# Receipt Generator Documentation

Jay Doshi, jaydoshi@usc.edu

## Concept

Basic sales tax is applicable at a rate of 10% on all goods, except books, food, and medical products that are exempt. Import duty is an additional sales tax applicable on all imported goods at a rate of 5%, with no exemptions.

When I purchase items, I receive a receipt which lists the name of all the items and their price (including tax), finishing with the total cost of the items, and the total amounts of sales taxes paid. The rounding rules for sales tax are that for a tax rate of n%, a shelf price of p contains (np/100 rounded up to the nearest 0.05) amount of sales tax.

**Write an application that prints out the receipt details for shopping baskets.**
The application's code should be **configurable**, **clean**, and **object-oriented**.

## High-Level requirements

This will be a Java application that takes input in the form of line items. Each line item includes the amount, the key types, the full product name, and its original price.

This application needs to keep track of the user's shopping cart which contains a list of all items. The appropriate tax rate for each item needs to be determined.

Upon completion of the program, the receipt needs to show all the items and amounts, the updated price including tax, the total sales tax, and the total amount to be paid.

Optimize the running time for this Java application. Consider what operations need to be performed on the data provided.

# Technical Specification

**Software requirements:** Eclipse IDE
**Language requirement:** Java

- Java classes (6 hours)

## Input configuration

The input should be configurable; the user can decide whether to feed input manually through the console, or provide a .txt file with the list of items.

Regardless of input method, each line item must follow this exact formatting in order for it to be parsed and categorized accurately. (Input is **case-insensitive** for both items from the exemption list and items the user inputs).

Example:

1 imported packet of headache pills at 9.75
[integer amount] [String key type] [String product name] at [double price]

Currently, the only key type recognized is "imported." However, in the future new key types can be added. For example, an item in the future could have the key type "luxury" and have an appropriate luxury tax applied. The product name must be the exact spelling to find it in the exemption list, mentioned below.

## Exemption list configuration

Another configurable feature should be how the program decides what is an exempted item or not. Our Java application comes with a default file that contains a large list of exempted items. When an item is inputted, it must be determined if it qualifies as a food, medical supply, or book to be exempted. We search our default file to check if the user's input is on our list of exempted items. If the user has a new list of exempted items, they should be able to choose their new exempted list instead of our default list of exemptions.

## Main Menu Options

Once the user has configured their preferred input and configuration method, there should be a menu of several options the user can choose from as the program runs. The menu is the same regardless of input and exemption choices. For example, a user could provide a .txt file of items but wants to still add to the shopping cart before saving the receipt.

The menu includes:
1) Add item
2) Remove item
3) Print receipt
4) Display shopping cart
5) Quit and print receipt

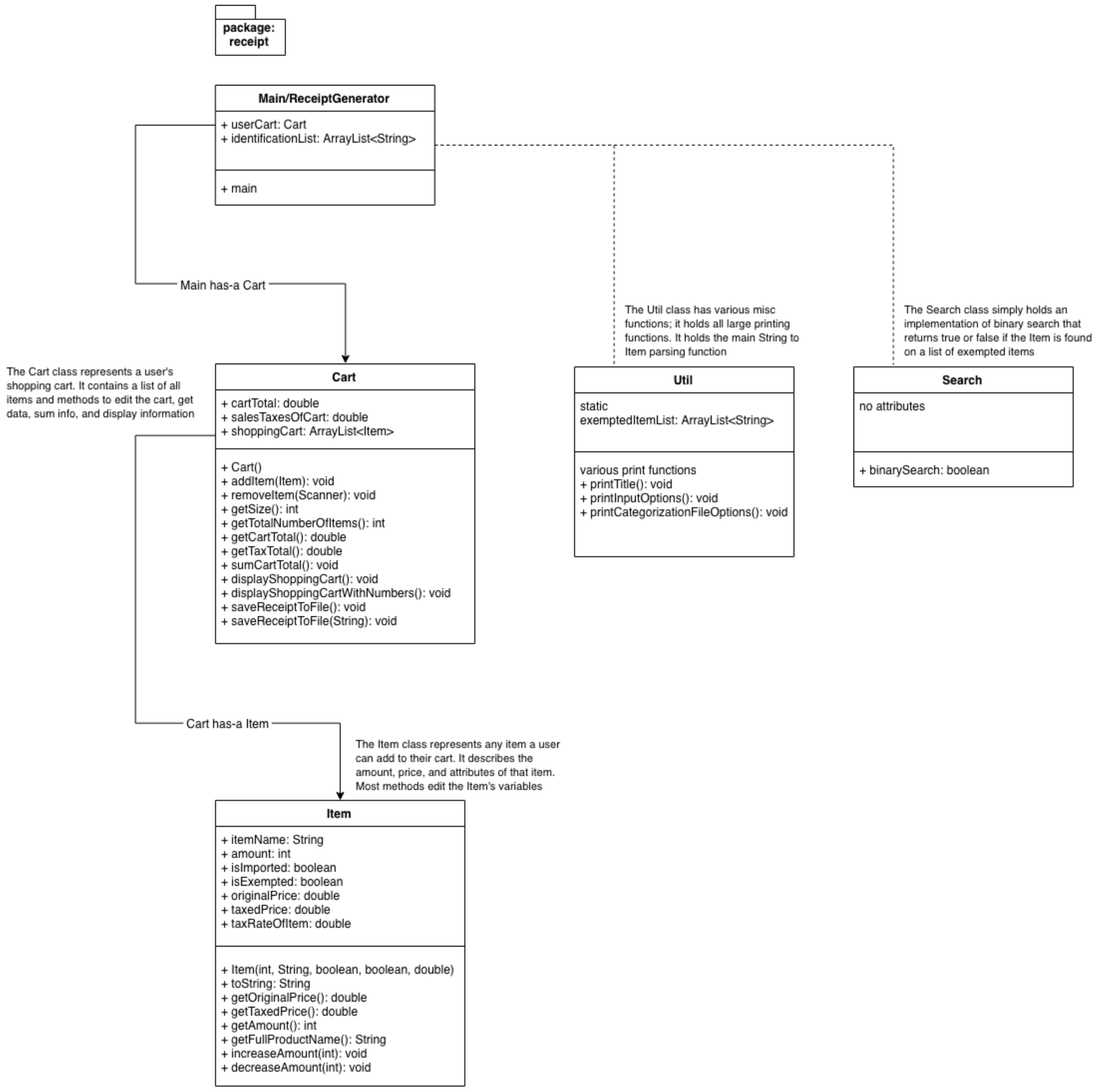
Add item - user inputs 1 line item to their shopping cart.
Remove item - display shopping cart, the user chooses which to remove
Print receipt - choose where to save the receipt to the file
Display shopping cart - display the shopping cart in the console
Quit and print receipt - print the receipt then quit the program

## Output configuration

The output should be configurable as well. The user can decide whether to save the receipt to a .txt file, save to the default file, or view the receipt in the console. To view the receipt, they can choose to view the shopping cart.

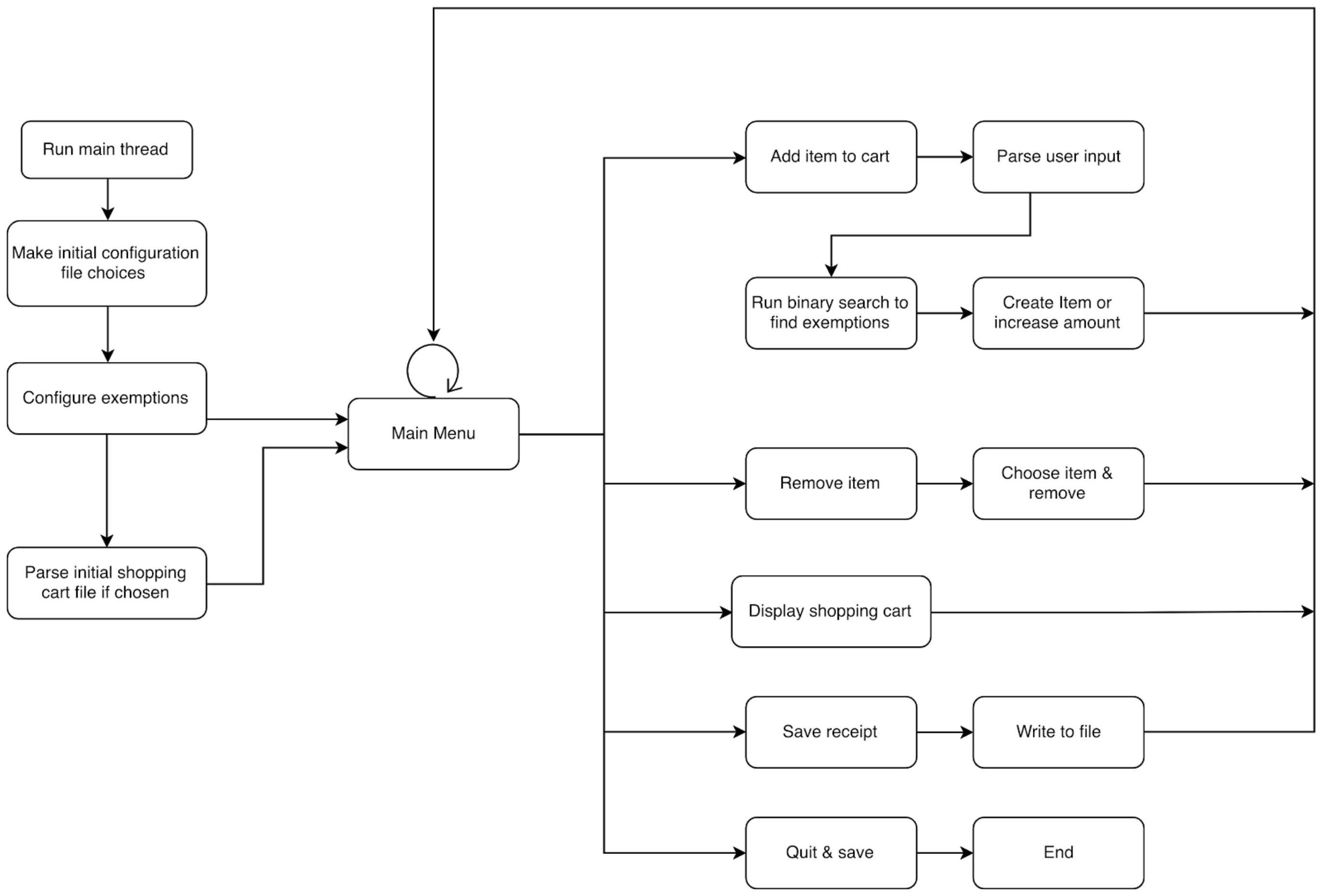
# Detailed Design

The class diagram:



package: receipt
Main/ReceiptGenerator
+ userCart: Cart
+ identificationList: ArrayList<String>
+ main
Main has-a Cart
The Cart class represents a user's shopping cart. It contains a list of all items and methods to edit the cart, get data, sum info, and display information
Cart
+ cartTotal: double
+ salesTaxesOfCart: double
+ shoppingCart: ArrayList<Item>
+ Cart()
+ addItem(Item): void
+ removeItem(Scanner): void
+ getSize(): int
+ getTotalNumberOfItems(): int
+ getCartTotal(): double
+ getTaxTotal(): double
+ sumCartTotal(): void
+ displayShoppingCart(): void
+ displayShoppingCartWithNumbers(): void
+ saveReceiptToFile(): void
+ saveReceiptToFile(String): void
The Util class has various misc functions; it holds all large printing functions. It holds the main String to Item parsing function
Util
static
exemptedItemList: ArrayList<String>
various print functions
+ printTitle(): void
+ printInputOptions(): void
+ printCategorizationFileOptions(): void
The Search class simply holds an implementation of binary search that returns true or false if the Item is found on a list of exempted items
Search
no attributes
+ binarySearch: boolean
Cart has-a Item
The Item class represents any item a user can add to their cart. It describes the amount, price, and attributes of that item. Most methods edit the Item's variables
Item
+ itemName: String
+ amount: int
+ isImported: boolean
+ isExempted: boolean
+ originalPrice: double
+ taxedPrice: double
+ taxRateOfItem: double
+ Item(int, String, boolean, boolean, double)
+ toString: String
+ getOriginalPrice(): double
+ getTaxedPrice(): double
+ getAmount(): int
+ getFullProductName(): String
+ increaseAmount(int): void
+ decreaseAmount(int): void

The state diagram:



## Design Notes

Main.java - handles the main method
Util.java - handles parsing and print methods (static class)
Exemption.java - handles exemption list creation
Cart.java - models a shopping cart, has a List<Item>, sums cart, save receipt
Item.java - models any product with attributes and computes the taxed price for (1) of itself
Search.java - standard implementation of binary search
CustomException.java - used to throw exceptions during parsing

receipt.txt - default save file
defaultID.txt - default list of exempted items

### Side notes

Adding items

- If you add an item with identical price and name to one that is already in your cart, the amount of the item in your cart will be increased. A new Item object will NOT be pushed into the shopping cart ArrayList

## Exception Handling

When programming the Receipt Generator program, it is important to keep in mind what exceptions could be thrown. There could be issues with malformed data from the console and the user provided file. In any case, we do not want the program to end because the user input was malformed and could not be parsed.

### Configuration Error Inputs

If the user chooses an option not listed for the introductory configurations, they will be prompted to choose a valid choice until they do.

#### Configuring to Default Files & Input/Output Handling

In the case that the user provides a shopping cart file that does not exist, an exception will be thrown and the an error message will be displayed to the user that their cart is empty because the file was not found.

In the case that the user provides an exemption file that does not exist, the program will fall back to default exemption file and use that to determine what items are exempted.

**Custom output files do not fall into file not found exceptions as they are created by Java**
**The default output save file is “receipt.txt”**

## Main Menu Error Input

If the user does not choose an option listed on the main menu, they will be prompted to choose again for a valid one.

## Handling Errors with Remove Item

If a user chooses to remove an item from their shopping cart, they first have to choose which item to remove.
Then, they will be prompted to choose an amount to remove.
If they choose an integer outside either of these ranges, they have to pick again.
Choosing the full amount when removing an item will remove the item entirely from the cart.

## Handling Exemption File Parsing

For the purposes of this application, every line in the exemption file is an exempted item. There is no special parsing for the exemption file, the user must make sure that each line is verbatim the name of an exempted product.

## Handling the Parsing of Malformed Input

The CustomException class allows us to notify to the user that there are errors with their input.

Note: Whitespace does not matter for the input
Note: The program is **case-insensitive**, so aPPLe is will be interpreted to be a product with the name apple

Proper input: 1 apple at 2.50

Case 1: Amount is not given/malformed
Examples:

    1apple at 2.50
    apple at 2.50
    1a apple at 2.50

Case 2: “at” is not included
Examples:

    1 apple 2.50
    1 apple a t 2.50
    1 appleat 2.50

Case 3: The price is malformed

1 apple at  2m50

1 apple at two fifty

Case 4: Only 1 word is inputted for the line item

Examples:

apple

Case 5: Leaving out the product description

Examples:

1 at 2.50

The above cases will all throw exceptions but will not end terminate the program.

If the program is parsing from the console, an error message will be displayed.

If the program is parsing a user's premade shopping cart file, error messages will be displayed **in addition** to a message that tells the user which line of the .txt file was skipped so they can manually re-enter it.

Sample error message:

Console:

```
What is the name of the file with items to add to cart? test1.txt
File chosen: test1.txt

Error: "at" was not found, please include "at"
Line 9 was skipped because it was malformed
```

The file test1.txt:

```
 1 1 imported bottle of perfume at 27.99
 2 3 bottle of perfume at 18.99
 3 1 packet of headache pills at 9.75
 4 20 imported box of chocolates at 11.25
 5 20 bagel at 9.00
 6 12 burger at 10.00
 7 1 batter at 1.99
 8 1 biscuit at 0.99
 9 3 cider 5.00
10 1 caviar at 100
```

## Testing

A list of sample test cases considered

### White box testing

- Test case 1
    - No premade cart, default exemptions, check shopping cart, print receipt, quit
- Test case 2
    - No premade cart, custom exemptions, check shopping cart, print receipt, quit

  Test case 3
    - Premade cart, default exemptions, check shopping cart, print receipt, quit
- Test case 4
    - Premade cart, custom exemptions, check shopping cart, print receipt, quit
- Test case 5
    - Premade cart, default exemptions, add item, print receipt, quit
- Test case 6
    - Premade cart, default exemptions, remove item, print receipt, quit
- Test case 7
    - Premade cart, default exemptions, remove all items, print receipt, quit

### Black box testing

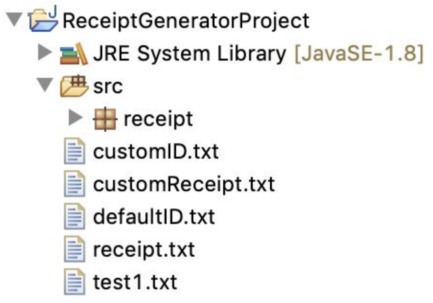
- Test case 1
    - Provided input1.txt, ensure it matches output1.txt
- Test case 2
    - Provided input2.txt, ensure it matches output2.txt
- Test case 3
    - Provided input3.txt, ensure it matches output3.txt
- Test case 4
    - No premade cart, custom exemptions, add exempted item, ensure no sales tax
- Test case 5
    - No premade cart, custom exemptions, add imported item, ensure import tax added

## Implementation

- Diagram the classes
- Write the Java classes
- Prepare the default exemption list
- Test the code with various cases
    - Write custom exception handling for edge cases
- Prepare to turn over to client

## Deployment

1. Push final code to GitHub repository
2. Ensure all code has been pushed
3. Download the .zip file as an attachment in an email or pull the code from the GitHub repository
4. Extract all the files
5. Import the Java project to Eclipse
    a. **Important:** Make sure the .txt files are located inside the Java project folder, but NOT in any folder within that such as src.



```
▼ ReceiptGeneratorProject
    ▶ JRE System Library [JavaSE-1.8]
    ▼ src
        ▶ receipt
    customID.txt
    customReceipt.txt
    defaultID.txt
    receipt.txt
    test1.txt
```

Example project setup, all .java files in the receipt package.
receipt.txt is the default save file for receipts, defaultID.txt is the default exemption source file.

6. Run Main.java from Eclipse

## Usage

Follow the messages on screen. You may optionally provide your own exemption file and your own premade shopping cart. **Ensure** both **receipt.txt** and **defaultID.txt** exist in your project folder outside of src. They are the two default .txt files

## Updates

- The Strings of exempted products is now being stored as a HashSet<String>, after considering the only operation needed on that data is lookup

    - Previously exempted Strings were stored in a ArrayList<String>, and combined with binary search the lookup runtime was O(log n)
    - Updated to HashSet for optimized searching
    - Now the lookup runtime is O(1) for any String
    - Also cuts down the runtime because no sorting takes place now
        - Previously had to sort exemptions for binary search

- The Search.java class is not really needed at the current implementation as contains() from HashSet is faster, O(log n) vs O(1) respectively

## Future

- When adding an item, we have to do a linear search on the shopping cart to check for duplicates (defined as having identical original price and full product name)
- In small shopping carts, this running time is acceptable, but if the shopping cart had an extremely large number of unique items linear search is not optimal
    - Fixes:
        - Write a comparator for the Item class and run merge sort on the Items in the shopping cart
        - Then run binary search on the shopping cart to find duplicates

- Configurable tax rate
    - For the purposes of this application, the tax rate is not configurable. The user should not be able to adjust the tax rate since that would result in inaccurate totals, only an admin should be able to change that when and if the tax rates change.

- User class
    - Instead of instantiating the userCart in the Main class, the userCart could belong to a User and the instantiation of User happens in Main.
        - That way a User has a shopping cart (Cart) and a Cart has Items