement class involves:

1. **Calculating Pre-Discount Charge:** This is done by multiplying the daily rental charge by the number of chargeable days and the quantity of tools rented.
2. **Calculating Discount Amount:** This involves applying the discount percentage to the pre-discount charge.
3. **Calculating Final Charge:** The final charge is obtained by subtracting the discount amount from the pre-discount charge.

Charge Day Calculation

In the ChargeProcessor class:

1. **Iterate Over Rental Days:** Loop through each day in the rental period.
2. **Check Holiday Status:** Use the HolidayProcessor to check if the current day is a holiday.
3. **Accumulate Charges:** If the day is not a holiday, accumulate the daily charge based on the type of day (weekday/weekend).

Holiday Detection

In the HolidayProcessor class:

1. **Iterate Over Holiday Rules:** Loop through each rule in the holiday rules list.
2. **Check if Date Matches Rule:** Apply each rule to see if the date is a holiday.

## Exception Handling

Invalid Rental Days

In the ShoppingCart and RentalAgreement classes:

if (rentalDays < 1) {

throw new IllegalArgumentException("Rental day count must be 1 or greater");

}

Invalid Discount Percentage

In the ShoppingCart and RentalAgreement classes:

if (discountPercent < 0 || discountPercent > 100) {

throw new IllegalArgumentException("Discount percent must be between 0 and 100");

}

Missing Charge Rule

In the RentalAgreement class:

ChargeRule chargeRule = chargeProcessor.getChargeRules().stream()

.filter(rule -> toolCode.equals(rule.getToolCode()))

.findFirst()

.orElseThrow(() -> new IllegalArgumentException("Charge rule not found for tool code: " + toolCode));

Logging

Logging is set up using the LoggerConfig class. All major steps, such as initialization and charge calculations, are logged at the INFO level, while exceptions and errors are logged at the SEVERE level. This provides a detailed trace of the system’s operations and helps in debugging and monitoring.

# Testing

## Test Cases

Testing the Tool Rental System involves a combination of validation tests to ensure that the input parameters are correct and functional tests to verify that the system performs the required calculations and generates the correct outputs. The following test cases have been implemented to cover the various aspects of the system.

Overview of Test Cases

1. **Validation Tests:**
   * Ensures the rental day count is valid.
   * Checks that the discount percentage is within the valid range.
2. **Functional Tests:**
   * Validates the calculation of rental charges.
   * Ensures the correct generation of rental agreements.
   * Verifies the handling of holidays in charge calculations.

### Validation Tests (e.g., rental day count, discount percent)

Validation tests ensure that the inputs to the system are correctly validated. These tests check for invalid values and ensure that the system handles them appropriately.

Rental Day Count Validation

@Test

public void testInvalidRentalDays() {

ShoppingCart cart = new ShoppingCart(chargeProcessor);

LocalDate checkOutDate = LocalDate.of(2024, 7, 4); // Independence Day

IllegalArgumentException thrown = assertThrows(

IllegalArgumentException.class,

() -> cart.addTool("LADW", "Ladder", "Werner", 0, 0, 1, checkOutDate), // Invalid rental days

"Expected addTool() to throw, but it didn't"

);

assertTrue(thrown.getMessage().contains("Rental day count must be 1 or greater"));

}

Discount Percentage Validation

@Test

public void testInvalidDiscount() {

ShoppingCart cart = new ShoppingCart(chargeProcessor);

LocalDate checkOutDate = LocalDate.of(2024, 7, 4); // Independence Day

IllegalArgumentException thrown = assertThrows(

IllegalArgumentException.class,

() -> cart.addTool("LADW", "Ladder", "Werner", 2, 110, 1, checkOutDate), // Invalid discount

"Expected addTool() to throw, but it didn't"

);

assertTrue(thrown.getMessage().contains("Discount percent must be between 0 and 100"));

}

### Functional Tests (e.g., calculating charges, generating rental agreements)

Functional tests validate that the system performs the required operations correctly. These tests ensure that the calculations for charges are accurate and that rental agreements are generated as expected.

Calculating Charges

The following test cases ensure that the system calculates the correct charges for different tools and rental periods, including handling holidays.

Ladder Rental Charge Calculation

@Test

public void testLadderRental() {

LocalDate checkOutDate = LocalDate.of(2024, 7, 4); // Independence Day

RentalAgreement agreement = new RentalAgreement("LADW", "Ladder", "Werner", 2, checkOutDate, 0, 1, chargeProcessor);

assertNotNull(agreement);

assertEquals("LADW", agreement.getToolCode());

assertEquals("Ladder", agreement.getToolType());

assertEquals("Werner", agreement.getToolBrand());

assertEquals(2, agreement.getRentalDays());

assertEquals(checkOutDate, agreement.getCheckOutDate());

assertEquals(checkOutDate.plusDays(agreement.getRentalDays() - 1), agreement.getDueDate());

assertEquals(BigDecimal.valueOf(1.99).setScale(2), agreement.getDailyRentalCharge().setScale(2));

assertEquals(1, agreement.getChargeDays());

assertEquals(BigDecimal.valueOf(1.99).setScale(2), agreement.getPreDiscountCharge().setScale(2));

assertEquals(BigDecimal.ZERO.setScale(2), agreement.getDiscountAmount().setScale(2));

assertEquals(BigDecimal.valueOf(1.99).setScale(2), agreement.getFinalCharge().setScale(2));

}

Generating Rental Agreements

The following test case verifies that the system generates correct rental agreements, including details of each tool rented.

Multiple Tools Rental Agreement Generation

@Test

public void testMultipleToolsHolidayWeekend() {

ShoppingCart cart = new ShoppingCart(chargeProcessor);

LocalDate checkOutDate = LocalDate.of(2024, 8, 29); // Thursday before Labor Day weekend

cart.addTool("JAKD", "Jackhammer", "DeWalt", 7, 0, 1, checkOutDate); // 1 jackhammer

cart.addTool("CHNS", "Chainsaw", "Stihl", 7, 0, 1, checkOutDate); // 1 chainsaw

List<RentalAgreement> agreements = cart.checkout();

assertNotNull(agreements);

assertEquals(2, agreements.size());

RentalAgreement jackhammerAgreement = agreements.get(0);

assertEquals("JAKD", jackhammerAgreement.getToolCode());

assertEquals("Jackhammer", jackhammerAgreement.getToolType());

assertEquals("DeWalt", jackhammerAgreement.getToolBrand());

assertEquals(7, jackhammerAgreement.getRentalDays());

assertEquals(checkOutDate, jackhammerAgreement.getCheckOutDate());

assertEquals(checkOutDate.plusDays(jackhammerAgreement.getRentalDays() - 1), jackhammerAgreement.getDueDate());

assertEquals(BigDecimal.valueOf(2.99), jackhammerAgreement.getDailyRentalCharge());

assertEquals(4, jackhammerAgreement.getChargeDays()); // Charge days should be 4 (Aug 29, 30, Sep 3, 4)

assertEquals(BigDecimal.valueOf(11.96), jackhammerAgreement.getPreDiscountCharge());

assertEquals(BigDecimal.ZERO.setScale(2), jackhammerAgreement.getDiscountAmount());

assertEquals(BigDecimal.valueOf(11.96), jackhammerAgreement.getFinalCharge());

RentalAgreement chainsawAgreement = agreements.get(1);

assertEquals("CHNS", chainsawAgreement.getToolCode());

assertEquals("Chainsaw", chainsawAgreement.getToolType());

assertEquals("Stihl", chainsawAgreement.getToolBrand());

assertEquals(7, chainsawAgreement.getRentalDays());

assertEquals(checkOutDate, chainsawAgreement.getCheckOutDate());

assertEquals(checkOutDate.plusDays(chainsawAgreement.getRentalDays() - 1), chainsawAgreement.getDueDate());

assertEquals(BigDecimal.valueOf(1.49), chainsawAgreement.getDailyRentalCharge());

assertEquals(5, chainsawAgreement.getChargeDays()); // Charge days should be 5 (Aug 29, 30, 31, Sep 1, 2)

assertEquals(BigDecimal.valueOf(7.45), chainsawAgreement.getPreDiscountCharge());

assertEquals(BigDecimal.ZERO.setScale(2), chainsawAgreement.getDiscountAmount());

assertEquals(BigDecimal.valueOf(7.45), chainsawAgreement.getFinalCharge());

jackhammerAgreement.printAgreement(1);

chainsawAgreement.printAgreement(1);

cart.printConsolidatedAgreement();

}

Handling Holidays in Charge Calculations

This test case ensures that the system correctly handles holidays in the charge calculations.

Charge Calculation with Holiday

@Test

public void testCalculateTotalCharge\_Ladder\_WithHoliday() {

LocalDate startDate = LocalDate.of(2024, 7, 4); // Independence Day

LocalDate endDate = LocalDate.of(2024, 7, 5); // The day after

// Find the rule for the ladder

ChargeRule ladderRule = chargeProcessor.getChargeRules().stream()

.filter(rule -> "LADW".equals(rule.getToolCode()))

.findFirst()

.orElseThrow(() -> new IllegalArgumentException("Charge rule not found for tool code: LADW"));

BigDecimal totalCharge = chargeProcessor.calculateTotalCharge(startDate, endDate, ladderRule);

// The first day is a holiday (no charge), the second day is a regular weekday (charge applies)

assertEquals(BigDecimal.valueOf(1.99).setScale(2), totalCharge);

}

**Summary**

The testing section covers both validation and functional tests to ensure that the Tool Rental System operates correctly. Validation tests check the correctness of input values, while functional tests verify that the system's core functionalities, such as calculating rental charges and generating rental agreements, work as expected. The provided code snippets and explanations demonstrate how each component is tested, ensuring robustness and reliability.

# Sample Output

### Example of Rental Agreement Output

The following section provides a sample output of the rental agreement generated by the Tool Rental System. This example illustrates the information provided for each tool rented, including tool details, rental period, charges, and any applicable discounts.

Individual Rental Agreement Output

\*\*\*\*\*\*\*\*\*\* Rental Agreement 1 \*\*\*\*\*\*\*\*\*\*

---------- Tool Information ----------

Tool code: LADW

Tool type: Ladder

Tool brand: Werner

Rental days: 2

Check out date: 07/02/24

Due date: 07/04/24

--------------------------------------

------------ Charges -----------------

Daily rental charge per tool: $1.99

Quantity: 1

Charge days: 2

Pre-discount charge: $3.98

Discount percent: 0%

Discount amount: $0.00

\*\* Final charge: $3.98 \*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\* Rental Agreement 2 \*\*\*\*\*\*\*\*\*\*

---------- Tool Information ----------

Tool code: CHNS

Tool type: Chainsaw

Tool brand: Stihl

Rental days: 3

Check out date: 07/02/24

Due date: 07/05/24

--------------------------------------

------------ Charges -----------------

Daily rental charge per tool: $1.49

Quantity: 1

Charge days: 3

Pre-discount charge: $4.47

Discount percent: 15%

Discount amount: $0.67

\*\* Final charge: $3.80 \*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Consolidated Rental Agreement Output

\*\*\*\*\*\*\*\*\*\* Consolidated Rental Agreement \*\*\*\*\*\*\*\*\*\*

---------- Tool 1 Information (LADW - Ladder) ----------

Tool code: LADW

Tool type: Ladder

Tool brand: Werner

Rental days: 2

Check out date: 07/02/24

Due date: 07/04/24

------------ Charges -----------------

Daily rental charge per tool: $1.99

Quantity: 1

Charge days: 2

Pre-discount charge: $3.98

Discount percent: 0%

Discount amount: $0.00

Final charge: $3.98

------------------------------

---------- Tool 2 Information (CHNS - Chainsaw) ----------

Tool code: CHNS

Tool type: Chainsaw

Tool brand: Stihl

Rental days: 3

Check out date: 07/02/24

Due date: 07/05/24

------------ Charges -----------------

Daily rental charge per tool: $1.49

Quantity: 1

Charge days: 3

Pre-discount charge: $4.47

Discount percent: 15%

Discount amount: $0.67

Final charge: $3.80

============== Summary ==============

Total pre-discount charge: $8.45

Total discount amount: $0.67

Total final charge: $7.78

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Conclusion

## Summary of the Solution

The Tool Rental Application provides a robust and efficient system for managing tool rentals. Key features include:

* **Tool Management:** A comprehensive structure for representing tools, including attributes such as tool code, type, and brand.
* **Rental Agreements:** Automated generation of rental agreements detailing the rental period, charges, and applicable discounts.
* **Charge Calculation:** A flexible charge processor that calculates rental charges based on tool type, rental period, and any discounts or holiday exemptions.
* **Holiday Management:** An integrated holiday processor that correctly identifies holidays and adjusts rental charges accordingly.
* **User Interaction:** A straightforward checkout process and detailed rental agreements that provide clear information to the customer.

The system is designed with a modular architecture, making it easy to extend and integrate into larger systems. The use of standard design patterns and thorough testing ensures reliability and maintainability.

## Future Enhancements

While the current implementation is robust, there are several areas for potential future enhancements:

* **Integration with Database Systems:** To support persistent storage and retrieval of tool and rental data, integrating a database system will be essential. This will allow for maintaining historical data, managing inventory, and ensuring data integrity.
* **User Interface Integration:** Developing a user-friendly interface, either as a web application or a mobile app, will enhance user experience. This interface could provide features such as tool search, rental history, and online payments.
* **Service-Oriented API Support:** Creating a set of APIs will enable integration with other systems. This would allow the Tool Rental Application to be part of a larger ecosystem, facilitating data exchange and interoperability with other services.
* **Advanced Reporting and Analytics:** Implementing advanced reporting tools and analytics will provide insights into rental patterns, customer preferences, and operational efficiency. This can help in making data-driven business decisions.
* **Notification and Reminder System:** Adding a notification system to remind customers of due dates, upcoming holidays, and promotional discounts can improve customer engagement and reduce late returns.
* **Dynamic Pricing and Promotions:** Implementing dynamic pricing strategies based on demand, season, or customer loyalty can optimize revenue. Additionally, running promotional campaigns and offering discounts to frequent renters can enhance customer retention.
* **Enhanced Security Features:** Incorporating advanced security measures, such as encryption, secure authentication, and authorization protocols, will ensure that sensitive data is protected and that only authorized users can access specific functionalities.
* **Globalization and Localization:** Adapting the system to support multiple languages and regional settings will make it accessible to a broader audience, catering to a diverse customer base.

These enhancements will not only increase the functionality of the Tool Rental Application but also improve its scalability, user experience, and integration capabilities, making it a more comprehensive solution for tool rental management.